ROTOR FATIGUE LIFE PREDICTION AND DESIGN FOR REVOLUTIONARY VERTICAL LIFT CONCEPTS

A Dissertation
Presented to
The Academic Faculty

By

Joseph Nathaniel Robinson

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology

December 2022

Copyright © Joseph Nathaniel Robinson 2022

ROTOR FATIGUE LIFE PREDICTION AND DESIGN FOR REVOLUTIONARY VERTICAL LIFT CONCEPTS

Approved by:

Prof. Dimitri Mavris
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Alexia Payan
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Kyle Collins
Dept. of Aerospace Engineering
Embry-Riddle Aeronautical University

Prof. Marilyn Smith
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: August 6, 2022

Most deadly errors arise from obsolete assumptions.

Frank Herbert, *Children of Dune*

To Rachel

ACKNOWLEDGEMENTS

Although only one name is listed as the author of this thesis, anyone who has gone through this process will know that it is impossible to complete it alone. I would like to begin this document by thanking the individuals without whom this work would have never been completed.

First, I need to acknowledge my defense committee members. My advisor, Prof. Dimitri Mavris, taught me how to think about aerospace problems from a designer's perspective and helped me refocus on the big picture when I got too excited about some specific technological challenge. Dr. Alexia Payan was instrumental in designing and executing my experiments and introduced me to the exciting world of rotorcraft flight safety. Prof. Kyle Collins saw my potential as a first-year grad student and gave me the skills I needed to succeed in rotorcraft design and analysis. Prof. Marilyn Smith introduced me to aeroelasticity as an undergrad and welcomed me into her own lab to work on fascinating and complex problems. Prof. Daniel Schrage provided me with a wealth of practical resources on helicopter design, many of which have been cited in this document.

Since this research is computational in nature, I want to thank the software developers who went above and beyond to make sure I was able to use their tools. Specifically, Allan Wood and Prof. Wenbin Yu, developers of VABS and PreVABS, were both extremely helpful and generous with their time. Larry Meyn, developer of RCOTOOLS, was quick to update his code when I found a bug that would have brought my progress to a halt. Dr. Régis Lebrun, a developer of OpenTURNS, assisted with my reliability analysis multiple occasions. I would also like to acknowledge all the other developers, contributors, and publishers of the open source software used in this research.

My fellow students at ASDL were essential in providing support and feedback throughout the process. Dr. Eric Inclan, Alexander Braafladt, Jeffrey Pattison, Rahul Rameshbabu, and Nikhil Iyengar all provided genuinely meaningful suggestions while listening to my

presentation far more times than should be required of any one person. Max-Daniel Sokollek wrote some excellent syntax highlighting rules for RCAS in `vim`, which I used throughout this research. A special thanks goes to Adrienne Durham, the Academic Program Manager, for helping me navigate the academic and bureaucratic requirements of graduate school.

I would also like to thank those who were not directly involved in my thesis research but had a positive influence on my career. Specifically, Charles Johnson of the FAA and Dr. Aditya Saraf of ATAC Corporation both played significant roles in my professional development and I will be endlessly grateful for their mentorship.

However, those who had the most influence on my success are my family members. My parents, Bert and Teresa Robinson, taught me how to write, and how to fret over spelling and grammar, from a young age. Unfortunately for my readers, they were not as concerned with the art of brevity, as evidenced by the length of this document. My brother, Nicholas Robinson, convinced me that MATLAB might not be the only programming language worth learning and inspired me on my (unrelated to this thesis but nearly as difficult) bread-making journey.

Finally, I need to thank my wonderful girlfriend, Rachel Martin, who has been a constant source of encouragement, inspiration, love, and support. No matter what happens in my career, I will always know that pursuing a PhD was worthwhile because, had I not enrolled in AE 6230 Structural Dynamics in the fall of 2018 to study for qualifying exams, I would not have met her. I can't wait to see where our lives together go from here.

Sincerely,
Joseph Nathaniel Robinson
Georgia Institute of Technology
July 2022

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Rotorcraft design and analysis is an inherently multidisciplinary effort. Equations, methods, and algorithms from a large body of scientific and engineering research have been used in this thesis. An effort was made to keep the nomenclature in this document consistent with its original authors' nomenclature, while also avoiding confusing conflicts. In some cases, one symbol or acronym may represent multiple concepts, or the same concept may be represented by multiple symbols. The author hopes that the meaning will be obvious in context, but apologizes in advance for any confusion.

Acronyms and Initialisms

AC . . . . . aerodynamic center

AHS . . . . American Helicopter Society

AMRDEC . Aviation and Missile Research, Development, and Engineering Center

ANN . . . . artifical neural network

ART . . . . . active rotor technology *or* Advanced Rotorcraft Technology, Inc.

BET . . . . . blade-element theory

CAMRAD . Comprehensive Analytical Model of Rotorcraft Aerodynamics and Dynamics

CBEMT . . . combined blade-element–momentum theory

CBM . . . . condition-based maintenance

CCD . . . . central composite design

CDF . . . . . cumulative density function

CFD . . . . . computational fluid dynamics

CG . . . . . center of gravity

COV . . . . coefficient of variation

CSD . . . . . computational structural dynamics

CTM . . . . Cost Too Much

CTOL . . . . conventional takeoff and landing

DES . . . . . discrete event simulation

DGW . . . . design gross weight

DoD . . . . . Department of Defense

DOE . . . . design of experiments

DOF . . . . degree of freedom

DS . . . . . directional simulation

MCP . . . . maximum continuous power

eVTOL . . . electric VTOL

FAA . . . . . Federal Aviation Administration

FORM . . . first-order reliability method

GA . . . . . general aviation

GAG . . . . ground–air–ground

GECB . . . . geometrically-exact composite beam

GPM . . . . gaussian process model

GRP . . . . . Sikorsky Generalized Rotor Performance

HCF . . . . . high cycle fatigue

HHC . . . . higher-harmonic control

IHST . . . . International Helicopter Safety Team

IRP . . . . . intermediate rated power

JHIMDAT . . Joint Helicopter Implementation Measurement Data Analysis Team

JHSAT . . . Joint Helicopter Safety Analysis Team

LCC . . . . . life-cycle costs

LCF . . . . . low cycle fatigue

LHS . . . . . Latin hypercube simulation

FVL . . . . . Future Vertical Lift

MC	Monte Carlo simulation
MDA	multidisciplinary analysis
MFOP	maintenance-free operating period
MRP	maintenance recovery period
MTBF	mean time between failures
RAF	Royal Air Force
NASA	National Aeronautics and Space Administration
NDARC . . .	NASA Design and Analysis of Rotorcraft
NLB	nonlinear beam
NTSB	National Transportation Safety Board
O&S	operating and support
OML	outer mold line
OpenMDAO	An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization
OpenTURNS	An Open-Source Initiative for the Treatment of Uncertainties, Risks, 'n' Statistics
PDF	probability density function
QMC	quasi-Monte Carlo
R&D	research and development
RAM-C . . .	reliability, availability, maintainability, and cost
RCAS	Rotorcraft Comprehensive Analysis System
RCOTOOLS	Rotorcraft Optimization Tools
RDT&E . . .	research, development, test, and evaluation
RFP	request for proposals
ROC	Rate of climb
ROM	reduced order model
RQ	research question
RSE	response surface equation

RSM . . . . response surface method

SLS . . . . . sea level standard atmosphere

SMA . . . . scheduled maintenance action

SMR . . . . single main rotor

SORM . . . second-order reliability method

SSE . . . . . sum of squared errors *or* sum of squared residuals

SST . . . . . total sum of squares

TBO . . . . time between overhauls

TOS . . . . . top-of-scatter

TTF . . . . . time to failure

TTR . . . . . time to repair

UAM . . . . urban air mobility

UMA . . . . unscheduled maintenance action

VABS . . . . Variational Asymptotic Beam Sectional Analysis

VAM . . . . variational asymptotic method

VTOL . . . . vertical takeoff and landing

X2TD . . . . Sikorsky X2 Technology Demonstrator

Symbols and Functions

a . . . . . crack length

A_o . . . . . operational availability

$\boldsymbol{b}$ . . . . . least-squares estimates of the β coefficients

b . . . . . activation function bias

C . . . . . cumulative fatigue damage

$\mathcal{D}(a)$ . . . . Dirac distribution at $x = a$

$\mathcal{D}$ . . . . . domain

D . . . . . distance

$\mathrm{E}[V]$ . . . . . expected value of the random variable V

e . . . . . residual

E_{ii} . . . . . Young's modulus along the i axis

EA . . . . . axial stiffness

EI_i . . . . . bending stiffness about the x_i-axis

F . . . . . force

$F_V(v)$ . . . . cumulative distribution function of the random variable V

$f_V(v)$ . . . . probability density function of the random variable V

$f(v)$ . . . . . function of the variable(s) v

$G(\boldsymbol{V})$ . . . . performance function

G_{ij} . . . . . Shear modulus in the direction j on the plane whose normal is i

GJ . . . . . torsional stiffness

H . . . . . generalized stiffness

$H(\boldsymbol{U})$ . . . . standardized performance function

h_d . . . . . Density altitude

$\boldsymbol{I}(v)$ . . . . . indicator function

I . . . . . moment of inertia

i . . . . . moment of inertia per unit length

$\mathcal{K}$ . . . . . kinetic energy per unit length

K . . . . . stress intensity factor

$k(\boldsymbol{v}, \boldsymbol{v}')$ . . . covariance function

$K(V, V')$ . . covariance matrix

k, n . . . . . number of elements in a set

ℓ . . . . . length

$\mathcal{L}(\mu_\ell, \sigma_\ell)$ . . Lognormal distribution whose underlying normal distribution is $\mathcal{N}\left(\mu_\ell, \sigma_\ell^2\right)$

M . . . . . moment

m . . . . . mass

$\mathcal{N}(\mu, \sigma^2)$ . . normal distribution of mean μ and variance σ^2

N . . . . . cycles to failure

$\Pr(E)$ . . . . probability of the random event E

P^* . . . . . most-probable failure point

R . . . . . stress ratio *or* resistance of a structure

r . . . . . load spectrum repetitions

$\mathrm{Std}(V)$ . . . standard deviation of the random variable V

S . . . . . load *or* stress *or* strain *or* strength

S_0 . . . . . initial static strength

S_r . . . . . residual strength

S_∞ . . . . . endurance limit

S_u . . . . . ultimate static strength

$\mathcal{T}(a, b, c, d)$. Trapezoidal distribution with vertices at $x = a$, $x = b$, $x = c$, and $x = d$

T . . . . . thrust

t . . . . . time

T_{IP} . . . . . isoprobabilistic transformation

$\mathcal{U}$ . . . . . strain energy per unit length

$\mathcal{U}(a, b)$ . . . Uniform distribution from $x = a$ to $x = b$

U . . . . . independent standardized normal random variable

$\mathrm{Var}(V)$ . . . variance of the random variable V

V . . . . . airspeed

$\mathcal{W}$ . . . . . work per unit length

$\mathcal{W}(\alpha, \beta, \gamma)$. Weibull distribution with shape parameter α, scale parameter β, and location parameter γ

W . . . . . Gross weight

w . . . . . weight parameter

X . . . . . random variable

x . . . . . input or independent variable

y . . . . . output or dependent variable

α . . . . . Weibull distribution shape parameter

β . . . . . reliability index *or* Weibull distribution scale parameter

$\boldsymbol{\beta}$ . . . . . coefficients of the RSE

γ . . . . . engineering strain *or* Weibull distribution location parameter

Δ . . . . . Change in a quantity

δ . . . . . Langrangean variation

$\mathit{\Delta}(a, m, b)$ . . Triangular distribution with a minimum $x = a$, a maximum at $x = b$, and a mode at $x = m$

ϵ . . . . . tail rotor cant angle

ε . . . . . error of a surrogate model

θ . . . . . GPM hyperparameter

κ . . . . . curvature

μ . . . . . mean *or* mass per unit length

ν . . . . . strength degradation parameter

ν_{ij} . . . . . Poisson's ratio corresponding to a contraction in direction j when extension is applied in direction i

σ . . . . . standard deviation

σ_f . . . . . signal standard deviation

$\phi(v)$ . . . . . activation function in a surrogate model

$\Phi_n(v)$ . . . . cumulative distribution function of the n-dimensional standard normal

$\phi_n(v)$ . . . . probability density function of the n-dimensional standard normal

χ^2_n . . . . . chi-squared distribution with n degrees of freedom

ψ . . . . . Rotor blade azimuth angle

ω . . . . . Turn rate

Superscripts and Diacritics

$\overline{\cdot}$ . . . . . mean

$\widehat{\cdot}$ . . . . . approximation or estimate

$\top$ . . . . . transpose

Subscripts

0 . . . . . initial

$1, 2, 3$ . . . . axes in a Cartesian coordinate system

amp . . . . . amplitude

B . . . . . blade

b . . . . . ballast

eq . . . . . equivalent

f . . . . . final *or* failure

h . . . . . horizontal

hs . . . . . hydrostatic

i, j . . . . . counters

l . . . . . lower

max . . . . . maximum

mean . . . . mean

min . . . . . minimum

NE . . . . . never-exceed

range . . . . range

s . . . . . survival *or* signed

t . . . . . total

tr . . . . . training set

u . . . . . upper

v . . . . . vertical

vm . . . . . von Mises

x, y, z . . . . axes in a rotor blade or vehicle coordinate system

Units

deg . . . . . degree

d . . . . . calendar day, equal to 24 h

FH . . . . . fight hour

ft . . . . . foot

Hz . . . . . hertz

HP . . . . . horsepower

h . . . . . hour

in . . . . . inch

kip . . . . . kip, equal to 1000 lbf

ksi . . . . . kip/in^2

lb . . . . . pound-mass

lbf . . . . . pound-force

MMH . . . . maintenance man-hour

MPa . . . . . megapascal

m . . . . . meter

min . . . . . minute

rev . . . . . revolution

RPM . . . . revolutions per minute

s . . . . . second

slug . . . . . slug, equal to 32.17404 lb

° . . . . . degree

SUMMARY

Despite recent technological advancements, rotorcraft still lag behind their fixed-wing counterparts in the areas of flight safety and operating cost. Competition with fixed-wing aircraft is difficult for applications where vertical takeoff and landing (VTOL) capabilities are not required. Both must be addressed to ensure the continued competitiveness of vertical lift aircraft, especially in the context of new military and civilian rotorcraft programs such as Future Vertical Lift and urban air mobility, which will require orders-of-magnitude improvements in reliability, availability, maintainability, and cost (RAM-C) metrics.

Lifecycle costs and accident rates are strongly driven by scheduled replacement or failure of flight-critical components. Rotor blades are life-limited to ensure that they are replaced before fatigue damage exceeds critical levels, but purchasing new blades is extremely costly. Despite aggressive component replacement times, fatigue failure of rotor blades continues to account for a significant proportion of inflight accidents. Fatigue damage in rotorcraft is unavoidable due to the physics of rotary-wing flight, but new engineering solutions to improve fatigue life in the rotor system could improve rotorcraft operating costs and flight safety simultaneously.

Existing rotorcraft design methods treat fatigue life as a consequence, rather than a driver, of design. A literature review of rotorcraft design and fatigue design methods is conducted to identify the relevant strengths and weaknesses of traditional processes. In rotorcraft design, physics-based rotor design frameworks are focused primarily on fundamental performance analysis and do not consider secondary characteristics such as reliability or fatigue life. There is a missing link between comprehensive rotor design frameworks and conceptual design tools that prevents physics-based assessment of RAM-C metrics in the early design stages.

Traditional fatigue design methods, such as the safe life methodology, which applies the Miner's rule fatigue life prediction model to rotorcraft components, are hindered by a lack of

physics-based capabilities in the early design stages. An accurate fatigue life quantification may not be available until the design is frozen and prototypes are flying. These methods are strongly dependent on extrapolations built on historical fatigue data, and make use of deterministic safety factors based on organizational experience to ensure fatigue reliability, which can lead to over-engineering or unreliable predictions when applied to revolutionary vertical lift aircraft.

A new preliminary fatigue design methodology is designed to address these concerns. This methodology is based on the traditional safe life methodology, but replaces several key elements with modern tools, techniques, and models. Three research questions are proposed to investigate, refine, and validate different elements of the methodology. The first research question addresses the need to derive physics-based fatigue load spectra more rapidly than modern comprehensive analysis tools allow. The second investigates the application of different probabilistic reliability solution methods to the fatigue life substantiation problem. The third question tests the ability of the preliminary fatigue design methodology to evaluate the relative impact of common preliminary fatigue design variables on the probability of fatigue failure of a conceptual helicopter's rotor blade.

Hypotheses are formulated in response to each research question, and a series of experiments are designed to test those hypotheses. In the first experiment, a multi-disciplinary analysis (MDA) environment combining the rotorcraft performance code NDARC, the comprehensive code RCAS, and the beam analysis program VABS, is developed to provide accurate physics-based predictions of rotor blade stress in arbitrary flight conditions. A conceptual single main rotor transport helicopter based on the UH-60A Black Hawk is implemented within the MDA to serve as a test case. To account for the computational expense of the MDA, surrogate modeling techniques, such as response surface equations, artificial neural networks, and Gaussian process models are used to approximate the stress response across the flight envelope of the transport helicopter. The predictive power and learning rates of various surrogate modeling techniques are compared to determine which is the

most suitable for predicting fatigue stress. Ultimately, shallow artificial neural networks are found the provide the best compromise between accuracy, training expense, and uncertainty quantification capabilities.

Next, structural reliability solution methods are investigated as a means to produce high-reliability fatigue life estimates without requiring deterministic safety factors. The Miner's sum fatigue life prediction model is reformulated as a structural reliability problem. Analytical solutions (FORM and SORM), sampling solutions (Monte Carlo, quasi-Monte Carlo, Latin hypercube sampling, and directional simulation), and hybrid solutions (importance sampling) are compared using a notional fatigue life problem. These results are validated using a realistic helicopter fatigue life problem which incorporates the fatigue stress surrogate model and is based on a probabilistic definition of the mission spectrum to account for fleet-wide usage variations. Monte Carlo simulation is found to provide the best performance and accuracy when compared to the exact solution.

Finally, the capabilities of the preliminary fatigue design methodology are demonstrated using a series of hypothetical fatigue design exercises. First, the methodology is used to predict the impact of rotor blade box spar web thickness on probability of fatigue failure. Modest increases in web thickness are found to reduce probability of failure, but larger increases cause structural instability of the rotor blade in certain flight regimes which increases the fatigue damage rate. Next, a similar study tests the impact of tail rotor cant angle. Positive tail rotor cant is found to improve fatigue life in cases where the center of gravity (CG) of the vehicle is strongly biased towards the tail, but is detrimental if the CG is closer to the main rotor hub station line. Last, the effect of design mission requirements like rate of climb and cruising airspeed is studied. The methodology is not sensitive enough to predict the subtle impact of changes to rate of climb, but does prove that a slower cruising airspeed will decrease probability of fatigue failure of the main rotor blade.

The methodology is proven to be capable of quantifying the influence of rotor blade design variables, vehicle layout and configuration, and certain design mission requirements,

paving the way for implementation in a rotorcraft design framework. This thesis ends with suggestions for future work to address the most significant limitations of this research, as well as descriptions of the tasks required to apply the methodology to conventional rotorcraft or conceptual revolutionary vertical lift aircraft.

CHAPTER 1
MOTIVATION AND RESEARCH OBJECTIVES

Rotary-wing vehicles have made incredible technological strides since the introduction of the first mass-produced helicopter nearly eight decades ago. The ensuing years have seen significant improvements in efficiency, speed, endurance, passenger comfort, utility, maneuverability, and more. However, rotorcraft still lag behind their fixed-wing counterparts in a number of areas. Key deficiencies include flight safety, operating costs, purchase price, noise, and vibrations [1].

In spite of these drawbacks, rotorcraft remain competitive because their vertical takeoff and landing (VTOL) capabilities enable certain missions, such as medical evacuation, offshore support, sight-seeing, and urban mobility, that would be impossible or impractical with conventional takeoff and landing (CTOL) vehicles. Nevertheless, it is necessary to improve the affordability and flight safety of rotary-wing aircraft to ensure their continuing viability in a sea of increasingly versatile and capable aerospace vehicles [1].

Despite the vast expenditures committed to the research, design, testing, certification, and production of vertical lift aircraft, operating costs are the primary driver of rotorcraft life-cycle costs (LCC) [2]. This greatly diminishes the affordability of owning and operating a fleet of rotary-wing aircraft. Comparatively, fixed-wing aircraft outrank rotorcraft in nearly every area of operating and support (O&S) cost [3]. Furthermore, rotary-wing vehicles suffer a proportionally higher rate of fatal accidents than their fixed-wing counterparts [4, 5]. In addition to the tragic loss of life, these accidents damage the public perception of rotorcraft safety and decrease public acceptance of vertical lift aircraft, regardless of their role. Both of these aspects must be improved if rotary-wing aircraft are to remain relevant and become more widely adopted.

In this chapter, helicopter life-cycle costs and accident rates are investigated thoroughly.

Common areas of improvement for each are identified and discussed in detail. Throughout the chapter, key observations are highlighted. Later, these observations are used to define research objectives for this thesis.

1.1 Life-Cycle Costs

In order to address the high cost of rotorcraft operations, the individual elements of LCC must first be decomposed and examined. Based on a 1975 survey of United States Army helicopters, Reddick [2] estimates O&S costs comprise 75% of LCC, with acquisition costs, which includes research and development (R&D) and production, making up the remaining 25%. Of the O&S costs, 75% are due to maintenance and parts, with personnel (10%) and consumables (15%) accounting for the remainder. Thus, maintenance and parts makes up a majority (56.25%) of LCC.

1.1.1 System-Level Decomposition

Maintenance costs can be further decomposed into the major systems on the vehicle. Reddick presents a summary of direct support maintenance costs for individual components on the Boeing CH-47 Chinook cargo helicopter. On this vehicle, the *rotor* (28.8%), *power plant* (27.4%), and *transmission* (11.5%) systems account for the majority of maintenance costs. The same three systems are ranked similarly on the Bell UH-1 utility helicopter, although the power plant replaces the rotor as the highest-cost system.

Within the rotor system, the blades account for nearly all (80%) of the maintenance cost. Rotor blade maintenance drivers include, in order of significance, foreign object damage, cracking, combat damage, overstressing, and debonding, among other reasons. Reddick attributes the high rate of blade damage to the extreme aerodynamic environment within which the blades operate.

Conversely, the maintenance cost drivers for the power plant are not dominated by any one component of the system. Again, foreign object damage is the most significant

maintenance cost contributor, followed closely by issues arising to improper maintenance, seal leakage, and erosion. However, only foreign object damage exceeds 10% of the total cost contribution, and a large number of maintenance items contribute less than 5%. Reddick posits that the lack of a single significant contributor decreases the efficacy of power plant maintenance improvement programs.

Maintenance costs incurred by the transmission system are largely driven by scheduled removal and replacement of wear items such as bearings and gears. An effective maintenance improvement program could involve advancing the design of transmissions to reduce wear and increase the time between overhauls (TBO), which would significantly reduce maintenance costs for this system.

1.1.2 Maintenance Actions

Harris [3] arrives at similar conclusions to Reddick and highlights the drastically higher maintenance cost of rotorcraft compared to their fixed-wing competitors. Harris argues that the primary operating cost drivers of rotorcraft are *maintenance actions* and replacement of *life-limited parts*.

Maintenance actions are frequently categorized as scheduled maintenance actions (SMAs) or unscheduled maintenance actions (UMAs). SMAs must occur at specific intervals specified by the manufacturer. For example, the maintenance manual for a Robinson R22 GA helicopter specifies a number of recurring maintenance tasks with intervals ranging from 29 FH to 2200 FH and four months to 12 years. Establishing the frequency of UMAs for a given vehicle is difficult, but Harris recommends a rule of thumb of 1 MMH of unscheduled maintenance per MMH of scheduled maintenance. SMAs and UMAs contribute to the O&S cost of the vehicle through part costs and labor costs.

Life-limited parts are those that must be replaced at specific intervals to maintain airworthiness of the vehicle. In most helicopters, many elements of the main rotor system are limited by their fatigue life, which is a period of safe operation after which a component

has a certain risk of failing due to fatigue fracture. For example, several flight-critical components of a Robinson R22 must be replaced at intervals ranging from 2200 FH to 6260 FH [3]. Predictably, the cost of replacement components is high, and replacement costs over several years of operation can easily rival the initial purchase price of the vehicle Thus, part replacement costs also contribute significantly to overall O&S costs.

1.1.3 Discussion

Harris concludes that high operating costs inherent to rotorcraft prevent direct competition between rotary-wing and fixed-wing vehicles in most cases. In the case of a hypothetical commercial airline operating a rotorcraft fleet, high ticket prices, driven primarily by O&S expenses, would make operations infeasible without government subsidies. In fact, this exact scenario played out in the mid-20th century when the Federal Aviation Administration (FAA) supported the establishment of three helicopter airlines with operating concepts notably similar to the modern concept of urban air mobility (UAM).[1] These airlines were never able to turn a profit without government subsidies and each ceased operations shortly after the subsidies were ended [3].

Conversely, in the early 20th century, fixed-wing airlines were able to successfully transition from businesses supported by government subsidies and airmail contracts to independent businesses focused primarily on carrying passengers. Thus, rotorcraft are restricted to applications in which their VTOL capabilities are mandatory; otherwise, lower O&S costs will force operators to choose fixed-wing aircraft. This leads to Observation 1.1:

Observation 1.1

High LCC associated with rotorcraft are driven by part replacement and repair requirements. Rotor blades make up a significant portion of this cost due to their extreme operating environment, high purchase price, and life-limited nature.

[1] UAM is a concept of operations which consists of small VTOL aircraft carrying passengers a short distance in an urban environment.

In the conclusion to his report, Reddick suggests increasing focus on vehicle reliability and maintainability during the development phase, which could reduce LCC despite increasing design costs. This effort must be accompanied by the development of more robust methods to predict O&S costs in the preliminary design phase rather than by extrapolating from historical operational data.

Similarly, Harris notes the difficulty of establishing realistic expectations for the time and cost required to maintain a helicopter fleet. Much of this difficulty is due to a lack of comprehensive historical financial data for helicopter operations and the challenge of comparing operational experience across a diverse helicopter fleet. Thus, the fundamental problem of high O&S costs is compounded by the difficulty of designing new rotary-wing vehicles that significantly improve upon their predecessors in terms of affordability due, in part, to the lack of appropriate design tools.

1.2 Accident Rates

Statistics gathered by the FAA indicate that helicopters suffer an average fatal accident rate of 1.02×10^{-5} accidents/FH, significantly greater than the overall (fixed- and rotary-wing) fatal accident rate of 0.84×10^{-5} accidents/FH [5].[2] According to Lombardo [4], helicopters experience a greater number of in-flight accidents than fixed wing aircraft, relative to the number of each type operating. A portion of this discrepancy can be explained by the use case: helicopters typically fly more risky missions than fixed-wing aircraft, including search and rescue, emergency medical services, offshore support, and troop insertion. These missions frequently occur in adverse weather conditions or at low altitudes, which are inherently dangerous. However, some causes of the excessive accident rate of rotorcraft may be traceable to the fundamental nature of the vehicles themselves. This section presents a more detailed analysis of helicopter accident rates and causes.

[2] Generally, accident analyses report accident rates as, for example, *1 accident per 100,000 FH*. In an effort to be more consistent with best practices when using units in a written context, this document will use the form 1×10^{-5} *accidents/FH*. The meaning is identical.

1.2.1 JHSAT Accident Analysis

A recent statistical analysis of rotorcraft accidents was published by the U.S. Joint Helicopter Safety Analysis Team (JHSAT) covering 523 helicopter accidents in 2000, 2001, and 2006 [6, 7]. This dataset includes rotorcraft accidents in the U.S. only. The JHSAT reports a total rotorcraft accident rate of 9.1×10^{-5} accidents/FH, 8.0×10^{-5} accidents/FH, and 5.7×10^{-5} accidents/FH for these three years, respectively, and notes that for 15 years prior to the publication of the report the rotorcraft accident rate has remained relatively constant, indicating stagnation in the safety record. Additionally, 49% of all rotorcraft accidents in the years studied included at least one injury, with 16% resulting in fatal injury. In total, 483 victims were injured in just three years, including 151 fatalities.

The JHSAT grouped these accidents into a number of different categories, including industry, activity, occurrence, and phase of flight. By industry, the most accidents occurred in the *personal/private* and *instructional/training* categories (18.5% and 17.9% of accidents, respectively), indicating that general aviation (GA) helicopters account for a large number of total accidents. In the occurrence category, which identifies the immediate cause of the accident, *loss of control* (41%), *autorotations* (32%), and *system component failures* (28%) account for the bulk of all accidents.[3] Thus, the majority of rotorcraft safety incidents may be attributed to pilot error; however, a significant proportion of accidents are beyond the pilots' control.

Loss of control occurrences include any accident in which the pilot loses control over the aircraft. According to the JHSAT report, the three most prominent causes of loss of control accidents are performance management issues (e.g., insufficient rotor power or RPM), dynamic rollover, and exceeding the established operating limits of the vehicle. These categories make up 15%, 6%, and 5%, respectively, of the total 523 accidents examined in the report.

[3]Each accident can have multiple assigned occurrences, which is why the sum of these percentages exceeds 100%.

Autorotation accidents typically occurred when the pilot(s) performed the maneuver incorrectly, either during training or during an actual in-flight emergency. The JHSAT attributes these incidents primarily to lack of experience on behalf of the pilot(s) or, in the case of training, the instructor's failure to intervene and prevent the accident.

System component failures are incidents in which the failure of a specific system on the vehicle leads to an accident. Organized by system, the system component failures occur primarily in the powerplant (45.8%), the rotor (38.2%), and the airframe (16.0%). Organized by initiating event, the bulk of system component failures (50.7%) are related to improper maintenance procedures, while lesser amounts are due to manufacturing error, pilot error, and unknown causes (21.5%, 18.1%, and 9.8%, respectively).

1.2.2 JHIMDAT Accident Analysis

A subsequent follow-on analysis by the U.S. Joint Helicopter Implementation Measurement Data Analysis Team (JHIMDAT) statistically compares the JHSAT data with new data collected during the three years from 2009 to 2011, which included 415 helicopter accidents [8, 9]. The JHIMDAT attempted to determine if there were any statistically significant[4] differences in the accident categories compared to the JHSAT dataset, thus indicating if corrective actions were improving or failing to improve the flight safety record.

The JHIMDAT found that the number of GA (*personal/private* and *instructional/training*) accidents increased, but the increase was not statistically significant. These classifications remained the most prominent within the industry category. Within the occurrence category, the number of loss of control accidents significantly increased, the number of autorotation accidents remained nearly constant, and the number of system component failures significantly decreased. Nevertheless, at 21.4% of all accidents, component failures remain a substantial portion of the total accident count.

Within the system component failure category, neither of the three systems discussed

[4]Note that the JHIMDAT used a particularly conservative p-value of 0.01 in this analysis.

previously showed any significant difference between the two datasets. However, when categorized by initiating event, the number of accidents attributed to manufacturing error significantly decreased, the number attributed to maintenance error and pilot error remained roughly constant, and, most worryingly, the number assigned to unknown causes increased sharply. Overall, the JHIMDAT concluded that, while the overall number of accidents decreased, the proportion of accidents in most categories either remained stagnant or increased, with very few categories showing significant improvements in the number of incidents.

1.2.3 NTSB Accident Data

Harris's own analysis of rotorcraft accidents covers a broader span of time than that of the JHAST/JHIMDAT [10]. Using National Transportation Safety Board (NTSB) data, Harris studied U.S. rotorcraft accidents from 1964 to 2011. Overall, the data show that the total number of rotorcraft accidents per year is steadily decreasing despite the increasing size of the rotorcraft fleet and escalating number of hours flown per year, as can be seen in Figure 1.1.

Harris credits the recent significant decrease in accident occurrences to the actions of the International Helicopter Safety Team (IHST), the organization which established the JHSAT and the JHIMDAT. However, the situation is not as optimistic as Figure 1.1 would perhaps make it appear. Harris highlights the appearance of *accident bubbles* that correspond closely to the introduction of new technologies to the helicopter fleet. For example, the total yearly number of accidents rises and falls significantly (relative to a straight downward-sloping trend line) from 1972 to 1986, with a peak during 1980. This correlates closely with the adoption of rotorcraft powered by a single turbine engine; the total number of these vehicles in service grew significantly from 1970 to 1985. Harris asserts that this bubble was a result of the inherent risks associated with new, and potentially poorly understood, technologies and vehicle configurations, a phenomenon known in the field of reliability engineering as

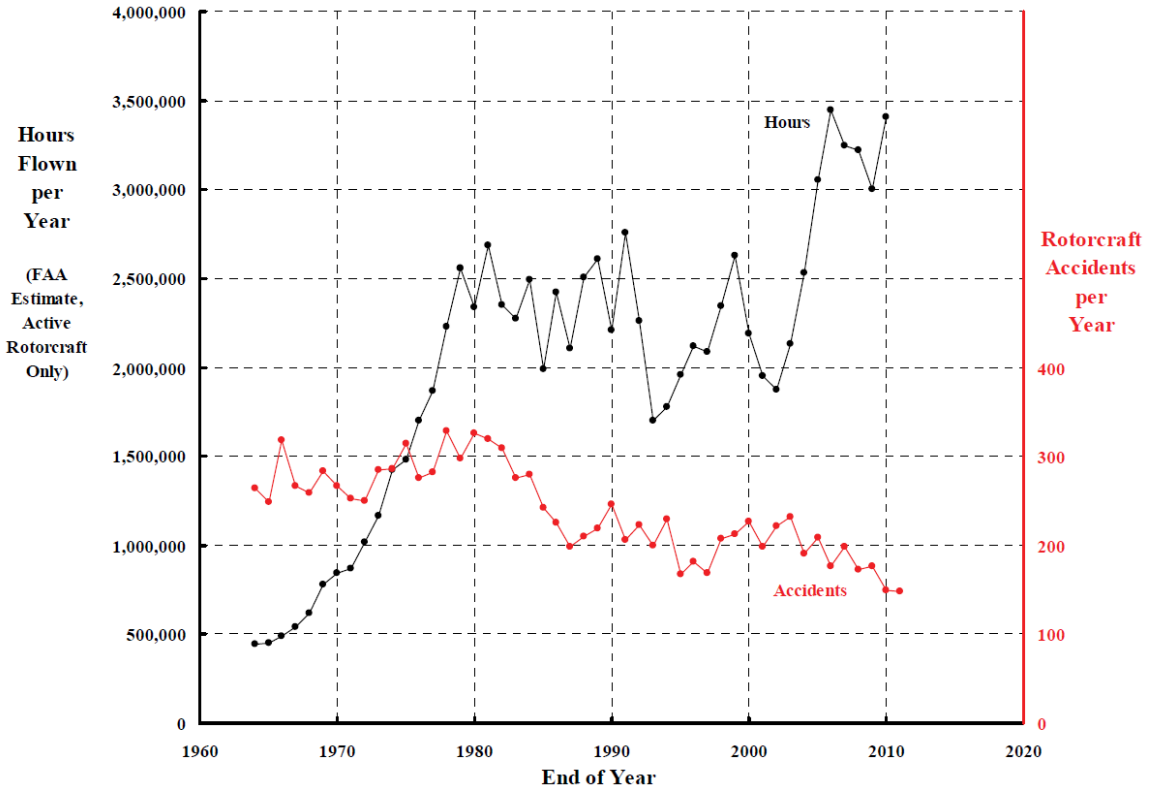


Figure 1.1: Rotorcraft usage and accident numbers from 1964 to 2011, reprinted from Harris [10] with permission.

infant mortality [11]. The possibility of additional accident bubbles bodes poorly for the safety of future vertical lift vehicles, which may use configurations and propulsion systems wildly different from those of the present day.

The NTSB also categorizes rotorcraft accidents by first occurrence, in a manner similar to the JHSAT report. Over the nearly 50 year period studied by Harris, the four most prominent accident causes were as follows: loss of engine power (26.6%), loss of control (15.4%), in-flight collision (14.76%), and airframe/component/system failure/malfunction (12.8%). Note that the NTSB first occurrence classifications do not align exactly with the JHSAT accident occurrences,[5] but in both studies loss of control and system component failures are among the most common causes. Figure 1.2 plots the change in accident proportions within each NTSB classification throughout the same time span.

[5]For example, accidents attributed to "autorotation" by the JHSAT are likely attributed to "loss of engine power" by the NTSB, since autorotation typically follows the loss of engine power.

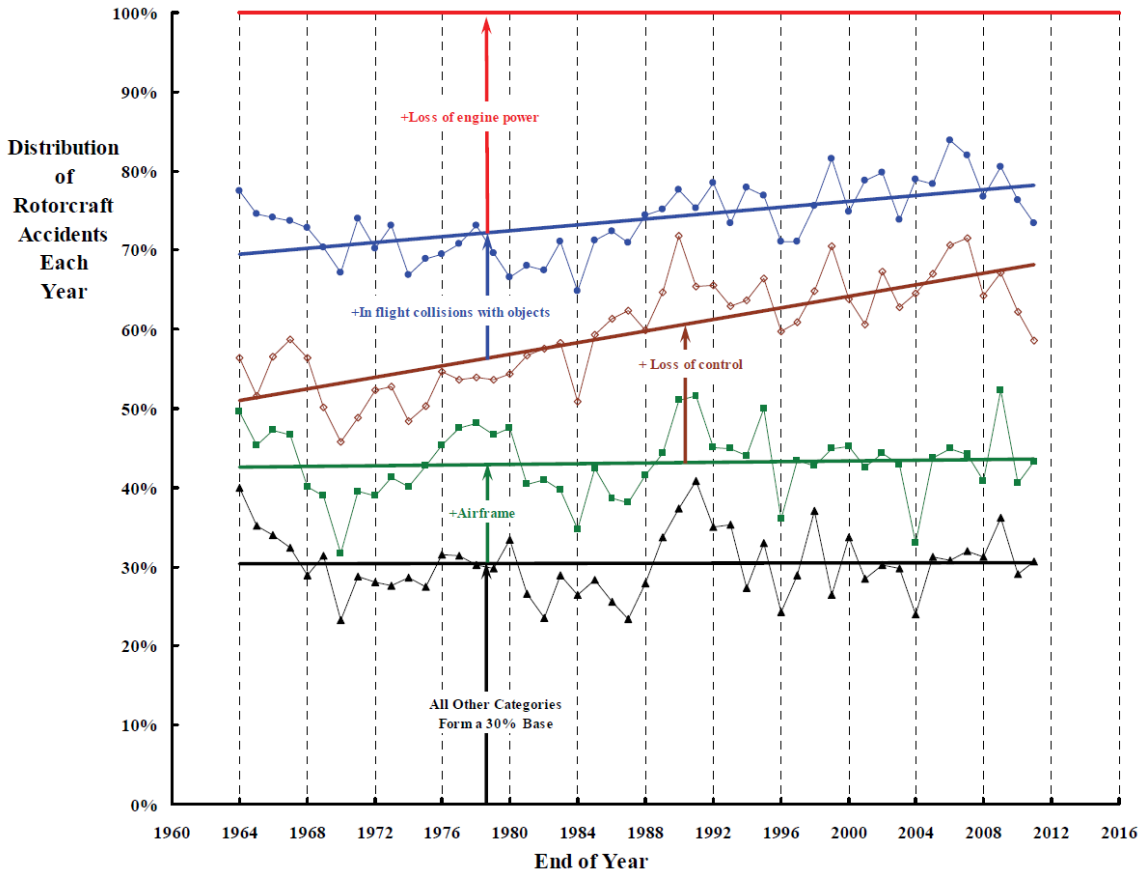


Figure 1.2: Accident proportions per first-occurrence classification from 1964 to 2011, reprinted from Harris [10] with permission.

The proportion of two of the accident classifications, *loss of engine power* and *in-flight collisions*, decreases steadily over time. This suggests that mitigating factors to address these issues have been successful. However, the proportion of *loss of control* accidents increases significantly throughout the time span. This could be a result of helicopters becoming more difficult to pilot, or being tasked to perform increasingly dangerous missions. It may also simply be that the proportion of loss of control accidents, which are the third most common by 2011, increased because the proportion of the two most common accidents as of 1964 decreased over time. Finally, the proportion of *airframe/component/system failures or malfunctions* remains relatively constant, indicating that rotorcraft may not have become significantly more reliable since 1964.

1.2.4 Discussion

The JHSAT and JHIMDAT each issued a number of recommendations to improve the accident rate. Many of these recommendations are operational: they are focused on improving pilot training and decision making methods or ensuring the implementation of proper maintenance procedures. Interestingly, neither the JHSAT nor JHIMDAT seem to suggest that safety records could be improved by making changes to the design of the aircraft.

Conversely, Harris recommends improvements to the vehicles themselves to improve the accident record. For example, Harris advocates increased rotor system inertia to improve the autorotational capability of single-engine helicopters, at the expense of increased weight empty and reduced useful load, as well as a more widespread adoption of stability augmentation systems to address loss of control incidents. Harris compares the safety record of rotorcraft to that of fixed-wing aircraft in the early 20th century and suggests that safety standards should be dramatically increased for future rotary-wing vehicles. This leads to Observation 1.2:

Observation 1.2

Component failures comprise a significant number of rotorcraft accidents and this proportion has not changed dramatically over time. It may be possible to reduce the rate of component failures through improvements to the vehicle designs themselves in addition to operational interventions.

1.3 Relevance to Future Rotorcraft Programs

The previous sections have established that LCC and flight safety are significant drawbacks that dampen the widespread adoption of rotary-wing vehicles. It is likely that these problems, and their potential solutions, will only become more important in future as the use of rotary-wing vehicles becomes more widespread in the GA, commercial, and military sectors.

1.3.1 Urban Air Mobility

The UAM operating concept was discussed previously in Section 1.1. Many academic, industrial, and governmental organizations are currently pursuing the development and implementation of the vehicles, regulatory standards, technologies, and aviation systems necessary to make UAM a reality [12, 13, 14]. This concept has the potential to revolutionize personal and public transportation. A complete UAM system could involve fleets of thousands of vehicles completing hundreds of thousands of trips daily in just one urban area. Commercial operations of this size will carry massive maintenance costs and passenger safety will be of the utmost concern. Therefore, vehicles designed for use in a UAM system may very well be required to meet higher standards of safety than current helicopters.

Presently, most conceptual UAM systems are based around small electric VTOL (eVTOL) configurations carrying only a few passengers [15]. These aircraft exist in a design space bounded by a large number of constraints: they must be light enough to hover, yet sturdy enough to protect their occupants in an accident; efficient in both hover and forward flight to maximize available battery power; and small enough to operate in confined urban environments, yet carry enough passengers to turn a profit. These strenuous and often conflicting requirements have resulted in a wide variety of exotic vertical lift configurations, many of which have not been previously flown.

Many proposed eVTOL concepts feature several rotor systems in novel arrangements that are uncommon in production rotorcraft. Examples include side-by-side dual rotors, multirotor systems, tilting rotors, slowed rotors, stopped rotors, variable speed rotors, ducted rotors, and more. There is a strong relationship between flight safety and rotor system configuration. It is likely that novel eVTOL designs will require substantial analysis and testing to ensure that all safety risks are appropriately modeled. Certain configurations have unique interactions between different aerodynamic components on the vehicle that may negatively impact safety. For example, Avera [16] showed that rotor–rotor interactions in an overlapping side-by-side rotor configuration can affect the magnitude of aerodynamic

loading on each rotor and that this effect is highly sensitive to the lateral and vertical separation of the rotors. This could have unintended consequences on the handling qualities of the vehicle, or the heightened aerodynamic loading could increase the chances of rotor blade failure.

Equally important is the public perception of these vehicles. Because of the proposed scale of UAM operations, eVTOL aircraft may become tightly integrated into the urban fabric and a perception and proven record of safety will need to be maintained. Ito and Furue [17] found that the perceived safety of a given aircraft drives public acceptance. Obviously, an in-flight failure, especially over a populated urban area, would significantly harm public acceptance of the UAM system and the eVTOL vehicle type.

It is unlikely that these aircraft will remain restricted to commercial roles. Several organizations [18, 19, 20] have announced the development of equally-exotic personal eVTOL aircraft which will be targeted towards the GA market and it is possible that other eVTOLs originally developed for UAM will eventually be sold as GA aircraft. This could lead to widespread proliferation of novel rotary-wing configurations in a market that is already subject to the largest proportion of rotorcraft accidents, as discussed in Section 1.2.

1.3.2 Military Programs

Note also that the militaries of several nations have been aggressively targeting reliability, availability, maintainability, and cost (RAM-C) improvements in their rotorcraft fleets for several decades. Improving the RAM-C characteristics of a vehicle allows the aircraft to be used more often, due to reduced maintenance downtime, and for less expense, due to reduced O&S cost. Recent decades have seen a maintenance paradigm shift from preventive maintenance, to condition-based maintenance (CBM), to the beginnings of a maintenance-free operating period (MFOP) paradigm [21]. The MFOP policy has been adopted by the U.S. Department of Defense (DoD) and the British Royal Air Force (RAF).

Under the Future Vertical Lift (FVL) program, the DoD is targeting a MFOP of 100 FH

and an operational availability (A_o) of 90%, compared to a current benchmark of less than 10 FH and 75%, respectively. For more detail on present and future efforts to reduce maintenance and cost burdens of military rotorcraft, the reader is encouraged to review the thesis by Bellocchio [21]. The future of military rotorcraft is relevant to the commercial and GA markets, because a large number of successful civilian rotorcraft are derived from military programs [1]. Thus, requirements set by military procurement offices will be embedded in the design of civilian rotorcraft for decades to come.

The previous discussion can be summarized in Observation 1.3:

Observation 1.3

Improving LCC and flight safety are key objectives or requirements in modern and future rotary-wing development programs, in both the civilian and military sectors.

1.4 A Common Denominator

Observations 1.1 and 1.2 summarize two of the most prominent deficiencies of rotorcraft compared to fixed-wing aircraft: high life-cycle costs and a poor safety record. Each of these issues is a significant challenge and overcoming both will require a large-scale coordinated effort involving rotorcraft operators, pilots, industry members, and regulators. However, Reddick and Harris, among other authors, believe that both problems can be addressed using engineering methods. That is, modifying helicopter designs before production and fielding to improve flight safety or reduce O&S cost is feasible. Of course, solutions that improve rotorcraft safety, such as enhanced stability augmentation systems, may not improve O&S cost. In fact, the opposite is likely true. Conversely, some efforts to make rotorcraft more affordable, such as lowering safety margins, come at the expense of flight safety, which is unacceptable.

However, a common denominator can be found in the category of component failures. In terms of O&S costs, Reddick and Harris both note that a large portion of operating cost is driven by the need to replace components that have either been damaged, failed, or worn

out. Further, failures or malfunctions in airframe components is listed as a leading cause of helicopter accidents by the JHSAT, the JHIMDAT, and the NTSB. Note from Figure 1.2 that the proportion of accidents induced by component failures has not decreased significantly over nearly 50 years, indicating that any efforts to mitigate this accident cause have been largely ineffective.

Hypothetically, an effort to reduce the rate at which onboard systems and components fail would improve both safety and O&S costs, at the expense of development and possibly manufacturing costs. Detailed trade-off studies between O&S costs, manufacturing costs, and development costs could potentially identify a level of investment at which LCC and safety are both improved. Before such a study can be performed, the primary causes of component failures on rotorcraft must be understood and classified. In this section, the operating cost and flight safety studies discussed previously will be analyzed in greater detail.

1.4.1 Component Replacement Costs

Reddick and Harris both discuss the cost of replacing failed or worn out components in their analyses of operating cost. In Reddick's analysis, the blades of the rotor system are an important driver of maintenance costs, making up 80% of the total rotor system support cost, which in turn makes up approximately 30% of the total vehicle maintenance cost. As discussed previously, many of the costs incurred by rotor blades are associated with blade damage or failure: foreign object damage (44%), cracking (9%), combat damage (9%), overstresses (9%), and debonding (6%) are the key cost contributors. Reddick also notes that rotor blades are difficult to repair; approximately half of all removals to address the aforementioned items result in the blade in question being scrapped and a new blade fitted for replacement.

As discussed previously, the cost of replacing life-limited parts is a major driver of O&S costs. Many of these replacement requirements are due to concerns of fatigue failure. These

parts are considered *fatigue-life limited* and must be replaced after a certain number of flight hours or years, even if they have not been visibly damaged. For example, the Robinson R22 main rotor blade life is limited to 2200 FH or 10 years, the Bell 206B main rotor blade is fatigue life limited to 5000 FH, and the Sikorsky S-76 main rotor blade is fatigue life limited to 28,000 FH. As of the time of this writing, replacement costs are $16,200, $47,530, and $162,390 per blade, respectively [3].[6] Although Harris does not attempt to rigorously calculate the contribution of these fatigue-life limited parts to the overall O&S cost, it is easy to see their significance.

1.4.2 Component Failure Causes

Additionally, NTSB data collected from 1964 to 2011 indicates that approximately 12% of all rotorcraft accidents are due to system or component failure. In this category, over 80% of failures occur in the main and tail rotor systems [10], which include the the rotor, drivetrain, and control system. In terms of failure modes, the leading cause of component failures was fatigue fracture, which was responsible for 23% of all component failures leading to accidents. Additionally, component failures are frequently caused by errors in assembly, installation, or maintenance.

Davies, Jenkins, and Belben [22] report that 54.8% of all component failures examined by the AugustaWestland (now Leonardo) Materials Technology Laboratory can be attributed to fatigue. Additional failure modes cited by the authors include corrosion (11.7%), wear (10.3%), and overload (10.0%). The JHSAT and JHIMDAT reports do not discuss accident statistics in enough detail to determine the failure mode of the component involved.

1.4.3 Discussion

It is clear that fatigue failure is a leading cause of component failure on rotorcraft, and that fatigue is also a primary driver of O&S costs and responsible for a significant proportion

[6] The R22 uses a two-bladed teetering rotor system and both blades must be replaced simultaneously, bringing total replacement cost to $32,400.

of helicopter accidents. Additionally, the rotor systems appear to be the common locations of fatigue failure; failures in these systems are made more serious by the high cost of rotor blade replacements and the near-inevitability of an accident after a rotor failure occurs. Fortunately, fatigue is a relatively well-understood failure mode and easier to analyze using engineering methods than other common rotor failure modes, such as foreign object damage or combat damage, which occur in a more unpredictable fashion. Thus, efforts to improve the resistance of rotor components to fatigue damage could result in safer, more affordable rotary-wing aircraft. This leads to Observation 1.4:

Observation 1.4

Targeting fatigue life improvements in the rotor system could improve rotorcraft O&S costs and flight safety simultaneously.

In the next section, the underlying causes of fatigue damage in rotorcraft will be described.

1.5 Causes of Fatigue Damage

Due to their fundamental nature, rotary-wing aircraft are highly susceptible to fatigue damage. In fact, they are much more susceptible to fatigue failure and resulting accidents than fixed-wing vehicles. The production of lift using a rotating wing produces highly dynamic airloads which are realized as oscillating loads applied at a high frequency [4]. In contrast, lift generation using a fixed wing produces relatively steady airloads and when dynamic loading occurs the associated frequencies and cycle counts are typically much lower. It is this dynamic loading environment that is responsible for the accumulation of fatigue damage and eventual fatigue failure if components are not regularly removed from service.

There are a number of factors contributing to the dynamic loading produced by rotating wings. Fundamentally, the rotor loading environment is made up of aerodynamic, centrifugal, inertial, and gravitational forces, which vary based on the flight condition of the vehicle. This

section will attempt to concisely describe the flight conditions and loading environments most relevant to fatigue life determination. For a more detailed explanation of fatigue loads in rotorcraft, the reader is encouraged to review the technical report by Lombardo [4].

1.5.1 Forward Flight

Perhaps the most common flight condition for any aircraft is steady forward flight. In a fixed-wing vehicle, lift is generated by air rushing over the wings. Because each portion of the wing experiences the same wind velocity at all points in time, the lift produced by the fixed wing is effectively steady. However, in rotary-wing vehicles, the rotation of the wing during edgewise flight alters the magnitude and direction of the relative wind at each point, both radially and azimuthally. In steady level flight, a specific radial point on the blade will experience azimuthal variation in airspeed and angle of attack, resulting in dynamic airloads that vary over the course of one revolution. These are known as *cyclic airloads* and each period of oscillation is referred to as a cycle. This phenomenon is illustrated in Figure 1.3. Typically, the forward velocity of the vehicle correlates positively with the magnitude of the load cycle. Knowledge of the magnitude and number of cycles accumulated over a specified period is a critical step in calculating the fatigue damage, and the remaining safe life, of a component.

Cyclic airloads are particularly harmful due to their rapid rate of accumulation. For example, Lombardo notes that a rotor angular speed of 5 Hz will result in load cycles accumulating at a rate of 18,000 cycles/FH. Over the course of a 10,000 FH service life, the rotor system will have accumulated approximately 10^8 cycles due to forward flight cyclic loading alone. Thus, cyclic loading is known as *high cycle fatigue* (HCF), in contrast to *low cycle fatigue* (LCF). This distinction will be discussed further later in this section.

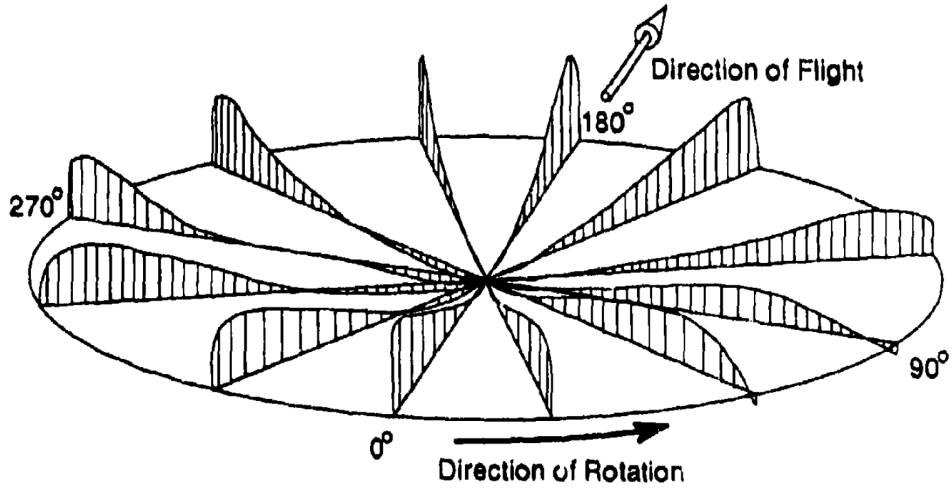


Figure 1.3: Conceptual illustration of the lift force produced by a rotor at different radial and azimuthal stations, reprinted from Lombardo [4] with permission.

1.5.2 Higher-Order Aerodynamic Effects

The aerodynamic environment is further complicated by rotor–fuselage interactions and rotor–rotor interactions, which produce additional unsteady loads. Further, higher-order aerodynamic effects such as high-speed compressibility, blade-tip vortices, reverse flow, and dynamic stall each add additional dynamic loads. In addition to oscillating aerodynamic loads, a rotor in edgewise flight also experiences oscillating inertial loads due to blade flapping and lagging, static centrifugal forces, and static gravitational forces.

In hovering flight or vertical climbs, the effect of cyclic aerodynamic loading is largely reduced. However, cyclic loading can still be induced by ambient wind, center of gravity imbalance, and rolling moments produced by the tail rotor in a single main rotor (SMR) configuration. Additionally, rotor–fuselage and rotor–rotor interactions continue to cause dynamic loading in axial flight.

The region between hover and forward flight, known as transitional flight, is also a significant source of fatigue loads. Because the flight condition is constantly changing during transitional flight, the blade response is similarly transient; this may produce greater deformations in the blade structure, which are accompanied by increased internal loads.

Lombardo notes that the transition from forward flight to hover generally occurs more slowly than the reverse operation because the pilot is simultaneously preparing to land. Thus, the vehicle spends more time in the transitional flight condition and more fatigue cycles are accumulated.

1.5.3 Low Cycle Fatigue

Two of the most important causes of LCF in rotorcraft are ground–air–ground (GAG) cycles and autorotation. The GAG cycle is simply a single cycle describing the oscillation between the minimum steady load on a part, which usually occurs on the ground, and the maximum steady load on the same part, which usually occurs in-flight, perhaps during a maneuver. GAG cycles result in LCF loading at a rate of 1 cycle/flight. LCF loads superimposed with HCF loads can cause said HCF loads to be more damaging than would otherwise be predicted.

If autorotation occurs during a flight, the magnitude of the GAG cycle is amplified because the rotor is operating in a manner opposite of its normal state, in essence acting as a windmill and extracting power from the air. This results in steady loads on some rotor components being nearly opposite of what would be experienced in normal flight, and the minimum load in the GAG cycle becomes the loads experienced during autorotation. Lombardo notes that autorotation is responsible for a large proportion of fatigue damage in training helicopters due to the frequency with which the maneuver is practiced.

1.5.4 Other Sources of Fatigue Damage

Fatigue loads can also be induced by taxiing, droop stop pounding, ground resonance, operating from uneven surfaces, operating from ships, and more. These loading patterns are inherent to and unavoidable in any vehicle configuration which includes a rotor in edgewise flight. Of course, the design and orientation of a specific rotor affects the fatigue loads that are inflicted upon it. For example, in a SMR configuration, the tail rotor typically

experiences a higher number of load cycles than the main rotor because they rotate at a higher speed. Additionally, yaw maneuvers result in increased cyclic loading on the tail rotor. In a tandem configuration, the effect of rotor–rotor interactions is much stronger than that of a SMR. Ultimately, predicting the strength and frequency of fatigue loads on rotor systems requires detailed knowledge of the rotor design, vehicle configuration, and expected operational profile of the vehicle.

1.5.5 Influence on Accident Rates

Support for the previous assertion that fatigue damage is more prevalent in rotary- than fixed-wing aircraft can be found in the analysis of accident reports. In 1983, Campbell and Lahey [23, 24] published a study of 1885 fatigue-related accidents in fixed- and rotary-wing aircraft occurring from the year 1927 to 1981. Of these accidents, a total of 1466 were attributed to fixed-wing vehicles and 419 were attributed to rotorcraft. Considering that, in a given year, the number of fixed wing aircraft is roughly two-to-three orders of magnitude greater than the number of operational helicopters [4], rotorcraft appear to be much more susceptible to fatigue accidents.

Campbell and Lahey report that the most common location of fatigue failure in rotorcraft is the engine or transmission, accounting for 32% of all observed accidents. The tail rotor is responsible for 24% of accidents, followed by the main rotor at 13%. If the tail rotor and main rotor are considered together, rotor systems account for 37% of all fatigue-related accidents in rotorcraft and are responsible for 155 accidents resulting in 248 fatalities and the destruction of 74 aircraft. Other common locations of fatigue failure in rotorcraft include the fight controls, airframe, and landing gear. At the time of the report's publication, rotorcraft experienced an average fatigue-related accident rate of 31 accidents/year, compared to 69 accidents/year for fixed-wing aircraft.

The previous discussion can be summarized in Observation 1.5:

Observation 1.5

Fatigue damage is unavoidable in rotorcraft due to the aerodynamic forces produced by a rotating wing in edgewise flight. This is reflected in the proportionally higher fatigue-related accidents in rotary-wing aircraft compared to their fixed-wing counterparts.

1.6 Research Objectives

The discussion of the previous sections is summarized in Table 1.1, which reprints all the key observations in Chapter 1 for reference.

Table 1.1: Summary of all key observations in Chapter 1.

Observations	
1.1	High LCC associated with rotorcraft are driven by part replacement and repair requirements. Rotor blades make up a significant portion of this cost due to their extreme operating environment, high purchase price, and life-limited nature.
1.2	Component failures comprise a significant number of rotorcraft accidents and this proportion has not changed dramatically over time. It may be possible to reduce the rate of component failures through improvements to the vehicle designs themselves in addition to operational interventions.
1.4	Targeting fatigue life improvements in the rotor system could improve rotorcraft O&S costs and flight safety simultaneously.
1.5	Fatigue damage is unavoidable in rotorcraft due to the aerodynamic forces produced by a rotating wing in edgewise flight. This is reflected in the proportionally higher fatigue-related accidents in rotary-wing aircraft compared to their fixed-wing counterparts.
1.3	Improving LCC and flight safety are key objectives or requirements in modern and future rotary-wing development programs, in both the civilian and military sectors.

Future rotorcraft development programs will be influenced by the twin objectives of mitigating rotor system O&S costs (Observation 1.1) and reducing in-flight component failures (Observation 1.2). Analyses of O&S cost drivers and accident causes suggest that improving the fatigue life of rotor system components could have a significant positive impact in both areas (Observation 1.4). Because fatigue damage is unavoidable in rotary-wing aircraft (Observation 1.5), the problem cannot simply be “designed out” of the vehicle.

Currently, revolutionary vertical lift vehicles with new and unique configurations and strenuous cost and safety requirements are being designed and prototyped (Observation 1.3). In order to meet these requirements, these vehicles should be designed from the beginning with the goal of maximizing the fatigue life of critical rotor system components. That is, fatigue life should be a design driver rather than a fallout property.

Although fatigue life is not typically considered as a primary design driver in modern helicopters, an analogy can be found in the wind turbine industry. The economic feasibility of a wind turbine is highly dependent on maintaining a low maintenance burden, especially in off-shore wind farms where access is difficult or expensive [25]. Thus, wind turbine components are typically designed for a fatigue life of 20 years or more [26]. Wind turbines and rotorcraft are not entirely dissimilar: each constantly accumulates fatigue damage in operation due to the cyclic nature of the airloads acting on the rotor. The intensity of these loads varies depending on the magnitude, direction, and turbulence intensity of the wind flowing through the turbine, producing a variety of loading environments similar to the rotorcraft flight conditions discussed in Section 1.5.

Several researchers have developed design frameworks to optimize wind turbine blade fatigue life in the early design stages. Ronold, Wedel-Heinen, and Christensen [26] developed a probabilistic model of the fatigue damage accumulated by a wind turbine rotor blade. Random variables were used to capture the stochastic nature of airloads and material strength. Simple stress–strain relationships, S-N curves, damage accumulation theories, and first-order reliability methods were used to determine the probability of fatigue failure during the 20 year design life. Toft and Sørensen [27] discuss a similar, but more complex, reliability-based design framework intended to estimate the reliability of a wind turbine blade in both fatigue failure and ultimate failure. Florian and Sørensen [25] incorporated a fatigue damage model into a maintenance and repair simulation to optimize the LCC of an individual wind turbine.

The existence of fatigue design frameworks for wind turbine rotor blades establishes a

precedent of feasibility. It is reasonable to assume that a similar fatigue design framework could be developed for rotorcraft. Such a framework would enable the use of fatigue life as a driver in the early stages of development, allowing engineers to conduct design exercises and trade-offs to improve fatigue life prior to the detailed design and construction of the vehicle.

However, the fatigue life prediction problem is complex and multi-dimensional. A complete analysis of the fatigue life of a specific component must consider the structural and material design of the component; the mean and oscillatory loads on the component, including aerodynamic, centrifugal, and gravitational forces; and the manner by which the vehicle's operational profile influences these loads. The primary focus of this research will be reviewing, testing, and implementing methods to efficiently solve the fatigue life problem in order to achieve cycle times suitable for use in a rotorcraft design framework. This leads to the fundamental research question of this thesis:

Research Question 0

How can the fatigue life of rotor system components be efficiently evaluated for use as a design driver in a rotorcraft design framework?

A flow chart summarizing the logical steps leading to the formulation of Research Question 0 is presented in Figure 1.4.

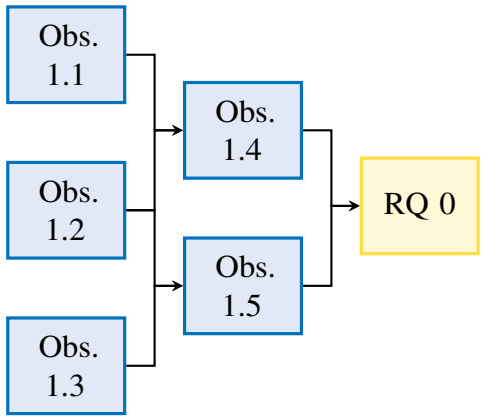


Figure 1.4: Flow chart summarizing the formulation of Research Question 0.

The remainder of this dissertation will involve answering Research Question 0 and any subsequent research questions that arise. To that end, the following research objectives will be pursued:

1. Review the strengths and weaknesses of traditional and modern rotorcraft fatigue design methods.
2. Formulate a new rotorcraft fatigue design methodology to address the weaknesses of traditional methods.
3. Conduct virtual experiments to test, validate, and refine different elements of the methodology.
4. Demonstrate the viability of the methodology using a hypothetical fatigue design exercise.

Chapter 2 will feature literature review necessary to satisfy Research Objective 1, covering rotorcraft design methods, traditional fatigue design practices, and modern improvements to those practices. Research Objective 2 will be the focus of Chapter 3, which concerns the formulation of the framework and subsequent research questions. Chapters 4 and 5 will address Research Objective 3 by constructing virtual experiments using industry-standard research and simulation packages. Finally, Research Objective 4 will be considered in Chapter 6, which will document a proof-of-concept fatigue design study to demonstrate the applicability of the new methodology.

CHAPTER 2

REVIEW OF ROTORCRAFT DESIGN METHODS

In order to answer Research Question 0 and address Research Objective 1, an in-depth literature review was conducted. The findings of the literature review are presented in this chapter.

This review attempts to answer three *literature questions* related to the subjects of vehicle design and fatigue design. These are distinct from *research questions* in that they are intended to be answered by reviewing preexisting research rather than by conducting original research.

First, the aerospace vehicle design process in general, and the rotorcraft design process in particular, will be reviewed. The objective of this activity is to understand how the fatigue design process integrates with the overall design process; that is, when and how fatigue design influences or is influenced by vehicle design. Additionally, popular rotorcraft design tools will be examined to assess their feasibility for fatigue design or other RAM-C design activities. This is captured by Literature Question 1:

Literature Question 1
How does fatigue design fit into the overall vehicle design process? Can existing rotorcraft design tools be effectively utilized for fatigue design?

Next, the traditional rotorcraft fatigue design process will be examined. This will include discussion of both fatigue damage theory and fatigue design methods. The function of specific theories, models, and practices will be identified and any disadvantages with respect to Research Question 0 will be highlighted. This leads to Literature Question 2:

Literature Question 2
What are the most important elements of the traditional fatigue design process? What are the associated disadvantages and drawbacks?

Finally, a survey of modern rotorcraft fatigue design studies or methodologies will be presented. Particular attention will be paid to those that seek to improve upon traditional methods using new computational methods or simulation packages. The completeness of each study or methodology, or the degree to which it satisfies Research Question 0, will also be assessed. This is summarized by Literature Question 3:

Literature Question 3
How have other researchers improved upon the traditional rotorcraft fatigue design process?

Throughout the chapter, key observations related to the aforementioned literature questions are highlighted. Following the review, specific gaps in the literature are derived from these observations.

2.1 Rotary-Wing Vehicle Design

Aircraft design is a complex, iterative, and multi-disciplinary process that seeks to derive the best possible solution to a set of requirements. Prior to constructing a rotor fatigue design framework, the various elements of the aircraft design process must be understood. This section provides an overview of the rotorcraft design process, followed by detailed discussions of tools used in the conceptual and preliminary design of rotary-wing vehicles. The section is concluded by examining the state of the art in rotor system preliminary design tools.

For reference, Literature Question 1, which is the focus of this section, is reprinted below:

Literature Question 1
How does fatigue design fit into the overall vehicle design process? Can existing rotorcraft design tools be effectively utilized for fatigue design?

2.1.1 Overview

Anderson [28] defines aircraft design as "the intellectual process of creating on paper . . . a flying machine to meet certain specifications and requirements established by potential users and/or pioneer innovative, new ideas and technology". This process is typically divided into a number of stages or phases based on the various design activities and the order in which they occur. The definition of each phase is somewhat arbitrary and varies between authors, institutions, or organizations. For example, Anderson, in a discussion of fixed-wing vehicle design, divides the design process into three stages termed conceptual design, preliminary design, and detail design. Johnson [29], discussing rotary-wing vehicles, describes only two stages, termed preliminary design and detail design. Leishman [30], also discussing rotorcraft, blends the conceptual design and preliminary design stages into a single stage termed conceptual and preliminary design, which is followed by detail design.

It appears that, in the case of rotorcraft design, the distinction between the conceptual and preliminary design stages may be insignificant or difficult to define. For the purposes of this research, a line is drawn between conceptual and preliminary design at the point at which rotor system analysis transitions from simple tools such as momentum theory and blade-element theory (BET) to more complex analyses using comprehensive codes and flexible blade models. This distinction will be discussed in further detail later in this section.

2.1.1.1 Requirements

Requirements are "a clear set of specifications, which are defined based on the needs of a potential customers or . . . the need to meet a specific military requirement" [30]. In aircraft design, requirements may be specified by the customer, the manufacturers themselves, or other stakeholders. Typically, requirements for a specific vehicle program are written into a contract after negotiations between the manufacturer and the customer. Alternatively, the customer will issue a request for proposals (RFP), and multiple manufacturers will submit competing designs in response. After winning the contract, the manufacturer is responsible

for designing and/or producing a vehicle that meets all the specifications laid out in the contract.

Leishman provides examples of typical requirements for civilian and military rotorcraft programs. Civilian rotorcraft are typically designed with an eye towards reducing cost and internal/external noise while increasing safety, passenger comfort, reliability, and maintainability. Comparatively, military rotorcraft are focused primarily on performance, speed, maneuverability, flexibility, damage tolerance, and survivability.

2.1.1.2 Conceptual Design

The conceptual design stage encompasses all design activities involved in determining the overall configuration and size of the vehicle [28]. Design decisions are driven by the vehicle requirements: the end goal of the conceptual design stage is to produce a high-level layout of a vehicle that is capable of meeting all specifications. This design should also be optimized; that is, the design selected should be the best possible design that satisfies the requirements. During conceptual design, engineers are primarily concerned with the aerodynamics, propulsion, performance, and weight of the aircraft. Anderson notes that processes such as structural design and control system design are typically not the focus of the conceptual design stage. However, certain decisions within the conceptual design stage may be made by qualitatively considering the impact on the vehicle structure or control system.

This stage also concerns the aerodynamic design of the rotor system(s). Leishman asserts that the main rotor is the single most important component of the helicopter and, as a result, "helicopter design is often synonymous with rotor aerodynamics" [30]. Because the performance of the vehicle is highly dependent on the performance of the rotor(s), in-depth design of this system begins early in the design process. This process typically begins with a selection of the rotor configuration (e.g., single main rotor, tandem, coaxial, etc.) and basic sizing [29]. Other rotor design variables of interest include the number of blades, solidity,

tip speed, and disk loading. Additionally, the designer may consider more complex variables such as the geometric twist distribution, planform shape, and airfoil(s). Consideration is also given to the aerodynamic environment in which the rotor will operate, defined by Mach number, advance ratio, and blade loading. Other rotor conceptual design activities include prediction or calculation of the weights of each component and estimation of basic performance measures such as equivalent lift-to-drag ratio, propulsive efficiency, and figure of merit (FM). Rotor conceptual design processes typically use elementary rotor theories such as momentum theory, BET, or combined blade-element–momentum theory (CBEMT) due to their speed and flexibility [30], which allows the designer to quickly explore a large design space.

A modern example of the rotor conceptual design process is given by Bagai [31], who describes the aerodynamic design of the Sikorsky X2 Technology Demonstrator (X2TD) rotor system. The X2TD rotor system consists of two hingeless coaxial lift-offset rotors. Additionally, the blades have a complex planform shape, a non-monotonically-decreasing geometric twist distribution, and a variety of airfoils. The aerodynamic design of the rotor system was accomplished by improving upon a previously-designed vehicle, the Sikorsky XH-59A, using the Sikorsky Generalized Rotor Performance (GRP) methodology. GRP uses a relatively simple rigid blade lifting surface model with uniform or non-uniform inflow, supplemented by airfoil lookup tables and stall models. Bagai notes the simplicity of the model allows for rapid design space exploration but resulted in loss of fidelity in certain cases. The primary design activities described by Bagai include blade geometric design, including airfoil selection, planform design, and blade twist specification; and rotor performance prediction at several values of lift-offset, shaft tilt, and blade loading. Note that, as discussed previously, it is difficult to classify this work as *conceptual* versus *preliminary* design. However, because Bagai focused primarily on rotor aerodynamics and did not make significant considerations for the structural design of the rotor blade, it is more appropriately classified as conceptual design.

2.1.1.3 Preliminary Design

In the preliminary design stage, the layout produced during conceptual design is further analyzed, refined, and expanded [28]. Only minor changes are made to the overall configuration of the vehicle: the primary focus is on structural analysis and design, as well as the design of certain important systems, such as the control system. Additionally, the analyses become more complex. Wind tunnel testing and computational fluid dynamics (CFD) are used to understand and improve the aerodynamics of the vehicle, leading to minor changes in the layout. By the end of the preliminary design stage, the configuration should be frozen and rigorously defined.

During this stage, the rotor design discussed previously is similarly refined and expanded. The rotor hub type (e.g., articulated, teetering, hingeless, etc.) will be selected based on the desired handling qualities, aeroelastic stability, and maintainability of the rotor [29]. The structure of the rotor blade will be considered, and a basic cross-sectional design may be created. Mass and stiffness distributions along the blade can be calculated, which, when combined with the hub design, enable more detailed analysis using more advanced tools such as comprehensive analysis programs. Comprehensive analyses can predict the complete aeromechanical behavior of a rotor system, synthesizing the fields of dynamics, structural dynamics, aerodynamics, and aeroelasticity. These tools also enable the prediction and improvement of rotor characteristics such as stability, vibration, and noise. Because the tools and theories used in rotor preliminary design require far greater runtime than the conceptual design techniques discussed earlier, preliminary design is focused only on exploring a small portion of the design space around the conceptual design point.

Blackwell and Millott [32] provide a modern example of the rotor preliminary design process. This work follows from Bagai's conceptual design of the X2TD rotor and concerns the dynamic design of the X2TD rotor system. The authors discuss a rotor preliminary design methodology that combines the Sikorsky KTRAN, RCAS, and CAMRAD II comprehensive codes. This methodology was used to conduct a trade study on blade thickness, spar

wall thickness, hub separation, blade precone, blade natural frequencies, and aerodynamic efficiency. The primary goals of this study were to provide sufficient inter-rotor tip clearance, reduce vibrations, achieve aeromechanical stability, measure blade and control system loads, and appropriately place blade natural frequencies to avoid resonance conditions.

These activities are classified as preliminary design because they follow on from the conceptual design activities discussed by Bagai, do not significantly alter the rotor configuration selected in conceptual design, and exhibit a greater focus on structural design and analysis of the rotor blades. Note also that more advanced aeromechanical simulations are used to support the preliminary design activities, in contrast to the simpler rigid blade aerodynamic models used in conceptual design.

2.1.1.4 Detail Design

Anderson colloquially defines detail design as the "nuts and bolts" stage. Prior to this stage, all design and analysis of aerodynamics, propulsion, structures, performance, and flight controls should be completed. Now, each component of the aircraft, such as ribs, spars, and skin, must be precisely designed to produce detailed manufacturing specifications. Fasteners are also selected, sized, and placed on the vehicle. At the end of the detailed design phase, the aircraft should be ready for manufacturing and eventual delivery.

Johnson mentions that, in the design of rotor systems, the detail design phase includes an in-depth structural analysis of all the components of the hub and blades as well as detailed measurements or predictions of all aerodynamic and inertial loads. At this stage, prototypes of the rotor system will be constructed for aerodynamic and structural analysis. According to Leishman, wind tunnel testing, ground testing, construction of a prototype vehicle, and flight testing may also occur during the detail design phase. Others, such as Arden, Chappell, and Reddick [33], consider construction and flight testing of the prototype vehicle to be part of a post-design phase they term "development". Needless to say, these activities are inherently expensive and time-consuming, and any major redesigns at this stage could result

in significant budget and schedule overruns.

2.1.1.5 Discussion

The rotorcraft design process discussed previously is roughly summarized in Figure 2.1. As the design process progresses, the level of detail and precision increases, from the general layout established in the conceptual design change to the precise manufacturing specifications established in the detail design phase. Simultaneously, the complexity and accuracy of the analysis tools used to support the design effort increases.

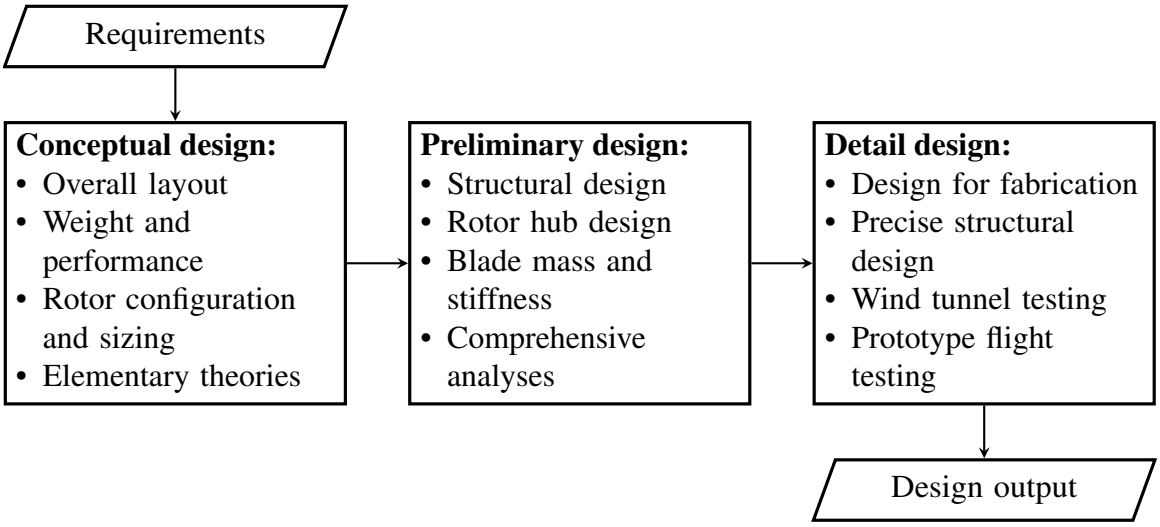


Figure 2.1: Flow chart summarizing the rotorcraft design process.

In the early design stages, low-fidelity tools are used to enable rapid exploration of the design space, trade studies, and optimization. In later stages, these tools are replaced with higher-fidelity simulation packages, wind tunnel tests, and possibly flight tests, which provide increased accuracy at the expense of much greater cycle times. Thus, later stages of design are focused more on exploring a very small space around the previously selected design.

Mavris [34] notes that due to the structure of the design process, the amount of design freedom decreases monotonically throughout each stage. That is, major decisions regarding the vehicle design are made in early design stages when their impacts may not be fully

understood due to the low fidelity of the analyses used in these stages. These decisions also have the most significant impact on the overall performance and cost of the final vehicle design. Thus, developing methodologies to use higher-fidelity tools in the conceptual and preliminary design stages is a common focus of modern aircraft design research. Related efforts will be discussed later in this document.

In the context of fatigue design, the preliminary design stage appears to be the earliest point at which fatigue life predictions can be reasonably accomplished. In the conceptual design stage, the rotor blade structural design is not advanced enough to determine blade loads with reasonable accuracy. Additionally, conceptual design tools are typically more concerned with the aerodynamic performance and efficiency of the rotor system and may not make any provisions to predict blade loads. Finally, the computational expense of rotorcraft fatigue design, to be discussed in Section 2.2, means that these analyses are too time-consuming for use in conceptual design activities.

However, in the preliminary design stage, the structural design of the rotor blade will progress to a point at which fairly accurate prediction of blade loads may be possible. Comprehensive tools currently used to assist in preliminary design could be leveraged to predict blade loads for fatigue analysis. The small size of the preliminary design space is more suited for the high-fidelity, computationally expensive programs used for fatigue life prediction. This leads to Observation 2.1:

Observation 2.1

The preliminary design stage is the earliest point in the rotorcraft design process at which fatigue design is feasible.

2.1.2 Vehicle Design Tools

The past few decades have seen a proliferation in the number of rotorcraft *vehicle design tools*. This terminology refers to software tools used primarily in the conceptual design phase to assist in the design and analysis of rotary-wing vehicles. The use of the word

vehicle implies tools that consider the entire aircraft in a multi-disciplinary manner, rather than other design tools that focus only on a specific system or sub-system on the vehicle.

This section provides an overview of some of these tools. They are divided into categories based on their primary use. *Performance-focused* tools are those that are primarily concerned with traditional conceptual design activities as described in Section 2.1.1. They are typically concerned primarily with vehicle weight, sizing, aerodynamics, propulsion, and performance. *RAM-C–focused* tools are those that are intended to predict the reliability, availability, maintainability, and cost aspects of the concept vehicle. These tools may be coupled with performance-focused tools to provide a more complete vehicle design environment or may be operated in a standalone manner.

2.1.2.1 Performance-focused Tools

Perhaps the most well-known example of a performance-focused tool is NASA Design and Analysis of Rotorcraft (NDARC), a rotorcraft design code developed at NASA Ames Research Center [35]. NDARC uses a robust collection of component models, including rotors, wings, fuselages, drive systems, and more, which enable the user to construct nearly any rotary- or fixed-wing vehicle concept imaginable. Separate design and analysis routines within NDARC allow the user to either size a "rubberized" concept based on one or more design missions and flight conditions, or analyze the performance of a sized concept in any number of missions or conditions. An outline of the NDARC program is included in Figure 2.2.

NDARC relies on low-fidelity models similar to those discussed in Section 2.1.1 to reduce computational expense and design iteration time. For example, many of the component weight and power consumption models are constructed using regressions of historical rotorcraft component data. Regression models used in NDARC are, in general, applicable to a wide variety of vertical lift concepts because they are used in a system-by-system buildup of the vehicle, rather than attempting to capture the performance of the entire vehicle in

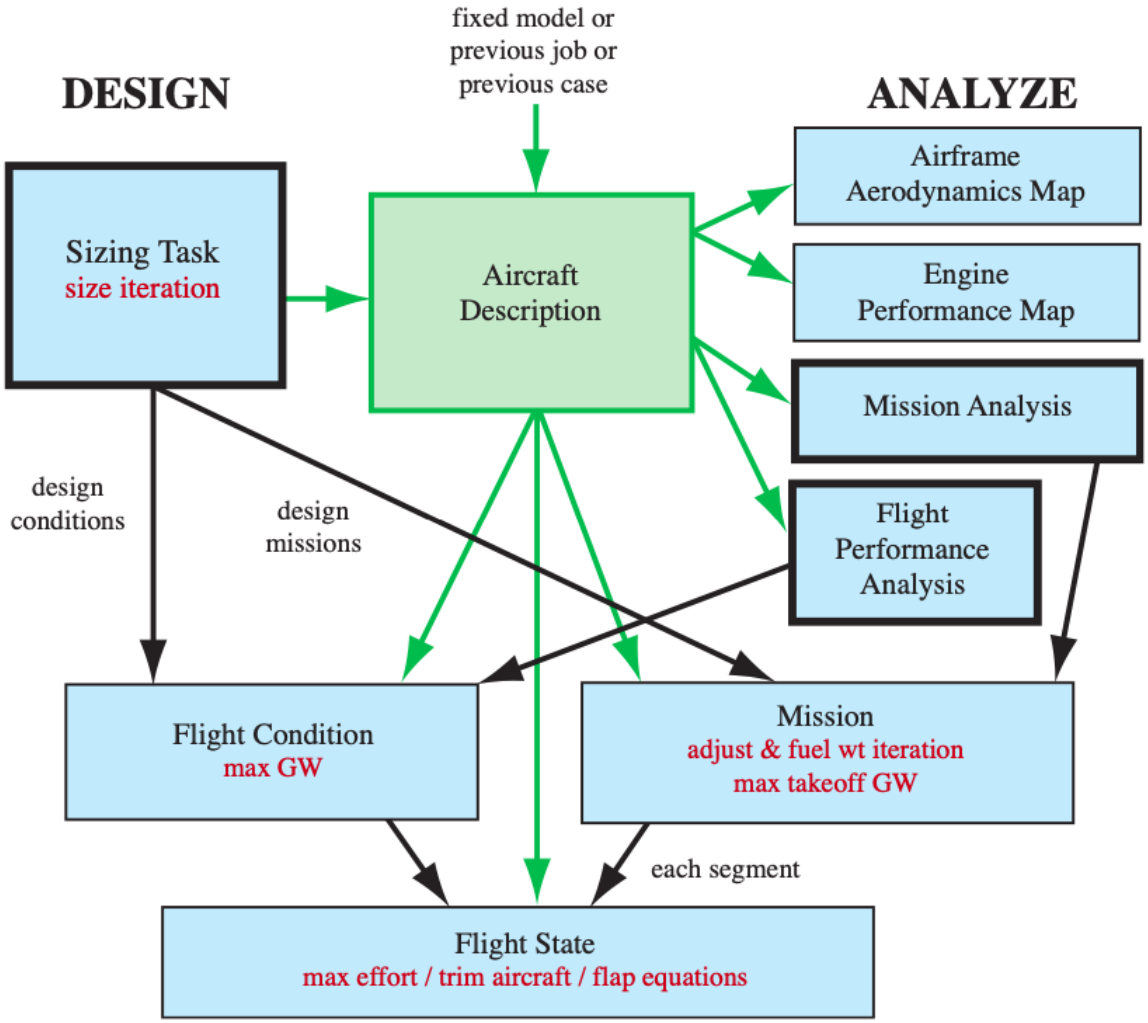


Figure 2.2: Outline of the NDARC program, reprinted from Johnson [36] with permission.

one model. Simple physics-based models such as momentum theory, BET, and lifting line theory are also used for the analysis of rotor aerodynamics and performance. Due to the low-fidelity and compartmentalized nature of the NDARC analysis, some interactions between systems may be lost. However, depending on how the designer chooses to model the vehicle, reasonable flexibility and accuracy can be achieved. NDARC outputs include a detailed vehicle weight breakdown, a description of the sized configuration, and detailed performance assessments and mission analyses. NDARC will be discussed further in Section 4.4.2.1.

Although NDARC is primarily performance-focused, some RAM-C capabilities have been integrated. Until recently, NDARC used a relatively basic cost model developed by

Harris and Scully [37], known as the "rotorcraft cost too much (CTM)" model. This model estimates unit cost and direct operating cost using a statistical analysis of data derived from historical aircraft. The resulting equations are based on airframe weight, empty weight, maximum takeoff weight, fuel weight, installed power, and the number of rotor blades.

Although this model may provide a reasonable first-order prediction of general cost characteristics, similar simple cost models have proved inadequate when predicting the outcomes of certain RAM-C trade-offs [38]. For example, simple cost models were unable to correctly predict the results of a historical helicopter upgrade program: the model predicted increased maintenance costs based on the gross weight increase when, in fact, the actual maintenance cost was much lower due to reliability improvements in the upgraded vehicle. Fortunately, there are a large number of more detailed rotorcraft RAM-C models available. Many tools implementing these models use a system-level buildup philosophy similar to that of NDARC, and some are even designed to interface directly with NDARC.

2.1.2.2 RAM-C–focused Tools

RAM-C–focused tools are a more recent development in the rotorcraft community. The development of these programs has been fueled by a recent push to produce more affordable and reliable rotorcraft, especially in the military sector, as discussed in Section 1.3. This section provides a brief overview of selected RAM-C–focused tools and methodologies.

Bellocchio [21] developed a discrete event simulation tool to model the reliability and availability of vertical lift systems. This program allows the user to build a reliability model of a vehicle by specifying the time to failure (TTF) and time to repair (TTR) of its component systems using statistical distributions. The tool then simulates vehicle operations: missions are "flown" and the vehicle must be "repaired" when components fail. The stochastic nature of system failure is captured using Monte Carlo simulation. The primary outputs from the simulation are aggregate predictions of MFOP and maintenance recovery period (MRP), which are used to assess the reliability and availability of the vehicle. The author used the

results of the simulation to identify necessary improvements to meet reliability targets and investigate the impact of different preventive maintenance policies. Notably, Bellocchio did not attempt to compute the cost of these maintenance actions or predict the specific reliability characteristics of individual systems; the necessary TTF and TTR distributions are generated randomly or, in some examples, derived from historical maintenance data.

Price, Ashok, Armstrong, et al. [39] developed a similar discrete event simulation, including a more detailed phased mission simulation and additional fault types. The authors developed the simulation to predict traditional RAM-C metrics including A_o, mean time between failures (MTBF), and maintenance man-hours per flight hour (MMH/FH), in addition to MFOP as studied by Bellocchio. The tool was intended to calculate the cost of the maintenance actions accumulated during the simulation, but difficulty obtaining realistic estimates of component replacement costs prevented study in this area. Notably, the authors also had difficulty determining realistic values of TTF and TTR for each component, noting that "because this environment is meant for use at the conceptual design level, part failure and repair time data are rarely available for the specific components of interest" [39]. A high-level overview of the authors' simulation is included in Figure 2.3.

Significant inroads into the problem of predicting rotorcraft LCC in the preliminary design phase were made by Scott et al. (see Scott [38], Scott, Schrage, and Sirirojvisuth [40], and Scott [41, 42]). Scott and his co-authors developed a RAM-C assessment environment that predicts maintenance and cost characteristics of a rotorcraft design using the Bell PC-Based Cost Model [43]. This model was enhanced with multiplicative *technology factors* that enabled the representation of RAM-C–focused system upgrades. Scott's environment was integrated with NDARC, enabling complex trade-offs between weight, performance, operating cost, and maintenance cost. Additional regression equations were integrated to capture necessary increases in procurement cost and research, development, test, and evaluation (RDT&E) costs due to RAM-C upgrades. This approach allows the prediction of RAM-C characteristics at a level of fidelity very similar to NDARC's weight and perfor-

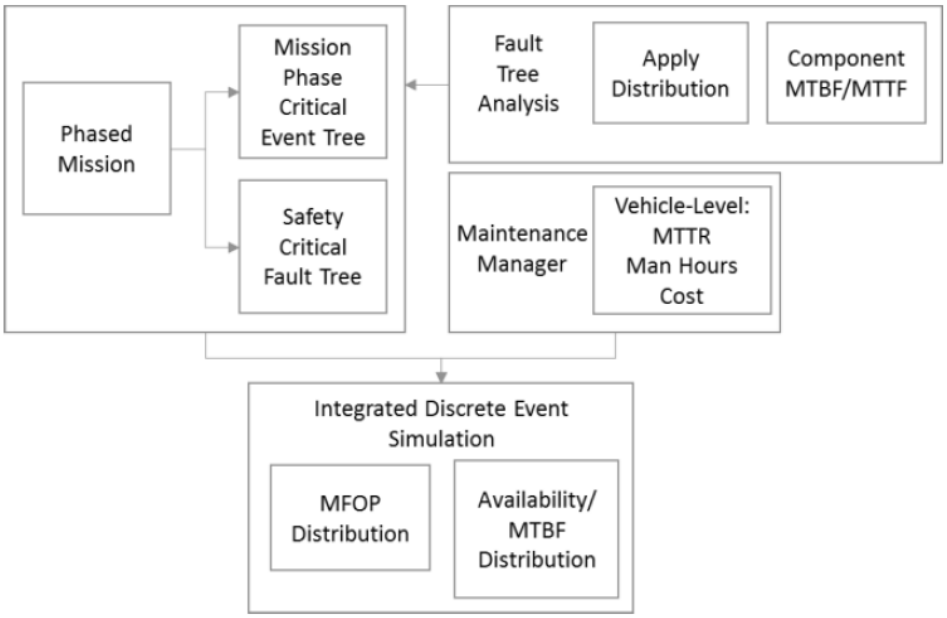


Figure 2.3: Outline of RAM-C simulation, reprinted from Price, Ashok, Armstrong, et al. [39] with permission.

mance modeling, and recent upgrades to the NDARC software have incorporated portions of Scott's cost model [36].

Other notable studies include the work of Bhattacharya et al. (see Bhattacharya, Nagaraju, Fiondella, et al. [44, 45] and Bhattacharya, Nagaraju, Spero, et al. [46]), who developed a more rigorous model of rotorcraft component failure rates. This work was specifically focused on modeling the impact of RAM-C investments and considering the impacts on the rotary-wing fleet as a whole, rather than individual vehicles.

2.1.2.3 Discussion

The aforementioned design tools and studies have significantly advanced the understanding of RAM-C characteristics of rotary-wing aircraft. Most importantly, they have helped identify trade-offs that can be made in the early design stages to significantly reduce LCC and overall fleet costs.

However, many of the cited publications explicitly discussed the challenge of meeting data requirements. Most simulation tools require several inputs for each system on the

vehicle. These inputs are often difficult to estimate realistically, especially in the case of the technology factors used in Scott's financial analysis. Harris [3] stated that a lack of well-documented and consistently-organized accounting and maintenance data makes the task of predicting RAM-C characteristics difficult, and this observation appears to be corroborated by the experiences of Bellocchio, Price, Scott, and others.

This difficulty is especially prevalent in the early design stages where detailed designs of each system and component, which are necessary for full-scale reliability engineering analysis, are not yet available. New design and analysis tools may be required to accurately predict the RAM-C characteristics of individual rotorcraft systems and components in the early design stages. This leads to Observation 2.2:

Observation 2.2

Most currently available RAM-C–focused design tools suffer from high data input requirements. These inputs can be difficult to populate due to a lack of appropriate resources to predict individual system characteristics in the early design stages.

2.1.3 Rotor Design Tools

Recent advances in computer technology have enabled vast possibilities in the field of rotor system analysis and design. *Rotor design tools* are tools, software packages, or design frameworks intended to aid in the design, optimization, and analysis of the rotor system(s), alone or in the context of the whole vehicle. These tools typically use fairly advanced simulation programs to provide a higher level of fidelity than is typical of the vehicle design tools discussed in Section 2.1.2. They may be used independently or coupled to vehicle design tools to improve the accuracy of the rotor aerodynamic theories inherent to the vehicle design tool.

This section provides an overview of some of these tools. First, key enablers to the development of rotor design tools are described. Next, a number of examples of modern rotor design tools are reviewed.

2.1.3.1 Enablers

Rotor design tools are enabled by the convergence of two disciplines, each developed simultaneously and, for the most part, independently. The first discipline of interest is rotor comprehensive analysis, discussed previously in Section 2.1.1. According to Johnson [47], the field of comprehensive analysis began in 1962 with the development of what later became known as the C81 program at Bell Helicopter. The field has expanded since, and as of the time of this writing, at least eight modern, full-scale comprehensive codes are in existence.

Typically, comprehensive codes combine a computational structural dynamics (CSD) solver with built-in aerodynamic theories, including airfoil lookup tables and a number of wake models, to calculate the motion, deformation, and loads of the various components of the rotor system. Recent developments in comprehensive analysis have focused on integrating CFD with these codes to replace the built-in aerodynamic theories [47]. In these efforts, CFD aerodynamic calculations are coupled with the comprehensive code to provide a highly accurate, albeit computationally expensive, prediction of airloads and rotor performance.

Recently, Smith and Moushegian [48] have developed dual-solver hybrid methods that reduce the cost of coupled CFD–CSD analysis by using Reynolds-averaged Navier-Stokes solvers in only the near field, while the wake is simulated with more cost-effective methods. When applied to a UH-60A main rotor model, these methods were demonstrated to be accurate to within 4% of a full CFD–CSD simulation while being 70% less expensive [49]. To the best of this author's knowledge, these hybrid methods have not yet been utilized within a rotor design environment, but they promise to dramatically increase the fidelity of these tools while maintaining low runtime. Comprehensive analysis is discussed further in Section 4.4.2.2.

, The second discipline of interest is engineering optimization. This field concerns the development of computer algorithms to automatically improve products or designs based

on one or several objective functions. Objective functions provide a means to calculate the performance of the product based on its design variables.

A limited history of engineering optimization in the context of rotor design is given by Tarzanin and Young [50] who, writing in 1998, describe 12 years of experimentation with the optimization of rotor systems at the Boeing Company. The authors and their colleagues attempted to generate rotor designs with substantial improvements in weight, vibration, and aerodynamic performance. The authors used a "single, tightly coupled, interdisciplinary rotor analysis" similar to the comprehensive codes described previously. The optimization task was accomplished using gradient-based, metaheuristic, and response surface methods. Predictably, they faced challenges related to a large number of design variables, the high computational cost of evaluating the objective functions, and the complexity of the design space. Nevertheless, their overall experience was promising and the paper concludes with optimism for the future of automated optimization and design of rotor systems.

2.1.3.2 Modern Examples

The convergence of comprehensive analysis with engineering optimization has resulted in a push to improve rotor system design by incorporating high-fidelity, physics-based prediction methods into the early stages of the process. If successful, these projects could improve vehicle performance, reduce design cycle time and cost, and mitigate uncertainty and risk.

For example, in 2010, Sinsay and Nuñez [51] presented the efforts of the U.S. Army's Aviation and Missile Research, Development, and Engineering Center (AMRDEC)[1] to develop what they termed *right-fidelity* rotorcraft design. Right-fidelity design is an attempt to create a flexible, rapid, physics-based design environment by combining low- and high-fidelity analysis tools intelligently with engineering optimization algorithms. Chief among these tools are CFD analyses of fuselages and lifting surfaces, comprehensive analysis of rotor systems, and higher-fidelity coupled CFD/CSD analysis of rotor systems.

Later, Sinsay and Alonso [52] presented a multidisciplinary analysis environment in-

tegrating the comprehensive code CAMRAD II, an in-house structural analysis code, and NDARC. This environment was successfully used to improve the performance of a vehicle featuring a coaxial lift-offset rotor, demonstrating its ability to analyze and optimize next-generation rotary-wing aircraft. Sinsay and Alonso's design framework is reproduced in Figure 2.4. Note the highly coupled and iterative nature of the analysis.

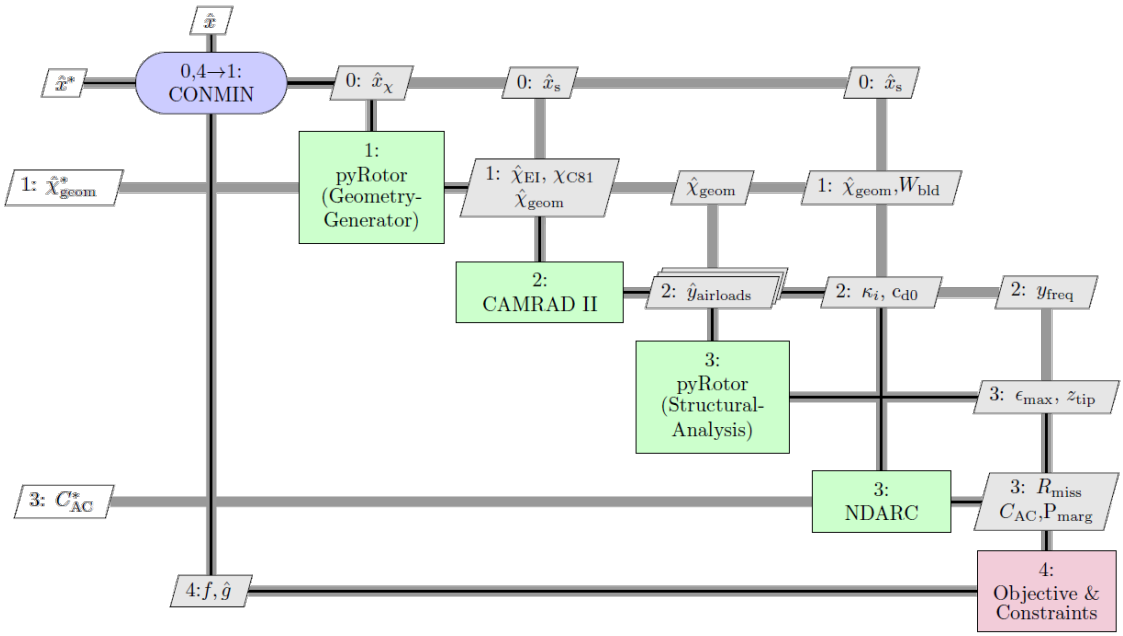


Figure 2.4: Extended design structure matrix of a multi-disciplinary rotorcraft optimization environment, reprinted from Sinsay and Alonso [52] with permission.

Another notable effort in the field of automated optimization of rotor systems is that of Collins et al. (see Collins, Bain, Rajmohan, et al. [53], Collins [54], and Collins and Sankar [55]). Over several years, Collins and his colleagues developed a mixed-fidelity (combining low-fidelity and high-fidelity tools) framework using CSD, CFD/CSD, and aeroacoustic analyses. This framework integrated comprehensive analysis code RCAS, aeroacoustics code PSU-WOPWOP, and CFD code GT-Hybrid to improve the performance, vibration, and external noise characteristics of a hingeless rotor system. Notably, this framework also made use of surrogate modeling techniques to reduce the expense of

[1]AMRDEC is now known as the Aviation & Missile Center (AMC)

repeatedly evaluating the objective functions during the optimization process. Surrogate modeling will be discussed further in Section 4.1. Ultimately, this effort was successful in generating a set of pareto-optimal rotor system designs that improved upon the baseline design in most of the metrics of interest.

2.1.3.3 Discussion

A large number of similar studies focusing on first principles, physics-based optimization of rotor systems using a variety of analysis tools have been published. An exhaustive review of these efforts is beyond the scope of this research. There are two common themes among the majority of these studies.

First, the computational expense and runtime of the simulation packages that serve as the foundation of rotor design tools make application to the early design stages difficult. As discussed in Section 2.1.1, conceptual and preliminary design environments rely on a fast cycle time and rapid iteration to explore the design space efficiently. Sinsay and Nuñez overcome this challenge through the use of right-fidelity frameworks, which combine low-fidelity and high-fidelity tools to improve overall runtime while retaining a certain level of accuracy. Collins' mixed-fidelity framework is philosophically similar to Sinsay and Nuñez's right-fidelity framework, but is supplemented by the use of surrogate modeling techniques, which can dramatically reduce computational expense if implemented appropriately. This leads to Observation 2.3:

Observation 2.3

The development of rotor design tools is complicated by the computationally expensive simulation packages on which they rely.

Second, most rotor design tools are focused primarily on fundamental aspects of rotor system performance, such as the conceptual design performance measures discussed in Section 2.1.1.2. Some rotor design tools have expanded to include objectives such as perceived noise, magnitudes and frequencies of hub vibration, and structural stability, which

are typically considered to be preliminary design considerations. Very few studies have considered in great detail the RAM-C characteristics of the rotor system, such production cost, O&S cost, and component failure rates.

Generally, the assumption is that if certain desirable parameters can be maximized while weight is minimized, the resulting design will be high-performance and affordable [30]. However, this is not always the case; affordability improvements can be accomplished without reducing component weight. For example, a rotor could be modified to improve the service life of a specific critical component at the expense of forward flight efficiency. The additional cost of extra fuel now required may well be offset by the cost saved in part replacements, resulting in an overall reduction in LCC.

Of course, the authors of the aforementioned studies should not be criticized for their focus on more fundamental parameters. In engineering, it is common practice to ensure the feasibility of a system before optimizing its secondary characteristics. The affordability of a rotor system is meaningless if it provides such poor performance that it can never be installed on a practical aircraft.

The previous discussion leads to Observation 2.4:

Observation 2.4

Most currently existing physics-based rotor design tools typically do not make considerations for the as RAM-C characteristics of the rotor system.

In the next section, the rotorcraft fatigue design process will be examined, including its relationship to the vehicle design processes and tools discussed previously.

2.2 Rotorcraft Fatigue Design

Fatigue design is the process of predicting the fatigue life of certain safety-critical components of a system and, if necessary, taking steps to improve the fatigue life [56]. This process is especially important in rotary-wing vehicles due to the constant accumulation of HCF loads during nearly all phases of flight, as discussed in Section 1.5. Fatigue design

spans each stage of the vehicle design process. The fidelity and accuracy of the methods used will vary depending on the available structural design information and loads data in each stage.

This section discusses the process of fatigue design in rotorcraft. First, a general review of fatigue damage theory is presented, with a focus on fatigue life prediction models and cycle counting methods. Next, fatigue design literature specifically related to the rotorcraft industry is reviewed and traditional fatigue design methods are discussed. Finally, several new approaches to rotorcraft fatigue design related to the rotor design tools discussed in Section 2.1.3 are examined.

For reference, Literature Question 2, which is the focus of this section, is reprinted below:

Literature Question 2

What are the most important elements of the traditional fatigue design process? What are the associated disadvantages and drawbacks?

2.2.1 Fatigue Damage Theory

Fatigue is a phenomenon in which materials subjected to cyclic loading accumulate damage over time, eventually leading to failure [57]. Notably, fatigue damage can be accumulated at load levels significantly lower than a given material's ultimate strength, potentially causing structural failures without exceedance of the ultimate load. Fatigue damage can also be exacerbated by temperature fluctuations, corrosion, contact between two surfaces, or fretting.

Suresh [57] describes five general stages of fatigue damage progression as follows:

1. Microscopic structural changes cause permanent damage.
2. Microscopic cracks are created.
3. Microscopic cracks grow and coalesce to form "dominant" crack(s).
4. Dominant crack(s) propagate through the structure.
5. The structure fractures or becomes unstable.

More simply, the first three stages can be considered *crack initiation* and the final two can be considered *crack propagation*. The objective of fatigue life prediction models is to estimate, for a given structure, the amount of load cycles until crack initiation or until structural failure.

Modern and historical literature contains a vast number of fatigue life prediction models with varying levels of sophistication, complexity, and applicability. An exhaustive review and comparison of each model is beyond the scope of this research. Instead, this section will contain a brief summary of models with demonstrated applications to rotorcraft fatigue analysis.

Degrieck and van Paepegem [58] classify these models into three categories. *Fatigue life models* do not consider the actual damage mechanisms; instead, these models use S-N curves derived from experiments in which material samples are loaded to failure at varying load levels. S-N curves can be combined with the known load history to produce a hypothetical level of damage accumulation and an associated fatigue life. *Phenomenological models* predict the degradation of material stiffness or strength which can be measured in a laboratory, unlike the damage accumulation factor predicted by fatigue life models. A failure criterion based on a certain level of residual stiffness or strength is established. Finally, *progressive damage models* predict the progress of the actual damage mechanism, such as the crack initiation and propagation stages described previously.

2.2.1.1 Fatigue Life Models

The most commonly-used fatigue life model is known as Miner's sum, Miner's rule, or the Palmgren-Miner rule [59]. Despite being published in 1945, this theory remains popular due to its low computational expense and the relatively small amount laboratory testing required for its implementation. Miner's rule is used in nearly all fatigue life predictions in the rotorcraft industry, as will be discussed in Section 2.2.2. Cansdale [60] hypothesizes that Miner's rule remains popular in this domain because subsequently-developed alternatives

do not show consistent improvements over a wide range of applications and are, in general, more complicated.

S-N Curve Fatigue life models such as Miner's rule begin with the definition of the S-N curve. Typically, material coupons or representative components are subjected to constant cyclic loading in a laboratory. The applied load, S, may be described in terms of force, moment, stress, strain or displacement. Since the load is alternating, S can be further described in terms of the maximum load, S_{max}, the load range, $S_{\text{range}} = S_{\text{max}} - S_{\text{min}}$, or the load amplitude, $S_{\text{amp}} = S_{\text{max}} - S_{\text{mean}}$. The test continues until structural failure occurs. The number of cycles required to reach failure, N, is recorded. If the sample has not failed after a predetermined number of cycles, the article is considered to have infinite fatigue life for this value of S, known as *runout*. A notional cyclic load history and its associated terminology are presented in Figure 2.5.

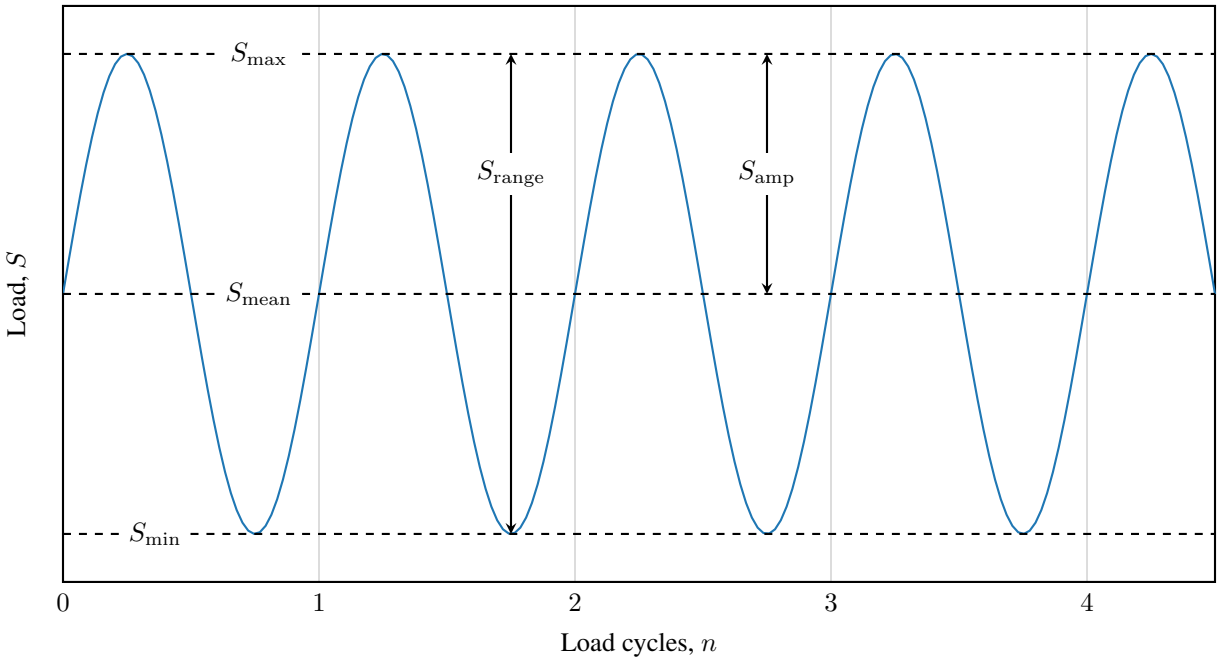


Figure 2.5: Notional cyclic load history.

In addition to the load amplitude, the mean load also has an impact on N. Typically, the

impact of the mean load is captured using the *stress ratio*, R:

$$R = \frac{S_{\text{min}}}{S_{\text{max}}} = \frac{S_{\text{mean}} - S_{\text{amp}}}{S_{\text{mean}} + S_{\text{amp}}} \tag{2.1}$$

Thus, $R = -1$ represents fully reversed loading ($S_{\text{min}} = -S_{\text{max}}$ and $S_{\text{mean}} = 0$) and $R = 1$ represents constant loading with no alternating component ($S_{\text{min}} = S_{\text{max}} = S_{\text{mean}}$).

A number of samples will be tested until failure at constant load amplitude, with the amplitude varied in each test to establish the impact of S on N. Typically, these tests are all performed at constant R. If R is varied during the test, the data points collected will not belong to the same S-N curve and will produce erroneous results. It is common in the rotorcraft industry to use only six representative component samples to save cost at the expense of increased uncertainty in the definition of the S-N curve [33]. If material coupons are used, more data points will likely be produced, reducing uncertainty. After completing all the tests, a S-N diagram is created by plotting each failure point with S on the ordinate and N on the abscissa. Then, a mathematical description of the S-N curve is derived by fitting an appropriate mathematical expression to the data using a technique such as the least-squares fit. For example, Och [61] provides two potential expressions:

$$S = S_\infty + (S_u - S_\infty)e^{-\alpha(\log N)^\beta} \tag{2.2}$$

$$S = S_\infty + \frac{B}{N^x} \tag{2.3}$$

In Equation (2.2), S_∞ represents the endurance limit, or the highest load level at which fatigue failure will never occur, and must be determined during the fitting process; S_u represents the ultimate static strength and can be determined using other simple tests; and α and β are additional parameters that must be determined during the fitting process. Equation (2.3) is a simpler equation that is easier to derive: only two parameters, B and S_∞, need to be determined during the fitting process. The exponent x is chosen based on the material; for example, $x = 1/2$ would be used for a steel sample. However, Equation (2.3) is

only accurate for the HCF range ($N > 10^5$), whereas Equation (2.2) covers the entire LCF and HCF range from $N = 1$ to $N = \infty$. A notional S-N diagram and associated S-N curves are presented in Figure 2.6.

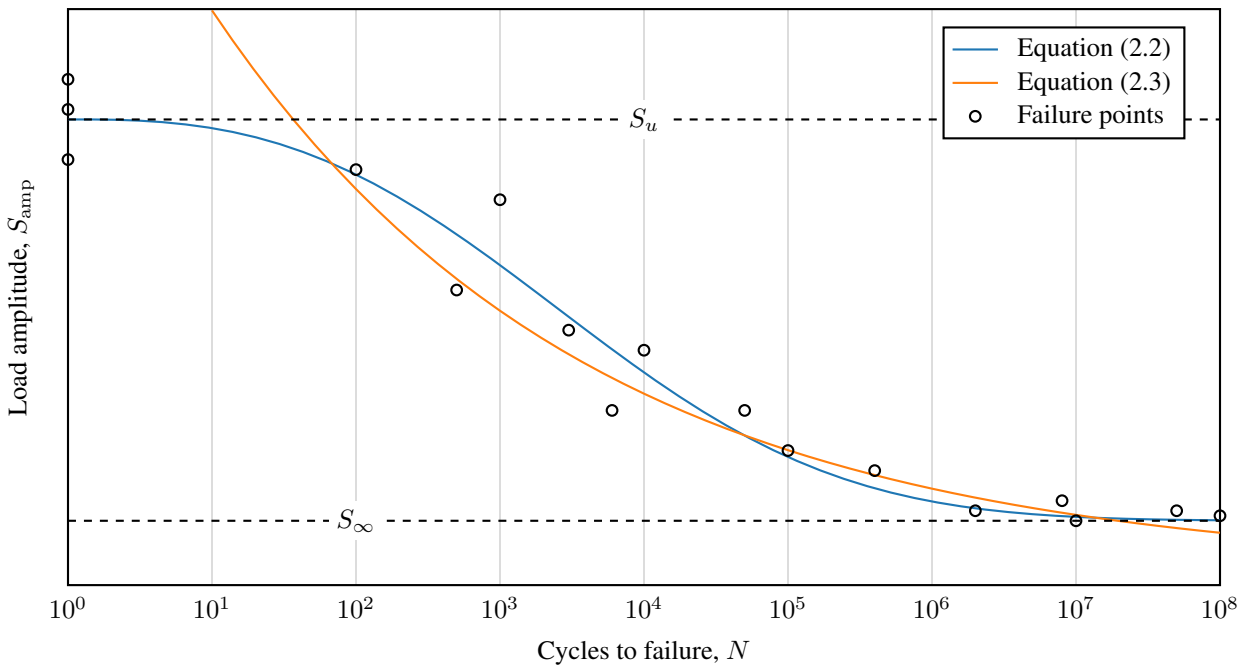


Figure 2.6: Notional S-N diagram and curves.

Mean Load Effects An single S-N curve such as the curve pictured in Figure 2.6 is only valid for a single value of R. However, components of a rotary-wing vehicle will experience variations in mean load throughout a single flight. For example, the mean load on a rotor blade will change with variations in gross weight, density altitude, or rotor speed. Typically, increasing the magnitude of the mean load reduces N, so mean load effects must be captured to produce a valid fatigue life prediction [61].

Additional fatigue tests can be performed at different R-values. Alternatively, various corrections can be used to predict the equivalent stress at $R = -1$, S_{eq}, for a given cycle defined by S_{amp} and S_{mean}. The most common correction used in the rotorcraft industry is

known as the Goodman relation and can be described as follows [61, 62]:

$$S_{\text{eq}} = \frac{S_{\text{amp}} S_u}{S_u - S_{\text{mean}}} \tag{2.4}$$

When plotted in the $(S_{\text{mean}}, S_{\text{amp}})$-space, the nature of the Goodman relation becomes more clear. Figure 2.7 depicts a notional Goodman diagram.

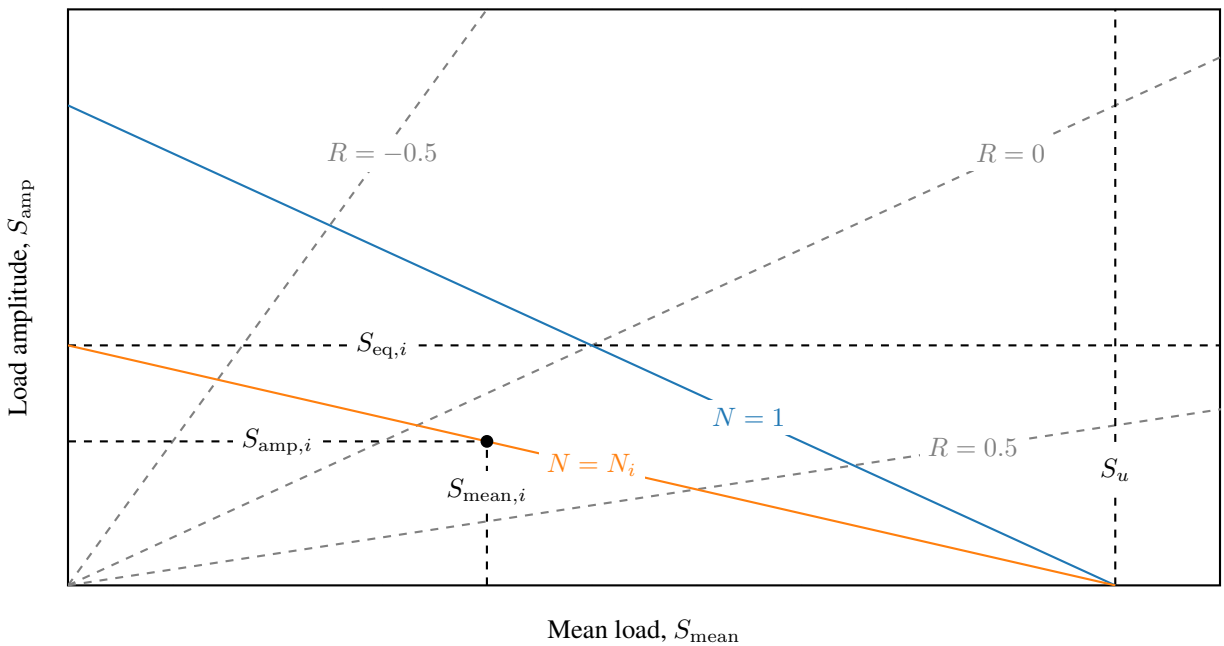


Figure 2.7: Notional Goodman diagram.

In Figure 2.7, constant-life lines are represented by solid colored lines. For a given cyclic loading pattern defined by $(S_{\text{mean},i}, S_{\text{amp},i})$, the equivalent load $S_{\text{eq},i}$ can be found by tracing a constant life line from the abscissa at S_u through the point $(S_{\text{mean},i}, S_{\text{amp},i})$ to the ordinate. Then, the fatigue life N_i can be found from an appropriate S-N curve, assuming the curve was derived using tests at $R = -1$. If the S-N curve was derived using a different R-value, $S_{\text{eq},i}$ can be found where the appropriate constant-R line, represented in Figure 2.7 as gray dashed lines emanating from the origin, intersects the constant-life line. Thus, the Goodman relation essentially allows for the construction of additional S-N curves at R-values differing from the original experiment.

Non-linear variations of the Goodman diagram exist; for example, the constant-life lines can be shifted along the abscissa or the constant-life lines can be constructed in a piecewise fashion. These variations may offer improved accuracy in some cases. However, these variations require additional fatigue life testing whereas the linear Goodman diagram requires only the S-N curve and knowledge of S_u. Thus, the linear form of the Goodman relation remains popular due to its simplicity.

Cumulative Fatigue Damage Finally, the accumulation of fatigue damage can be calculated. The *load spectrum* is a fixed load sequence intended to capture the complete variation of cyclic loading a component will encounter in service. The load spectrum is composed of k segments each containing, for example, a defined mean load, $S_{\text{mean},i}$, and load amplitude, $S_{\text{amp},i}$, which occur for a certain number of cycles, n_i. Then, using an appropriate S-N curve and mean load correction, an equivalent fatigue life, N_i, can be established for the ith segment in the spectrum. The cumulative damage is calculated using Miner's rule:

$$C = \sum_{i=1}^{k} \frac{n_i}{N_i} \tag{2.5}$$

where C is representative of the cumulative fatigue damage. Typically, a failure criterion is established such that $C = 1$ at failure. The total fatigue life can be found by repeating the load spectrum until $C = 1$ is reached.

A number of variations on Miner's rule exist. For example, Nijssen describes a number of non-linear variations, with the general form

$$C = \sum_{i=1}^{k} \left[A\frac{n_i}{N_i} + B\left(\frac{n_i}{N_i}\right)^D \right] \tag{2.6}$$

where A, B, and D are additional parameters that must be found by examining experimental data. Note that this form reduces to Equation (2.5) by setting $A = 1$, $B = 0$, and $D = 0$. Hashin and Rotem [63] developed a complex non-linear multi-stage fatigue life model which

showed improvements over Miner's rule in certain cases. However, Miner's rule remains the dominant fatigue damage theory in the rotorcraft industry and other industries due to its economical nature, as discussed previously.

2.2.1.2 Phenomenological Models

Phenomenological models have an advantage over fatigue life models in that they track cumulative fatigue damage using material properties that can be measured macroscopically [58]. For example, the cumulative fatigue damage parameter, C, in Equation (2.5), does not have any true physical meaning and cannot be measuring during the fatigue life experiment. On the other hand, phenomenological models predict the degradation of material strength or material stiffness, which can be measured. Strength-degradation models have the advantage of an inherent failure criterion: the sample fails when the strength degrades to a point that it is exceeded by the load. However, the strength of a material cannot be measured without degrading or destroying the sample. Stiffness-degradation models have the advantage that stiffness can be measured easily without degrading the sample, but the failure criterion is more arbitrary and usually depends on a predefined stiffness threshold.

Despite the wide variety of phenomenological models that have been developed, this author could only find one that had been used in the context of rotary-wing systems. A strength degradation model developed by Schaff and Davidson [64, 65] was used in the analysis of rotor blade fatigue by Li, Volovoi, and Hodges [66] and the analysis of wind turbine blade fatigue by Nijssen [62]. This model is developed specifically for composite materials and is notable for its limited experimental input requirements and flexibility. In fact, the model requires only a few additional inputs beyond the S-N curve and mean load corrections discussed previously.

For constant amplitude loading, the residual strength can be calculated using

$$S_r(n) = S_0 - (S_0 - S_{\max}) \left(\frac{n}{N} \right)^{\nu} \tag{2.7}$$

where S_r is the residual strength, S_0 is the initial static strength, and ν is termed the "strength degradation parameter", which must be determined experimentally. If S_0 is assumed to be equal to the ultimate static strength, S_u, then ν is the only additional parameter that must be derived in order to implement this model. A graphical depiction of Equation (2.7) is presented in Figure 2.8.

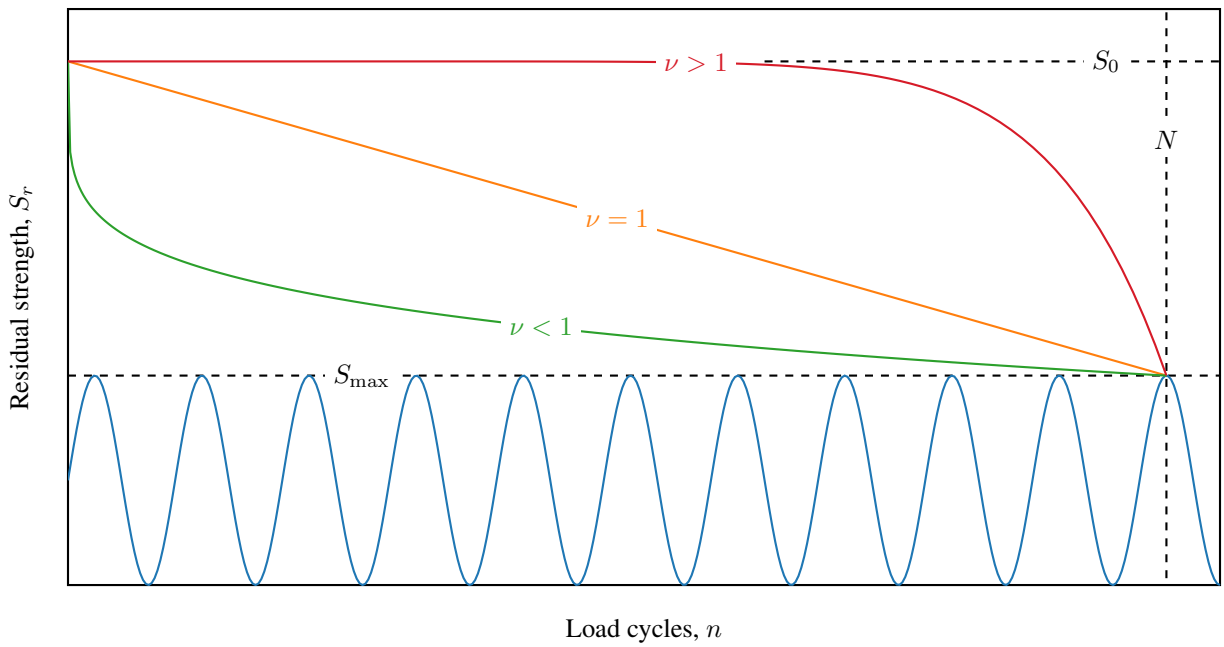


Figure 2.8: Notional residual strength degradation curves.

Figure 2.8 demonstrates the impact of ν on the strength degradation pattern. Schaff and Davidson term the $\nu = 1$ case "linear degradation", the $\nu > 1$ case "sudden death", and the $\nu < 1$ case "early degradation". A specific value of ν for a given material is derived by fitting data obtained during fatigue life experiments.

Equation (2.7) can be extended to spectrum loading as

$$S_r\left(\sum_{i=1}^{j} n_i\right) = S_0 - (S_0 - S_{\max,i})\left(\frac{n_j + n_{\text{eff},i}}{N_j}\right)^{\nu_j} \tag{2.8}$$

where j is the current segment in the spectrum, i denotes the previous segments, and $n_{\text{eff},j}$ is

the effective number of cycles for the jth segment. The latter is calculated using

$$n_{\text{eff},j} = N_j \left(\frac{S_0 - S_r \left(\sum_{i=1}^{j-1} n_i \right)}{S_0 - S_{\max,j}} \right)^{\frac{1}{\nu_j}} \tag{2.9}$$

As before, N_j is determined using an appropriate S-N curve and mean load correction. Note that because the calculation of S_r at each block depends on the strength degradation in each previous segment, the model must be applied to each segment in turn, unlike Equation (2.5).

Schaff and Davidson also introduce additional modifications to the model to account for complex spectrum loading phenomena, such as strength conversion and cycle mix effects. Experimental methodology for deriving the model parameters for a specific sample is also developed. The model is shown to out-perform Miner's rule in certain cases with composite materials and complex spectrum loading definitions. However, the model also requires more experimental effort to derive and more computational resources to evaluate.

2.2.1.3 Progressive Damage Models

Progressive damage models are perhaps the most "realistic" method to assess fatigue life. These models predict the development and propagation of a specific damage mode through the structure. For example, in metallic materials, progressive damage models typically predict the crack propagation phase discussed previously. The structure is considered to have failed when the crack propagates completely through its thickness. In composite materials, these models may predict a number of different damage modes, including matrix cracking and delamination. Generally, the growth of damage within the structure can be measured, either microscopically or macroscopically, allowing the rate of damage propagation to easily be tracked during the experiment [58].

Salvetti, Cavallini, and Fediani [67] provide a review of progressive damage models within the context of rotorcraft fatigue design. The primary focus of the review is on crack propagation models. Fundamentally, these models depend on establishing a functional

relationship between the crack growth rate and a quantity known as the stress intensity factor:

$$\frac{\mathrm{d}a}{\mathrm{d}n} = f(\Delta K, R) \tag{2.10}$$

where a is the crack length and K is the stress intensity factor. ΔK represents the total stress intensity factor range during a particular load cycle. Methods used to calculate K depend on the geometry of the crack and the material but generally follow the form

$$K = \sigma\sqrt{\pi a}\, F(g, a) \tag{2.11}$$

where σ is a reference stress and g captures the geometry of the problem.

The true form of Equation (2.10) can only be derived through experimental measures. The relationship between crack growth rate and ΔK is typically sigmoidal in nature if plotted on a log–log axis. That is, at very low values of ΔK, the crack growth rate is almost imperceptible. At intermediate values, $\frac{\mathrm{d}a}{\mathrm{d}n}$ varies linearly with ΔK, and at high values, $\frac{\mathrm{d}a}{\mathrm{d}n}$ increases dramatically. The typical impact of R is to increase the crack growth rate. A notional plot of this relationship is presented in Figure 2.9.

A number of different forms of Equation (2.10) can be used depending on the specific problem and the desired level of accuracy. The Paris law takes the form

$$\frac{\mathrm{d}a}{\mathrm{d}n} = C_p\, \Delta K^m \tag{2.12}$$

where C_p and m are constants which must be determined experimentally. This equation is limited in that it represents only the linear portion of Figure 2.9 and does not include stress ratio effects. Additional forms discussed by Salvetti et al. include

$$\frac{\mathrm{d}a}{\mathrm{d}n} = C_F\, \Delta K^m \left[(1-R)K_c - \Delta K\right]^{-1} \tag{2.13}$$

$$\frac{\mathrm{d}a}{\mathrm{d}n} = C_1 + C_2 \tanh^{-1} \frac{\log\left[\frac{K_c K_t}{K_{\max}^2}(1-R)^s\right]}{\log \frac{K_t}{K_c}} \tag{2.14}$$

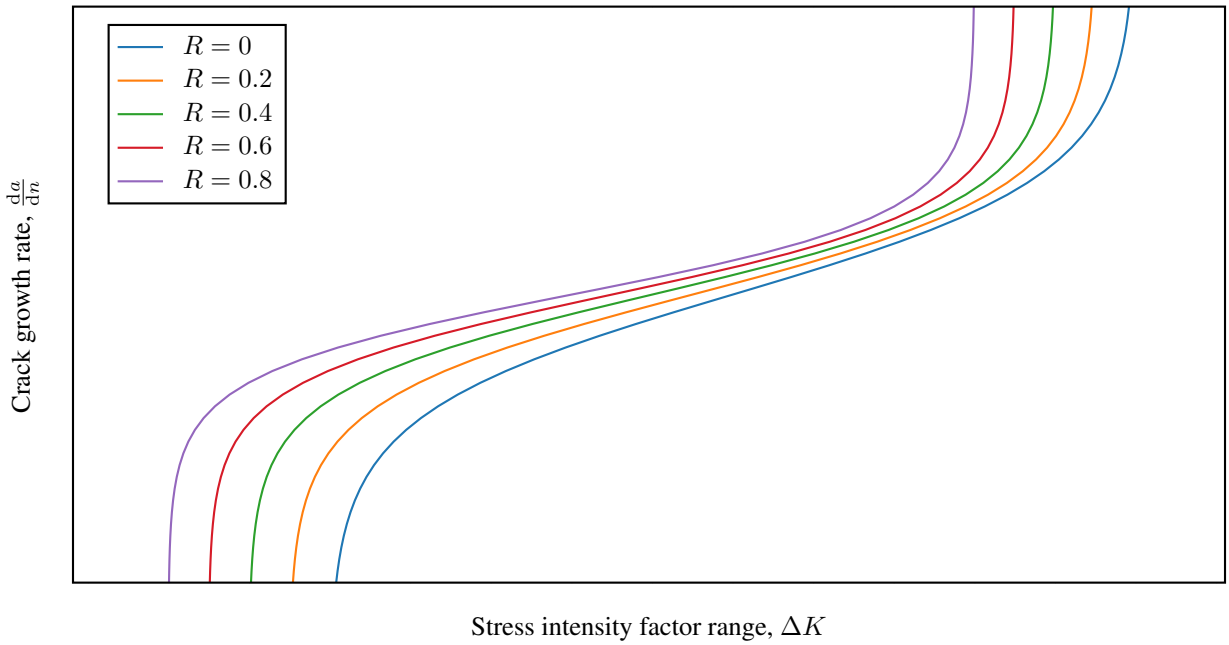


Figure 2.9: Notional crack growth rate curves.

where C_F, C_1, C_2, and s are constants that must be determined experimentally. K_c and K_t represent the critical and threshold values of K, which correspond to the high-rate and low-rate regions of Figure 2.9, respectively. These equations improve upon the Paris law in that they more appropriately capture the sigmoidal shape of the crack growth rate curves as well as the stress ratio effects. However, additional fatigue life experiments must be performed to derive values for each constant.

In constant amplitude loading, it is rather simple to calculate the total fatigue life by rearranging and integrating Equation (2.10):

$$N = \int_{a_0}^{a_f} \frac{\mathrm{d}a}{f(\Delta K, R)} \tag{2.15}$$

where a_0 and a_f represent the initial and final crack length, respectively. However, in the case of spectrum loading, additional complications are added due to interaction effects. For example, the presence of a single peak load, known as an overload, in a constant amplitude loading program will cause a decrease in the subsequent crack growth rate. Conversely, the

presence of an underload can cause an increase in the subsequent crack growth rate. These interaction effects can change the total fatigue life of the specimen by factors of two or more. These effects can be captured in Equation (2.15) by introducing appropriate corrective factors in specific cycles.

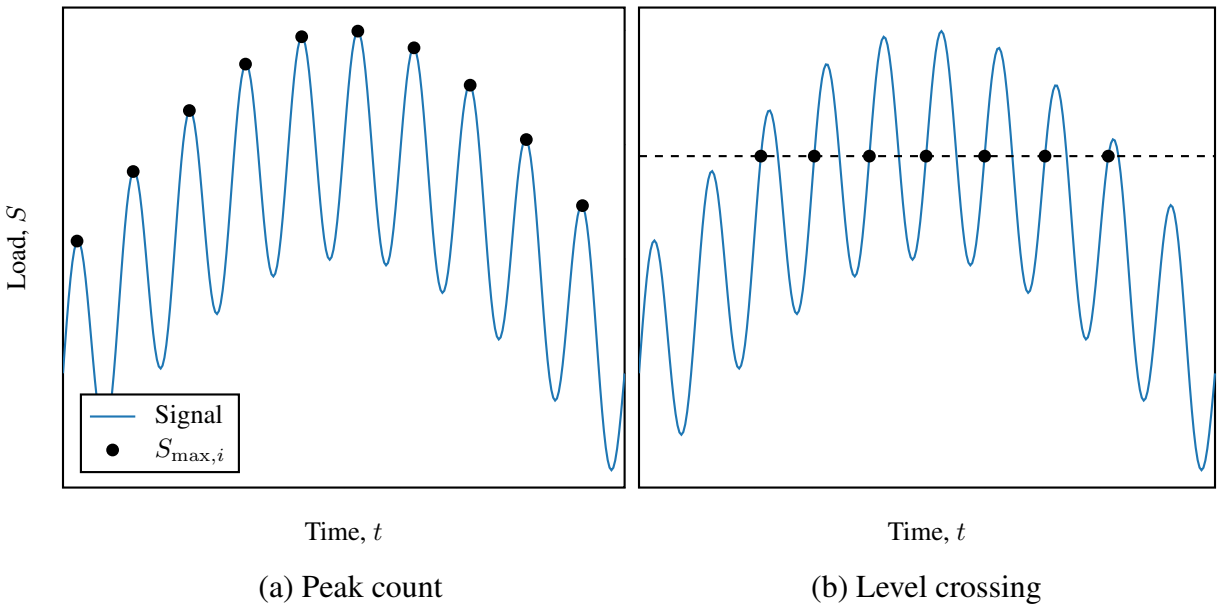
Degrieck and van Paepegem describe additional progressive damage models tailored specifically to composite materials. Many of these models use forms similar to the Paris law to describe the growth of damage modes such as delamination and matrix cracking. Notably, many models are specific to the damage mode under consideration and, as of the time of this writing, there is no single general model that is applicable in all situations.

2.2.1.4 Cycle Counting Methods

Each of the fatigue life prediction models discussed previously relies, in part, on an accurate description of the loading spectrum. In the case of constant amplitude loading, this definition is rather simple: one only needs to specify S_{mean}, S_{range}, and n to define the entire loading program. However, constant amplitude loading is rarely applicable to real-world scenarios. As discussed in Section 1.5, rotary-wing vehicle components will experience a highly diverse loading spectrum over the course of a single flight. The process of deriving a well-defined loading spectrum from a load signal is known as *cycle counting*. The objective of the cycle counting process is to produce a sequence of load peaks ($S_{\text{max},i}$) and valleys ($S_{\text{min},i}$), or load means ($S_{\text{mean},i}$) and ranges ($S_{\text{range},i}$), along with an associated number of cycles (n_i). In some methods, the order of cycles is preserved. This section will present a brief review of cycle counting methods with a focus on those used in rotary-wing literature.

Peak Count and Level Crossing Perhaps the simplest cycle counting method is the peak count method [68]. In this method, the local value of S_{max} is recorded each time a change in the slope of S is noted. Identical or very similar values of S_{max} are grouped and counted together to determine the total cycle count. In some variations, the troughs may

be recorded as well. A similar method, known as the level crossing method, defines one or more threshold values of S. A count is accumulated each time the load signal crosses the threshold in an upward direction [68, 69]. This can be used to filter out small variations in amplitude or remove cycles below the endurance limit that would otherwise be counted using the peak count method. In some variations, downward crossings across a separate threshold level are also counted. Examples of the peak count and level crossing methods for a hypothetical load signal are presented in Figure 2.10.



(a) Peak count (b) Level crossing

Figure 2.10: Examples of the peak count and level crossing cycle counting methods.

Range-Mean An obvious drawback of the peak count and level crossing methods is that only $S_{\max}$ is counted, thus neglecting the ability to calculate S_{mean} and determine the value of R. A more complex method that captures R is known as the range-mean method [62]. In this method, the load signal is divided into segments, each of which is bounded by a peak and its adjacent valley. $S_{\text{mean},i}$ and $S_{\text{range},i}$ is calculated for each segment and stored. The order of segments can be preserved or discarded depending on the requirements of the fatigue life prediction model. Note that in the range-mean method, half-cycles are counted instead of full cycles. An example of the range-mean counting method is presented in

Figure 2.11.

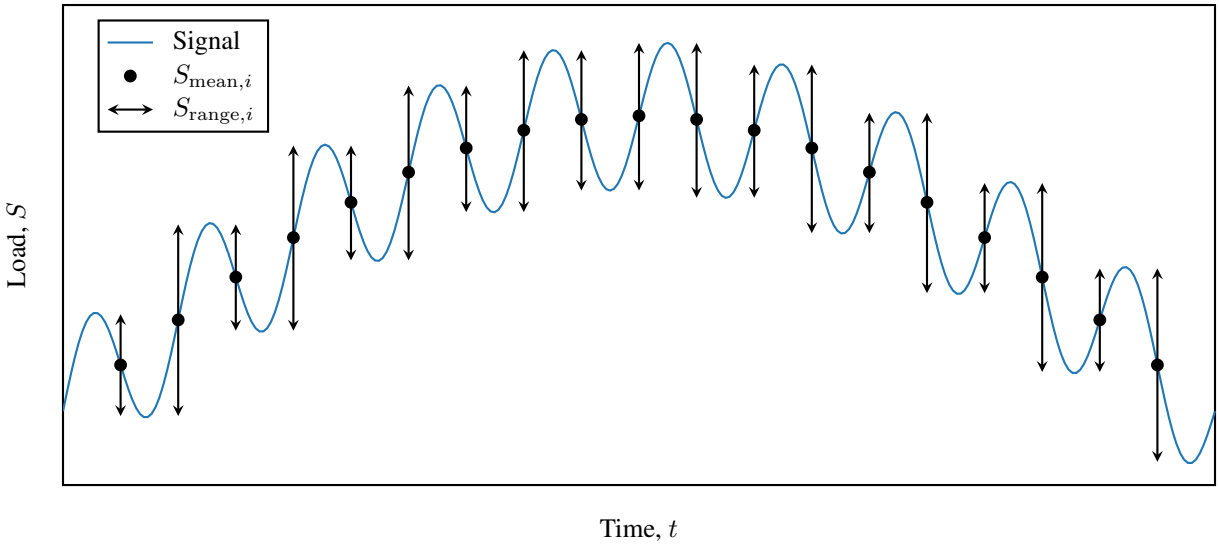
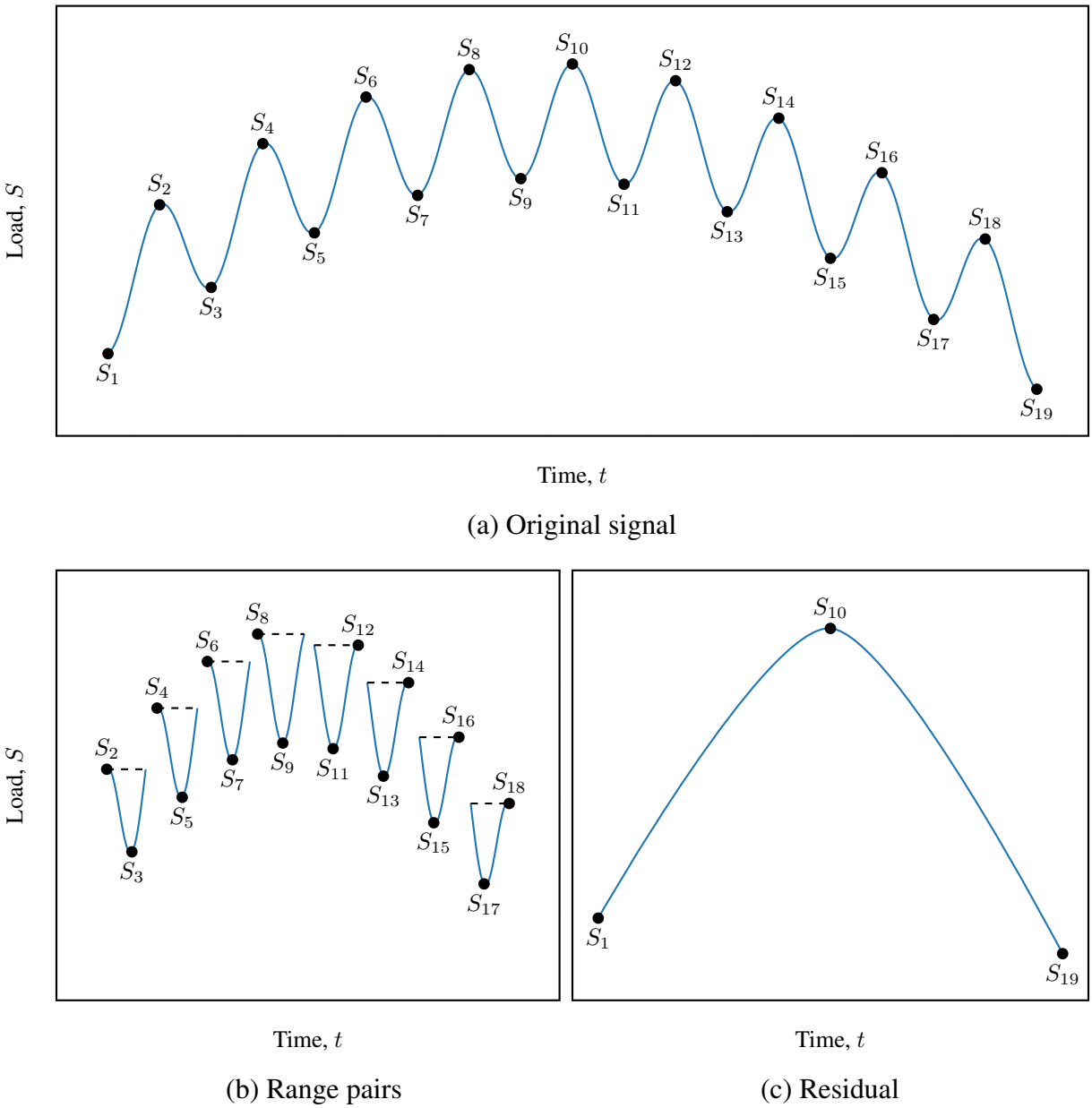


Figure 2.11: Example of the range-mean cycle counting method.

The primary deficiency of the range-mean method can be seen clearly in Figure 2.11. Only the small load oscillations are counted and information regarding the large underlying load oscillation is lost. Considering that a larger value of S_{range} results in more fatigue damage, cycle counting using the range-mean method could lead to non-conservative results if patterns similar to Figure 2.11 are present in the signal. An realization of these patterns in rotary-wing vehicles is the GAG cycle, discussed previously in Section 1.5. In the case of rotorcraft, range-mean counting may capture only HCF cycles while ignoring the damaging LCF cycles.

Rainflow A cycle counting method known as the Rainflow method [62, 68] or the range pair-range [69] method corrects this issue.[2] The Rainflow method consists of a two-phase algorithm. First, the load signal is converted into a sequence of peaks and troughs, which can be represented as a sequence $\boldsymbol{S} = \{S_1, S_2, S_3, \ldots S_k\}$. A hypothetical load sequence is presented in Figure 2.12a.

[2]The Rainflow method was developed in Japan and first published in 1967. Simultaneously, the range pair-range method was developed in the Netherlands and first published in 1969. Both methods produce identical results [69]. In this text, this algorithm will be referred to as the Rainflow method.



(a) Original signal

(b) Range pairs

(c) Residual

Figure 2.12: Example of the Rainflow cycle counting method.

The first phase begins by selecting the first four extrema, $\{S_1, S_2, S_3, S_4\}$. Since the inner extrema, S_2 and S_3, are "contained" within the outer extrema, S_1 and S_4, the range pair S_2–S_3–S_2 is extracted and counted.[3] S_2 and S_3 are removed from the sequence and the algorithm steps to the next four extrema, $\{S_1, S_4, S_5, S_6\}$. If the inner extrema are not contained within the outer extrema, such as is the case when the algorithm reaches

[3] S_2 and S_3 are "contained" within S_1 and S_4 because $S_1 < S_2 < S_4$ and $S_1 < S_3 < S_4$.

$\{S_1, S_{10}, S_{11}, S_{12}\}$, the range pair is not extracted and S_{10} and S_{11} remain within $\boldsymbol{S}$. Instead, the algorithm steps to the next four extrema, $\{S_{10}, S_{11}, S_{12}, S_{13}\}$ and continues until S_{19} is reached. The range pairs extracted in the first phase are depicted in Figure 2.12b.

In the second phase, the remaining points, $\{S_1, S_{10}, S_{19}\}$, are grouped together, extracted, and counted as single ranges. These points are known as the *residual*, and capture the large underlying load oscillation in the original signal. The residual is depicted in Figure 2.12c. A more detailed description of the Rainflow method, including analysis of the results, is given by de Jonge [69].

Several variations of the Rainflow algorithm exist. For example, Nijssen developed a cycle counting method termed the "Rainflow-equivalent range-mean count", which combines the Rainflow and range-mean methods discussed previously into a method which is more suitable for use in strength degradation or crack growth fatigue life prediction models, in which the load spectrum must be evaluated sequentially. This algorithm produces similar results to the Rainflow method but preserves the approximate order of the load cycles.

2.2.1.5 Discussion

The number of fatigue life prediction models, only a minute fraction of which have been discussed in the preceding sections, is large and continually increasing as new models are developed and published. The correct choice of model depends on the structural design, material choice, and load spectrum, but perhaps most importantly on the level of information available to the engineer at the design stage in which the fatigue life prediction is undertaken. For example, application of the progressive damage models requires knowledge of the most critical damage mode, its location, and its geometry. In general, this requires the component or a representative model thereof to be constructed and tested experimentally before the necessary model parameters can be derived [67]. This restricts application of these models to during or after the detail design stage, discussed previously in Section 2.1.1.4.

Fatigue life models and certain phenomenological models are more easily adaptable to

earlier design stages. Because the model parameters can be populated by testing material coupons, they are applicable to parts that have not yet been produced, assuming a sufficient database of material fatigue life data exists. For example, Li, Volovoi, and Hodges applied Schaff and Davidson's strength degradation model to the preliminary design of main rotor blades; this effort will be discussed further in Section 2.3.1. Once the part is constructed and the load spectrum is known, the model parameters can be modified to improve accuracy and the same models can be recycled for a more detailed and accurate analysis.

The choice of material also has a strong influence on the selection of the fatigue life prediction model. Many components of modern rotorcraft, especially those in the rotor system, are constructed of composite materials. Some argue that linear fatigue life models such as Miner's rule are inappropriate for the analysis of composite fatigue due to its non-linear nature. However, Nijssen demonstrated that Schaff and Davidson's strength degradation model *did not* improve significantly upon Miner's rule in the context of wind turbine rotor blade fatigue life prediction. The author concludes that Miner's rule is preferable due to its reduced computational expense and the relative lack of strength degradation data available in material databases.

Progressive damage models are also difficult to use with composite materials: Salvetti, Cavallini, and Fediani state that "as a consequence of the complexity and variety of the damage forms . . . existing [fracture mechanics] methodologies do not lead to useful applications, from an engineering point of view, in the field of composite materials" [67]. Further developments in this field are required to develop a single progressive damage model that is generally applicable to a wide range of materials, geometries, and damage modes.

This discussion, and the discussion in Section 2.1.1, leads to Observation 2.5:

Observation 2.5

Fatigue life models such as Miner's rule and phenomenological models such as Schaff and Davidson's strength-degradation model are most appropriate for use in the rotorcraft preliminary design stage. Miner's rule is preferable due to its simplicity and ease of implementation.

A cycle counting method should be selected based on the fatigue life prediction model of choice. Literature from the rotorcraft industry [4, 56] suggests the Rainflow method is the *de facto* standard for use in rotary-wing vehicles. However, if the fatigue life prediction model in use incorporates sequence effects then the Rainflow-equivalent range-mean counting method developed by Nijssen, which produces similar results to the Rainflow method, is preferable.

2.2.2 Fatigue Design Methods

This section explores the process of applying the theory discussed in Section 2.2.1 to rotorcraft components. First, a brief history of rotorcraft fatigue design is given. Next, traditional processes are described in detail, with a focus on describing competing fatigue design methodologies and aligning individual fatigue design activities to the overall design process discussed in Section 2.1.1.

2.2.2.1 History

Rotorcraft designers have been concerned with the service life of rotor blades since near the inception of the helicopter. In 1935, inventor Jaun de la Cierva specified a service life of 75 FH for the blades of his Cierva C.30 autogyro, which were constructed using a steel spar surrounded by wooden spars and skin [70].

Writing in 1969, Twelvetrees [71] discusses the subsequent evolution of rotor blade construction from that early period until the mid-1960s. Twelvetrees describes the manner by which wooden rotor blades were phased out due to their poor service life. It appears that,

at the time, the primary driver of rotor blade service life was not fatigue damage but the slow warping of the wooden rotor blade until the outer mold line (OML) was altered to the point of being unusable. Wooden blades, which by then had obtained a service life of around 600 FH, were slowly replaced by metal blades. Metal blades did away with the warping issues associated with wooden blades, but are more susceptible to fatigue damage than wood. Early metal rotor blades, typically made of steel, were fatigue-life limited to 1800 FH. Over time, the construction of metal blades was improved until service lives neared 5000 FH. Simultaneously, designs were improved to allow for easier inspections.

Twelvetrees also discusses the possibility of composite blade construction, which, at the time, had not yet been realized. The author correctly predicts that the high strength and low weight of certain composite materials will allow engineers to produce lighter, stiffer blades with longer service lives. It seems that Twelvetrees recognized the extreme importance of a well-designed, long-lived rotor blade, concluding the report by predicting that

> in [the] future, reconsideration of priorities will result in a well engineered blade that is a commercial proposition and thus make a significant contribution to bringing the most sought-after means of transport into successful competition with other types of aircraft in the civil as well as the military fields of operation [71].

In 1961, Ward and Ludi [72] presented an in-depth review of rotor loads research with a particular focus on applications to fatigue life substantiation. This is the earliest substantial work on this subject known to this author. Ward and Ludi discuss the three main building blocks of the fatigue life substantiation process:

1. A list of possible flight conditions and associated load cycle data,
2. The total or percentage of time the vehicle will spend in each flight condition,
3. The rate at which the structure accumulates fatigue damage.

These three factors are the primary focus of nearly every subsequent work related to fatigue life prediction in rotorcraft.

At the time, the main difficulty in the process appeared to be in the prediction of load cycles associated with various flight conditions. Because advanced rotor aerodynamics theories, and the powerful computers needed for their calculation, had not yet been developed, the authors and their contemporaries were required to derive fatigue loads from wind tunnel experiments and flight tests of heavily instrumented rotorcraft. The knowledge of fatigue theory was sophisticated enough to derive a fatigue life specification for a given component once the loads were determined, but this analysis was necessarily done *a posteriori*; that is, fatigue life could not be determined until the vehicle had been developed to the point that wind tunnel tests of the rotor system and flight tests of the prototype aircraft were possible. As discussed in Section 2.1.1, a change to a critical component due to insufficient service life at this stage in the design process would likely be extremely costly and time-consuming.

2.2.2.2 Modern Methods

Rotorcraft fatigue design methods have progressed significantly since the time of Cierva. Advances in computational power, load prediction and measurement, and aeroelastic theories have enabled more rigorous fatigue life predictions that enable rotorcraft designers to specify longer component lifetimes with greater confidence. The fatigue life of modern rotorcraft components is specified using one of two methodologies: the *safe life* methodology or the *damage tolerance* methodology [33].

Safe Life Methodology Components designed according to the safe life methodology are specified with a service life or replacement time and removed once that time is elapsed, regardless of the operational history or current condition of the component. The service life is selected such that the chance of fatigue failure is extremely remote. For example, the U.S. Army specifies that safety-critical components must have a probability of failure of one in a million [73]. This requirement is known colloquially as *six nines reliability*, since the specified probability of failure implies a reliability of 0.999999. A diagram of the safe

life methodology as presented by Lombardo [4] is included in Figure 2.13.

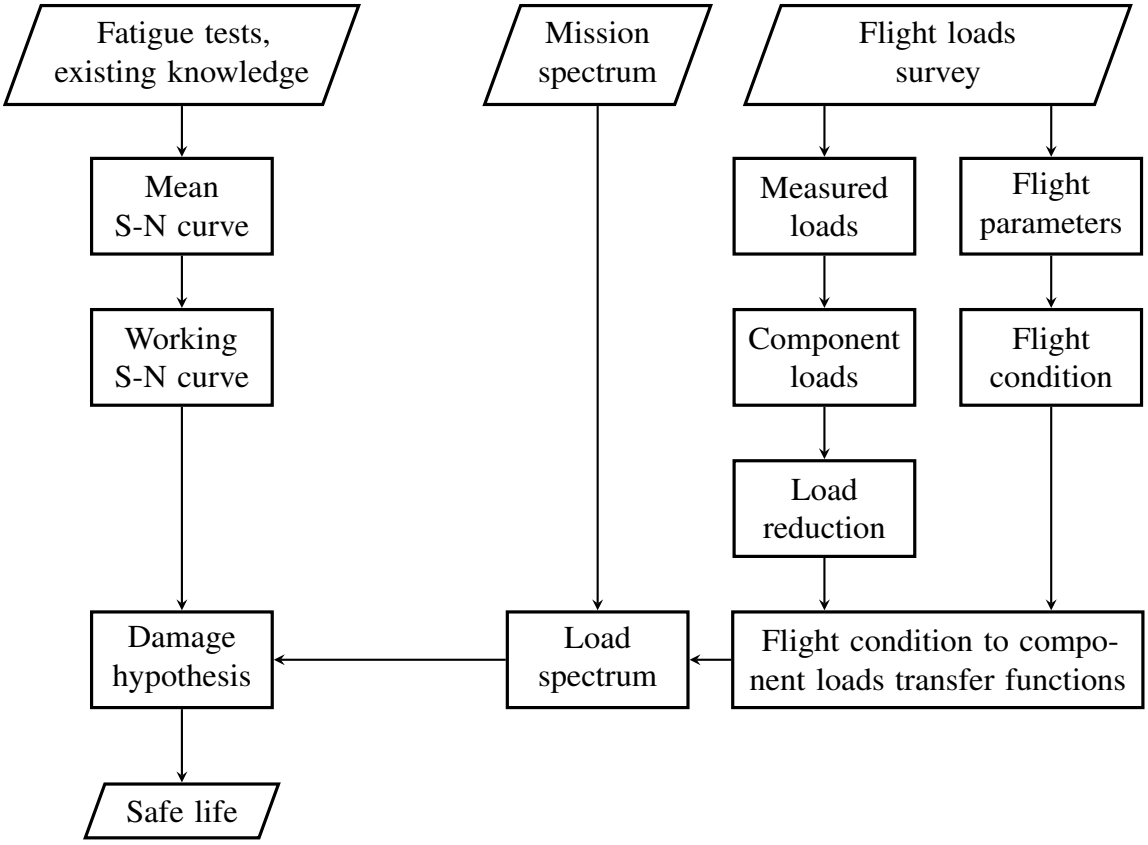


Figure 2.13: Diagram of the safe life methodology, adapted from Lombardo [4] with permission.

Fatigue Tests The top left portion of Figure 2.13 has been described previously in Section 2.2.1.1. Fatigue tests of material coupons and representative components are combined with existing knowledge on prior tests to produce a S-N curve, as in Figure 2.6. Note that the failure points in Figure 2.6 do not align precisely with the curve. Stochastic variability in the material properties, coupon/component construction, and applied load cause scatter in the data. As such, the curve in Figure 2.6 represents the mean fatigue life of all components in the population: 50% of all components will exhibit a higher fatigue life and 50% a lower fatigue life. If the safe life was calculated using the mean S-N curve, half of all components fielded would fail before their specified retirement time, causing an unacceptable loss of life and equipment.

Instead, the safe life calculation is conducted using the *working* S-N curve, which is derived from the mean S-N curve using a number of reductions, or safety factors. Methods used to produce the working S-N curve vary between manufacturers. One common option is to use the $\mu - 3\sigma$ curve, where μ is the mean fatigue life and σ is the standard deviation of the fatigue life. Other options include taking 80% of the mean value curve or drawing the S-N curve through only the lowest points. Some manufacturers use different reduction methods for LCF and HCF regions. These reductions are based mainly on organizational experience [4]. Additional methods for constructing the working curve are given by Facchin and Raggi [74].

Mission Spectrum The rightmost portion of Figure 2.13 describes the process by which the load spectrum, discussed previously in Section 2.2.1.1, is established. First, the mission spectrum is defined. The mission spectrum is a table which defines the expected flight conditions the vehicle will experience and the percentage of time it will spend in each within its operational life. It is typically derived using one or more mission profiles, which are generally included in the RFP as discussed in Section 2.1.1.1. It can also be derived by recording a number of flights of a typical mission type and calculating the percentage of time spent in each condition.

Because helicopters are highly versatile and the actual usage of the vehicle may vary from the manufacturers' original intentions, it is common to include some level of conservatism in the definition of the mission spectrum. For example, rotorcraft used for training will accumulate higher levels of fatigue damage due to the high rate of practice autorotations, as discussed in Section 1.5. This must be preempted in the definition of the mission spectrum to ensure flight safety for the entire fleet. An example transport helicopter mission spectrum given by Stievenard [75] is included in Table 2.1.

Note that each flight condition in Table 2.1 is further divided, or *prorated*, into one of three density altitudes; density altitude has a strong impact on fatigue loads as discussed in

Table 2.1: Example transport helicopter mission spectrum, from Stievenard [75].

	Percentage of total time (%)			
Condition	Sea level	1500 m	3000 m	Total
Hover in-ground-effect (IGE)	4.450	0.500	0.050	5
Hover OGE	4.450	0.500	0.050	5
Rearwards flight at maximum permitted speed	0.356	0.040	0.004	0.4
Start of forward flight from rearward flight	0.089	0.010	0.001	0.1
Sideways flight to the right at maximum speed	0.890	0.100	0.010	1
Sideways flight to the left at maximum speed	0.890	0.100	0.010	1
Rudder reversal from turn to right in hover	0.445	0.050	0.005	0.5
Rudder reversal from turn to left in hover	0.445	0.050	0.005	0.5
Transition speed	2.670	0.300	0.030	3
Turn to right at transition speed	0.890	0.100	0.010	1
Turn to left at transition speed	0.890	0.100	0.010	1
$0.5V_{\mathrm{NE}}$	7.120	0.810	0.070	8
$0.7V_{\mathrm{NE}}$	6.206	0.724	0.070	7
Turn to right at $0.7V_{\mathrm{NE}}$ (40°)	1.335	0.150	0.015	1.5
Turn to left at $0.7V_{\mathrm{NE}}$ (40°)	1.335	0.150	0.015	1.5
$0.85V_{\mathrm{NE}}$	15.120	1.710	0.170	17
Turn to right at $0.85V_{\mathrm{NE}}$ (40°)	1.780	0.200	0.020	2
Turn to left at $0.85V_{\mathrm{NE}}$ (40°)	1.780	0.200	0.020	2
$0.9V_{\mathrm{NE}}$	25.810	2.900	0.290	29
$0.95V_{\mathrm{NE}}$	1.980	0.498	0.022	2.5
V_{NE}	0.890	0.098	0.012	1
$1.11V_{\mathrm{NE}}$	0.445	0.050	0.005	0.5
Oblique climb at maximum power	2.136	0.240	0.024	2.4
Vertical climb at maximum power	2.136	0.240	0.024	2.4
Autorotation at minimum RPM	0.090	0.009	0.001	0.1
Autorotation at maximum RPM	0.090	0.009	0.001	0.1
Load factor less than unity	0.445	0.050	0.005	0.5
Approach	3.015	0.350	0.035	3.5
Flare	0.356	0.040	0.004	0.4
Quick stop	0.090	0.009	0.001	0.1

Section 1.5. It is also common to prorate the conditions further based on gross weight or center of gravity location [76]. Also note that even though level cruise segments at various speeds account for 64.5% of the time in this mission spectrum, far less common conditions are also included because they can produce far more fatigue damage than level flight despite their minimal duration.

Load Spectrum Once the mission spectrum is defined, the fatigue loads associated with each condition must be determined. The most accurate method for determining component loads is to conduct a flight survey of the conditions in Table 2.1 using a heavily-instrumented vehicle. These instruments typically include transducers which can measure displacement, velocity, acceleration, or strain on a component; component loads can later be derived from these measurements. These data can be stored on the vehicle for later analysis or transmitted wirelessly during the flights. Additional measurements of flight parameters are used to define an associated flight condition for each portion of the load history. The process of component load measurement is described in more detail by Jorio [77].

Because load data measured during flight is subject to scatter in a manner similar to the S-N data, additional reductions or safety factors are applied to the load data to ensure acceptable reliability. A common method is to fly the same condition multiple times and use the highest loads recorded; this is known as the *top-of-scatter* (TOS) method [78]. After load reduction is complete, a load spectrum is formed by correlating the mission spectrum with the component loads in each condition using a transfer function.

Safe Life Calculation Finally, the safe life of the component is calculated by applying an appropriate damage hypothesis. The damage hypothesis consists of a fatigue damage theory and a cycle counting method, as discussed in Sections 2.2.1 and 2.2.1.4. The determined safe life is translated to a replacement time specification, possibly involving further application of safety factors, which is included in the maintenance manual for the vehicle model. The component in question must be replaced after its safe life is consumed, regardless of the condition of the component, for the vehicle to remain airworthy. The safe life methodology achieves the required level of component reliability through combined reductions or safety factors on the S-N curve, the component loads, and the mission spectrum [79].

Damage Tolerance Methodology Components designed according to the safe life methodology suffer a number of drawbacks. Primarily, the requirement to retire each component after the safe life elapses results in the premature retirement of most parts. For example, assuming six nines reliability, only one part in 1,000,000 will actually suffer fatigue failure by the end of its safe life. The remaining 999,999 parts will be discarded despite having useful fatigue life remaining. Additionally, it is difficult to capture the influence unpredictable phenomena such as manufacturing defects, maintenance errors, or operational damage [33]. Finally, parts designed according to the safe life methodology may be overly conservative, resulting in a weight penalty [80].

The damage tolerance methodology offers an alternative approach to deriving fatigue life and can result in significant cost savings if applied appropriately. In general, damage-tolerant structures are designed to *contain* damage rather than avoid it, as in the safe life methodology. A regular inspection interval is prescribed in order to detect any damage before it leads to failure of the component. The duration of the inspection interval is set based off of the calculation of the time between the coalescence of visible damage and the failure of the structure. The damage tolerance methodology is closely related to and dependent upon the progressive damage models discussed in Section 2.2.1.3.

Arden, Chappell, and Reddick [33] describe the two primary categories of damage-tolerant structures, which they term *fail-safe* and *safe crack growth*. Fail-safe structures are those that can successfully experience damage without causing a significant flight safety risk. For example, structures such as the fuselage can be designed with multiple load paths such that if one fails, another is available to take the remaining load. Structures can also be designed with features to arrest crack growth before the crack propagates catastrophically. Safe crack growth implies that the (single load path) structure has been sized such that the crack will grow slowly at a stable rate not propagate completely through the structure before the next inspection. In both cases, the structure is designed to maintain acceptable strength to allow for safe operation in the interval between inspections.

Due to the wide variety of damage-tolerant approaches and the complexity thereof, the damage tolerance methodology is not described in great detail in this document. The building blocks are similar to those of the safe life methodology, except that a progressive damage theory such as a crack propagation model is used instead of Miner's rule. Additionally, the safety and reliability of the structure must be predicted in undamaged and damaged states. Finally, rigorous inspection methods must be designed and proven to be capable of detecting the damage mode in question. A more detailed treatment of the damage tolerance methodology is given by Arden, Chappell, and Reddick [33], Amer [80], and Reddick [81].

Although damage-tolerant structures may offer better safety and longer fatigue lives than those designed using the safe life methodology, there are some difficulties in the application of this methodology. Primarily, the damage tolerance methodology is subject to the same drawbacks associated with progressive damage models, as discussed in Section 2.2.1.3. The lack of a unified, generally applicable model for composite materials adds additional time and expense to the fatigue substantiation process, as models may have to be tailored to individual components and damage modes. Additionally, it is difficult to apply the damage tolerance methodology to some of the most critical parts of the vehicle. For example, helicopter dynamic components typically only have one load path, preventing the use of the fail-safe approach. Inspections of these components may be impossible or overly difficult and time-consuming, complicating the use of the safe crack growth approach. Thus, while the damage tolerance methodology can be easily applied to certain components, like the airframe, the safe life methodology remains the dominant choice for many others [4].

2.2.2.3 Phases of Fatigue Design

The fatigue design methodologies presented in the previous section may seem monolithic but in fact are an idealization of a complex procedure that is executed in a staged, iterative manner throughout the design, development, production, and in-service phases of the aircraft fleet's life-cycle. This section provides an overview of the various phases of rotorcraft fatigue

design and draws connections to the design stages discussed in Section 2.1.1. For reference, a diagram of the phases of fatigue design, adapted from Arden, Chappell, and Reddick, is presented in Figure 2.14.

Design Phase The design phase encompasses the requirements definition, conceptual design stage, preliminary design stage, and detail design stage, as outlined in Figure 2.1. As part of the requirements process, the customer or another stakeholder will specify the required mission profile(s), the desired service life of critical components, the fatigue design methodology to be used, and the methods of qualification. Typically, only military customers include fatigue life as a requirement; for civilian rotorcraft, desired fatigue life is decided upon by the manufacturer. These decisions will strongly influence the service life, and in turn the safety and expense, of the final vehicle [33].

According to Arden, Chappell, and Reddick, little fatigue design is done in the conceptual design stage. As discussed in Section 2.1.1.2, this stage is mainly concerned with defining to the overall configuration and performance of the vehicle, and the level of knowledge necessary to define the fatigue life of a specific component is not yet available.

The initial fatigue design activities take place during the preliminary and detail design stages. Traditionally, the structural layout and material selection of critical components is established to some degree during the preliminary design stage. During the detail design stage, the structural layout and sizing of critical components is refined to achieve the desired fatigue life. Because physical components of the vehicle are most likely not available in this stage, fatigue life data is populated using fatigue testing of material coupons rather than representative components.

Clearly, the flight loads survey specified in Figure 2.13, which is a critical component of any fatigue design methodology, is also unachievable at this stage. Instead, fatigue loads and the resultant fatigue life may be estimated using low-fidelity analytical studies or extrapolations based on historical data. For example, the Army Materiel Command [82]

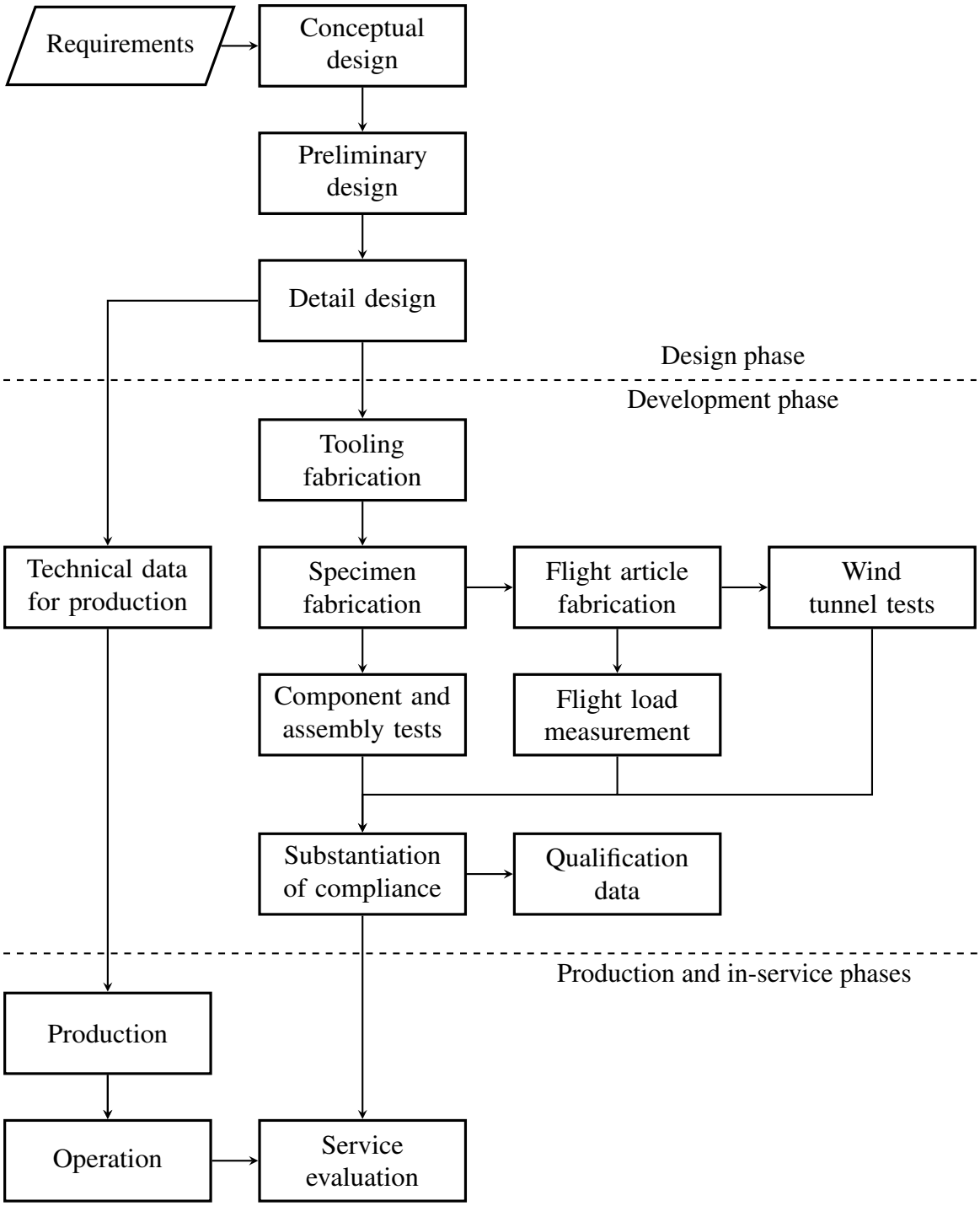


Figure 2.14: Phases of rotorcraft fatigue design, adapted from Arden, Chappell, and Reddick [33] with permission.

describes a method by which fatigue life can be estimated using only a single load condition.

The component loads associated with a specific flight condition, such as high speed flight,

are measured using a preexisting similar vehicle or predicted using analytical methods such as the comprehensive analyses discussed in Section 2.1.3. Then, corrective factors are applied to these loads to account for changes from the baseline in material, flight condition, and desired fatigue life. These corrective factors depend on certain fundamental physical parameters of the vehicle and are to be derived using statistical analysis of flight load surveys from previously-designed helicopters.

Alternatively, designers can use a standardized fatigue loading sequence, known as *loading standards*, to substitute for an actual load spectrum. Loading standards are defined fatigue load spectra that are intended to represent a specific structure or use case. They are intended to be used a basis for material fatigue testing and for the assessment of analytical fatigue life prediction techniques. Although loading standards are not intended for use in the final fatigue testing of specific aircraft components, they can provide a reasonable preliminary estimation of the loads a component may be subjected to in service [83]. For helicopters, the HELIX and FELIX [84, 85] loading standards can be used. These standards are intended to represent the loads produced by articulating and rigid rotor systems, respectively, and are derived from flight load surveys of helicopters of each type.

Development Phase The development phase begins once the manufacturer commits resources to the production of the vehicle. Although Figure 2.14 implies the development phase is fully subsequent to the design phase, in reality, their may be overlap between the two for scheduling purposes. This overlap blurs the distinction between the two phases; some authors, such as Leishman [30], consider certain development phase activities to be a part of the late detail design stage.

A critical fatigue design activity during the development phase is the construction and testing of full-scale representative components for laboratory testing. Fatigue data derived from these experiments will replace the coupon test data used during the design phase. Simultaneously, prototypes of the complete aircraft and the rotor system will be constructed

for flight and wind tunnel testing. This enables accurate flight load measurements, which will replace the load estimates used previously in the design phase.

With all the key elements in place, the fatigue life of each component is reevaluated using the desired fatigue design methodology and compliance with the fatigue requirements is evaluated by the qualifying agency. If the required fatigue life is not met, redesign of the components in question will likely be required. This requires developing new production tooling, constructing new representative components, repeating the component fatigue tests, and possibly repeating wind tunnel and flight testing, which significantly impacts the cost and schedule of the program.

Production and In-Service Phases Although the majority of fatigue design activities have been completed by the conclusion of the development phase, certain processes continue into the production and in-service phases. For example, quality control is constantly conducted during the production phase. A few specimens of each component are removed from the production line regularly and tested using the same fatigue testing methods as in the development phase. If the test fails, then it is likely that some drift in production quality has caused the components to fall below the defined fatigue life requirements and production may be paused to allow time for correction. Quality assurance is especially important in the case of composite materials because small differences in the manufacture of these components can result in wide variation in material properties [33].

Additionally, the mission spectrum may be revised during the in-service phase based on the actual usage of the aircraft, leading to reevaluations of the specified service life. Accident records collected during the operational life the model can be analyzed to determine if redesigns or modifications are necessary. For example, as of 1983, the Bell 47, a light GA helicopter, experienced 42 accidents attributable to fatigue failure of the tail rotor blade, seven caused by the tail rotor blade yoke, and six caused by the tail rotor retention bolt [24]. It is not known to this author whether corrective action was taken in this case.

2.2.2.4 Discussion

The traditional fatigue design methods discussed in this section are typified by their conservatism as well as their reliance on flight test data. The need for conservative methods is self-evident: it is much more disastrous to specify an overly-optimistic service life than a pessimistic one. In order to prevent catastrophic loss of life and equipment, designers and regulatory agencies have established stringent safety requirements, such as the six nines reliability requirement. Uncertainty in material fatigue properties, mission spectra, and fatigue loading force engineers to design to the "worst case" fatigue life: that is, a vehicle constructed of the weakest material, performing the most damaging maneuvers, and subject to the highest possible loads for each flight condition should still be safe for the specified service life. It is well-understood that this practice leads to the waste of some components due to aggressive retirement schedules, but this is considered to be an acceptable trade-off for increased flight safety.

However, the methods used to derive this conservatism are somewhat arbitrary and vary from organization to organization. This section discussed the manner by which safety factors or reductions are applied to fatigue test data, flight load data, and mission spectra to produce an appropriate level of reliability for the entire vehicle fleet. Generally, these reductions are not standardized and are left to the discretion of the manufacturer, subject to approval from the qualifying agency [33]. This can result in wide variability in results. For example, the American Helicopter Society (AHS)[4] posed a hypothetical pitch link fatigue life problem, asking a number of rotorcraft manufacturers to predict the fatigue life of the pitch link based on the same source data [86]. The results of this exercise are given by Lombardo and included in Table 2.2.

Although Table 2.2 may be an extreme example, it is evident that the lack of standardization in the safe life methodology results in variation across manufacturers. Additionally, because many of the safety factors and reductions are based on the given manufacturer's ex-

[4]The AHS is now known as the Vertical Flight Society (VFS).

Table 2.2: Calculated fatigue lives for the hypothetical pitch link problem, from Lombardo [4].

	Predicted fatigue life (FH)	
Manufacturer	Block counting method	Manufacture's preferred cycle counting method
Aerospatiale	9	58
Augusta	804	6450
Bell	1831	27,816
Boeing-Vertol	1294	22,523
Hughes	2594	24,570
Kaman	861	56,901
Sikorsky	240	470

perience, the reasoning behind each may be difficult to trace. This leads to Observation 2.6:

Observation 2.6

Traditional methods used to achieve high fatigue life reliability vary across organizations and are based on organizational experience rather than rigor.

Additionally, the safe life and damage tolerance fatigue design methodologies are highly dependent on flight load surveys. These surveys are a necessary and irreplaceable part of the final fatigue life substantiation and qualification process, as they provide the most accurate representation of the loads a component will experience in service. The previous literature review also indicates that flight surveys play a major role in the early stages of design: preliminary and detail design activities are dependent upon historical flight load surveys of similar helicopters to predict load data for the current vehicle. Although some sources [4, 33] mention the use of analytical load prediction techniques, it appears these are used less frequently and treated with greater suspicion than flight loads data.

These practices are most likely not appropriate for the revolutionary rotary-wing configurations as described in Section 1.3. In advanced and revolutionary vertical lift configurations, historical loads data gathered from SMR helicopters and other traditional configurations are no longer applicable. Complex configuration-dependent interactions, such as those identified by Avera and discussed in Section 1.3, will not be captured by historical datasets

and fatigue life predictions derived from this data may be flawed. If traditional methods are used in the design of complex rotary-wing vehicles, components may have unacceptably low fatigue lives, necessitating frequent component replacements or costly redesigns to improve their service lives. This leads to Observation 2.7:

Observation 2.7

Historical flight load surveys used in the rotorcraft fatigue design process may produce erroneous predictions when applied to revolutionary rotary-wing configurations.

2.3 New Approaches to Fatigue Design

The traditional fatigue design methods discussed previously comprise the most common procedures in use by the rotorcraft industry, its customers, and its regulators. These methods have evolved over the decades. Namely, the introduction of the damage tolerance methodology has lead to a decrease in the prevalence of parts designed using the safe life methodology, where applicable. New, more thorough fatigue life prediction models have helped improve reliability and decrease uncertainty in the service life estimates. However, while elements of the methodology evolve, the overall process remains similar to the processes depicted in Figures 2.13 and 2.14.

In this section, several new, and largely academic, approaches to the fatigue design process are reviewed and discussed. The primary focus is on approaches that are intended to integrate with or complement the vehicle design tools discussed in Section 2.1.2. In general, these approaches also tend to make use of the physics-based rotor design tools discussed in Section 2.1.3.

For reference, Literature Question 3, which is the focus of this section, is reprinted below:

Literature Question 3

How have other researchers improved upon the traditional rotorcraft fatigue design process?

2.3.1 Structural Design Against Fatigue Failure for Composite Rotor Blades

Li, Volovoi, and Hodges [66] (see also Li [87]) developed a composite rotor blade design process which uses fatigue failure as a constraint. The design framework integrates the blade cross-sectional design and analysis tool VABS [88], the rotorcraft comprehensive code Dymore [89], and an in-house fatigue analysis based on Schaff and Davidson's strength degradation model. Li's design framework is depicted in Figure 2.15.

The framework uses a single objective function which seeks maximum performance by minimizing cross-sectional mass and ensures aerodynamic stability by minimizing the distance between the aerodynamic center and the shear center. Additional constraints ensure the first six natural frequencies of the blade are not near the integer multiples of the rotor speed to avoid resonant conditions. The minimum acceptable fatigue life of the rotor blade is also included as a design constraint.

The optimization process begins by generating a mesh of the rotor blade cross-section in VABS. The cross-section is defined by a VABS geometry template which, when combined with the initial design variables, generates a VABS input file. The design variables enabled by the template include skin thickness and layup angle, web thickness and layup angle, several geometric properties of the D-spar, and the weight of a non-structural leading edge mass. VABS calculates sectional mass and stiffness properties for the cross-section, which are then used to build a multi-body dynamic analysis model in Dymore. VABS will be discussed further in Section 4.4.2.3.

Because the aeroelastic analysis and trim procedure is very computationally intensive, the natural frequency constraints are checked using a simpler *in-vacuo* analysis. If the resonance constraints are satisfied, then the rotor is trimmed to the desired flight condition(s) in Dymore. The spanwise loads on the rotor blade are recorded for the duration of the flight condition and used for the process of stress and strain recovery in VABS. The recovery process produces a time history of the stress and strain fields at each node in the cross-section mesh; this data is then used to calculate the fatigue life as described in Section 2.2.1.3.

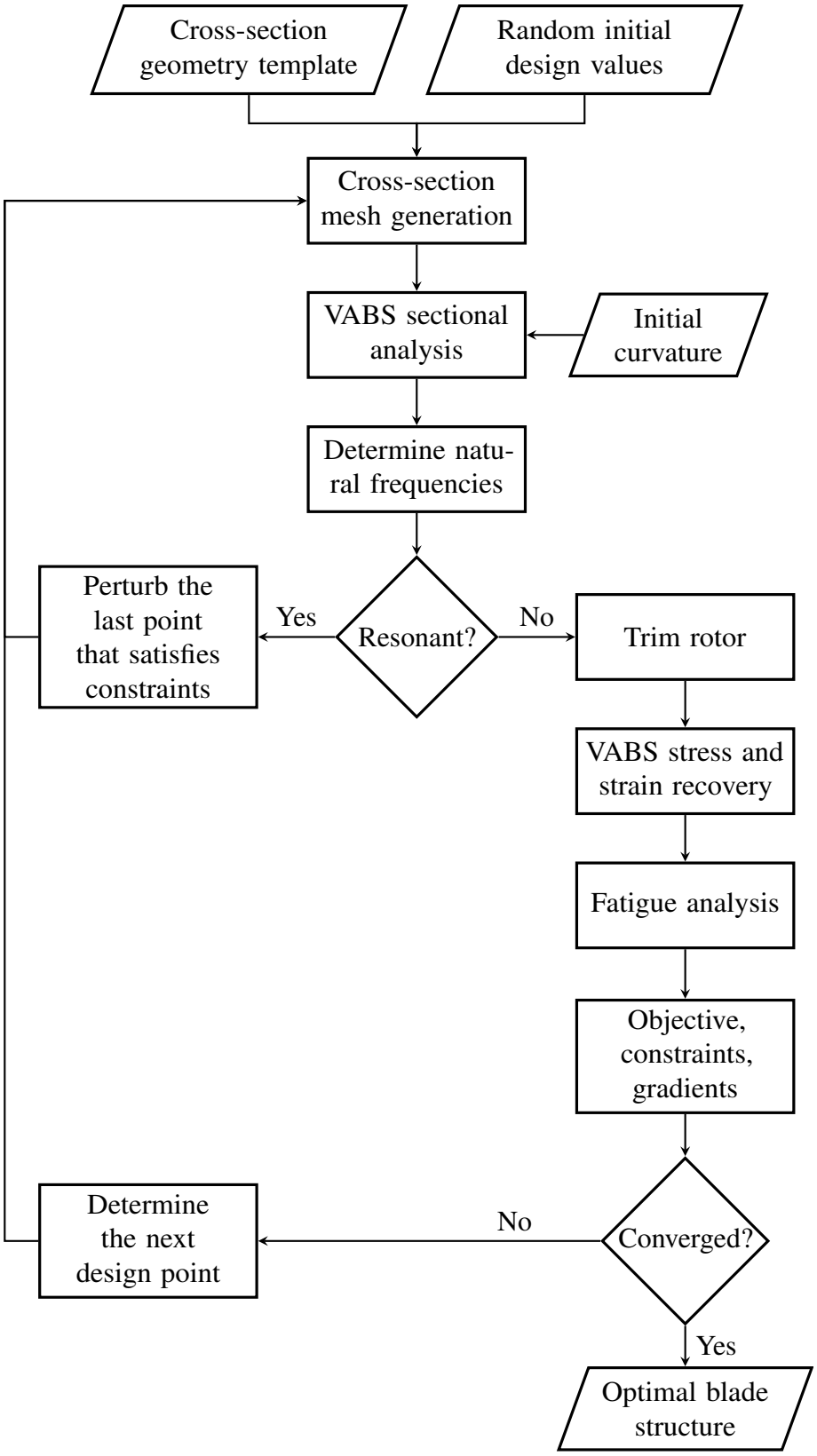


Figure 2.15: Structural design framework developed by Li, adapted from Li, Volovoi, and Hodges [66] with permission.

Next, the objective function, constraint functions, and gradients thereof are calculated. The framework uses a sequential quadratic programming optimization algorithm to search for the lightest and most stable cross-section design that meets the fatigue life requirement. Li's case study centers on the optimization of a three-bladed hingeless rotor system. Only a single flight condition is analyzed: steady forward flight at $V = 123\,\mathrm{ft/s}$ and $T = 1500\,\mathrm{lbf}$. The fatigue life requirement is set to at least 10,000 FH. Sixteen randomized initial designs were used to produce five local optima that satisfied all the constraints. The best of these optima was significantly lighter than the baseline design and features a calculated fatigue life of 15,800 FH. The fatigue life of the baseline design is not known.

2.3.2 Impact of Active Rotor Technologies on Fatigue Life

Arruda, Hamel, and Collins [90] developed a quantitative technology analysis framework intended to assess the impact of active rotor technologies (ARTs) on rotorcraft performance, effectiveness, and LCC. This work essentially "bridges" the rotor design tools discussed in Section 2.1.3 with the RAM-C–focused vehicle tools discussed in Section 2.1.2.2. The impact of a given ART is assessed using physics-based rotor analysis tools and the resulting impact on LCC is determined using a discrete event simulation (DES).

In the example provided in the work, Arruda, Hamel, and Collins demonstrated a framework intended to test the impact of higher-harmonic control (HHC) on pitch link fatigue life. HHC systems supplement the 1/rev swashplate control inputs using additional control inputs of higher frequencies. The amplitude, frequency, and phase of the HHC inputs can be varied to tune the rotor blade response to produce the desired results. Generally, HHC is intended to reduce hub vibrations, but in this case is used to minimize the pitch link load amplitude.

Arruda, Hamel, and Collins used RCAS to model the rotor system with HHC added. The rotor system was simulated across a number of flight conditions, such as hover, steady climb, transient maneuvers, and forward flight at various airspeeds, reflecting the operational reality

of the vehicle in question. Ideal values for HHC phase, frequency, and amplitude were found by sweeping the values of these parameters over a predefined range and selecting the combination which most reduced pitch link load amplitude and vertical hub force amplitude. The pitch link loads were recorded and applied to an ANSYS finite-element model of the pitch link to determine the peak stresses on the part. The authors then applied Miner's rule to the pitch link to determine the fatigue life of the pitch link. It was estimated that the improvements in fatigue life obtained by implementing HHC would result in a reduction in pitch link replacements of 20% for a notional helicopter fleet.

2.3.3 Applications of Surrogate Modeling to Fatigue Design

The discussion in Sections 2.2.1 and 2.2.2 repeatedly emphasizes the dependence of fatigue life prediction methods on vast quantities of data. Constructing the appropriate fatigue life prediction model requires an extensive fatigue life testing program using material coupons or representative components. Developing the load spectrum requires a thorough flight loads survey, which must capture a large number of flight conditions and maneuvers to adequately account for the expected in-service usage of the entire fleet. Because of the stochastic nature of these processes, identical tests must be carried out multiple times to provide an estimate of the mean and the variance of the results. For example, a single maneuver in a flight loads survey may be repeated multiple times to capture variations in the resulting fatigue loads [78].

These testing requirements contribute significantly to the cost of vehicle development and certification. In some cases, the amount of testing may be reduced for economic and schedule reasons. This forces the manufacturer to apply more conservative safety factors or reductions to maintain adequate fatigue reliability given reduced certainty [4].

Surrogate modeling is a tool that could potentially be used to reduce the number of test cases required in certain situations. The basic principle of surrogate modeling is to approximate a complicated or expensive function, $f(\boldsymbol{x})$, such as a rotor system aeroelastic

analysis, by building a surrogate model, $\widehat{f}(\boldsymbol{x})$, that approximates the output of $f(\boldsymbol{x})$ but can be executed more rapidly. Because $\widehat{f}(\boldsymbol{x})$ does not reproduce $f(\boldsymbol{x})$ exactly, there will be some associated error $\varepsilon(\boldsymbol{x})$. The goal is to use an appropriate surrogate modeling technique to reduce the magnitude of $\varepsilon(\boldsymbol{x})$ for the desired range of the input variables, $\boldsymbol{x}$. This surrogate model can then be evaluated in the place of the original function, generating new outputs without requiring costly and time-consuming analysis [91].

2.3.3.1 Regression Analysis of the Load Spectrum

Several applications of surrogate modeling in the context of the rotorcraft fatigue design problem can be found in the literature. The first is a study on establishing a relationship between fatigue loads and flight condition using response surface methods by Zion [76]. The objective of this study was to analyze trends in fatigue loads for specific critical components of the CH-53 heavy-lift transport helicopter by building a surrogate model based on data collected during flight loads surveys. Then, the influence of each independent variable can be analyzed leading to more informed construction of further flight loads surveys. Additionally, the surrogate model could be used to predict fatigue loads for flight conditions or maneuvers that have not been flown, potentially reducing the number of test points in the program. This would allow the manufacturer to construct a detailed load spectrum at a low cost.

Response surface methods are a surrogate modeling technique in which a functional relationship is established between the dependent variable, y, and the k independent variables, $\boldsymbol{x}$, using regression analysis [92]. Zion uses linear response surface equations of the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + \varepsilon_i \tag{2.16}$$

where β_j are linear coefficients and i denotes the sample point. The values of β_j are estimated using a linear regression process which minimizes the sum of squared errors (SSE)

for all n sample points, which is defined as

$$\text{SSE} = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \ldots - \beta_k x_{ik})^2 \tag{2.17}$$

The resultant least-squares estimates of the β_j coefficients are termed b_j and used to form the response surface equation:

$$\widehat{y}_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_k x_{ik} \tag{2.18}$$

Equation (2.18) can then be used to generate outputs for additional sets of dependent variables not included in the original n data points. Assuming that the surrogate model is well-developed, the error should be low for most new cases, as long as the independent variables are kept within the ranges used to construct the response surface equation. Response surface methods are discussed further in Section 4.1.1.

Zion used surrogate models to predict steady and vibratory loads for the main rotor pushrod and the tail rotor spindle of the CH-53. Separate response surface equations were produced for each flight condition and maneuver of interest, including level cruise, steady state climb, 30° turns, 45° turns, and climbing turns. Dependent variables included load factor, airspeed, density altitude, gross weight, engine torque, rotor RPM, rate of climb, pilot control positions, and vehicle attitude, although not all dependent variables were found to be significant influences on fatigue loads in all cases. Zion demonstrated that, with the application of appropriate modeling techniques, well-developed surrogate models can be used to predict fatigue loads for all flight conditions with reasonable accuracy. The author also noted the importance of evenly sampling a wide range of the dependent variables to produce a flexible surrogate model with low error. However, it is not known to this author whether Zion's work has been applied to the fatigue design of any production vehicles.

2.3.3.2 Prediction of Stress and Strain

Another study of interest focuses on the use of surrogate modeling to predict the maximum stress and strain in a rotor blade cross section by Schank [93]. Schank constructed a rotor blade analysis framework to determine optimal trim conditions for complex rotary-wing aircraft with non-unique trim solutions. The trim conditions are constrained by the ultimate strength of the rotor blade material, which must resist aerodynamic loads imparted by gusts of wind. In order to evaluate these constraints, Schank uses the aforementioned VABS tool to calculate the internal stress and strain at critical points in the rotor blade. Load data is provided by the comprehensive analysis program RCAS.

Schank notes that the process of recovering internal stress and strain from applied loads and moments in VABS is extremely time consuming, requiring approximately of 7 h for a 5 s gust simulation with 1000 time steps. To combat this, the author implemented an artificial neural network to predict the maximum strain in the blade cross-section based on the applied loads and moments. Neural networks are a non-linear surrogate modeling technique inspired by analogy to a the human brain [91]. This type of surrogate modeling will be discussed further in Section 4.1.2. In general, this modeling technique is much more flexible, but also much more complex, than the linear response surface methods discussed previously.

The neural network was able to predict the maximum strain with no more than 5% error in approximately 1 s. However, Schank notes that this technique is only cost-effective if the total neural network training time is lower than the time required for a complete stress and strain recovery in VABS. The training process must be repeated each time the design of the cross-section is changed, possibly limiting the application of this process in the preliminary design phase where the cross-section design may change rapidly. Although the author does not apply his technique to fatigue life prediction, he notes that the computational cost of this problem could be greatly reduced using similar methods.

2.3.4 Reliability of Fatigue Life Predictions

As discussed in Section 2.2.2.2, the safe life methodology achieves an appropriate fatigue reliability through the use of aggressive reductions and safety factors. This allows engineers and regulators to specify retirement times that ensure flight safety despite variations in material strength, rotor loads, and vehicle usage. However, the level of reliability provided by the safe life methodology is not typically quantified; rather, it is accepted that the probability of failure, P_f, is "extremely remote" based on organizational and operational experience [79].

Thompson and Adams [79] developed a numerical Monte Carlo simulation to quantify the reliability level of safe life calculations. Monte Carlo simulation will be discussed further in Section 5.1.3. The objective of the study was to do away with commonly used safety factors to better represent the "true" nature of fatigue damage. Thus, key parameters were modeled with random variables and the fatigue life prediction was accomplished probabilistically rather than deterministically.

The simulation requires statistical distributions for material strength, fatigue loads in each flight condition, and frequency of occurrence of each flight condition. These distributions were created by fitting normal, lognormal, and Weibull distributions to historical data recorded during material tests, flight loads surveys, and usage surveys. The calculation proceeds by simulating the operations of a single UH-60 vehicle with component strengths assigned by sampling the material strength distribution. Throughout the simulation, different maneuvers and flight loads are applied to the component by sampling from their respective distributions. When the cumulative fatigue damage reaches its limit, the age of the component at failure is recorded and the simulation is reset and repeated.

If the simulation is repeated a sufficient number of times, an approximation of the true P_f distribution is obtained. Then, the fatigue life at a desired P_f can be calculated by reading from the cumulative distribution function of P_f. In reality, calculating extremely small values of P_f using a Monte Carlo simulation requires an impracticably large number

of simulations.[5] The authors were only able to achieve $P_f \approx 1 \times 10^{-3}$ using over 1000 repetitions of the simulation. These results were manually extrapolated to $P_f = 1 \times 10^{-6}$ to obtain estimates of the retirement time necessary to achieve six nines reliability.

Thompson and Adams found that the retirement times calculated using the traditional safe life methodology were extremely similar to those calculated using the aforementioned methods. The authors also conducted sensitivity analyses and concluded that variability in material strength is the most important factor in the fatigue life prediction, followed by flight load variability and usage variability.

2.3.5 Discussion

The framework described by Li proposes a novel alternative to the fatigue design processes discussed in Section 2.2.2. Essentially, this framework incorporates fatigue life as a constraint in the preliminary design of the rotor system, using tools and concepts similar to those of Section 2.1.3. This allows rotor performance, weight, stability, and fatigue life to be considered simultaneously, which is not possible in the traditional rotor design process. The segmented nature of the optimization process reduces computational expense by evaluating many of the constraints prior to the full aeroelastic analysis of the rotor system, enabling more rapid design space exploration.

However, the comprehensive analysis of the rotor is limited to a single steady forward flight condition in which the rotor system is modeled in isolation. This neglects a wide variety of extremely damaging flight conditions discussed in Section 1.5, such as transient maneuvers, the GAG cycle, and transitional flight. In effect, the mission spectrum portion of the safe life methodology (see Figure 2.13) is neglected entirely and, as a result the load spectrum is extremely simplified. Additionally, the effects of rotor–fuselage and rotor–rotor interactions are not captured. As a result, the calculated fatigue life of 15,800 FH may be significantly overestimated.

[5]Recall that to achieve six nines reliability, $1 - P_f = 0.999999$. Thus, $P_f = 1 \times 10^{-6}$. This is referred to as a *weak* probability.

Furthermore, the only design variables considered by Li are related to the cross-sectional design of the blade itself. The potential impacts of variations in other design variables, such as the rotor hub, the position of the rotor relative to other components on the vehicle, or the cruise speed, are neglected. Nevertheless, the approach presented by Li is especially promising and has the potential to be expanded into a more complete analysis.

Arruda, Hamel, and Collins improved upon Li's framework in some respects by properly capturing the vehicle's mission spectrum by modeling a number of flight conditions and maneuvers. They also demonstrated the potential of modifying design parameters other than those related to the cross-section of the blade to improve fatigue life. This work establishes that the fatigue life of a rotor system component can be influenced by a number of internal and external design decisions.

However, this analysis also depended on a single rotor modeled independently of the remainder of the vehicle, neglecting rotor–rotor and rotor–fuselage interactions. Additionally, Arruda, Hamel, and Collins did not attempt to rigorously optimize the parameters of the HHC system or investigate any other ARTs, which could potentially improve the fatigue life further. Thus, this study serves as a promising proof-of-concept but does not constitute a complete design environment.

Zion and Schank demonstrated the applicability of surrogate modeling to the rotorcraft fatigue design problem. Both authors demonstrated the potential to replace time-consuming and expensive processes with rapid evaluations of simple functions. However, neither author demonstrated the technique in the context of a fatigue design framework. In this context, the application of a surrogate model requires a cost–benefit trade-off analysis. The benefit of rapid evaluation may be outweighed by the cost of evaluating the original function at each of the training points used in the construction of the surrogate model. Additionally, the error induced by the surrogate model would reduce confidence in the fatigue life prediction; additional measures may be required to maintain a satisfactory level of reliability.

Thompson and Adams demonstrated that the safety factors and reductions key to the

traditional safe life methodology can be eliminated using probabilistic calculations to determine reliability. Unfortunately, the authors were not able to capture the extremely weak probabilities required by the six nines requirement in a reasonable amount of time. The extrapolation method used to predict these values was not especially rigorous and may not be generically applicable to other cases. Additionally, the Monte Carlo simulation used in this study is rather crude and leaves room for improvement.

This discussion leads to Observation 2.8:

Observation 2.8

Several new approaches to the rotorcraft fatigue design have demonstrated the potential applicability of physics-based design tools. In each case, limitations in scope prevent their application in a complete fatigue design environment.

2.4 Gaps in the Literature

The guiding literature questions and derived observations from the literature review in the previous sections are reprinted in Table 1.1 for reference.

Based on the observations in Table 1.1, a number of gaps in the literature can be identified. Observation 2.1 states that the fatigue design in certain rotorcraft components is possible as early as the preliminary design stage. Based on the discussion in Section 2.1.1, it is desirable to conduct such analyses as early as possible, in order to ensure the vehicle will achieve desirable fatigue life characteristics and avoid costly redesigns during detail design or development if requirements are not met. Early knowledge of component service lives is also necessary to accurately predict maintenance and cost requirements for the complete aircraft.

However, preexisting preliminary design tools developed for rotor systems and rotary-wing vehicles are not capable of predicting the fatigue life of rotor components or projecting vehicle RAM-C characteristics using this information, as noted by Observations 2.2 and 2.4. New approaches to the rotorcraft fatigue design problem presented in works by Li, Volovoi,

Table 2.3: Summary of literature questions and observations in Chapter 2.

Literature Questions	
1	How does fatigue design fit into the overall vehicle design process? Can existing rotorcraft design tools be effectively utilized for fatigue design?
2	What are the most important elements of the traditional fatigue design process? What are the associated disadvantages and drawbacks?
3	How have other researchers improved upon the traditional rotorcraft fatigue design process?
Observations	
2.1	The preliminary design stage is the earliest point in the rotorcraft design process at which fatigue design is feasible.
2.2	Most currently available RAM-C–focused design tools suffer from high data input requirements. These inputs can be difficult to populate due to a lack of appropriate resources to predict individual system characteristics in the early design stages.
2.3	The development of rotor design tools is complicated by the computationally expensive simulation packages on which they rely.
2.4	Most currently existing physics-based rotor design tools typically do not make considerations for the as RAM-C characteristics of the rotor system.
2.5	Fatigue life models such as Miner's rule and phenomenological models such as Schaff and Davidson's strength-degradation model are most appropriate for use in the rotorcraft preliminary design stage. Miner's rule is preferable due to its simplicity and ease of implementation.
2.6	Traditional methods used to achieve high fatigue life reliability vary across organizations and are based on organizational experience rather than rigor.
2.7	Historical flight load surveys used in the rotorcraft fatigue design process may produce erroneous predictions when applied to revolutionary rotary-wing configurations.
2.8	Several new approaches to the rotorcraft fatigue design have demonstrated the potential applicability of physics-based design tools. In each case, limitations in scope prevent their application in a complete fatigue design environment.

and Hodges [66], Zion [76], Thompson and Adams [79], Li [87], Arruda, Hamel, and Collins [90], and Schank [93] serve as promising proofs-of-concept but do not constitute complete fatigue design frameworks, as stated by Observation 2.8. This leads to Gap 1:

Gap 1

There is a missing link between rotor design tools and rotary-wing vehicle design tools that prevents physics-based prediction of component service lives and resulting RAM-C characteristics in the preliminary design stage.

Addressing Gap 1 is hindered by inherent difficulties in the rotorcraft fatigue design process. Primarily, establishing the load spectrum, which is critical for determining an accurate representation of in-service fatigue loads, is costly and time-consuming. In the preliminary design phase, load spectra are typically approximated by applying correction factors to historical flight load surveys. As noted by Observation 2.7, this historical data is not applicable to complex vertical lift configurations and its continued use could result in erroneous and potentially non-conservative fatigue life predictions.

Rotor aeroelastic analysis tools, such as the comprehensive analysis programs discussed in Section 2.1.3, offer a potential solution by enabling the analytical prediction of component loads for arbitrary rotary-wing aircraft configurations. However, as noted by Observation 2.3, the computational expense of these programs complicates their use in conceptual and preliminary design frameworks. The vast amount of runtime required to construct a complete load spectrum prohibits the direct application of comprehensive analyses to the fatigue design problem. This leads to Gap 2:

Gap 2

No well-developed methods are available to rapidly derive the load spectrum of revolutionary vertical lift configurations in the preliminary design stage.

Traditional fatigue design methods used in the rotorcraft industry use acceptably accurate fatigue life prediction methods, as noted by Observation 2.5. Sufficient reliability and confidence in fatigue life predictions is achieved by introducing conservative biases wherever uncertainty is present. These biases take the form of reductions and safety factors that vary between aircraft development programs, manufacturers, and qualifying agencies, and are typically based on a given organization's historical experience, as noted by Observation 2.6.

These approaches are difficult to trace and reproduce and may provide overly conservative solutions in some cases, leading to increased weight and expense of the final component.

Attempts to do away with these safety factors using probabilistic methods can only be found in one-off studies which do not appear to have resulted in widespread adoption in the industry. More rigorous, robust, transparent, and flexible methods of achieving sufficient fatigue life reliability are necessary. This leads to Gap 3:

Gap 3

There are no consistent and repeatable methods to ensure high fatigue life reliability in regular use in the rotorcraft industry.

The connections between the observations from Chapter 2 and the three gaps identified previously are illustrated in Figure 2.16.

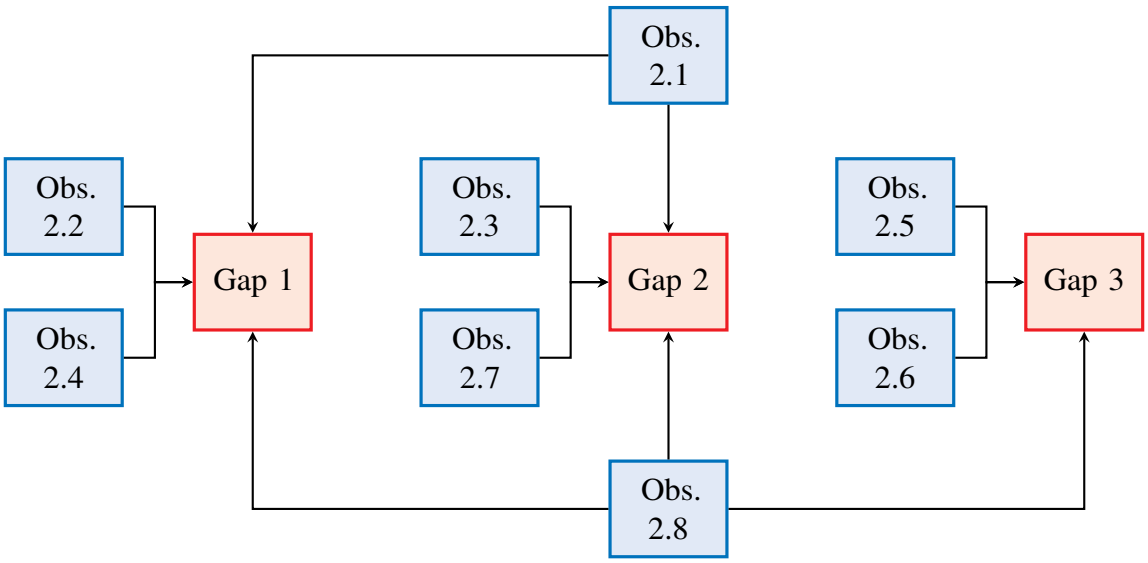


Figure 2.16: Flow chart summarizing the identification of gaps in the literature.

The following chapter will address the formulation of a research plan to close the gaps identified in this section.

CHAPTER 3

RESEARCH FORMULATION

In Chapter 1, Research Question 0 was derived based on a review of rotorcraft life-cycle costs, accident rates, causes of fatigue damage, and future vertical lift aircraft development programs. For convenience, Research Question 0 is reprinted below:

Research Question 0

How can the fatigue life of rotor system components be efficiently evaluated for use as a design driver in a rotorcraft design framework?

The literature review in Chapter 2 covered rotary-wing vehicle design, including the design process, vehicle design tools, and rotor design tools; rotorcraft fatigue design, including fatigue damage theory and traditional fatigue design methods used in the rotorcraft industy; and several new academic approaches to rotorcraft fatigue design. Key observations from each subject area were used to identify three main gaps in the literature, as illustrated in Figure 2.16. These gaps are summarized in Table 3.1.

Table 3.1: Summary of literature gaps identified in Chapter 2.

	Gaps
1	There is a missing link between rotor design tools and rotary-wing vehicle design tools that prevents physics-based prediction of component service lives and resulting RAM-C characteristics in the preliminary design stage.
2	No well-developed methods are available to rapidly derive the load spectrum of revolutionary vertical lift configurations in the preliminary design stage.
3	There are no consistent and repeatable methods to ensure high fatigue life reliability in regular use in the rotorcraft industry.

In this chapter, a conjecture to Research Question 0 is formed. Next, further research questions are developed and an initial outline of a new preliminary fatigue design methodology is presented.

3.1 Conjecture to Research Question 0

Gap 1 motivates the development of a new fatigue design methodology. Such a methodology would need to be consistent with the activities and knowledge level of the preliminary design stage (see Section 2.1.1.3) in order to ensure compatibility with the vehicle and rotor design tools described in Sections 2.1.2 and 2.1.3, respectively. For reference, Figure 2.1, which summarizes the stages of rotorcraft design, is reprinted below.

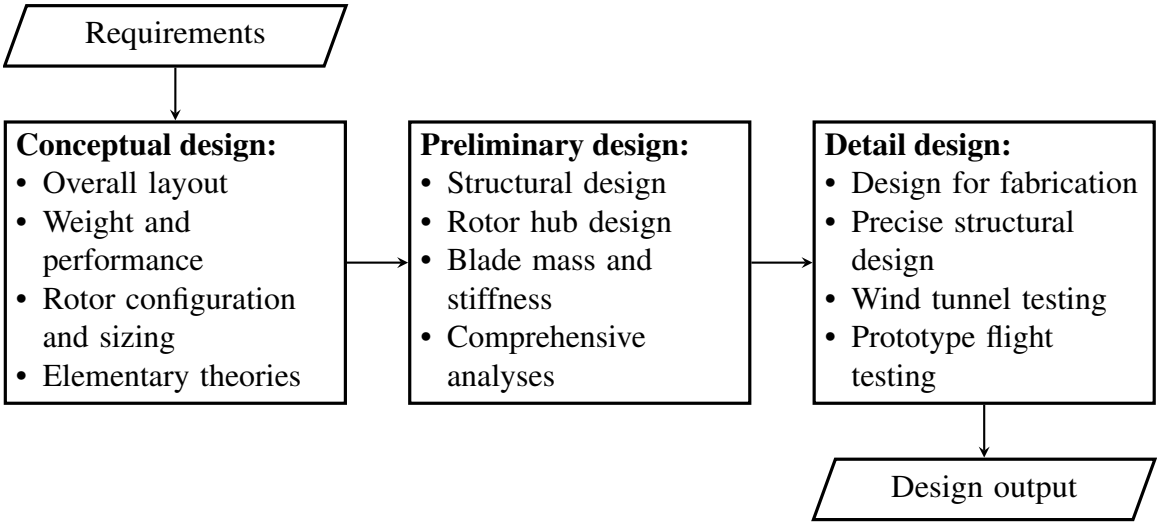


Figure 2.1: Flow chart summarizing the rotorcraft design process (reprinted from Page 33).

A primary requirement for a preliminary design methodology is to produce results quickly. Although the most wide-ranging design space exploration is complete by the end of the conceptual design stage, the preliminary design stage will contain some design excursions and trade studies to seek an optimal design point near the configuration identified previously. If fatigue life predictions take an inordinate amount of time to complete, such design exploration would be impossible or prohibitively time-consuming.

An additional requirement is that the level of knowledge required by the preliminary design methodology must be consistent with the knowledge produced before and during the preliminary design stage. As described previously, the preliminary design stage typically involves wind tunnel testing and CFD simulations to further the aerodynamic design of the

vehicle. Rotor hub design and rotor blade structural design will advance to a point such that comprehensive aeroelastic analyses can be used to predict rotor performance, stability, loads, and handling qualities. However, detailed structural layouts and manufacturing plans will not be created until the detail design stage. This limits the available structural analysis methods. For example, certain fatigue life prediction methods, such as the progressive damage models discussed in Section 2.2.1.3, are likely unsuitable for use in a preliminary design methodology. Of course, deriving the load spectrum from flight load surveys is also impossible because prototype vehicles will not be flown until the development phase. Alternative methods are needed to construct the load spectrum.

The aforementioned requirements necessitate modifications to existing rotorcraft design methodologies which, as described in Section 2.2.2, are fully applicable only in the detail design stage or the development phase. Enhancements to these methodologies are necessary to produce fatigue life predictions of comparable quality and reliability in the preliminary design stage. These modifications are derived from the rotorcraft design and analysis tools discussed in Sections 2.1.2 and 2.1.3 and the new fatigue design approaches discussed in Section 2.3. This can be stated as a conjecture to Research Question 0:

Conjecture 0

A new preliminary fatigue design methodology can be created by enhancing traditional fatigue design methodologies with modern rotorcraft analysis tools and integrating new methods to improve flexibility and runtime. This enables the use of rotor component fatigue life as a design driver for future rotary-wing vehicle development programs.

To support Conjecture 0, a new preliminary fatigue design methodology will be developed and tested, satisfying Research Objective 2. This methodology must generate fatigue life predictions of a fidelity similar to those of the aforementioned traditional fatigue design methodologies. Additionally, runtime must be sufficiently low to allow for design space exploration analogous to that which may take place in the preliminary design stage. The capabilities and drawbacks of this methodology will be assessed by application to hypothetical

rotorcraft design exercises.

3.2 Research Question 1

Constructing a complete load spectrum is critical to the assessment of the fatigue life of a flight-critical component. This is especially important in the context of revolutionary vertical lift vehicles, which may experience complex loading environments due to their novel configurations. Sections 1.5 and 2.2.2 emphasize the importance of capturing all elements of the load spectrum, as seemingly insignificant segments may be responsible for a majority of the fatigue damage a component experiences.

The most promising method of predicting the load spectrum in the preliminary design stage is to leverage comprehensive analysis tools to analytically predict component loads and stresses. This technique is used in the context of fatigue life prediction by Li [87], Arruda, Hamel, and Collins [90], and Schank [93] (see Sections 2.3.1 to 2.3.3) and in the context of rotor design environments by Sinsay and Alonso [52] and Collins [54] (see Section 2.1.3).

However, long runtimes are inherent to comprehensive analysis programs due to the complexity of the non-linear equations that must be solved and the short time steps required for accuracy. Analysis time increases significantly if CFD coupling is introduced, which may be necessary to accurately predict the airloads produced by unconventional rotor configurations. For example, Collins [54] reported a runtime of approximately 9 h for a single analysis point using loose CFD/CSD coupling on 12 processors, which he considered to be "very fast". Consider that constructing a load spectrum for the relatively simple transport helicopter mission spectrum described in Table 2.1 would require, at minimum, 90 analysis points or 810 h of processing time assuming runtimes are similar to those reported by Collins. This process would need to be repeated each time the vehicle or mission design is modified. The process of constructing a load spectrum for use in the preliminary fatigue design methodology quickly becomes intractably difficult and time-consuming. This leads to Research Question 1:

Research Question 1

How can a complete load spectrum be rapidly derived using physics-based comprehensive analysis tools?

3.3 Research Question 2

Gap 3 identifies the lack of available methods to ensure fatigue life prediction reliability in traditional rotorcraft design methods. The reductions and safety factors described in Section 2.2.2.2 do produce acceptable levels of reliability in most cases, as demonstrated by Zion [78] and Thompson and Adams [79]. However, reductions vary across organizations and are typically derived from a given organization's operational experience. Thus, reductions that produce high fatigue life reliability for conventional rotorcraft may not be similarly successful when applied to arbitrary rotary-wing configurations.

Lappos [94] argues that the practice of depending upon safety factors to address modeling uncertainty when designing new rotorcraft leads to overly-conservative designs. Instead, designers should use probabilistic methods that capture and quantify this uncertainty, leading to a more robust design with potential savings in weight and cost.

The proposed preliminary fatigue design methodology should make use of similar principles to ensure high reliability while minimizing the use of traditional reductions and safety factors. This has the potential to reduce the weight and cost of the resultant designs and provide additional flexibility with respect to changing levels of uncertainty and variable reliability requirements. This leads to Research Question 2:

Research Question 2

How can probabilistic methods be applied to efficiently remove the dependence of traditional fatigue design methodologies on reductions and safety factors?

Research Questions 1 and 2 will be used to satisfy Research Objective 3.

3.4 Proposed Methodology

This section outlines the initial formulation of the preliminary fatigue design methodology proposed in Conjecture 0.

3.4.1 Reference Methodology

This methodology will be based upon the safe life fatigue design methodology detailed in Figure 2.13. The safe life methodology was selected over the damage tolerance methodology because safe life remains the dominant methodology of use for single-load-path components in the rotorcraft industry. Additionally, the damage tolerance methodology is less appropriate for application in the preliminary design stage (see Section 3.1). For reference, Figure 2.13, which diagrams the safe life methodology, is reprinted below.

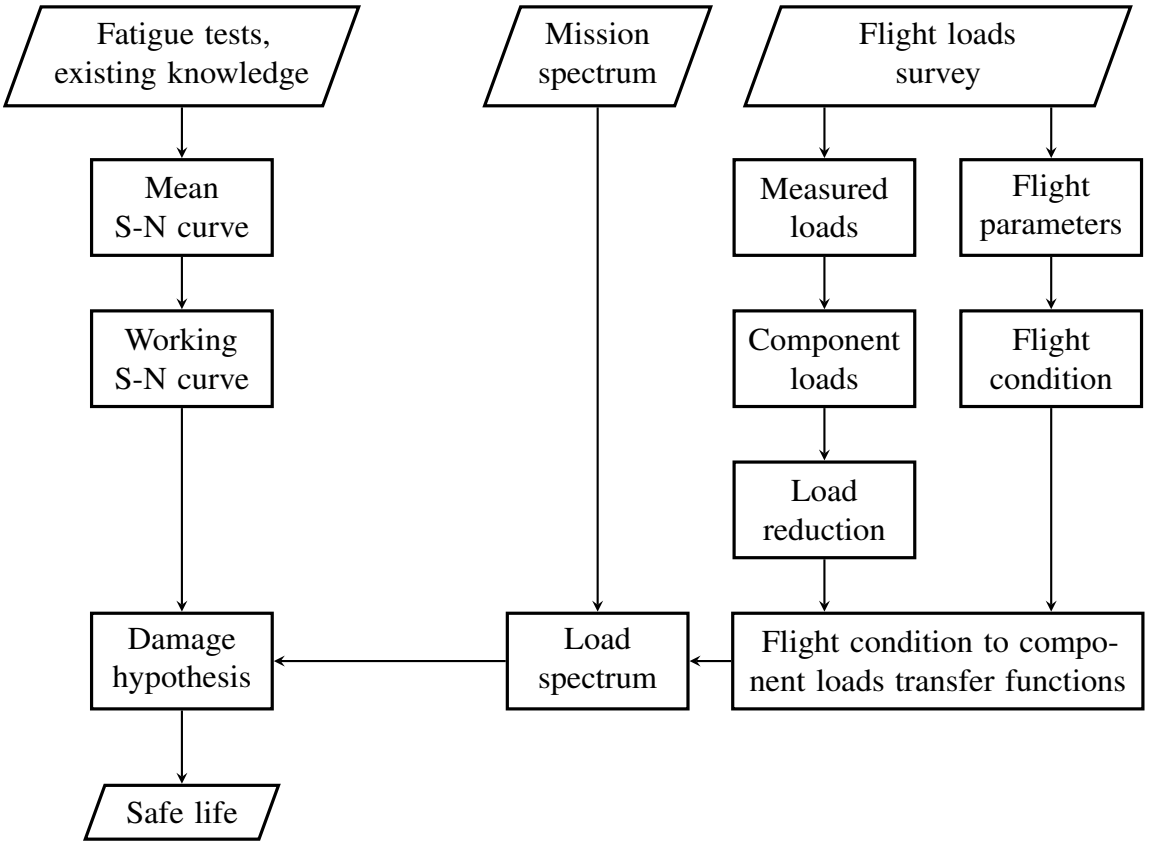


Figure 2.13: Diagram of the safe life methodology (reprinted from Page 67).

3.4.2 Preliminary Fatigue Design Methodology

The proposed preliminary fatigue design methodology replaces elements of the safe life methodology with modern tools, techniques, and models detailed in the previous chapters. A diagram of the proposed methodology is presented in Figure 3.1.

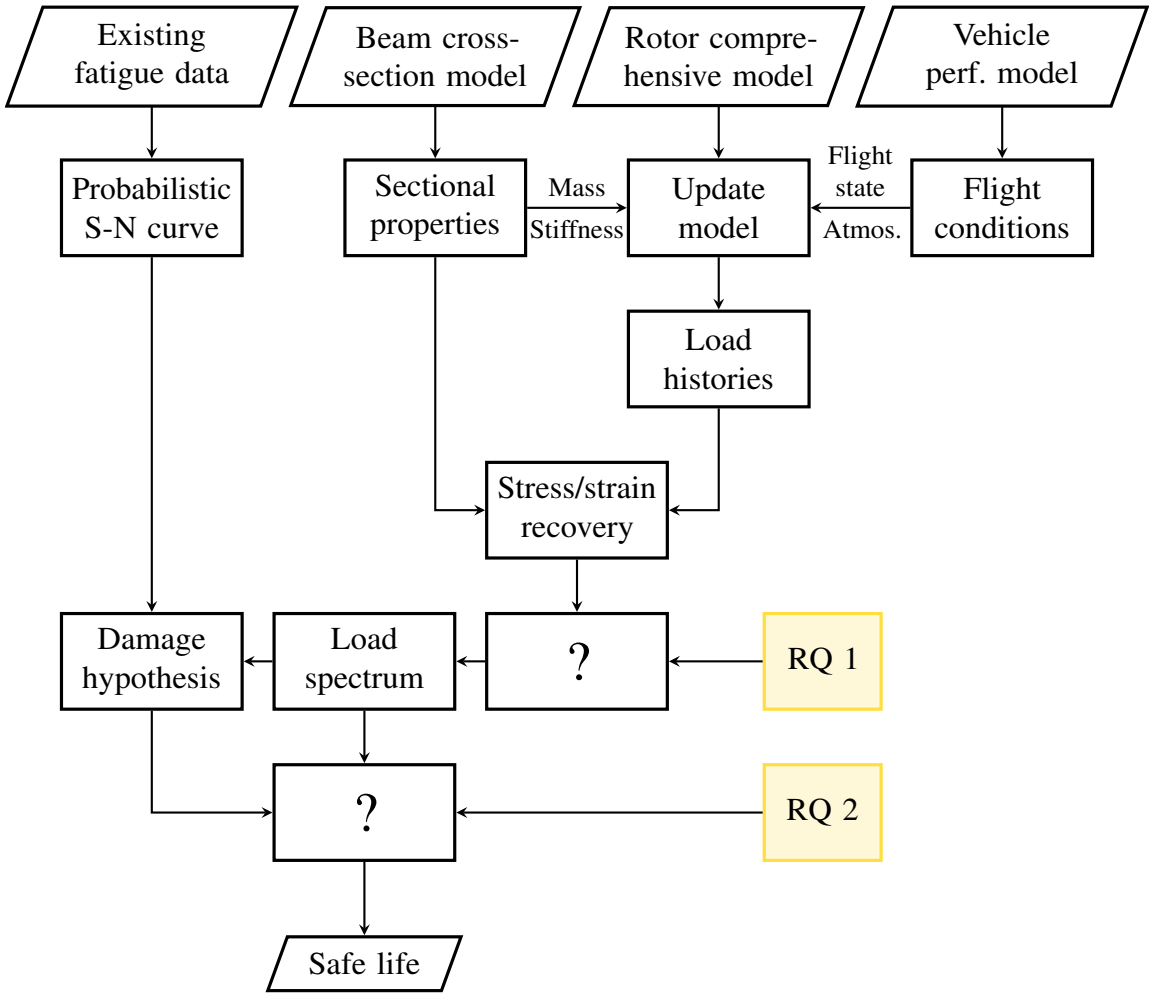


Figure 3.1: Initial diagram of the preliminary fatigue design methodology.

The first difference between the proposed design methodology and the safe life methodology is the construction of the S-N curve. In the proposed methodology, information relating to the variance of the S-N data is retained to create a probabilistic S-N curve, rather than deriving a working curve from the mean curve. This probabilistic data will be captured as uncertainty in the probabilistic analysis.

Rather than rely on present or historical flight load surveys to produce a load spectrum, the proposed methodology makes use of beam cross-section modeling programs such as VABS (see Sections 2.3.1 and 2.3.3), rotor comprehensive analysis programs such as RCAS or Dymore (see Section 2.1.3) and vehicle performance analysis tools such as NDARC (see Section 2.1.2). The beam analysis tool provides sectional mass and stiffness properties for the component design. The vehicle analysis tool provides a description of the vehicle state, including airspeed, atmospheric conditions, gross weight, attitude, and pilot control positions, based on a predefined design mission profile or point performance conditions.

This data is used to update the comprehensive analysis model, which can then be executed repeatedly in different flight conditions. Once load histories for the appropriate flight conditions are obtained from the comprehensive model, the cross-section model will be executed again in stress/strain recovery mode to extract stress and strain fields. A complete load spectrum for the component of interest can then be derived. Research Question 1, explored further in Chapter 4, will address ways to speed up this process at runtime to avoid the computational expense of repeated calls to the comprehensive analysis and cross-section model.

Finally, the load spectrum and damage hypothesis will be used to predict the component's safe life at the desired level of reliability. Research Question 2, which is addressed in Chapter 5, will explore ways to replace traditional reductions and safety factors with modern probabilistic methods. Thus, the methodology described in Figure 3.1 will be refined and updated based on the outcome of original research intended to answer Research Questions 1 and 2.

3.5 Research Question 3

The utility of the aforementioned methodology must be demonstrated through an appropriately-realistic case study. The test case is structured as a hypothetical preliminary design exercise intended to improve the fatigue life of a baseline conceptual design. This will involve

determining a baseline safe life the rotor blades and seeking methods to improve their lifespan. Conceptual and preliminary design variables defining the rotor blade cross-section, vehicle geometry, and design mission profile will be varied to study the resulting impact on the fatigue life prediction. This case study is framed as Research Question 3:

Research Question 3

Does the preliminary fatigue design methodology enable evaluation of the relative impact of common preliminary design variables on the probability of fatigue failure of a flight-critical component in a conceptual helicopter design?

Further details related to the case study will be provided in Chapter 6. Research Question 3 will be used to satisfy Research Objective 4.

3.6 Summary

In this section, the initial design of a preliminary fatigue design methodology to address weaknesses in traditional rotorcraft design processes was proposed. Two research questions which will inform the development of this methodology were identified, and a third is used to frame a proof-of-concept study of the methodology's capabilities. The logical process used to formulate these research questions is depicted in Figure 3.2.

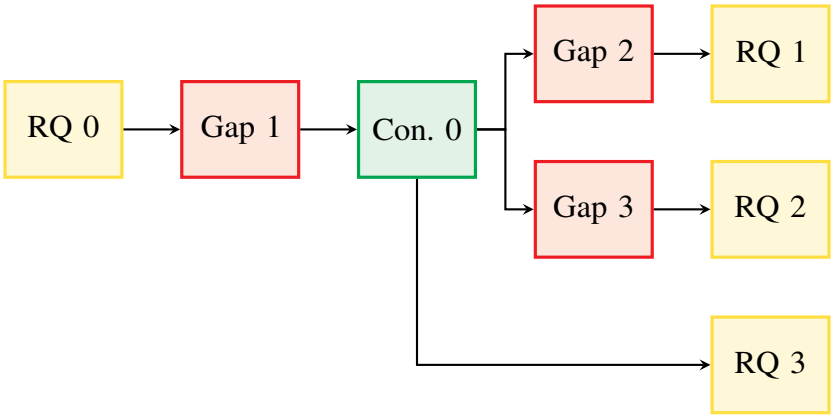


Figure 3.2: Flow chart summarizing the research formulation, not including the observations described in Table 2.3.

The next chapters will involve additional literature review necessary to formulate hypotheses to the aforementioned research questions. Virtual experiments will be conducted to address each hypothesis in turn, and the proposed methodology will be refined as necessary. The capabilities of the methodology will be tested and compared to preexisting traditional methods.

CHAPTER 4

PREDICTION OF FATIGUE LOADS USING SURROGATE MODELING

Research Question 1 (see Section 3.2) asks if the load spectrum derivation task can be sped up despite the long runtime of comprehensive analysis programs. For reference, Research Question 1 is reprinted below.

Research Question 1

How can a complete load spectrum be rapidly derived using physics-based comprehensive analysis tools?

The literature review in Chapter 2 suggests that surrogate modeling methods may be able to rectify this problem. Zion [76] demonstrated that response surface methods can be used to predict oscillatory and mean component loads based on inputs such as gross weight, density altitude, airspeed, pilot control positions, and vehicle attitude. Schank [93] successfully used artificial neural networks to predict maximum strain in a rotor blade cross-section based on the applied forces and moments. Either or both of these methods could be used to reduced the total computational expense of building the load spectrum, but further research is required to determine the most appropriate surrogate modeling method, which dependent and independent variables should be considered, and if a significant time savings can be realized.

This section begins with a brief review of surrogate modeling and a comparison of several methods of interest. Next, a hypothesis to Research Question 1 is formed. Hypothesis 1 will be tested with Experiment 1, and the results will be used to refine the preliminary fatigue design methodology.

4.1 Review of Surrogate Modeling Techniques

Surrogate modeling, which was discussed briefly in Section 2.3.3, is the process of creating an approximate model, $\widehat{f}(\boldsymbol{x})$, of an expensive function, $f(\boldsymbol{x})$, such as a CFD analysis, finite element analysis, or rotor comprehensive analysis. The approximate model can then be used in place of the expensive function to reduce calculation time when a large number of executions are required. Surrogate modeling is also known as *regression analysis*; in this text, the term *regression analysis* is avoided in order to prevent confusion with the process of statistical hypothesis testing.

The general process of developing a surrogate model is as follows [95]:

1. *Identify the input variables $\boldsymbol{x}$, and their ranges, $\boldsymbol{x}_u - \boldsymbol{x}_l$*: The set of input variables should be chosen appropriately to balance flexibility and complexity. If the set of input variables are too limited, the model will be inflexible and possibly not well-suited to its task. If a large number of input variables are required, then the size of the input space must be increased and the surrogate model itself will become more complex. Similarly, the length of each input range must be chosen with care. Because extrapolation outside of the design space may produce wildly inaccurate predictions, the input ranges must be sufficiently large to capture all possible use cases. However, a large input range also increases sampling time and makes fitting the model more difficult.
2. *Sample the design space*: The sampling process involves defining n sample or training points, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, and evaluating $f(\boldsymbol{x})$ n times to find the corresponding values of the response variable, $y_1, y_2, \ldots, y_n$. The process of selecting the sample points is known as design of experiments (DOE). A large number of DOEs have been developed and may be selected based on the nature of the experiment, the response, and the surrogate modeling method.
3. *Fit the surrogate model to the responses*: The process of fitting the surrogate model

involves "training" the model by varying its parameters (or hyperparameters) until the responses, y_i, are well predicted. The level of prediction accuracy (goodness-of-fit) achieved is usually measured by assessing the residual between the predicted responses and the measured responses, $e_i = y_i - \widehat{y}_i$. The training process and the goodness-of-fit metrics used vary widely between different surrogate modeling methods. This step will be the primary focus of this section.

4. *Check the surrogate model for goodness-of-fit*: After the surrogate model is trained, it is best practice to validate the model by checking its accuracy. A number of metrics and procedures can be used to check and improve the fit. *Model fit error* describes the difference between the predicted and actual responses in the training set. *Model representation error* describes the difference between the predicted and actual responses in a new set of data which is generated for validation purposes and not used in the training process. Each of these errors must be acceptably low for the predictions generated by the surrogate model to be considered accurate [96].

The field of surrogate modeling is wide-ranging and has connections with the fields of statistical analysis and machine learning. In the remainder of this section, several surrogate modeling methods of interest, including response surface methods (RSM), artificial neural networks (ANN), and Gaussian process modeling (GPM), are discussed and compared. This is by no means intended to be an exhaustive review of the field of surrogate modeling, which is beyond the scope of this thesis.

4.1.1 Response Surface Methods

Response surface methods involve fitting a polynomial function to the sample points using a process called linear regression. The complexity of the model depends on the order of the polynomial. For example, Equation (2.16), reprinted below for reference, defines a

first-order response surface equation (RSE).

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + \varepsilon_i$$

Second-order interactions between the input variables can be captured by including multiplication terms of the form $\beta_{12} x_{i1} x_{i2}$ or self-interaction terms of the form $\beta_{11} x_{i1}^2$. It is also possible to capture third- and higher-order interactions using a similar pattern, but second-order models are sufficient for many cases [97]. The set of x-terms used in the RSE are known as the *basis functions*. It is more convenient to write Equation (2.16) using matrix–vector notation as

$$\boldsymbol{y} = \Phi \boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{4.1}$$

where

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \Phi = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix}, \quad \text{and } \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Then, least-squares estimates of the coefficients can be found by minimizing the SSE with respect to $\boldsymbol{\beta}$

$$\min_{\boldsymbol{\beta}} \text{SSE} = \min_{\boldsymbol{\beta}} \boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon} \tag{4.2}$$

This can be solved analytically using

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\beta}} \boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon} \Big|_{\boldsymbol{b}} = 0 \tag{4.3}$$

which leads to

$$\boldsymbol{b} = \left(\Phi^\top \Phi \right)^{-1} \Phi^\top \boldsymbol{y} \tag{4.4}$$

where $\boldsymbol{b}$ are the least squares estimators of $\boldsymbol{\beta}$. Finally, predictions can be made using

$$\widehat{\boldsymbol{y}} = \Phi \boldsymbol{b} \tag{4.5}$$

An identical process would be used for higher-order models, with appropriate modifications to the Φ matrix and the $\boldsymbol{\beta}$ vector.

4.1.1.1 Demonstration

A demonstration of RSM fitting is presented in Figure 4.1. Sample points were generated by sampling a normal distribution with a standard deviation of $\sigma = 0.05$ around the function $y = \sin(x)$ for the range $[0, 2.5]$. Four separate models were created, with bases ranging from zeroth- to third-order. The zeroth-order basis is $\{x^0\}$, the first-order basis is $\{x^0, x^1\}$, and so on. Equation (4.4) was used to estimate the regression coefficients and Equation (4.5) was used to generate predictions.

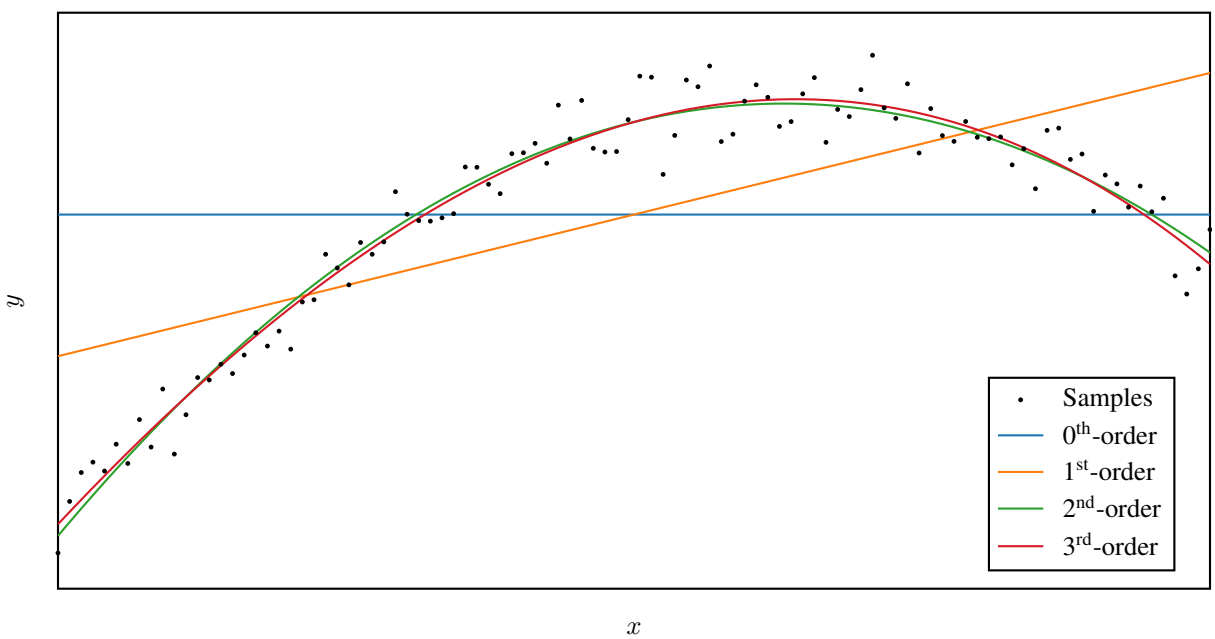


Figure 4.1: Demonstration of response surface model fitting.

Note that the model generated using the zeroth-order basis is simply the mean of the sample points. The first-order basis provides a linear fit of the sample points and captures

the overall trend. Satisfactory fit is achieved with the second-order basis, and the third-order basis offers little additional benefit.

4.1.1.2 Goodness-of-Fit

A number of goodness-of-fit metrics can be used to assess the uncertainty or error in the predictions produced by the model [96]. The coefficient of determination, R^2, is calculated using

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} \tag{4.6}$$

where SSE is the residual sum of squares and SST is the total sum of squares, defined as

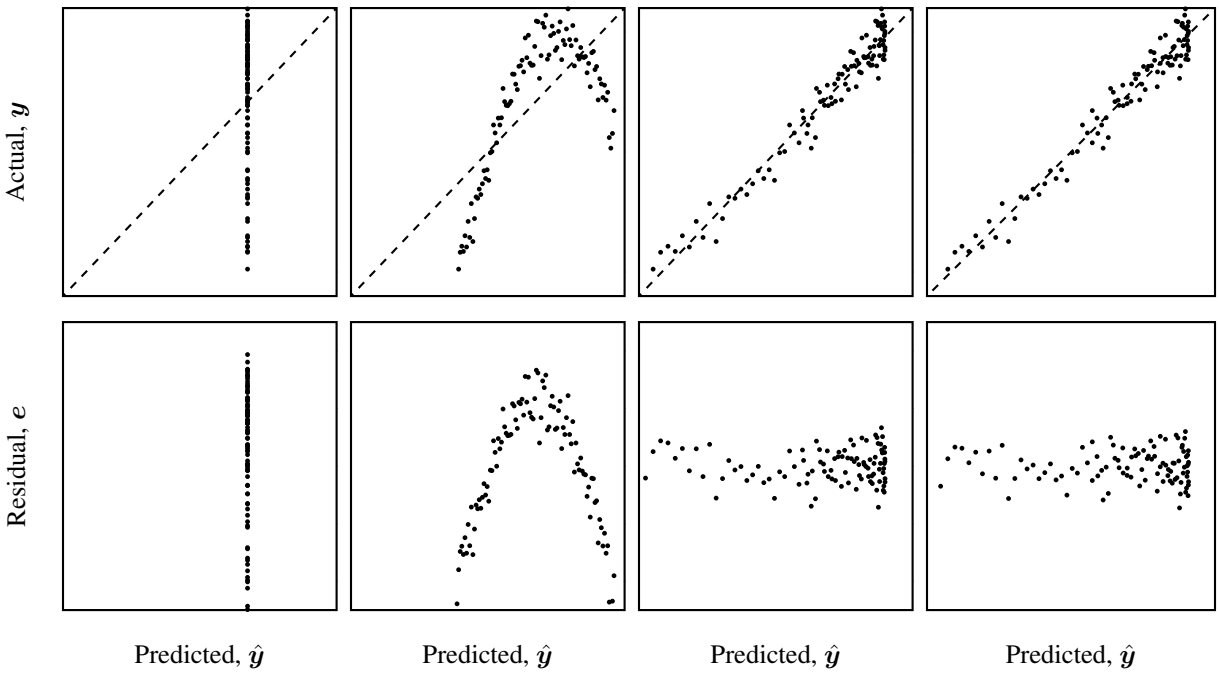
$$\text{SSE} = \boldsymbol{e}^\top \boldsymbol{e} \tag{4.7}$$

$$\text{SST} = (\boldsymbol{y} - \overline{\boldsymbol{y}})^\top (\boldsymbol{y} - \overline{\boldsymbol{y}}) \tag{4.8}$$

where $\boldsymbol{e} = \boldsymbol{y} - \widehat{\boldsymbol{y}}$ and $\overline{\boldsymbol{y}}$ is the mean of $\boldsymbol{y}$. R^2 takes a value between negative infinity and one. Generally, a low value of R^2, such as $R^2 < 0.8$, indicates a poor fit, while a high value of R^2 may suggest but does not guarantee a good fit. For example, the values of R^2 for the four models in Figure 4.1 are 0, 0.4787, 0.9595, and 0.9610, from lowest- to highest-order. Note that $R^2 = 0$ for the zeroth-order model because $\widehat{\boldsymbol{y}} = \overline{\boldsymbol{y}}$ for all sample points.

Another common goodness-of-fit check is to plot $\boldsymbol{y}$ versus $\widehat{\boldsymbol{y}}$ and $\boldsymbol{e}$ versus $\widehat{\boldsymbol{y}}$. The former plot, known as the *actual by predicted* plot, should show that most points are clustered around the $\boldsymbol{y} = \widehat{\boldsymbol{y}}$ line with even distribution and no discernible pattern. This indicates that the predicted responses are closely aligned with the actual responses. The latter plot, known as the *residual by predicted* plot, should show a random distribution of residuals with small overall variation. For example, the aforementioned goodness-of-fit checks for the models in Figure 4.1 are presented in Figure 4.2.

Figure 4.2 indicates that both goodness-of-fit metrics improve dramatically as the model order increases, but the difference between the second- and third-order models is marginal.



(a) Zeroth-order (b) First-order (c) Second-order (d) Third-order

Figure 4.2: Demonstration of RSM goodness-of-fit checks. Note that the axes of the actual by predicted and residual by predicted plots are to different scales.

Additionally, the clumping towards the right of the second- and third-order models indicates that these models do not accurately predict the highest observations in the sample set.

Additional goodness-of-fit checks include analyzing the distribution of the model fit error, which is the difference between actual and predicted values of the sample set, and model representation error, which is the difference between the actual and predicted values of a separate testing or validation set, which is not used during the fitting process. Each of these distributions should have a mean near zero and an acceptably small spread. The model representation error is especially important as it quantifies the power of the model to predict observations outside of the sample set. It can also be used to estimate the overall uncertainty in predictions produced by the model.

4.1.1.3 Improving Goodness-of-Fit

If the model fit is unacceptable, the model can be improved through the use of transformations or higher-order terms [92, 96]. Transformation fits are applied by applying a functional transformation, such as a logarithm, power, or trigonometric function, to either the dependent or the independent variables. For example, a logarithmic dependent-variable transformation of Equation (2.16) would take the form

$$\log y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + \varepsilon_i \tag{4.9}$$

The remainder of the fitting process is identical, but results must be "un-transformed" prior to use of the model. Transformations allow the model to adapt to underlying patterns in the true function that may be difficult to capture with polynomial functions alone. For example, a sinusoidal transformation may improve fit in the demonstration in Figures 4.1 and 4.2.

Higher-order terms can be included by adding higher-order basis functions to the model. This is useful if complex interactions exist between input variables exist in the true function. However, this may lead to *overfitting*, a phenomenon whereby the model predicts the true function well only at the training points and does not perform well in general. This behavior can be identified by analyzing the model representation error. In some cases, higher-order terms do not significantly improve the fit, as in Figure 4.2. Thus, the marginal improvement gained may not be worth the extra computational time required to fit a more complex model.

4.1.2 Artificial Neural Networks

Artificial neural networks, sometimes referred to simply as neural networks, simulate the brain structure and learning methods of biological organisms [98]. The network is composed of simple building blocks known as *perceptrons*, analogous to neurons, which can be arranged and connected in different ways to create complex topologies. The network is trained in a manner similar to RSM, except a nonlinear regression technique known as

backpropagation is used to find parameter values that minimize prediction error. ANNs are highly adaptable and are well-suited to uses such as classification and surrogate modeling.

The perceptron is essentially a mathematical function that takes in multiple inputs and gives a single output. This function is typically idealized as a step function that is "off" until a certain threshold value is reached. In a sense, a single perceptron is a surrogate model that produces binary predictions based on its inputs. It can be represented mathematically as Equation (4.10):

$$\widehat{y} = \phi\left(b + \sum_{i=1}^{k} w_i x_i\right) \tag{4.10}$$

where ϕ is the activation function, b is the bias, and w_i are the weights of each input. The form of the activation function is selected based on the purpose of the network and are generally similar to the step function. Common activation functions include the identity function, the sign function, the sigmoid function, hyperbolic tangent function, and the rectified linear unit function [98]. The bias and input weights are determined automatically during the training process. Note that if the activation function in Equation (4.10) is the identity function, the equation is identical to that of a first-order RSE.

4.1.2.1 Layering

The complexity of the network is increased by *layering* the neurons. For example, the outputs of multiple perceptrons in the first layer can be summed and fed into another perceptron in the second layer by expanding Equation (4.10) into

$$\widehat{y} = \phi\left[b + \sum_{j=1}^{m} w_j \phi_j \left(b_j + \sum_{i=1}^{k} w_{ij} x_i\right)\right] \tag{4.11}$$

To avoid complex mathematical notation, ANNs can be represented graphically using a mathematical network in which each node is a perceptron, as in Figure 4.3.

In Figure 4.3, the first layer is referred to as the *input layer* and the second layer is

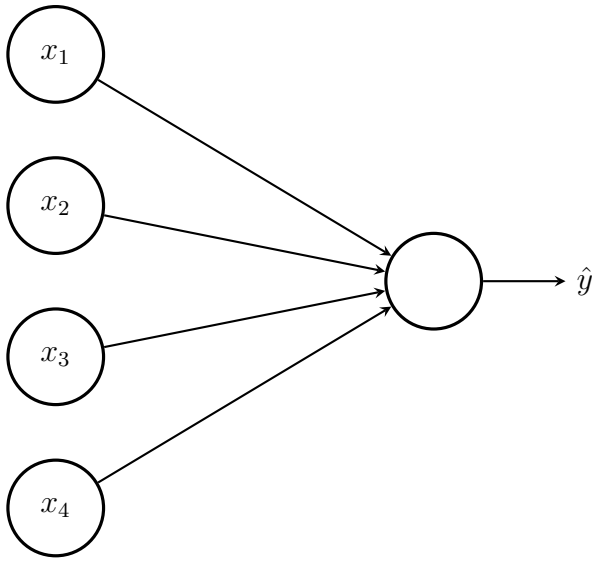


Figure 4.3: A simple artificial neural network.

referred to as the *output layer*. Additional layers can be added between the input and output layers; these are referred to as *hidden layers*. An example of a more complex network with one hidden layer is given in Figure 4.4.

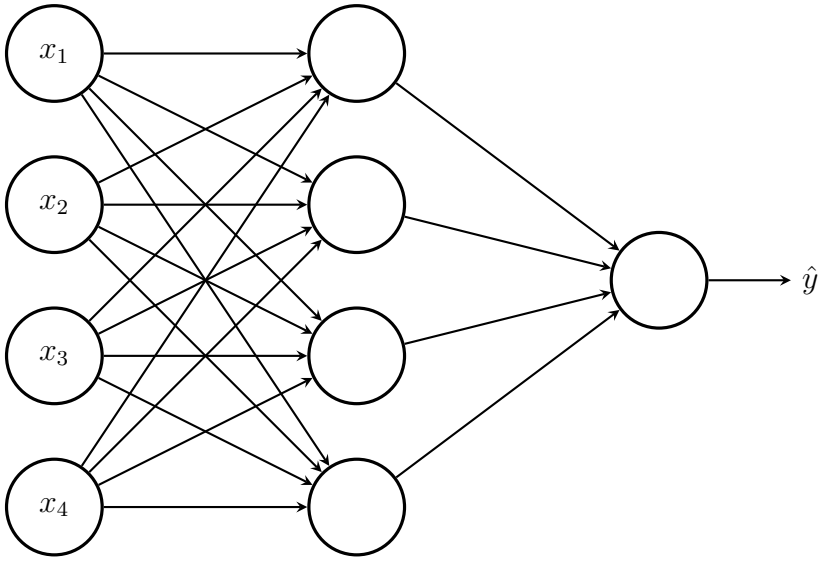


Figure 4.4: A more complex artificial neural network.

4.1.2.2 Topology

The topology of the ANN can be modified in a number of ways. The number of hidden layers and the nodes on each layer can be varied. A network with one hidden layer and many nodes in each layer is called a *shallow* network; a network with multiple layers and only a few nodes in each is called a *deep* network. Although a large shallow network is theoretically capable of predicting the output of any function, the high number of node interconnections, each of which has some associated weight parameter, will make training the network more difficult. Both the training time and the required size of the sample set increases as the number of nodes increases. Deep networks can model similarly complex responses using fewer nodes; complexity is obtained through the convolution of activation functions with the activation functions on subsequent layers. However, deep networks have inherent issues which make training more difficult [98].

The neuron interconnections can also be modified. *Fully connected* networks have connections between every neuron in each layer, as in Figure 4.4. *Partially connected* networks have connections between only some of the neurons in subsequent layers to reduce the number of weights that must be determined. *Feed-forward* networks only connect neurons to neurons in subsequent layers. *Recurrent* networks connect neurons to neurons in the same layer or in preceding layers, creating complex recursive models [99].

Additionally, the number of nodes in the output layer may be modified to produce multiple outputs from the same model; this is especially useful in surrogate models that aim to predict multiple responses. For surrogate modeling applications, output nodes typically use the linear activation function, $\phi(v) = v$, to produce real-valued outputs. Other nodes in the network may use other activation functions; Aggarwal [98] states that the rectified linear unit function is a common choice for modern neural networks because it eases the process of training deep neural networks.

4.1.2.3 Training

Training the ANN involves determining the weight and bias parameter values that minimize the prediction error using a process called backpropagation. Initially, values for all parameters are generated randomly. The backpropagation process has two phases: forward and backward. In the forward phase, the response for a single sample in the training set is computed by executing the entire model. The prediction error and the gradient of the prediction error with respect to the output is computed. In the backward phase, the gradient of the prediction error with respect to the weight and bias parameters is determined for each node, starting from the output node. These gradients are used to update the parameters and the process is repeated with the next sample point [98, 99]. The rate at which weight parameters are varied is controlled using the *learning rate* parameter.

The backpropagation process concludes at an arbitrary stopping point, which may be determined through a variety of different convergence criteria. Because ANNs have a tendency to overfit the data, especially for networks with large numbers of nodes, goodness-of-fit checks similar to those used for RSM should be applied [98]. Additionally, a number of automated procedures are available to reduce overfitting, including early stopping, ensembling, dropout, and regularization [99]. For a more detailed discussion of these techniques, the reader is encouraged to review the text by Aggarwal [100].

4.1.3 Gaussian Process Models

Gaussian process modeling is a non-parametric, stochastic machine learning technique with applications to classification and surrogate modeling [101]. Regression begins by specifying a *prior distribution* of functions. The prior distribution is an infinitely-large set of basis functions with common characteristics; these characteristics are dictated by the *hyperparameters* of a covariance function which defines the process. The *posterior* distribution is formed by applying the sample set to the prior distribution and "filtering out" those basis functions that do not pass through the sample points. The user is left with a

distribution which defines the expected value and the variance of the prediction at each point. Thus, GPM provides both a predicted response, $\widehat{y}$, and its uncertainty. The learning task involves determining the hyperparameters which cause the posterior distribution to best fit the training data [102].

Covariance Functions The prior distribution is determined by a mean function and a covariance function. Typically, the mean function is assumed to be zero, some non-zero constant, or a polynomial function. The covariance function describes the relationship between the function values of two points based on the distance between the coordinates of each point [101]. For example, the squared exponential covariance function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{x}'|^2}{2\ell^2}\right) \tag{4.12}$$

has a higher value when the point $\boldsymbol{x}$ is near $\boldsymbol{x}'$ than when they are far apart. Thus, similar points are expected to have similar function values. In Equation (4.12), σ_f and ℓ are the hyperparameters of the covariance function. The characteristic length, ℓ, defines how rapidly the basis functions may change value, and the signal standard deviation, σ_f, controls the magnitude of these changes. The covariance of sets of points is described by a covariance matrix, $K(X, X')$.

4.1.3.1 Predictions

Predictions require a training set, X_{tr}, and associated responses, $\boldsymbol{y}_{\text{tr}}$. Assuming the mean of the prior distribution is zero, the joint distribution of the predictions and the training responses is

$$\begin{bmatrix} \widehat{\boldsymbol{y}} \\ \boldsymbol{y}_{\text{tr}} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) & K(X, X_{\text{tr}}) \\ K(X_{\text{tr}}, X) & K(X_{\text{tr}}, X_{\text{tr}}) \end{bmatrix}\right) \tag{4.13}$$

where $\mathcal{N}$ is the normal distribution. Using Bayes' theorem, the conditional distribution of the predicted responses is found to be

$$\widehat{\boldsymbol{y}} \sim \mathcal{N}\left(K(X_{\text{tr}}, X_{\text{tr}})^{-1} K(X_{\text{tr}}, X)\boldsymbol{y}_{\text{tr}}, K(X, X) - K(X, X_{\text{tr}})K(X_{\text{tr}}, X_{\text{tr}})^{-1}K(X_{\text{tr}}, X)\right) \tag{4.14}$$

The mean and variance of the prediction for a single test point, $\boldsymbol{x}$, can then be found using

$$\mathrm{E}[\widehat{y}] = k(\boldsymbol{x}_{\text{tr}}, \boldsymbol{x})^{\top} K(X_{\text{tr}}, X_{\text{tr}})^{-1} \boldsymbol{y}_{\text{tr}} \tag{4.15}$$

$$\mathrm{Var}(\widehat{y}) = k(\boldsymbol{x}, \boldsymbol{x}) - k(\boldsymbol{x}_{\text{tr}}, \boldsymbol{x})^{\top} K(X_{\text{tr}}, X_{\text{tr}})^{-1} k(\boldsymbol{x}_{\text{tr}}, \boldsymbol{x}) \tag{4.16}$$

4.1.3.2 Training

In GPM, learning is accomplished by tuning the hyperparameters, $\boldsymbol{\theta}$, to maximize the *marginal likelihood*. The marginal likelihood is the probability of obtaining the training point responses conditioned on the training point coordinates and the hyperparameters, $p(\boldsymbol{y}_{\text{tr}}|X_{\text{tr}}, \boldsymbol{\theta})$. This is traditionally accomplished by minimizing the negative logarithmic transformation:

$$-\log p(\boldsymbol{y}_{\text{tr}}|X_{\text{tr}}, \boldsymbol{\theta}) = \frac{1}{2}\log|K(X_{\text{tr}}, X_{\text{tr}})| - \frac{1}{2}\boldsymbol{y}_{\text{tr}}^{\top} K(X_{\text{tr}}, X_{\text{tr}})^{-1}\boldsymbol{y}_{\text{tr}} - \frac{n}{2}\log 2\pi \tag{4.17}$$

In Equation (4.17), the first term acts as a complexity penalty, the second term captures the data fit, and the third term is a normalization constant. Thus, when the equation is minimized, the GPM naturally assumes a balance between complexity and model fit, which may be difficult to achieve with other surrogate modeling methods. Because the covariance function is infinitely differentiable, the optimization task can be achieved using a gradient-based optimizer [103, 104].

4.1.3.3 Demonstration

A demonstration of the GPM surrogate modeling method is presented in Figure 4.5. The function $y = x \sin x$ was sampled at six points distributed evenly on the range $[0, 2\pi]$. A GPM surrogate model with a squared exponential covariance function was trained with these sample points using the `fitrgp` function in the MATLAB Statistics and Machine Learning Toolbox™. The hyperparameters were found using a quasi-Newton optimizer: the optimized values are $\ell = 1.0220$ and $\sigma_f = 2.3104$. The variability in the prediction is captured by plotting a 95% confidence interval ($\mu \pm 1.96\sigma$) around the prediction.

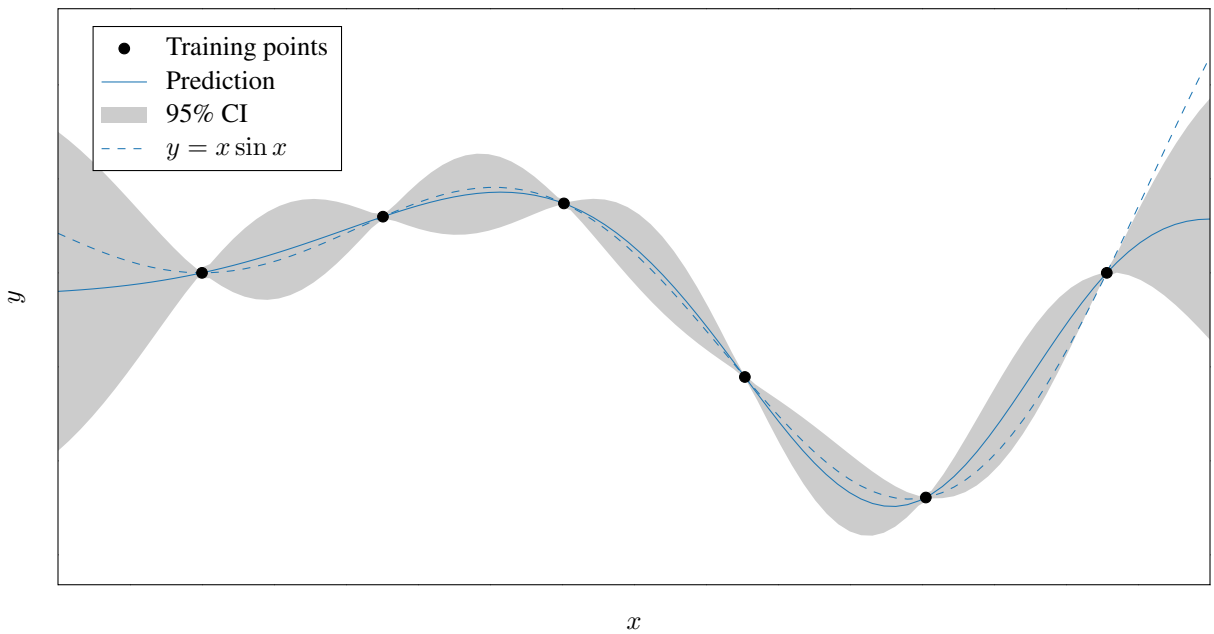


Figure 4.5: Demonstration of a GPM surrogate model.

Note that the fit of this model is relatively poor; a low number of sample points was chosen to create large confidence intervals for visualization purposes. In fact, the true function is not captured within the 95% confidence interval at all points due to the poor fit. However, the uncertainty reduces to zero at each training point because all prior functions that do not pass through these points have been discarded. Also note that the prediction uncertainty increases dramatically beyond the limits of original sample space. This is also the case with other surrogate modeling techniques.

A complete discussion of GPM and its applications is beyond the scope of this thesis. For a more detailed treatment, the reader is encouraged to review the text by Rasmussen and Williams [105].

4.1.4 Comparison

The surrogate modeling methods discussed previously each have unique advantages and disadvantages. Although all are technically capable of creating a model given the same set of sample points, the quality of the model produced by each method will not be identical. The correct choice of modeling method depends on the characteristics of the true function and on the desired properties of the resultant surrogate model.

4.1.4.1 Response Surface Methodology

RSM is perhaps the simplest modeling method surveyed. Response surface equations have a small number of parameters, which means that fewer sample points are required to train the model. This is an advantage when the true function is extremely costly to evaluate. The form of the trained model is also simple: a complete RSE can typically be written in one or several lines of text depending on the number of basis functions used. The polynomial equation is also relatively easy to understand. This has advantages when transferring, deploying, or publishing the model.

However, RSM is not well-suited to modeling highly non-linear responses. Traditional first- and second-order equations are not well suited to capturing the response of functions that are, for example, exponential in nature. Although dependent-variable transformations are able to improve the fit in some cases, the process is not guaranteed to work in all cases and increases the amount of manual input required during the training task. Higher-order terms can be added to the basis function set, but this reduces the benefits of RSM's inherent simplicity. Determining which higher-order terms to include is also a manual diagnostic procedure.

Response surface equations are also limited in that they are single-output, deterministic models. If multiple outputs are required, as is the case in a rotor blade load or stress prediction model, an additional model must be trained for each output. This increases the number of goodness-of-fit checks required and the total modeling time. The deterministic nature of RSM is also a disadvantage when it is desired to quantify the uncertainty in the model. A measure of uncertainty can be derived from the model representation error, but this is a global quantity that applies to every point in the sample space rather than a local quantity that varies between points. This may lead to overestimation of the model uncertainty at some points.

4.1.4.2 Artificial Neural Networks

ANNs avoid the primary drawbacks of RSM by enabling the creation of hugely complex models from simple building blocks. Complexity can be achieved by increasing the number of hidden layer nodes in a shallow neural network or increasing the number of layers in a deep neural network. This flexibility means that ANNs can successfully approximate highly-nonlinear functions that would be difficult to capture with RSM.

Additionally, it is simple to produce multiple outputs with one network by increasing the number of nodes in the output layer. Although the goodness-of-fit should still be checked for each output individually, training a single model to produce multiple outputs is advantageous because potential correlations between the outputs provide additional information that, if captured, can increase the accuracy of the model. For example, in a rotor blade load or stress modeling application, each response is likely positively correlated. Such a correlation could be exploited in a multiple-output ANN but would be lost with RSM.

However, the flexibility of ANN is also its primary weakness. The amount of parameters that must be trained in a networks scales rapidly as the complexity of the network increases. For example, the model depicted in Figure 4.3 has $k + 1$ parameters: one weight for each of the k inputs plus a bias parameter. The model depicted in Figure 4.4 has $k + 1$ input

parameters for each of the four nodes in the hidden layer, plus five additional parameters for the output layer. Aggarwal [98] recommends that the total number of training points be at least two to three times the number of parameters in the model. Thus, ANN is at a severe disadvantage if the true function is expensive to evaluate. The large number of parameters also leads to an inherent tendency towards overfitting and poor generalization; a number of methods are available to counteract this tendency, as described previously, but no method is guaranteed to work in all cases.

The designer of a neural network must make decisions regarding the network architecture, the types of node interconnections, the activation functions on each layer, and the training methods to be used. These choices may not be obvious from the outset and multiple networks may need to be trained and evaluated before an appropriate choice for the problem in question is found. The complexity of the definition of the trained model makes interpretation, transfer, and deployment more difficult than RSM. ANN is also a deterministic modeling technique in the same sense as RSM, leading to similar issues when quantifying model uncertainty.

4.1.4.3 Gaussian Process Models

Perhaps the single most significant advantage of GPM is the probabilistic nature of the model. Because the expected value and the variance of the prediction is produced for each training point, GPM is inherently well-suited to applications which require quantification of modeling uncertainty, such as the structural reliability methods to be discussed in Chapter 5. These models are also capable of capturing the effects of noisy, rather than exact, inputs although this capability is not generally needed when the true function is a deterministic computer program [103].

GPM also provides a middle ground between the simplicity of RSM and the complexity of ANN. Because the model is non-parametric, fitting the model does not reduce its flexibility. GPM draws from an infinite set of basis functions, so the choice of which basis functions to include is not required. The hyperparameter optimization process balances model fit and

model generalization, reducing the overfitting issues inherent to ANN while still maintaining the ability to model highly non-linear functions. The design of a GPM also involves fewer choices than the design of a neural network; essentially, the only major choices to be made are the formulation of the covariance function and the number of sample points. The formulation of the covariance function depends on the nature of the training points. For example, if a periodic pattern is observed, a periodic covariance function is required to capture this effect properly.

However, GPM is generally limited to single-output models, requiring multiple models when multiple responses are desired. Rasmussen and Williams describes some methods by which correlation between multiple outputs can be captured in a GPM framework, but this is an active area of research and development that may not be sufficiently mature for engineering applications. Most popular surrogate modeling toolkits, such as MATLAB and scikit-learn, do not include this capability at the time of this writing.

Additionally, the training process for GPM is not trivial. Equation (4.17) can have multiple local minima, each corresponding to a different interpretation of the input data. Human intervention may be required to ensure that appropriate minimum is reached by the optimizer. For example, the optimization process may need to be repeated with multiple starting points. Poor hyperparameter choices may lead to a model that technically fits the sample points but otherwise has so much uncertainty that it is not useful for prediction.

4.2 Hypothesis to Research Question 1

It is not obvious which of the three previously discussed techniques is the best choice for application in the rotor fatigue stress prediction problem. RSM likely will require the lowest number of training points, but may suffer in terms of accuracy since, as discussed in Section 1.5, rotor loads are highly non-linear in both mean and amplitude with respect to the current flight condition. The primary advantage of ANN in this application is the capability to easily produce multiple outputs from a single model. The stochastic nature of GPM lends

itself better to probabilistic applications than either RSM or ANN.

In the fatigue stress prediction application, the number of outputs required depends on the fidelity of the model. In the simplest case, the first harmonic of a single stress invariant on a single point on a single blade on a single rotor requires at least two responses, mean stress and stress range, to predict. If all elements of the stress tensor (S_{11}, S_{12}, S_{13}, S_{22}, S_{23}, and S_{33}) at that point are required then at least 12 responses are needed. The number of responses increases further if it is desired to capture loads at multiple blade stations, using multiple harmonics of the periodic stress signal, on multiple independent blades, or on multiple rotors. Eventually, this problem leaves the domain of scalar surrogate modeling and would be better handled by reduced order models (ROMs), which are beyond the scope of this research. Since all these stress responses are likely correlated, the ability of a neural network to model and exploit this correlation gives a significant advantage over training multiple models using RSM or GPM. If the number of outputs is reduced at the expense of model fidelity, then RSM or GPM may be more appropriate than ANN.

As discussed previously, the fact that GPM predictions include both the expected value and the variance at each test point is extremely useful in structural reliability applications. This simplifies the process of determining the uncertainty in the load predictions and appropriately reflects the variable nature of uncertainty across the design space. In Figure 4.5, variance is lowest near the training points and largest at the maximum distance between training points. Compared to a global estimate of uncertainty derived from model representation error, the GPM result will likely lead to higher reliability levels since the nature of model uncertainty is better understood.

Based on the qualitative review and discussion of surrogate modeling techniques, it appears that scalar surrogate modeling is applicable to this problem. The application of surrogate models should provide a runtime advantage over individual calls to the comprehensive analysis program. This can be stated as a hypothesis to Research Question 1:

Hypothesis 1

At least one of the surveyed surrogate modeling methods can be used to derive a complete load spectrum from comprehensive analysis results, enabling rapid design space exploration.

The null hypothesis to Hypothesis 1 can be stated as follows:

> None of the surrogate modeling methods surveyed can be used to derive a complete load spectrum more rapidly than comprehensive analyses.

This will depend on the number of load predictions required to produce a fatigue life assessment. This process will be discussed further in Section 4.3.

4.3 Experiment 1 Overview

Experiment 1 is intended to establish that scalar surrogate modeling can be used to predict the fatigue loading on a rotor blade in arbitrary flight conditions. Additionally, it will compare the ability of the aforementioned surrogate modeling methods to predict rotor blade fatigue loads throughout the flight envelope. This section discusses the processes that will be used to compare the predictive capability and computational expense of each technique.

Experiment 1 is divided into two sub-experiments. Experiment 1a covers the development of a multidisciplinary analysis (MDA) tool, the implementation of a helicopter model in that tool, and the process of determining a *critical fatigue point* on the rotor blade. Experiment 1b involves training different surrogate models to predict the stress response at the critical fatigue point and comparing model performance.

4.4 Experiment 1a

Figure 4.6 provides an overview of Experiment 1a. The following sections detail the design of the experiment; implementation of the MDA, the helicopter model, and supporting analyses; results; and conclusions.

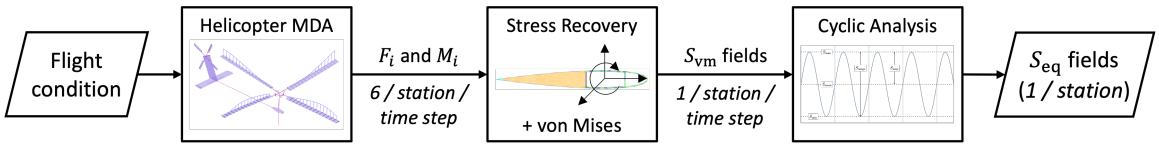


Figure 4.6: Overview of Experiment 1a.

4.4.1 Experimental Design

The first step of Experiment 1a is to develop the environment necessary to run the experiment, collect data, and analyze the results. The environment consists of a custom MDA tool based on the preliminary fatigue design methodology proposed in Section 3.4 (see Figure 3.1).

Then, a generic single-main-rotor (SMR) helicopter model will be implemented in the MDA. Although the preliminary fatigue design methodology is intended to be applicable to any arbitrary rotorcraft configuration, including exotic vertical lift aircraft currently being studied for implementation as UAM vehicles, this research makes use of a single main rotor transport helicopter for a number of reasons. First, the SMR configuration is well-studied and reference implementations exist in a many relevant software tools. Second, a conventional helicopter model can be tuned to match existing rotorcraft designs, which would enable validation of the predicted loads, stresses, and fatigue life produced by this methodology. This would not be possible for conceptual or prototype aircraft that have not yet undergone strenuous flight testing and flight loads surveys. Finally, because this methodology is built entirely upon physics-based models, it can be assumed to be applicable to arbitrary rotorcraft configurations as long as the models accurately capture that configuration's unique characteristics. Although the results may not be similar to or even follow the same trends as the generic SMR configuration, the methodology itself will be equally effective.

The generic SMR helicopter model is based off the Sikorsky UH-60A Black Hawk, which was selected because of the wide variety of reference implementations of this helicopter type in various design and analysis tools. Because many of the specific details of the UH-60A are subject to publication restrictions, the model implemented in this experi-

ment does not aim to exactly replicate the behavior of this helicopter but instead provide a reasonable approximation of a generic helicopter in the medium-lift utility class.

The generic SMR helicopter model will be executed in a number of extreme flight conditions, detailed by Table 4.1. Each flight condition is defined by six variables: airspeed (V), density altitude (h_d), gross weight (GW), rate of climb (ROC), turn rate (ω), and center of gravity station line offset (CG). These variables were chosen to allow the simulation of nearly all steady-state flight conditions relevant to fatigue damage accumulation, although certain conditions like pull-up and push-over cannot be simulated, and transient conditions are not considered.

Table 4.1: Extreme flight condition survey for Experiment 1a.

Case	Name	V (kt)	h_d (ft)	GW (lb)	ROC (ft/min)	ω (deg/s)	CG (ft)[a]
0	Baseline	130	5000	16,000	0	0	0.5
1	Hover	0	5000	16,000	0	0	0.5
2	High speed	160	5000	16,000	0	0	0.5
3	Low altitude	130	0	16,000	0	0	0.5
4	High altitude	130	10,000	16,000	0	0	0.5
5	Low weight	130	5000	11,500	0	0	0.5
6	High weight	130	5000	21,000	0	0	0.5
7	Descent	130	5000	16,000	−1400	0	0.5
8	Climb	130	5000	16,000	1400	0	0.5
9	Left turn	130	5000	16,000	0	−6	0.5
10	Right turn	130	5000	16,000	0	6	0.5
11	Forward CG	130	5000	16,000	0	0	−0.3
12	Aft CG	130	5000	16,000	0	0	1.6
13	Reverse	−20	5000	16,000	0	0	0.5

[a] CG offset is referenced to the main rotor hub station line.

The flight conditions in Table 4.1 were selected to approximately match the limits of the UH-60A flight envelope [106], except for the baseline case which is intended to represent a typical cruise condition and serve as a frame of reference for the other flight conditions. It is assumed that the edges of the flight envelope correspond to the most-damaging flight conditions in terms of fatigue damage accumulation.

For each flight condition, the MDA will predict force and moment components (F_x, F_y,

F_z, M_x, M_y, and M_z), where x, y, and z denote the axial, chordwise, and flapwise axes in the blade reference system, respectively. These components will be predicted at each station along the rotor blade and each time step in a complete blade revolution, producing a large number of load conditions.

Next, each load condition will be passed back into the cross-section model in stress recovery mode to predict the stress tensor field across the entire cross-section. The signed von Mises stress invariant will be calculated to reduce the six components of the stress tensor to a single value. Signed von Mises was chosen over standard von Mises because it can account for compression and tension loading, which is important in fatigue analysis.

A simple first-harmonic analysis of the cyclic stress signal at each point on the cross-section will be used to extract the mean stress, S_{mean}, and stress amplitude, S_{amp}. Goodman's relation will be applied to ultimately produce an equivalent stress, S_{eq}, field for the entire cross section. These equivalent stress fields will be analyzed to select an appropriate critical fatigue design point for use in Experiment 1b.

Because the generic SMR helicopter used in this research is not intended to exactly replicate an actual helicopter model, the results produced in this experiment and subsequent experiments cannot be validated. Instead, the MDA was implemented using software tools whose results have been independently validated. In this research, it is assumed that the results predicted by the MDA are sufficiently realistic to enable development and demonstration of different elements of the methodology. Verification and validation efforts will be discussed further in Chapter 7.

The next sections will describe the development of the MDA, helicopter model, and supporting tools necessary to implement the previously-described experimental plan.

4.4.2 Multidisciplinary Analysis

Figure 3.1 provides the building blocks for the MDA. The primary elements are the *vehicle performance model*, the *rotor comprehensive analysis*, and the *beam cross-section model*.

In this section, each of the primary elements and the software tools chosen for each element will be discussed in turn, including a brief review of the relevant theory and the capabilities of each tool.

4.4.2.1 Vehicle Performance Model

As described in Section 3.4, the vehicle performance model is responsible for calculating and providing a description of the vehicle flight state in all flight conditions in the mission spectrum. The flight state includes all parameters necessary to initiate a rotor loads prediction program, such as airspeed, atmospheric conditions, gross weight, center of gravity location, and pilot control positions necessary for trimmed flight. The vehicle performance model must be capable of storing the vehicle description, storing the definition of each flight condition, and rapidly determining the trim solution for each flight state.

This research will use NDARC, discussed previously in Section 2.1.2, to provide the vehicle performance model. NDARC was chosen due to its ability to accurately represent revolutionary vertical lift concepts using an extremely flexible component library. The program also has a wide user base in the rotorcraft academic community, which has led to the development of several useful extensions, such as RCOTOOLS, an NDARC–Python interface [107]. This section contains a brief review of the elements of NDARC that are applicable to this research; more detailed theory and description is provided by Johnson [36]. For reference, the NDARC program outline previously presented in Figure 2.2 is reprinted below.

NDARC Theory The central component of an NDARC job is the *aircraft description*, which defines the component systems of the vehicle. This description can be a fixed model or can be created by a conceptual design sizing task. For each component, properties such as the location, geometry, weight, aerodynamic model, power consumption or production model, and more, are specified. The overall aircraft properties are obtained from the combination

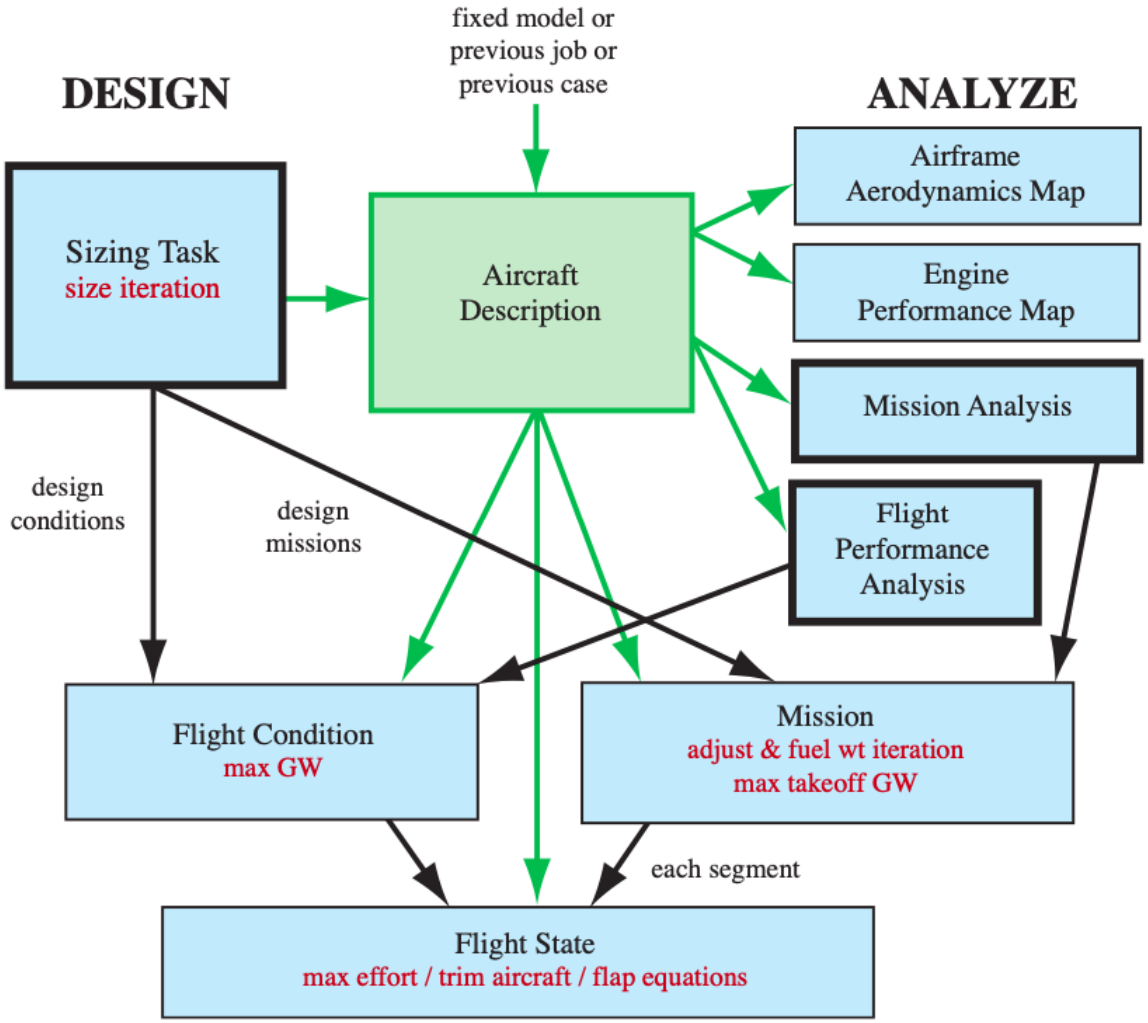


Figure 2.2: Outline of the NDARC program (reprinted from Page 36).

of the component properties. These attributes are then used for mission analysis, flight performance analysis, and performance mapping tasks.

Flight state solutions are produced by both the mission analysis and flight performance analysis tasks. In NDARC, a mission is a series of segments where vehicle properties, such as fuel weight, payload weight, and configuration, are applied consistently to subsequent segments. Each segment represents a different portion of the mission, such as taxi, take-off, hover, climb, cruise, and so on. The flight condition defined in the current segment is used to calculate a flight state solution, and that information is used to determine power required, power available, fuel burn rate, and other important parameters. Conversely, flight

performance analysis consists of defining a set of unrelated flight conditions and producing flight state solutions for each condition. This is commonly used for *point performance* calculations, such as maximum speed, service ceiling, and maximum takeoff gross weight. In the context of the current research, either analysis type could be used to produce the necessary data; mission analysis is beneficial if one desires to track gross weight and CG shift throughout a mission as fuel is burned.

The flight state solution process involves trimming the aircraft. Trim is achieved by solving for the pilot control positions and vehicle attitude that produce equilibrium, which is achieved when all the forces and moments on the vehicle sum to zero. Thus, solving the trim solution requires a detailed accounting of all the forces and moments produced by or acting on each component of the vehicle. These forces and moments are found using aerodynamic models that are tailored specifically to each component. For example, rotor loads are calculated using BET to determine the blade motion and airloads, and momentum theory to calculate the inflow. NDARC is capable of modeling some higher-order effects, such as non-uniform inflow, tip loss, and interference with other components. Limitations include restricting the blade flapping motion to rigid blades with no hinge offset: only the coning and 1/rev flapping components are considered. Forces in the rotating frame are transformed into the non-rotating frame to calculate the hub forces, which in turn are applied to the airframe.

The trim solution is found numerically using the Newton-Raphson method. Trim variables, such as pilot controls[1] and vehicle Euler angles, are adjusted until the trim targets, typically the summed forces and moments, reach zero within some predefined tolerance. Once the trim solution is found for a given flight state, the pilot control positions, the definition of the aerodynamic environment, and any vehicle properties such as weight and inertia can be extracted and transferred to the comprehensive analysis for rotor loads calculations.

[1]In NDARC, pilot controls (collective, cyclic, etc.) are mapped to component controls (swashplate position, elevator position, etc.) using a control mixing matrix.

NDARC Validation NDARC was validated by comparing its predictions to known flight performance data, weight statements, and comprehensive code predictions for four different real-world rotorcraft. Johnson [108] used the UH-60A to represent the SMR configuration, the CH-47D to represent the tandem configuration, the XH-59A to represent the coaxial lift-offset configuration, and the XV-15 to represent the tiltrotor configuration. NDARC predictions were found to closely match the truth data in each case, although the degree of similarity is dependent on tuning rotor power consumption models to match comprehensive analysis results. Thus, the NDARC model of the generic SMR helicopter, which will be described in Section 4.4.3.1, could be tuned to match an existing helicopter, unlocking the predictive power of NDARC.

4.4.2.2 Comprehensive Analysis

Comprehensive analysis programs were discussed previously in Section 2.1.1. These codes combine the disciplines of structural dynamics, aerodynamics, controls, and flight dynamics to produce detailed predictions of rotorcraft attributes such as performance, loads, vibration, noise, stability, and handling qualities. Most programs are founded on rigorous dynamics theories, allowing for precise analysis of rotating flexible beams. In this application, comprehensive analysis will be used primarily for calculating the internal forces and moments in the rotor blades, or another rotor component of interest.

The preferred comprehensive analysis program for this research is RCAS, developed by Advanced Rotorcraft Technology, Inc. (ART) under a contract from the Aeroflightdynamics Directorate of AMRDEC. RCAS was chosen due to its widespread adoption in the rotorcraft research community (see Section 2.1.3) and the author's own familiarity with the program. Additionally, RCAS offers a mixed-fidelity modeling approach: the same model can be adapted for low-fidelity (e.g. rigid blade), medium-fidelity (e.g. nonlinear flexible blades), and high-fidelity (e.g. CFD/CSD coupling) analyses. This allows the user to tailor the analysis for his or her desired levels of accuracy and runtime, which is especially useful

for research which requires a large number of cases to be executed rapidly. This section contains a brief review of RCAS theories and capabilities; a more detailed description of the program is provided by Saberi, Khoshlahjeh, Ormiston, et al. [109].

RCAS Theory RCAS is capable of modeling a wide range of rotary-vehicle configurations in hover, trimmed flight, and maneuvering flight. Defining a vehicle in RCAS requires the construction of any number of interconnected, coupled models: the structural model, the aerodynamic model, the engine/drivetrain model, and the control system model. The RCAS model hierarchy is presented in Figure 4.7. Only the structural model and the aerodynamic model will be discussed here, as this research will not make use of the other models.

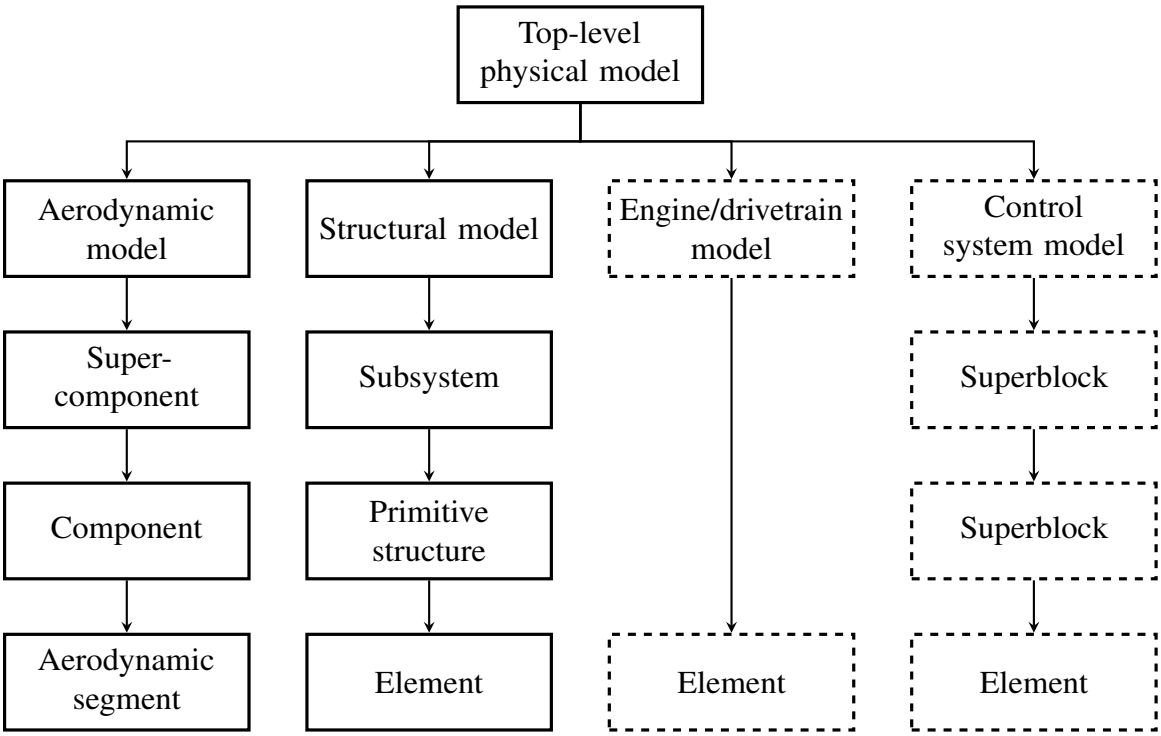


Figure 4.7: RCAS physical model hierarchy. Components with dashed outlines will not be used in this research.

Structural Model The structural model is decomposed into subsystems, such as the fuselage, rotor system, tail boom, empennage, and so on. Each subsystem is composed of primitive structures, such as rotor blades, wings, or stabilizers. The primitive structures are

in turn built of finite elements, which are the fundamental building blocks of the structural model. Elements are defined by their type, their properties, the location of their nodes, the connectivity between elements, and the constraints on each element. RCAS features an extensive library of structural elements to aid model construction; a portion of these elements are defined in Table 4.2.

Table 4.2: Examples of RCAS structural elements.

Type	DOF	Features	Applications
6-DOF rigid body	1 to 6	Body axis velocities as DOFs	Large maneuver motion
Rigid body mass	0	With CG offset	Rigid fuselage, stores, blades, rotating masses
Rigid bar	0	With mass and CG offset	Hinge offsets, very stiff components
Hinge	0 to 2	Free or controlled with linear spring and damper	Flap, lag, pitch bearing, gimbal for rotor hub
Slide	0 to 1	Free or controlled with linear spring and damper	Pitch link input from swashplate, landing gear
Spring	0	Linear, nonlinear, translational, rotational	One-dimensional rods, control stiffness
Damper	0	Linear, nonlinear, translational, rotational	Elastomeric bearings, snubbers, viscous dampers
Nonlinear beam	1 to 9	With elongation DOF, reduced geometric coupling, and material anisotropy	Rotor blades, tail boom, fuselage elastic parts, other elastic components

The nonlinear beam (NLB) element is perhaps the most important of the elements listed in Table 4.2. The NLB is used to model flexible rotor blades, among other structures. The element is based on the moderate deformation beam theory described by Hodges and Dowell [110]. Constructing a rotor blade or similar structure of multiple NLBs allows arbitrarily large deformations to be calculated precisely. Each NLB has nine degrees of freedom (DOFs), but any arbitrary degree of freedom can be removed to simplify the model or create a rigid blade. Elastic properties and structural twist can be defined for the blade

independent of the number and location of NLB elements. The properties are then linearly interpolated to Gaussian quadrature points defined along the beam. An alternative NLB formulation known as the geometrically-exact composite beam (GECB), based on updated theories by Hodges [111], provides greater accuracy but must be defined with a series of 6×6 mass and stiffness matrices instead of scalar stiffness values.

Aerodynamic Model The RCAS aerodynamic model uses a similar hierarchy of components. The aerodynamic model is built from supercomponents, such as rotors, wings, bodies, and auxiliary rotors. Each supercomponent is constructed from three types of components: lifting surfaces (e.g., blades and wings), lifting bodies, and auxiliary disk rotors. Components function like a mesh of connected aerodynamic segments. Aerodynamic segments are discrete representations of airfoils or bodies; each has an aerodynamic computation point at the segment center which is used for local velocity and force calculations.

The airloads on each segment are determined from the local flow velocity, which is a function of the rigid body motion of the complete model, the inflow on the component, and the dynamic response of the specific point. Airloads are calculated from linear or nonlinear analytical functions or lookup tables which consider velocity, angle of attack, Mach number, compressibility corrections, tip loss corrections, and yawed flow effects. RCAS also has the capability to model linear or nonlinear unsteady flow effects, such as dynamic stall, trailing edge separation, and vortex shedding.

The inflow on a rotor or wing is calculated through the use of customizable inflow models and interference models. Inflow models include uniform inflow, blade-element momentum thoery (BEMT), the Peters–He generalized dynamic wake model, a prescribed vortex wake model, and two free vortex wake models. Rotor-to-rotor interference models include simple rotor-to-rotor interaction, horseshoe vortex wake, cylindrical vortex sheet, and prescribed or free vortex wake. Interference with wings and bodies is calculated using horseshoe vortex wake, prescribed vortex wake, or source–sink models.

Solution Procedures At their core, RCAS solutions are similar to conventional finite element approaches, with some modifications. The system equations are formed first by forming the system DOFs, $\boldsymbol{X}$, from the element DOFs, $\boldsymbol{x}$. Then, the element equations are expressed in terms of the residual force vector, $\boldsymbol{q}$. The element residuals are collected into the system residual vector, $\boldsymbol{Q}$, which is then linearized by perturbation to form:

$$\boldsymbol{Q} = F - M\ddot{\boldsymbol{X}} - C\dot{\boldsymbol{X}} - K\boldsymbol{X} \tag{4.18}$$

where

$$M = \frac{\partial Q}{\partial \ddot{\boldsymbol{X}}}, \quad C = \frac{\partial Q}{\partial \dot{\boldsymbol{X}}}, \quad K = \frac{\partial Q}{\partial \boldsymbol{X}}$$
$$\ddot{\boldsymbol{X}} = \frac{\mathrm{d}^2\boldsymbol{X}}{\mathrm{d}t^2}, \quad \text{and } \dot{\boldsymbol{X}} = \frac{\mathrm{d}\boldsymbol{X}}{\mathrm{d}t}$$

and F is the vector of forces. The dimensionality of Equation (4.18) can be reduced using *modal reduction*, whereby certain vibratory modes are dropped from the model, leaving only the most influential modes and improving processing time [112]. Equation (4.18) is numerically integrated in time using the Newmark-Beta method, and the Newton-Raphson method is used within each time step to drive the residual vector to zero.

RCAS contains three primary analysis types: periodic analysis, trim analysis, and maneuver analysis. Periodic analysis involves solving the equations of motion for a defined time period, T, such that the solution at time t is equal to the solution at time $t + T$. For example, if T is set to the length of a single rotor revolution, then the periodic solution would provide a complete simulation of the model in a fixed and steady flight condition. The periodic solution can be found by numerically integrating the system equations in time until all the transients have decayed or by applying the harmonic balancing method in the frequency domain.

The trim solution consists of a series of repeated periodic solutions, between which the

trim variables are adjusted until the trim responses match their targets. Generally, the trim responses are averaged over one revolution to account for the natural fluctuation in rotor loads over a single revolution in forward flight. This process proceeds in a manner similar to the NDARC trim solution discussed previously. First, the trim sensitivity matrix is formed by perturbing each of the trim variables in turn and then the Newton-Raphson method is used to complete the trim process.

Finally, maneuver analyses allow the user to examine the nonlinear transient response of the model subject to some perturbation. This solution process involves first establishing a periodic or trim solution as the initial conditions, then numerically integrating the nonlinear system equations in the time domain. Parameters such as the pilot control inputs can be varied continuously during the maneuver analysis.

After a given solution has been completed, the analyst can extract and visualize any arbitrary model parameter. In this research, the blade forces and moments will be extracted at every spanwise blade station for each time step in the periodic solution. This data will then be transferred to the cross-section model to find the internal stress and strain field in the rotor blade.

RCAS Validation Saberi, Khoshlahjeh, Ormiston, et al. [109] validated RCAS by comparing its results to simple elastic problems, structural dynamic problems, and rotor system studies. First, RCAS results were shown to match the exact solution of an elastic beam with a tip moment, even when the model exhibits extreme deformation. Next, the authors demonstrated that the RCAS nonlinear beam element also matches the results of the Princeton Beam experiment, which measured the static deformation and natural frequencies of a cantilever beam with a tip weight at various pitch angles. Further RCAS validation efforts included predicting the structural dynamics of a rotating blade with a swept tip, and predicting UH-60A rotor blade loads. In the latter case, RCAS predictions showed good agreement except for the torsional moment predictions.

Bir [112] performed further RCAS validation experiments using wind turbine models. RCAS results were compared with analytical results, or predictions from the UMARC and ADAMS comprehensive codes. RCAS was found to be as accurate as the other two tools, although its adoption in the wind turbine industry has been hindered by the difficulty of learning to operate and extracting simulation data from RCAS.

4.4.2.3 Cross-Section Model

In the methodology presented in Figure 3.1, the beam cross-section model plays two critical roles. First, it is responsible for calculating beam elastic properties from the cross-sectional blade design; these properties are later imported into the nonlinear beam definition in the comprehensive model. Additionally, the cross-section model is used to find the stress and strain fields on the blade cross-section; these results are then used along with a S-N curve to calculate cumulative damage on the component using Miner's sum.

In this work, VABS will be used to provide the cross-section model. VABS is a beam sectional analysis program developed primarily by Prof. Dewey Hodges of the Georgia Institute of Technology and Prof. Wenbin Yu of Utah State University [88]. VABS has been continuously developed since 1989 and currently constitutes a mature, industry-ready software program. Although the majority of VABS use cases involve the analysis of composite rotor blades, it is applicable to many slender structures constructed from a variety of materials [113]. VABS was selected for use in this research due to its ability to accomplish both the elastic property calculation and stress recovery tasks, its previous use in similar applications (see Sections 2.3.1 and 2.3.3), and the author's previous experience with this program. This section includes an overview of VABS theory and operations.

VABS Theory The theory behind VABS is based upon a geometrically exact description of nonlinear one-dimensional beam kinematics and a decomposition method known as the variational asymptotic method (VAM). VAM allows the decomposition of the three-

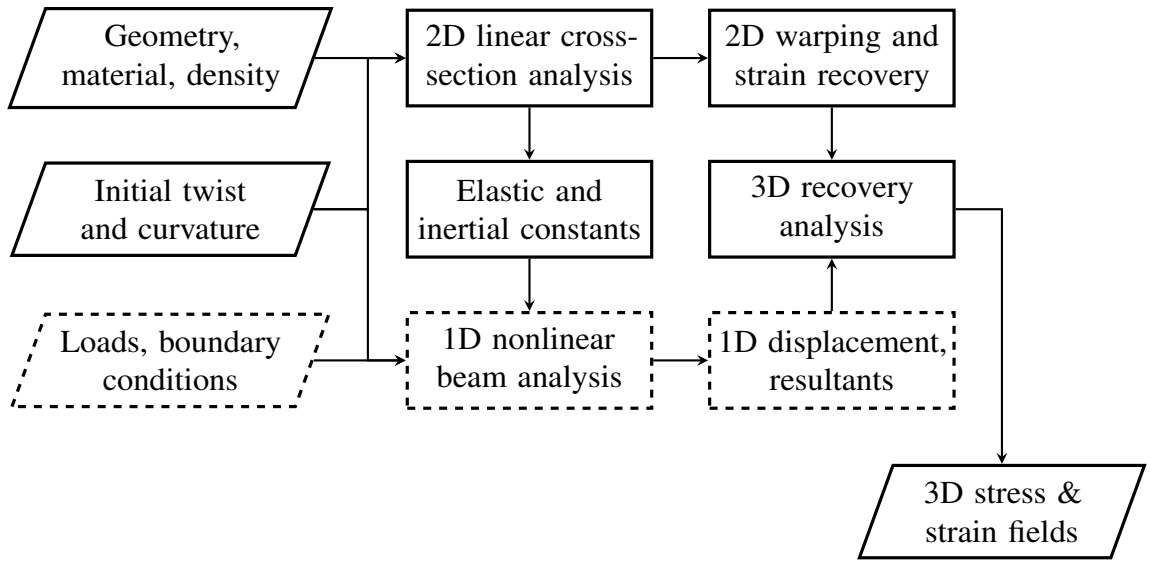


Figure 4.8: VABS analysis process. Dashed boxes represent components that will be replaced by the comprehensive analysis solution in this research.

dimensional nonlinear beam elasticity analysis into a nonlinear one-dimensional beam analysis and a linear two-dimensional cross-sectional analysis. This process is illustrated in Figure 4.8.

In the two-dimensional linear cross-section analysis, VABS uses a finite-element representation of the beam cross-section, which requires complete definitions of geometry, material stiffness, and material density, to produce spanwise elastic and inertial properties. Materials can be isotropic, orthotropic, or anisotropic [113]. The finite-element mesh can be created using a third-party FEA program such as ANSYS or a VABS pre-processor known as PreVABS [114].

The outputs of this process can be tailored to the required fidelity. For example, in simple applications, the *classical* beam model can be used:

$$\mathcal{U} = \frac{1}{2} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix}^\top \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{12} & H_{22} & H_{23} & H_{24} \\ H_{13} & H_{23} & H_{33} & H_{34} \\ H_{14} & H_{24} & H_{34} & H_{44} \end{bmatrix} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix} \tag{4.19}$$

where $\mathcal{U}$ is the strain energy per unit length, γ_{11} is the axial strain, κ_1 is the elastic twist, κ_2 and κ_3 are the elastic bending curvatures, and H_{ij} are generalized stiffness coefficients. Note that by convention the x_1-axis is oriented along the blade reference line and the x_2-axis is oriented chordwise pointing from the trailing edge to the leading edge. In the simplest case of homogeneous prismatic beams made of isotropic materials for which the x_2 and x_3 axes are aligned with the principal axes and originating from the shear center, Equation (4.19) reduces to the more familiar

$$\mathcal{U} = \frac{1}{2} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix}^\top \begin{bmatrix} EA & 0 & 0 & 0 \\ 0 & GJ & 0 & 0 \\ 0 & 0 & EI_2 & 0 \\ 0 & 0 & 0 & EI_3 \end{bmatrix} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix} \tag{4.20}$$

where EA is the axial stiffness, GJ is the torsional stiffness, and EI_2 and EI_3 are the bending stiffnesses.

The *generalized Timoshenko* model includes transverse shear strain:

$$\mathcal{U} = \frac{1}{2} \begin{bmatrix} \gamma_{11} \\ 2\gamma_{12} \\ 2\gamma_{13} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix}^\top \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} & H_{15} & H_{16} \\ H_{12} & H_{22} & H_{23} & H_{24} & H_{25} & H_{26} \\ H_{13} & H_{23} & H_{33} & H_{34} & H_{35} & H_{36} \\ H_{14} & H_{24} & H_{34} & H_{44} & H_{45} & H_{46} \\ H_{15} & H_{25} & H_{35} & H_{45} & H_{55} & H_{56} \\ H_{16} & H_{26} & H_{36} & H_{46} & H_{56} & H_{66} \end{bmatrix} \begin{bmatrix} \gamma_{11} \\ 2\gamma_{12} \\ 2\gamma_{13} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{bmatrix} \tag{4.21}$$

and the *generalized Vlasov* model adds restrained warping effects:

$$\mathcal{U} = \frac{1}{2} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \\ \kappa_1' \end{bmatrix}^\top \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} & H_{15} \\ H_{12} & H_{22} & H_{23} & H_{24} & H_{25} \\ H_{13} & H_{23} & H_{33} & H_{34} & H_{35} \\ H_{14} & H_{24} & H_{34} & H_{44} & H_{45} \\ H_{15} & H_{25} & H_{35} & H_{45} & H_{55} \end{bmatrix} \begin{bmatrix} \gamma_{11} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \\ \kappa_1' \end{bmatrix} \tag{4.22}$$

where κ_1' is the twist rate. A more complex representation can be used to incorporate the trapeze effect, which is not covered here. The cross-sectional analysis procedure also includes calculating the cross-section inertia matrix, which is identical for each model.

The formulation of the spanwise strain energy is used in the elastodynamic beam equations, which are derived from Hamilton's extended principle as follows:

$$\int_{t_1}^{t_2} \int_0^{\ell} \left[\delta(\mathcal{K} - \mathcal{U}) + \overline{\delta \mathcal{W}} \right] \mathrm{d}x_1 \, \mathrm{d}t = 0 \tag{4.23}$$

where δ is the Langrangean variation, $\mathcal{K}$ is the kinetic energy per unit length, $\overline{\delta \mathcal{W}}$ is the applied virtual work per unit length, t_1 and t_2 are arbitrary times, and ℓ is the length of the beam [88]. The complete derivation of VABS equations using VAM is beyond the scope of this text; an in-depth description of this procedure is given by Hodges [111].

The choice of cross-sectional stiffness model is dictated by the fidelity of the one-dimensional beam analysis. The one-dimensional nonlinear beam analysis component of VABS is capable of incorporating any of the models discussed previously, providing extremely precise results. However, in this research, the one-dimensional nonlinear beam analysis component of VABS will be replaced with the comprehensive analysis blade airloads solution as discussed in Section 4.4.2.2. The simpler NLB element formulation used by RCAS to model rotor blades mandates the use of the classical model, which results in a slight loss in fidelity. However, the more advanced GECB formulation enables direct

application of the generalized Timoshenko mass and stiffness matrices.

Once the spanwise blade airloads are determined, VABS will be used again to calculate the three-dimensional stress and strain fields in the cross-section. In VABS terminology, this process is known as *recovery*. The recovery process produces displacement, stress, and strain values at each node and Gaussian integration point in the finite-element mesh. Averaged values of each variable are also calculated for the entire element to aid in visualization. Each variable is reported in the beam coordinate system and the material coordinate system, which is necessary for fatigue failure analysis.

VABS Validation Yu, Volovoi, Hodges, et al. [115] validated VABS using a number of different test cases. First, an elliptic bar was modeled analytically using VAM, and the solution was identical to that derived from the theory of elasticity. Next, VABS was used to calculate the shear center location for various cross-section geometries, which agreed with common engineering assumptions used to approximate the shear center. Finally, VABS results were compared to the three-dimensional finite element code ABAQUS. In this study, the stress and strain fields recovered by VABS showed strong agreement with the 3D stress and strain values predicted by ABAQUS.

4.4.2.4 Implementation

The previous sections described the core components necessary to construct the MDA. In order to complete the MDA, the tools must be connected in a manner that enables them to act as a single consistent solver. A diagram of the necessary connections is presented in Figure 4.9.

First, the PreVABS+VABS model of the rotor blade cross-section is executed. PreVABS provides an XML-based language for defining the rotor blade cross section, including material properties, composite layups, and geometry. PreVABS meshes the cross-section model and generates input files for VABS, which is then predicts mass and stiffness matrices

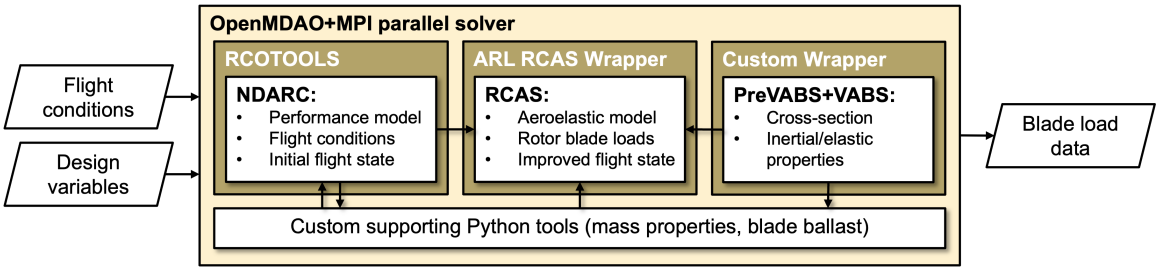


Figure 4.9: Summary of the multidisciplinary analysis tool.

for the cross section. The mass and stiffness matrices are passed directly to RCAS, and the mass per unit length of the blade is used to calculate the total blade mass, which is then passed to NDARC.

Next, the NDARC model of the vehicle is executed. NDARC predicts weights for each of the primary components of the vehicle, such as the fuselage, rotor, engine, and stabilizers. These weights, and associated moments of inertia, are passed to RCAS. NDARC also solves the trim solution for the flight condition of interest. The converged values for pitch, roll, collective, lateral cyclic, longitudinal cyclic, and pedal position are used as the initial condition for the RCAS trim solution. NDARC also provides a summary of the atmospheric environment in that flight condition, including the viscosity, density, and speed of sound, which are also passed to RCAS.

After all the other analyses are complete, RCAS is executed in trim solution mode. When the trim solution is complete, RCAS outputs the force and moment components at each blade station for each time step in the periodic solution. This constitutes the final output of the MDA.

Later, the PreVABS+VABS model will be executed in stress recovery mode to produce stress tensor fields for each load case provided by RCAS. In this case, PreVABS+VABS is executed in a standalone fashion and not alongside any other program. The stress tensor field will be processed as described in Section 4.4.1 to produce S_{eq} fields for each load case.

All data transfer between applications will be accomplished using Python. Python was selected for this application because it provides a single integrated environment for data

collection and analysis. Specifically, the OpenMDAO package [116] simplifies data transfer between and execution of various software tools.

To enable communication through OpenMDAO, each software tool needs a *wrapper* that maps specific OpenMDAO variables to the appropriate position in that programs' input or output files. The basic process is to read the input file, convert that file into a Python-native data structure, update values within that data structure, then regenerate a new input file with the desired updates. This enable automatic execution of different models without requiring any manual intervention by a human operator.

NDARC makes use of the RCOTOOLS wrapper described in Section 4.4.2.1. For RCAS, Michael Avera of the Army Research Laboratory (ARL) provided a prototype Python wrapper. Because this wrapper was originally intended to be used only with isolated rotor models, further developments were necessary to enable operation with a complete helicopter model. Additional development was required to enable parsing of the tabular force and moment history files produced by RCAS. The code for the RCAS wrapper is included in Appendix A.1.

For PreVABS+VABS, a completely custom Python wrapper was developed based on the `xmltodict` Python package. This wrapper is able to modify arbitrary attributes of the PreVABS cross-section model, such as material properties, layup definitions, or even geometry. Other elements of the wrapper were developed to read elastic properties and stress tensor fields from the VABS output files. The code for the PreVABS+VABS wrapper is included in Appendix A.2.

Further details related to the MDA and its elements will be provided in Sections 4.4.3 and 4.4.4.

4.4.3 Generic SMR Helicopter Model

As described in Section 4.4.1, a generic single main rotor helicopter similar to the UH-60A Black Hawk is used as a test case for Experiment 1. This section describes the

implementation of this vehicle in each of the components of primary components of the MDA.

4.4.3.1 NDARC Model

The NDARC model of the generic SMR helicopter is based on the "helicopter" example distributed with NDARC. Most attributes of this model were tuned to more closely match the known characteristics of the UH-60A. Additionally, the model was converted from a scalable model used for sizing tasks to a fixed-dimension model which is more appropriate for performance analysis. The basic attributes of the NDARC model are described in Table 4.3.

Table 4.3: Basic attributes of the generic SMR helicopter NDARC model.

Property	Units	Value
Design gross weight	lb	16,000
Operating weight	lb	11,308.2
Empty weight	lb	10,533.2
Installed horsepower	HP	3200
Main rotor		
Radius	ft	26.8
Number of blades	—	4
Blade chord	ft	1.75
Hub type	—	articulated
Incidence	deg	3
Tail rotor		
Radius	ft	6
Number of blades	—	4
Blade chord	ft	1.75
Hub type	—	hingeless
Cant[a]	deg	0

[a] The UH-60A has a tail cant of approximately 20°; this was removed for simplicity.

A 3D sketch of the NDARC model is presented in Figure 4.10. Note that this model is not intended to be a physically accurate description of the helicopter, but is useful for

visualizing the relative sizes and positions of each component.

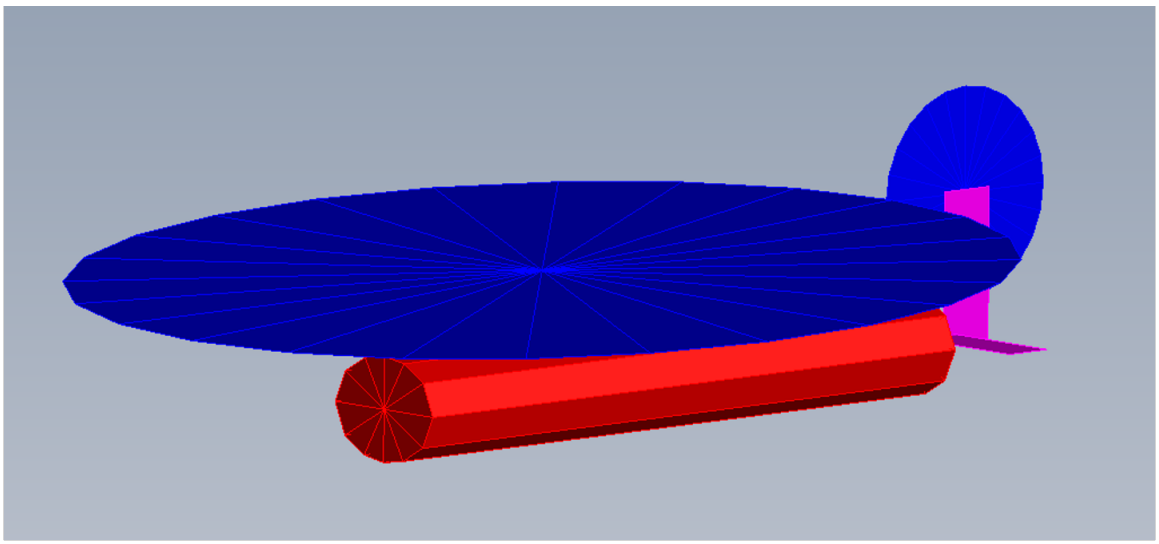


Figure 4.10: 3D representation of the generic SMR helicopter NDARC model.

In order to inform the experimental design presented in Table 4.1, the basic performance attributes of the NDARC model were analyzed. The results of the performance study are presented in Table 4.4.

Table 4.4: Performance of the generic SMR helicopter NDARC model.

Property	Units	Value	Notes
Maximum takeoff weight	lb	21,866.09	Sea level standard atmosphere (SLS), intermediate rated power (IRP)
Maximum cruise speed	kt	159.34	SLS, design gross weight (DGW), maximum continuous power (MCP)
Best range speed	kt	129.31	SLS, DGW
Hover ceiling	ft	11,759.32	Standard atmosphere, DGW, IRP
Cruise ceiling	ft	18,340.84	SLS, DGW, MCP, 130 kt
Maximum climb rate	ft/min	1447.70	SLS, DGW, MCP, 130 kt
Maximum turn rate	deg/s	11.88	SLS, DGW, MCP, 130 kt

Note that these values are based only on power required; specifically, NDARC solves each case such that the power required is equal to the power available. NDARC does not make any considerations for aerodynamic, aeroelastic, or structural effects that may limit performance in certain extreme conditions.

The files required to model the generic SMR helicopter in NDARC are included in Appendix B.1. Tables B.1 to B.3 describe the mapping between NDARC's input and output variables and the OpenMDAO variables. The OpenMDAO variables will be described further in Section 4.4.4.

4.4.3.2 RCAS Model

The RCAS model of the generic SMR helicopter is geometrically identical to the NDARC model. Different elements of the helicopter are modeled using the RCAS building blocks described previously in Table 4.2. The rotor system of the RCAS model is derived from "Training Example 4" distributed with RCAS, which was expanded from an isolated rotor model to a complete 6-DOF helicopter model.

The main rotor consists of a series of flexible and rigid beams, each with its own mass and inertia. The rotor blades themselves are modeled using the GECB formulation of the NLB element. The beam is composed of 11 nodes and has two degrees of freedom in the axial, lead–lag, flap, and torsion directions, as well as both shear directions. Each of the 10 elements has six Gauss integration points. At runtime, the 6×6 mass and stiffness matrices required to define the GECB element are passed from VABS to RCAS. The fully-articulated swashplate is constructed from rigid beam elements, hinges, and pitch bearings. A spring in series with the pitch link simulates control stiffness.

Aerodynamically, the main rotor is composed of 15 aerodynamic segments, each of which models a quasi-nonlinear NACA0012 airfoil. The swept tip of the UH-60A was removed for simplicity, but the $-8°$ aerodynamic and structural blade twist was retained. Blade element monentum theory (BEMT) was used for the inflow model in order to keep runtime low. Correction factors for yawed flow, tip loss, linear unsteady effects, and compressibility effects are present in the model. Additionally, the main rotor is simulated using the single blade analysis option, where the motion of only one rotor blade is calculated then duplicated to the other three blades. This improves runtime and, since all simulated

flight conditions are steady-state, does not impede accuracy.

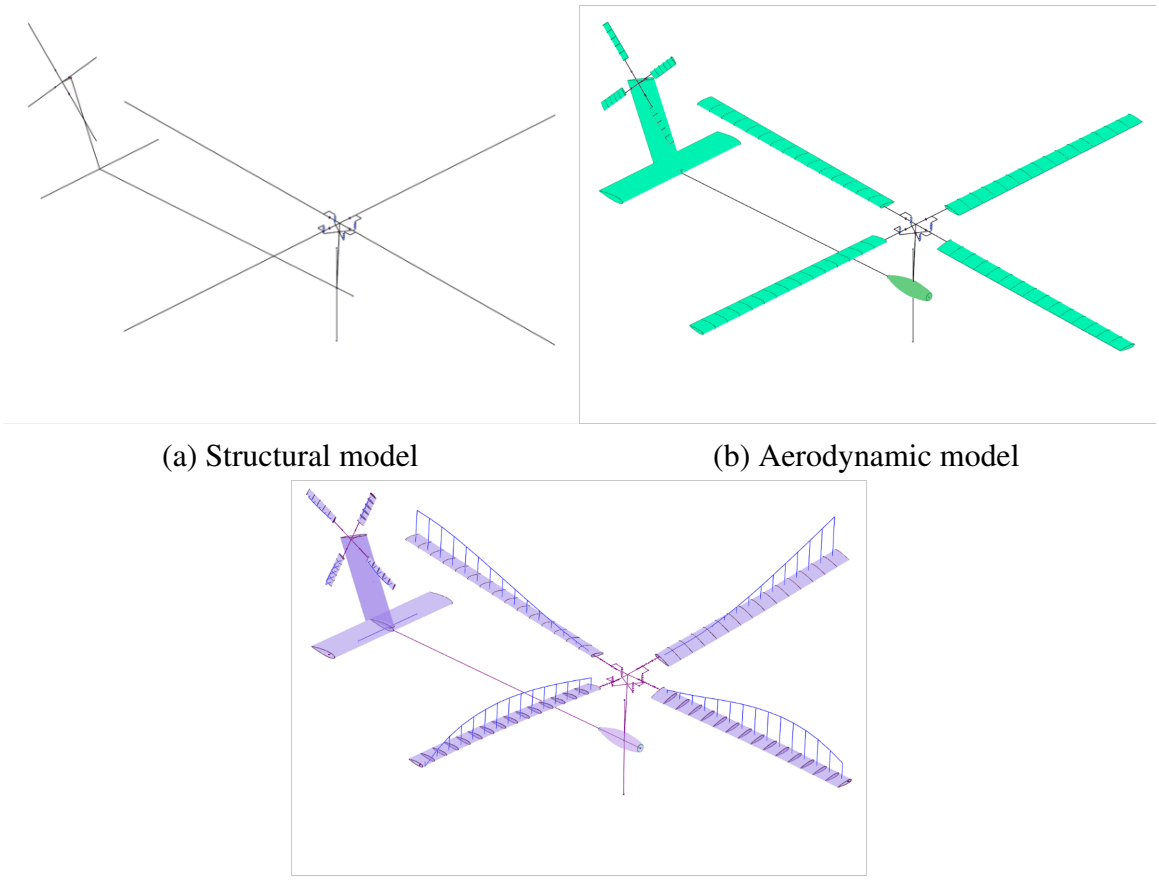
Since the main rotor is the focus of this research, the remainder of the helicopter is modeled using simpler elements. The fuselage is composed of rigid beams with mass, inertia, drag, lift, and pitching moment. The horizontal and vertical stabilizers are composed of rigid beams with mass and inertia. Aerodynamically, the stabilizers use simple linear airfoil definitions with uniform inflow.

The tail rotor is constructed of rigid beams with mass and inertia. A swashplate is not modeled, but a pitch bearing is included to enable yaw control. The tail rotor uses a NACA0012 airfoil with uniform inflow. Each rotor blade is constructed of five aerodynamic segments. The tail rotor also uses the single blade analysis option.

Particular attention was paid to ensuring consistency between the NDARC and RCAS models. Where applicable, all aerodynamic properties in the NDARC model, such as lift and drag coefficients of the fuselage, stabilizers, and tail rotor, were copied to the RCAS model. OpenMDAO ensures that all weight properties of the NDARC model, which may change for each run, are copied to the RCAS model at runtime. This process is described further in Section 4.4.4.

Figure 4.11 presents different renderings of the RCAS model of the generic SMR helicopter. Figure 4.11a depicts the beam and point mass elements that make up the structural definition. Figure 4.11b overlays the aerodynamic elements of the model over the structural model. Figure 4.11c shows a freeze-frame of the model in forward flight. The z-component of the aerodynamic loads on the main rotor are plotted to show the lift distribution on the rotor blade. The rotor blade is noticeably deforming, as would be expected in any flight condition.

The RCAS file used to model the generic SMR helicopter could not be made publicly available. A detailed description of the RCAS model, including the various components of the structural and aerodynamic models, is presented in Appendix B.2. The descriptions and tables in this appendix provide enough information for the reader to reconstruct the RCAS



(a) Structural model

(b) Aerodynamic model

(c) Deformation and loads

Figure 4.11: Different 3D representations of the generic SMR helicopter in RCAS.

model if desired. Tables B.28 to B.31 describe the mapping between the variables in this file and the OpenMDAO variables. The OpenMDAO variables will be described further in Section 4.4.4.

4.4.3.3 PreVABS+VABS Model

Unlike the NDARC and RCAS models, the PreVABS+VABS model does not simulate the entire vehicle. Instead, this model represents only the two-dimensional cross section of the main rotor blade. The PreVABS+VABS model is derived from a composite cross-section developed by Rohl, Cesnik, Dorman, et al. [117]. These authors, and later Kumar [118], modeled a parametric rotor blade cross section and used numerical optimization to match the inertial and elastic properties of that model to known properties of the UH-60A rotor

blade. Because the internal structure of the UH-60A rotor blade is not publicly available, Rohl's model serves as a useful substitute.

Rohl's cross section model was recreated in PreVABS using the NACA0012 outer mold line to match the RCAS aerodynamic model. The model consists of five materials: IM7, E-glass, S-glass, plascore, and steel. The box spar of the rotor blade is composed of IM7 with a 0°/45°/–45°/90° layup. The entire outer mold line is covered in an E-glass overwrap with a 0°/45°/–45°/0° layup. The leading edge is reinforced with IM7 under the overwrap, and is covered in a steel erosion strip, which serves to protect the blade from dust or large particles which can cause pitting. The extreme trailing edge is filled with S-glass, and the remainder of the area between the S-glass fill and the box spar is filled with plascore. Each material was modeled orthotropically using the values in Table 4.5.

Table 4.5: Orthotropic materials used in the PreVABS+VABS model, from Rohl, Cesnik, Dorman, et al. [117].

Property	Units	IM7	E-glass	S-glass	Steel	Plascore
ρ	slug/ft^3	3.01	3.34	3.61	15.13	0.09
E_{11}	lb/ft^2	3.45×10^8	4.32×10^8	9.06×10^8	4.28×10^9	1.44×10^5
E_{22}	lb/ft^2	1.84×10^8	4.32×10^8	2.51×10^8	4.28×10^9	2.88×10^6
E_{33}	lb/ft^2	1.84×10^8	4.32×10^8	2.51×10^8	4.28×10^9	1.44×10^5
G_{12}	lb/ft^2	1.02×10^8	8.55×10^7	7.52×10^7	1.61×10^9	5.01×10^5
G_{13}	lb/ft^2	1.02×10^8	8.55×10^7	7.52×10^7	1.61×10^9	1.44×10^5
G_{13}	lb/ft^2	1.02×10^8	8.55×10^7	7.52×10^7	1.61×10^9	8.35×10^5
ν_{12}	—	0.34	0.15	0.28	0.30	0.01
ν_{13}	—	0.34	0.15	0.28	0.30	0.30
ν_{23}	—	0.30	0.30	0.30	0.30	0.01

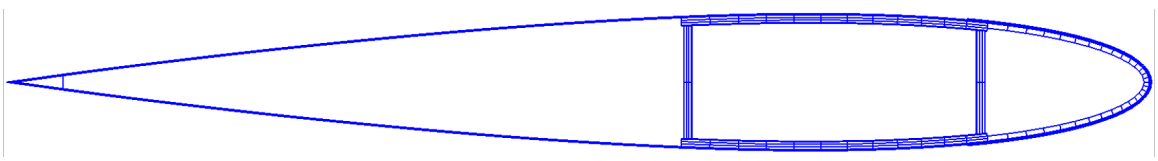
Rohl, Cesnik, Dorman, et al. tuned the ply thicknesses of each structural element to match the UH-60A elastic properties. The results require that a number of plies of each material of varying thickness are used in the cross-section design. Although this may not be realistic from a manufacturing standpoint, the ply thickness specifications were left unchanged in this research.

Figure 4.12 presents the complete PreVABS+VABS cross section model. Figure 4.12a

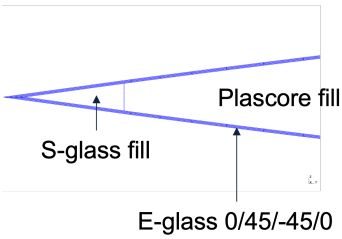
Table 4.6: Ply thicknesses used in the PreVABS+VABS model.

Component	Material	Ply number	Thickness (ft)	Angle (deg)
Spar	IM7	1	0.00410	0
Spar	IM7	2	0.00372	45
Spar	IM7	3	0.00372	–45
Spar	IM7	4	0.00100	90
Overwrap	E-glass	1	0.00050	0
Overwrap	E-glass	2	0.00050	45
Overwrap	E-glass	3	0.00050	–45
Overwrap	E-glass	4	0.00050	0
Erosion strip	Steel	1	0.00164	0
LE reinforcement	IM7	1	0.00782	0

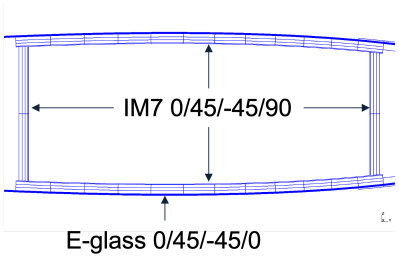
displays the geometry of the cross section. Figures 4.12b to 4.12d show detail views of the trailing edge, box spar, and leading edge, respectively. Figure 4.12e shows the mesh produced by PreVABS using the built-in automesher with a minimum size of 0.010 ft.



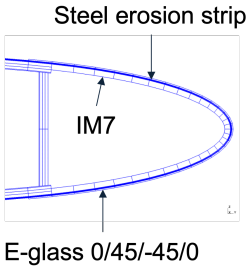
(a) Geometry



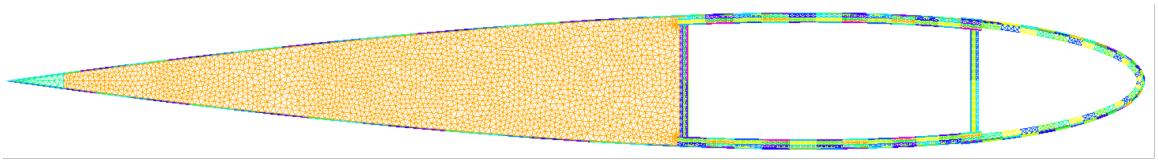
(b) Trailing edge detail



(c) Spar detail



(d) Leading edge detail

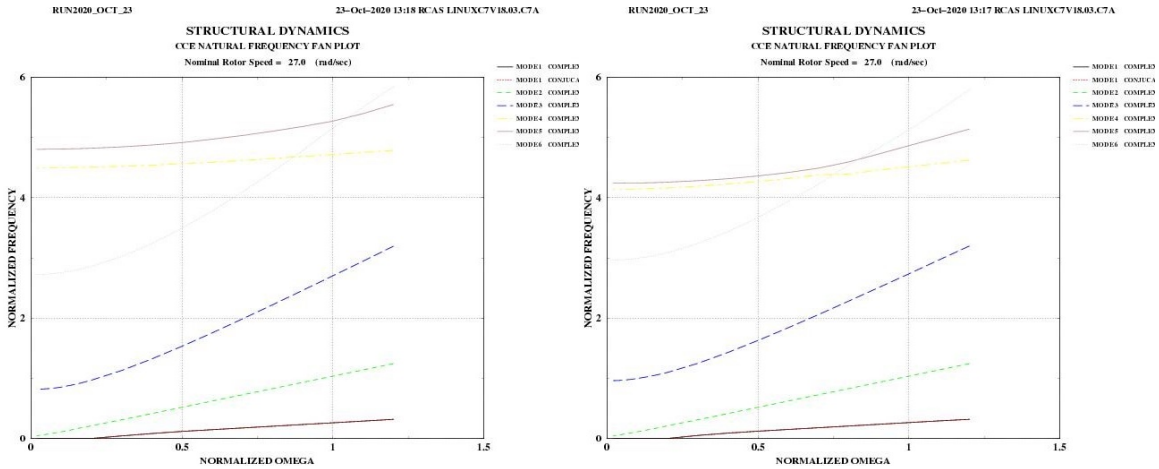


(e) Mesh

Figure 4.12: Representations of the PreVABS+VABS rotor blade cross section model.

After creating the PreVABS+VABS model, its mass and stiffness matrices were trans-

ferred to the RCAS model. A fan plot was created to compare the original NLB formulation from the RCAS training example file[2] and the new GECB formulation. The results are presented in Figure 4.13.



(a) Nonlinear beam (b) Geometrically exact composite beam

Figure 4.13: Fan plot comparison of original NLB formulation to new GECB formulation.

Note that most of the mode frequencies produced by the NLB (see Figure 4.13a) are very similar to those of the GECB (see Figure 4.13b). The frequency of the third mode has increased slightly at low rotor speed, while the frequencies of the fifth and sixth modes have decreased slightly across the entire range. The similarity suggests that the PreVABS+VABS cross section model accurately reproduces the characteristics of the UH-60A rotor blade, even though the outer mold line is not representative. Most importantly, the structural stability of the RCAS model's main rotor will not be affected by transitioning from the NLB to the GECB.

The files used to define the PreVABS+VABS model are included in Appendix A.2. Table B.32 describes the mapping between the PreVABS+VABS input/output variables and the OpenMDAO variables. The OpenMDAO variables will be described further in Section 4.4.4.

[2]It is assumed that the RCAS training example file is also intended to represent a UH-60A rotor system, although the elastic definition has been simplified to comply with data restrictions.

4.4.4 OpenMDAO Modules and Supporting Tools

In addition to the three primary software tools describe previously, the MDA requires a number of other supporting tools to maintain consistency between the different models. Each of these tools is implemented directly in Python as a subclass of the OpenMDAO `ExplicitComponent` class. Finally, these tools are combined into a OpenMDAO `Group`, which handles the execution order and variable mapping. This section will describe each of the supporting tools and the group in turn.

4.4.4.1 Blade Ballast Calculator

The PreVABS+VABS rotor blade cross section model does not include any ballast. In order to ensure that the rotor blade remains aerodynamically stable in the presence of design changes, the CG must remain ahead of the aerodynamic center (AC). The blade ballast calculator analyzes the mass matrix calculated by VABS and calculates a necessary ballast mass and location to maintain the CG of the baseline rotor blade design.

The VABS mass matrix takes the form of Equation (4.24) [113]:

$$M = \begin{bmatrix} \mu & 0 & 0 & 0 & \mu x_{m_3} & -\mu x_{m_2} \\ 0 & \mu & 0 & -\mu x_{m_3} & 0 & 0 \\ 0 & 0 & \mu & \mu x_{m_2} & 0 & 0 \\ 0 & -\mu x_{m_3} & \mu x_{m_2} & i_{22} + i_{33} & 0 & 0 \\ \mu x_{m_3} & 0 & 0 & 0 & i_{22} & i_{23} \\ -\mu x_{m_2} & 0 & 0 & 0 & i_{23} & i_{33} \end{bmatrix} \tag{4.24}$$

where μ is the mass per unit length, x_{m_2} and x_{m_3} are the center of mass location along the second and third axes, and i_{22}, i_{23}, i_{33} are the moments of inertia per unit length.

The blade ballast calculator first extracts these values, then calculates the necessary

ballast mass and location using Equations (4.25) to (4.27):

$$\mu_b = \mu_f - \mu_0 \tag{4.25}$$

$$x_{m_2,b} = \frac{x_{m_2,f}\mu_f - x_{m_2,0}\mu_0}{\mu_b} \tag{4.26}$$

$$x_{m_3,b} = \frac{x_{m_3,f}\mu_f - x_{m_3,0}\mu_0}{\mu_b} \tag{4.27}$$

where the b subscript denotes the ballast, the 0 subscript denotes the original values of each parameter read from the VABS output file, and the f subscript denotes the final desired values of each parameter. The implementation includes a check to ensure that Equation (4.25) does not return a negative value for the ballast mass.

Finally, the moments of inertia are reconstructed using Equations (4.28) to (4.30):

$$i_{22,f} = i_{22,0} + \mu_b x_{m_3,b}^2 \tag{4.28}$$

$$i_{33,f} = i_{33,0} + \mu_b x_{m_2,b}^2 \tag{4.29}$$

$$i_{23,f} = i_{23,0} + \mu_b x_{m_2,b} x_{m_3,b} \tag{4.30}$$

and a new VABS mass matrix with the effects of the ballast included is assembled using Equation (4.24).

The code defining the blade ballast calculator is included in Appendix C.2. The OpenMDAO variable names of the inputs and ouptuts are listed in Table C.1.

4.4.4.2 Blade Weight Calculator

The blade weight calculator calculates the complete mass of all four rotor blades given the mass per unit length of the cross section. The equation is given by

$$m_B = n_B \mu \ell_B + m_H \tag{4.31}$$

where m_B is the blade mass, n_B is the number of blades, ℓ_B is the length of a single blade, and m_H is the mass of the rigid beam used to model hinge offset. In the generic SMR helicopter model, $n_B = 4$, $\ell_B = 25.3\,\mathrm{ft}$, and $m_H = 0.122\,\mathrm{slug}$.

4.4.4.3 Mass Calculator

The mass calculator module is responsible for ensuring that the mass and inertia parameters of the helicopter components are consistent between the NDARC and RCAS models. It also calculates the payload CG required to maintain the specified vehicle CG given the payload weight and vehicle gross weight.

The module makes use the component weights and locations defined by the NDARC model. The inertial properties of each component are estimated by idealizing that component as a simple three-dimensional object. For minor components, inertia is neglected.

This module calculates the inertia of an ellipsoid using Equations (4.32) to (4.35):

$$I_{xx} = \frac{1}{5} m \left(a^2 + b^2\right) \tag{4.32}$$

$$I_{yy} = \frac{1}{5} m \left(a^2 + c^2\right) \tag{4.33}$$

$$I_{zz} = \frac{1}{5} m \left(b^2 + c^2\right) \tag{4.34}$$

$$I_{xy} = I_{xz} = I_{yz} = 0 \tag{4.35}$$

where a, b, and c are the semi-major axes of the ellipsoid along the z, y, and x axes, respectively.

The inertia of a cylinder oriented along the x axis is modeled using Equations (4.36) to (4.39):

$$I_{xx} = \frac{1}{2} m R^2 \tag{4.36}$$

$$I_{yy} = \frac{1}{12} m \left(L^2 + 3R^2\right) \tag{4.37}$$

$$I_{zz} = \frac{1}{12} m \left(L^2 + 3R^2\right) \tag{4.38}$$

$$I_{xy} = I_{xz} = I_{yz} = 0 \tag{4.39}$$

where R and L are the radius and length of the cylinder, respectively. The equations can easily be modified for cylinders oriented along the y and z axes.

The inertial properties of a rectangular prism are given by Equations (4.40) to (4.43):

$$I_{xx} = \frac{1}{12} m \left(L^2 + H^2\right) \tag{4.40}$$

$$I_{yy} = \frac{1}{12} m \left(W^2 + H^2\right) \tag{4.41}$$

$$I_{zz} = \frac{1}{12} m \left(L^2 + W^2\right) \tag{4.42}$$

$$I_{xy} = I_{xz} = I_{yz} = 0 \tag{4.43}$$

where L, W, and H are the length, width, and height of the rectangular prism, respectively. The prism is assumed to be oriented with width on the x axis, length on the y axis, and height on the z axis.

Finally, a flat disc component can be modeled using Equations (4.44) to (4.47):

$$I_{xx} = \frac{1}{2} m R^2 \tag{4.44}$$

$$I_{yy} = \frac{1}{4} m R^2 \tag{4.45}$$

$$I_{zz} = \frac{1}{4} m R^2 \tag{4.46}$$

$$I_{xy} = I_{xz} = I_{yz} = 0 \tag{4.47}$$

where the flat disc is assumed to be oriented along the x axis.

The exact inertia model used for each RCAS component is listed in Table 4.7. This table also describes the mapping between NDARC systems and RCAS components.

Table 4.7: Inertia models for the generic SMR helicopter model.

RCAS component	NDARC component(s)[a]	Inertia model	Inertia parameters[b]
Fuel	Fuel	None	None
Fuselage	Fuselage group, systems and equipment, vibration reduction, fuel systems	Ellipsoid	$a = 3\,\text{ft}$, $b = 4\,\text{ft}$, $c = 19.97\,\text{ft}$
Useful load	Fixed useful load	None	None
Gear	Alighting gear	None	None
Main rotor blade ($\times 4$)	Main rotor blades	None[c]	None
Main rotor hub	Main rotor hub and hinge	Flat disc	$R = 1.25\,\text{ft}$
Tail rotor blade ($\times 4$)	Tail rotor[d]	Rectangular prism	$W = 4.8\,\text{ft}$, $L = 0.754\,\text{ft}$, $H = 0.090\,\text{ft}$
Tail rotor hub	Tail rotor[d]	Flat disc	$R = 1.2\,\text{ft}$
Horizontal stabilizer	Horizontal stabilizer	Rectangular prism	$W = 14.66\,\text{ft}$, $L = 2.93\,\text{ft}$, $H = 0.35\,\text{ft}$
Vertical stabilizer	Vertical stabilizer	Rectangular prism	$W = 8.55\,\text{ft}$, $L = 0.71\,\text{ft}$, $H = 3.54\,\text{ft}$
Engine group	Drive system, engine structure, engine system	Cylinder	$L = 9.985\,\text{ft}$, $R = 2\,\text{ft}$
Payload	Payload	None	None

[a] Masses from several different NDARC components can be combined for a single RCAS component. The inertia model uses the sum of these masses.

[b] The inertial model of each component is defined using the axes of its reference frame in the RCAS model.

[c] The inertial properties of the main rotor blade are captured by the nonlinear beam model in RCAS.

[d] The tail rotor blades are assumed to sum to 55% of the tail rotor system, with the hub and shaft making up the other 45%.

The total mass of the vehicle is calculated using Equation (4.48):

$$m_t = \sum_{i=1}^{n} m_i \tag{4.48}$$

where n is the number of components. The CG of the entire system can then be found using Equations (4.49) to (4.51):

$$x_{\mathrm{CG}} = \frac{\sum_{i=1}^{n} m_i x_i}{m_t} \tag{4.49}$$

$$y_{\mathrm{CG}} = \frac{\sum_{i=1}^{n} m_i y_i}{m_t} \tag{4.50}$$

$$z_{\mathrm{CG}} = \frac{\sum_{i=1}^{n} m_i z_i}{m_t} \tag{4.51}$$

The mass calculator module calculates the necessary payload CG position for a prescribed vehicle CG position. This is accomplished by first calculating the total mass and CG without the payload included. Then, the station line (x position) of the payload is determined using

$$x_{\mathrm{PL}} = \frac{x_{\mathrm{CG}} \left(m_0 + m_{\mathrm{PL}}\right) - x_{\mathrm{CG},0} m_0}{m_{\mathrm{PL}}} \tag{4.52}$$

where the PL subscript denotes quantities relating to the payload and the 0 subscript denotes quantities related to the vehicle not including the payload. This module does not modify the butt line (y) or water line (z) of the payload or the combined CG position.

The code needed to run the mass calculator module is given in Appendix C.3. Table C.1 lists the inputs and outputs of the mass calculator module.

4.4.4.4 Pre-RCAS Modules

Due to differences in the way the NDARC and RCAS models are defined, a number of simple tools are used to convert specific variables into the correct format.

First, the *ROC Negatizer* makes the rate of climb negative. Because RCAS uses a z-down reference frame to define trimmed flight conditions, a negative ROC corresponds to

a positive altitude rate.

The *Payload Negatizer* makes the payload CG station line negative. In the NDARC model, station line is positive towards the aft of the helicopter, but the RCAS model uses an x-forward reference frame for the fuselage, so station line is positive towards the front of the helicopter. Both models use the rotor hub as the reference line, so no additional conversions are needed.

Next, the *Turn Radius Calculator* is used to calculate the turn radius given a turn rate, which RCAS needs to define a steady state turn condition. This is calculated according to Equation (4.53):

$$R = \begin{cases} \frac{V}{|\omega|}, & \text{if } \omega \neq 0 \\ 0, & \text{if } \omega = 0 \end{cases} \tag{4.53}$$

where R is turn radius.

NDARC calculates pilot control positions required for trimmed flight, but in certain extreme flight conditions it may be difficult to initialize the RCAS trim solution with those control positions. It is easier to initialize RCAS with less extreme control positions and allow RCAS to trim to the model to the correct solution. To accomplish this, the *Control Damper* multiplies the collective, lateral cyclic, longitudinal cyclic, and pedal positions predicted by NDARC by a factor of 0.75.

4.4.4.5 MDA Group

The MDA Group combines the previously discussed software programs, wrappers, and supporting modules into one location. It defines a number of independent variables to enable simple definition of the flight condition. The group also controls the execution order of the different elements of the multidisciplinary analysis. Recall overview of the MDA presented previously in Figure 4.9.

The execution order is as follows:

1. The PreVABS+VABS model is initialized and executed. This model outputs the blade

inertial and elastic properties. If any blade design variables have been modified, they will be incorporated into this step.

2. The blade ballast calculator determines the necessary amount and position of ballast to maintain the aerodynamic stability. A new VABS mass matrix is constructed to account for the ballast.
3. The blade weight calculator calculates the mass of a main rotor blade given the mass per unit length of the cross section.
4. NDARC is initialized and executed. This first execution of NDARC is intended to account for potential changes in the rotor blade weight; the weights of other systems, such as the main rotor hub, may increase or decrease as a result. Helicopter component weights and positions are read from the outputs.
5. The mass calculator reads the component weights and positions and calculates the payload CG location, the system CG location, and the component moments of inertia.
6. NDARC is initialized and executed a second time using the updated CG locations from the mass calculator. The user-defined flight condition is simulated and the trimmed control positions and atmospheric properties are returned to OpenMDAO.
7. The pre-RCAS modules are executed to prepare the RCAS run.
8. RCAS is initialized and executed. The RCAS model takes component mass and inertia inputs from NDARC and the mass calculator, trimmed control positions from NDARC, atmospheric conditions from NDARC, and rotor blade mass and stiffness matrices from VABS and the blade ballast calculator. RCAS returns a complete set of forces and moments on every blade station at every time step in the trimmed, converged periodic solution.

Thus, a complete field of rotor blade loads can be automatically calculated for every flight condition listed in Table 4.1, or any other arbitrary flight condition that can be defined by those six variables. The RCAS model uses 11 blade stations and 72 time steps per rotor blade revolution, so a complete MDA run produces 792 load cases. The code to implement

the MDA group is included in Appendix C.1.

After the MDA execution is complete, each load case is passed through the Pre-VABS+VABS model, now running in recovery mode, which returns a large number of stress tensors representing the internal 3D stress field in the rotor blade cross section. Two additional tools are used to process the stress tensor field into a form that is easier to interpret.

4.4.4.6 Von Mises Stress Calculator

As described in Section 4.4.1, each stress tensor is converted into a signed von Mises stress invariant, which is a scalar value.

The von Mises stress calculator calculates signed von Mises stress for each Gauss point on the cross section model, according to Equations (4.54) to (4.56):

$$S_{\mathrm{hs}} = \frac{1}{3}\left(S_{11} + S_{22} + S_{33}\right) \tag{4.54}$$

$$S_{\mathrm{vm}} = \sqrt{\frac{1}{2}\left[\left(S_{11} - S_{22}\right)^2 + \left(S_{22} - S_{33}\right)^2 + \left(S_{33} - S_{11}\right)^2\right] + 3\left(S_{12}^2 + S_{23}^2 + S_{13}^2\right)} \tag{4.55}$$

$$S_{\mathrm{vm},s} = \begin{cases} -S_{\mathrm{vm}}, & S_{\mathrm{hs}} < 0 \\ S_{\mathrm{vm}}, & S_{\mathrm{hs}} \geq 0 \end{cases} \tag{4.56}$$

where S_{hs} is the hydrostatic stress, S_{vm} is the von Mises stress, and $S_{\mathrm{vm},s}$ is the signed von Mises stress.

The von Mises stress calculator produces one stress field for each of the 792 load cases produced by the MDA. These calculations can apply to the entire stress field or to a single point of interest. The code to run the von Mises stress calculator is included in Appendix C.4. Table C.3 lists the inputs and outputs of this module.

4.4.4.7 *Stress Analyzer*

The stress analyzer performs cyclic analysis on the signed von Mises stress fields produced by the von Mises stress calculator. Because this research considers only steady-state flight conditions, a simple first-harmonic analysis is used instead of the complex cycle counting methods described in Section 2.2.1.4. This analysis assumes that the 1/rev components of the stress signal are the most important components and that higher-order frequencies can be neglected.

For each Gauss point, the maximum and minimum signed von Mises stress values in the cycle, S_{max} and S_{min}, are extracted. The amplitude and mean of the cycle is calculated using Equations (4.57) and (4.58):

$$S_{\text{amp}} = S_{\text{max}} - S_{\text{min}} \tag{4.57}$$

$$S_{\text{mean}} = S_{\text{min}} + \frac{S_{\text{amp}}}{2} \tag{4.58}$$

Then, the equivalent stress is calculated using the linear form of Goodman's relation (see Equation (2.4) and Figure 2.7). In Experiment 1a, the ultimate stress, S_u, is assumed to be 160 ksi. This value will be refined based on specific material properties in Experiment 2.

In this application, the stress analyzer produces 11 equivalent stress fields from the 792 signed von Mises stress fields. Equivalent stress is proportional to the rate at which fatigue damage is accumulated. Therefore, the point with highest equivalent stress is the most likely to be the first point of fatigue failure. The code required to run the stress analyzer is included in Appendix C.5. Table C.4 lists the inputs and outputs of this module.

4.4.5 Results and Analysis

In order to locate the critical fatigue point, each of the flight conditions in Table 4.1 was simulated using the previously-described MDA environment and the generic SMR helicopter model. The 792 load cases produced by RCAS were passed through the PreVABS+VABS

model in recovery mode, and the recovered stress tensor field was condensed into a signed von Mises equivalent stress scalar field using the von Mises stress calculator and stress analyzer modules.

In this section, the force, moment, and equivalent stress fields for each of the extreme flight conditions will be analyzed. Particular interest is paid to the spanwise, chordwise, and flapwise position of the maximum equivalent stress point in each case.

4.4.5.1 Case 0: Baseline

The baseline case represents a typical cruise flight condition for the generic SMR helicopter. This case is intended to serve as a reference to which the extreme flight conditions can be compared.

Figure 4.14 shows contour plots of the six blade load components over a complete revolution. A blade revolution is defined by the azimuth angle, ψ, which is 0° when the blade is pointing aft and 180° when the blade is pointing forwards. F_x, F_y, and F_z correspond to spanwise, chordwise, and flapwise force components, respectively. M_x is a twisting moment about the blade axis, M_y is a flapwise bending moment, and M_z is a bending moment in the lead–lag direction.

Note the strong centrifugal forces visible in the F_x plot. Centrifugal forces are caused by the rotation of the blade and, at each station, are proportional to the weight of the outboard segment. Thus, especially near the root, the blade experiences a large amount of tension.

The shear forces, F_y and F_z, are much smaller in magnitude than the centrifugal force. The chordwise shear forces near the root are likely caused by interactions with the pitch link and the lead–lag damper. The flapwise shear forces show interesting spanwise oscillations that are evidence of higher-order bending modes. The strong negative flapwise shear force near the root on the advancing blade in the region where $45° < \psi < 135°$ is likely due to interactions with the pitch link, which pulls the leading edge of the rotor blade down. Note that shear forces are expected to be small; in simpler aeroelastic models, rotor blade shear

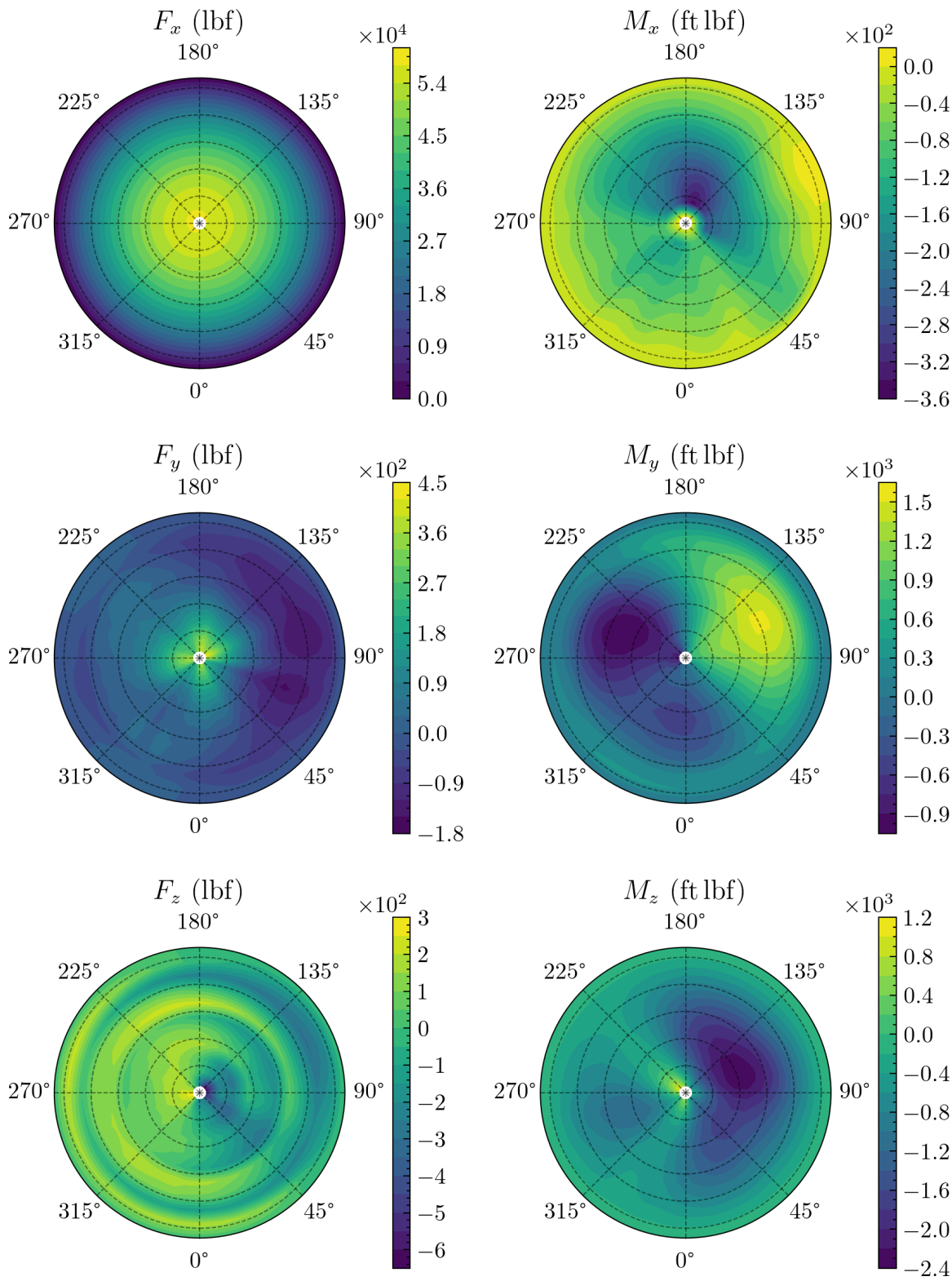


Figure 4.14: Rotor blade forces and moments for Experiment 1a, case 0.

stiffness and shear loads are neglected.

There is a corresponding strong negative (pitch down) blade moment near the root of the blade where $45° < \psi < 225°$, but pitching moments are small elsewhere. The generic SMR helicopter model uses a symmetric NACA0012 airfoil which does not produce significant pitch moments. The M_y and M_z plots portray strong flapwise and chordwise moments across the blade root and midspan. These moments are most likely a consequence of interactions between the aerodynamic environment and the rotor blade structure. A large lift distribution on the midspan of the advancing rotor blade produces significant positive flapwise bending moments, and the corresponding increase in drag produces significant negative chordwise bending moments. These effects are partially reversed for the retreating blade, which experiences weaker, but negative, flapwise bending moments and weaker negative chordwise bending moments.

Finally, note that all forces and moments decay to zero at the tip of the blade. This is expected as a rotor blade is essentially a fixed–free beam, and the tip loss model in RCAS ensures that no aerodynamic forces are generated exactly at the rotor blade tip.

Figure 4.15 plots the spanwise variation of $S_{\mathrm{eq,max}}$ and its chordwise and flapwise position on the cross section. The highest value of S_{eq} occurs at station 1, the root of the rotor blade, which is located 1.25 ft outboard of the center of the shaft. The value of S_{eq} at this point is 40.04 ksi.

Note that $S_{\mathrm{eq,max}}$ first drops, then rises slightly throughout the midspan of the rotor blade, then drops again to zero at the tip. This is reflective of the chordwise and flapwise bending moments seen in Figure 4.14. Figure 4.16 plots the full S_{eq} field on the rotor blade cross section. The $S_{\mathrm{eq,max}}$ point is highlighted in red.

There are essentially two regions of elevated S_{eq} on the cross section. The first is the auxiliary (forward) web of the box spar. Recall from Section 4.4.3.3 that the box spar is responsible for carrying most of the blade loads and is composed of the composite material IM7. There is another elevated stress region on low pressure surface near the leading edge

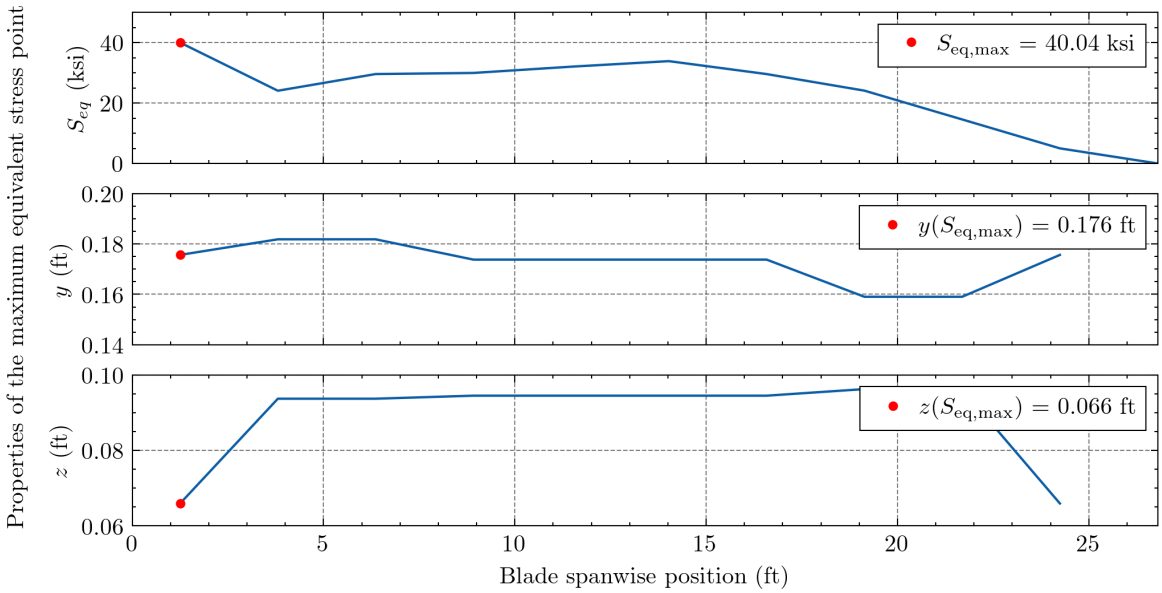


Figure 4.15: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 0.

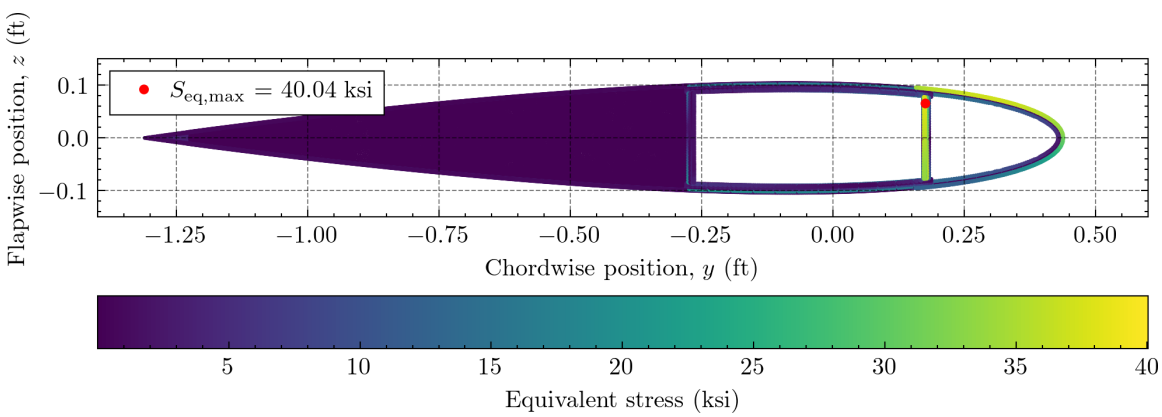


Figure 4.16: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 0.

of the blade. This corresponds to the location of the thin steel erosion strip, which protects the blade from dust and debris.

4.4.5.2 Case 1: Hover

Figure 4.17 plots the rotor blade loads for the hover case. Although the centrifugal forces are similar to the baseline case, all other loads vary significantly. Because the vehicle is stationary, there is little cyclic variation of airloads on the rotor blade. However, the rotor still produces some pitching and rolling moment to compensate for the CG position and tail

rotor thrust.

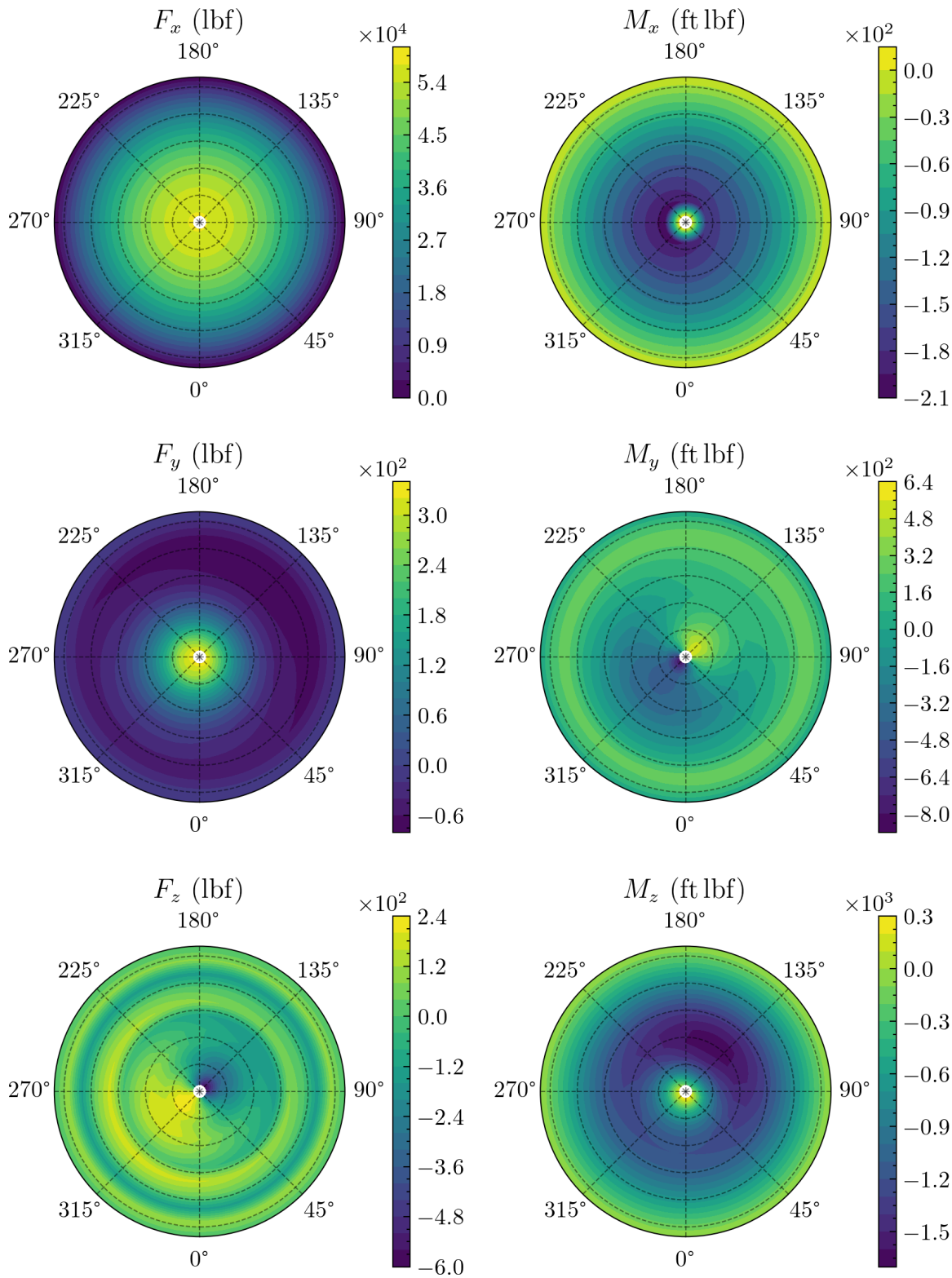


Figure 4.17: Rotor blade forces and moments for Experiment 1a, case 1.

The shear forces are also similar to the baseline case, albeit with slightly reduced amplitudes. The twisting, flapwise bending, and chordwise bending moments lose the characteristic 1/rev oscillations present in the baseline case, as there is no longer an "advancing" or "retreating" blade. The flapping and chordwise bending moments also have lower peak-to-peak amplitudes than the baseline case. Figure 4.18 shows the spanwise variation of $S_{\text{eq,max}}$.

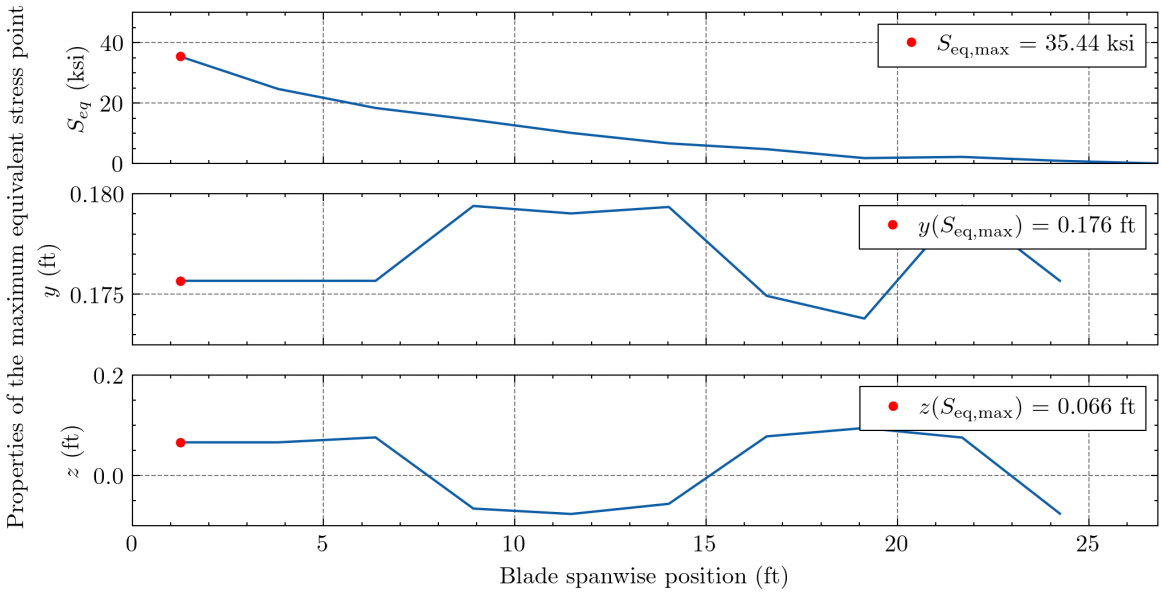


Figure 4.18: $S_{\text{eq,max}}$ and its location for Experiment 1a, case 1.

The peak stress point is still located at the root of the blade, and, at 35.44 ksi, is lower than the peak stress of the baseline case. However, the spanwise variation of stress changed dramatically. In this case, S_{eq} decreases throughout the rotor blade and no elevated stress is present in the midspan. Recall that S_{eq} is driven by both S_{amp} and S_{mean}. Although S_{mean} is still present, S_{amp} is nearly zero in the mid-span segment, as evidenced by Figure 4.17, resulting in a corresponding decrease in S_{eq}. Figure 4.19 plots the equivalent stress field at the root of the blade.

In Figure 4.19, the peak stress location is very similar to the baseline case, although the highest value of S_{eq} is lower. The region of elevated stress on the steel erosion strip is also present.

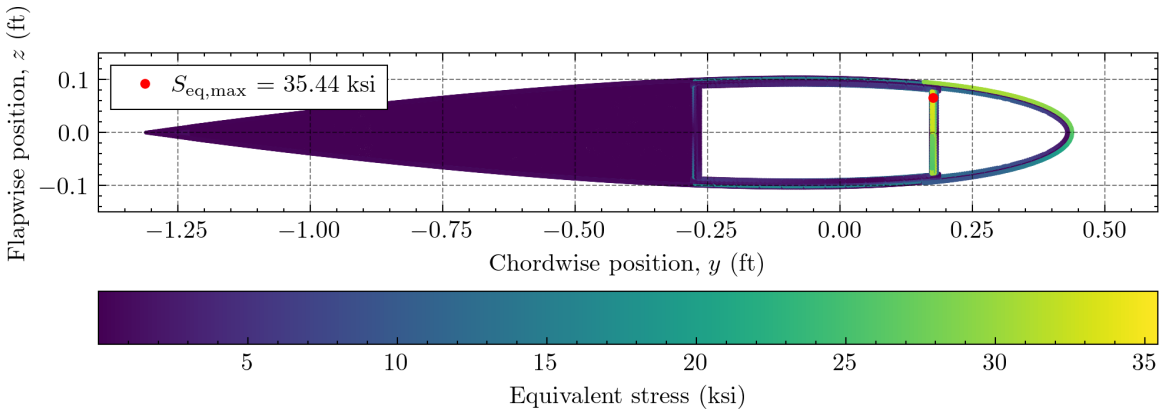


Figure 4.19: S_{eq} field at blade station 1 ($x = 1.25\,\text{ft}$) for Experiment 1a, case 1.

4.4.5.3 Case 2: High Speed

Figure 4.20 plots the rotor blade loads for the high speed case. Overall, the patterns of the rotor blade loads are similar to the baseline case, although peak-to-peak amplitudes are higher in all loads except centrifugal force.

In particular, there is a stronger pitch up (positive) moment near the tip on the advancing blade. The retreating blade experiences oscillations between positive and negative chordwise moments on the retreating blade, where $180° < \psi < 360°$. At high flight speeds, the area of the retreating blade near the root experiences a phenomenon known as *reverse flow* where the relative local wind speed is less than or equal to zero. This can have unpredictable aerodynamic effects, including flow separation and stall. The quasi-nonlinear aerodynamic model used by the RCAS model is capable of modeling steady-state reverse flow effects because the coefficients of the NACA0012 are defined for angles-of-attack from −180° to 180°.

Figure 4.21 plots the spanwise variation of $S_{\text{eq,max}}$. The pattern is very similar to the baseline case, albeit with notably higher S_{max} overall. At the root, $S_{\text{eq,max}} = 49.73\,\text{ksi}$. The complete equivalent stress field at the root is presented in Figure 4.22.

In this case, the maximum equivalent stress point is located near the leading edge of the blade on the low pressure surface. The exact position corresponds to the steel erosion strip.

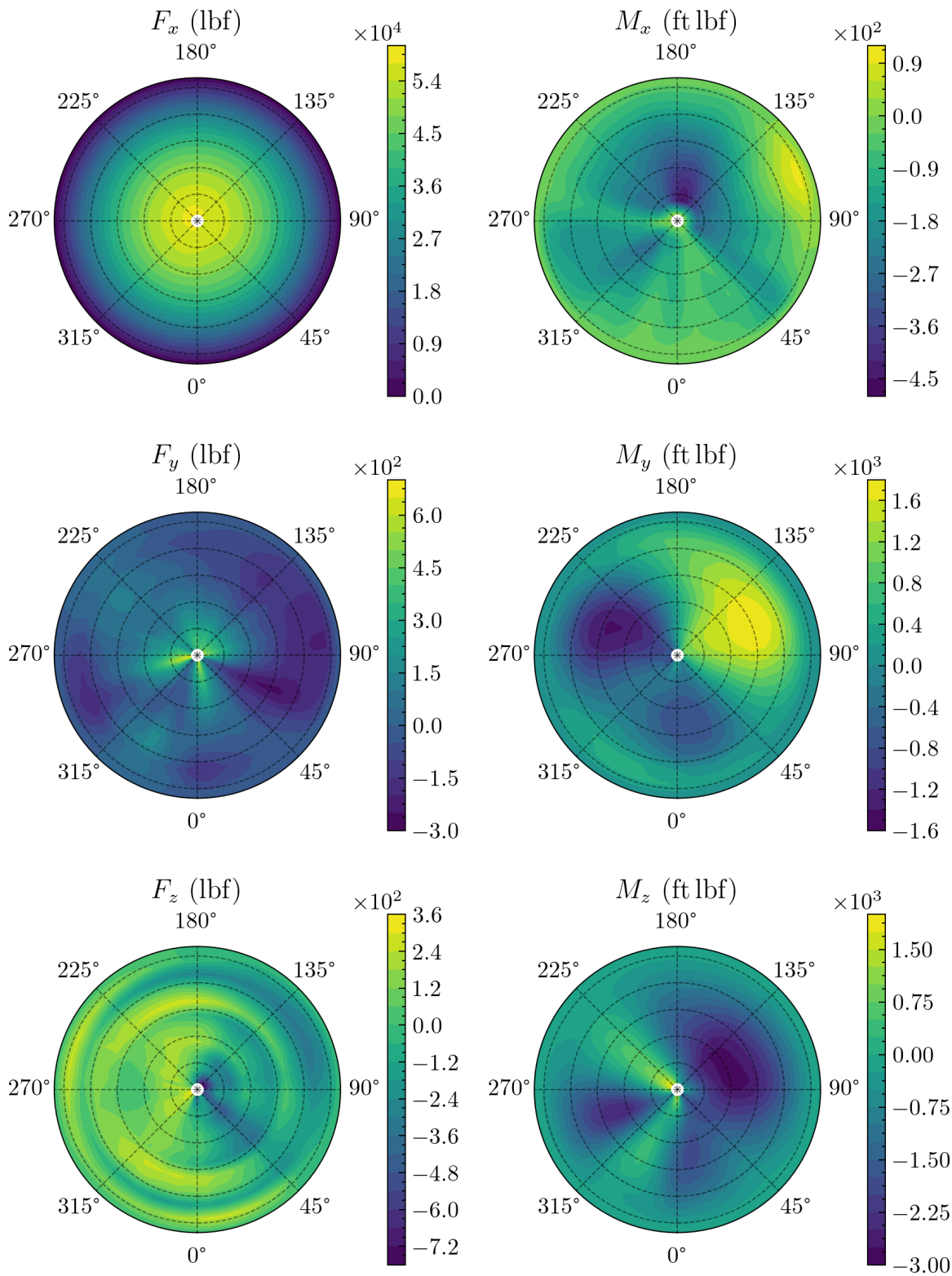


Figure 4.20: Rotor blade forces and moments for Experiment 1a, case 2.

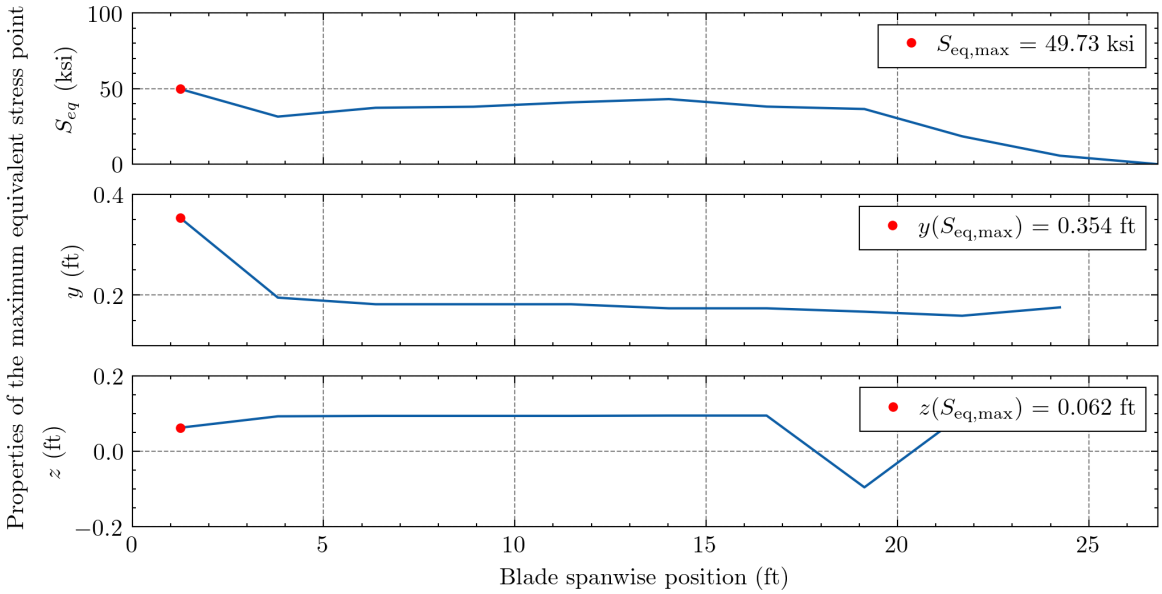


Figure 4.21: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 2.

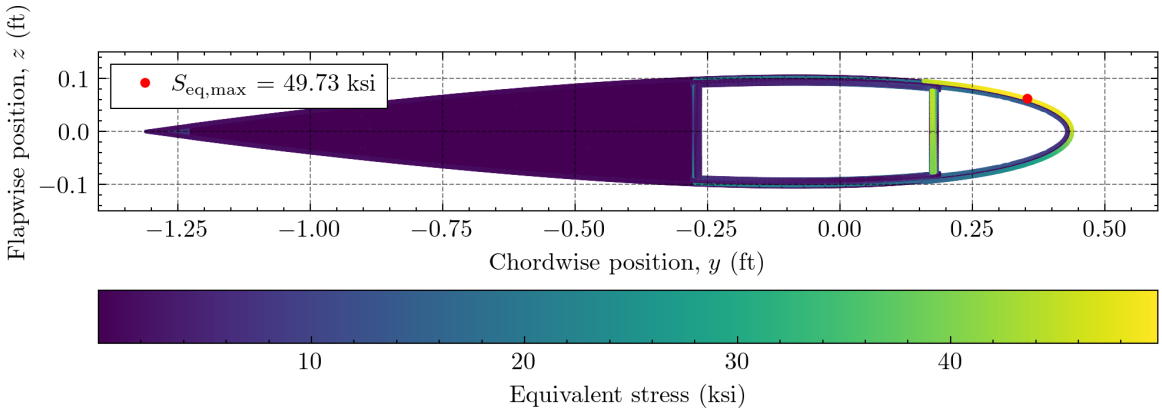


Figure 4.22: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 2.

It appears that the greater deformations of the rotor blade in the high speed flight condition cause the steel erosion strip, which has higher moduli of elasticity than the composite materials (see Table 4.5), to experience higher stress. However, note that the auxiliary web of the box spar also experiences high equivalent stress.

4.4.5.4 Case 3: Low Altitude

The rotor blade loads produced by the low altitude case are plotted in Figure 4.23. Overall, this case is nearly indistinguishable from the baseline case. The magnitude of the flapwise

shear forces are slightly lower, but the magnitudes of the pitching and chordwise bending moments are slightly higher than the baseline case. This is possibly related to higher air density at low altitudes increasing the aerodynamic loading on the rotor blade.

Figure 4.24 plots the spanwise variation of $S_{\mathrm{eq,max}}$ and its position. At 42.59 ksi, the peak equivalent stress is slightly higher than the baseline case, but the shape of the spanwise variation is otherwise similar.

Figure 4.25 presents the cross-sectional distribution of equivalent stress at the blade root. Other than the increased magnitude of $S_{\mathrm{eq,max}}$, this stress field is effectively identical to the baseline case.

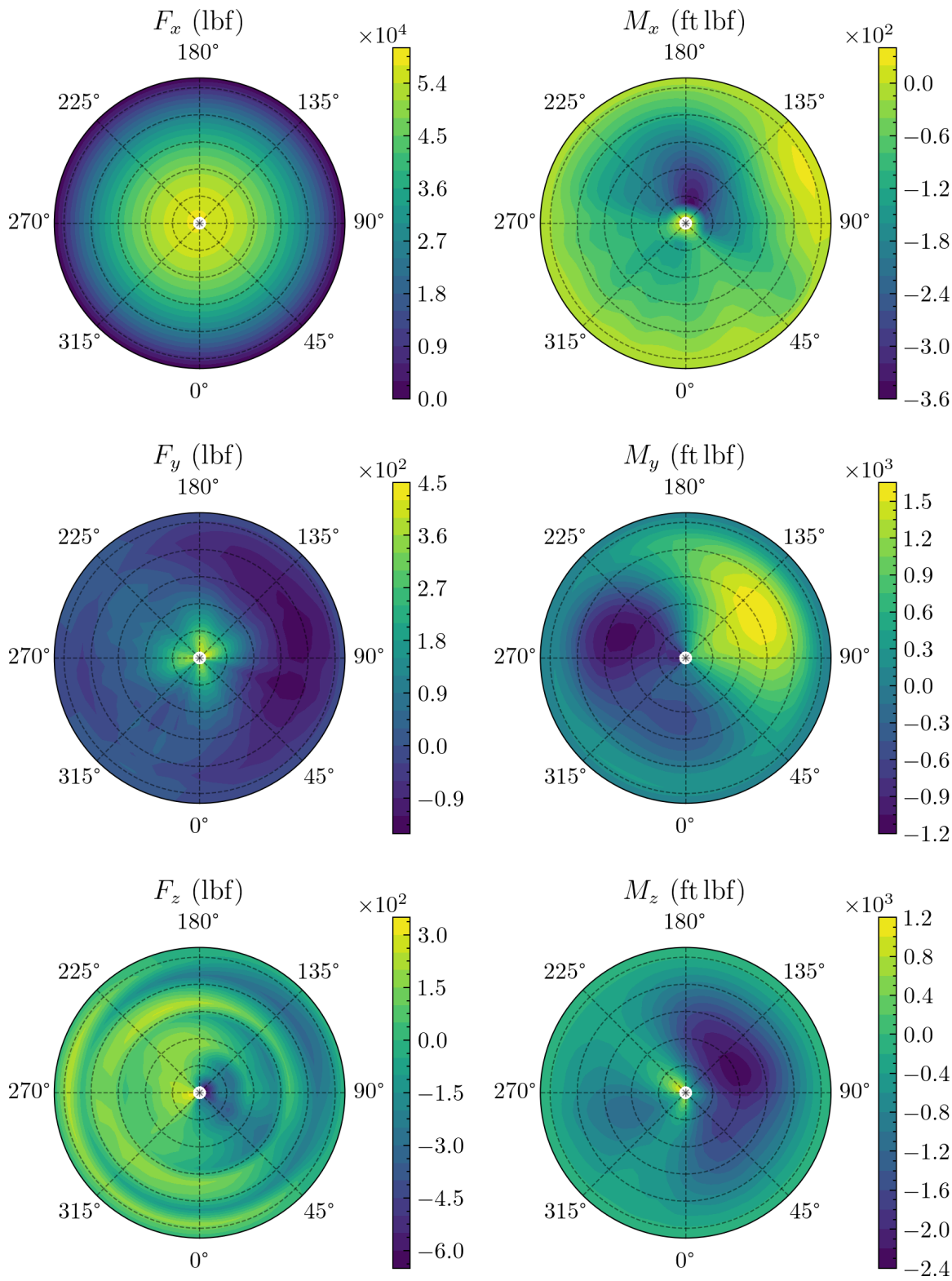


Figure 4.23: Rotor blade forces and moments for Experiment 1a, case 3.

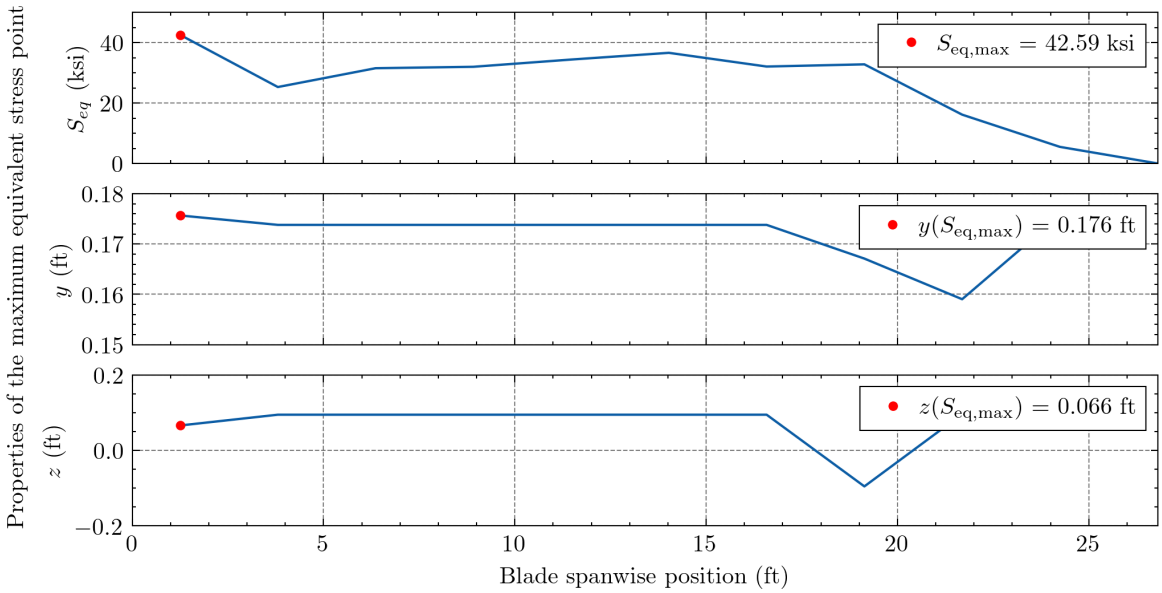


Figure 4.24: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 3.

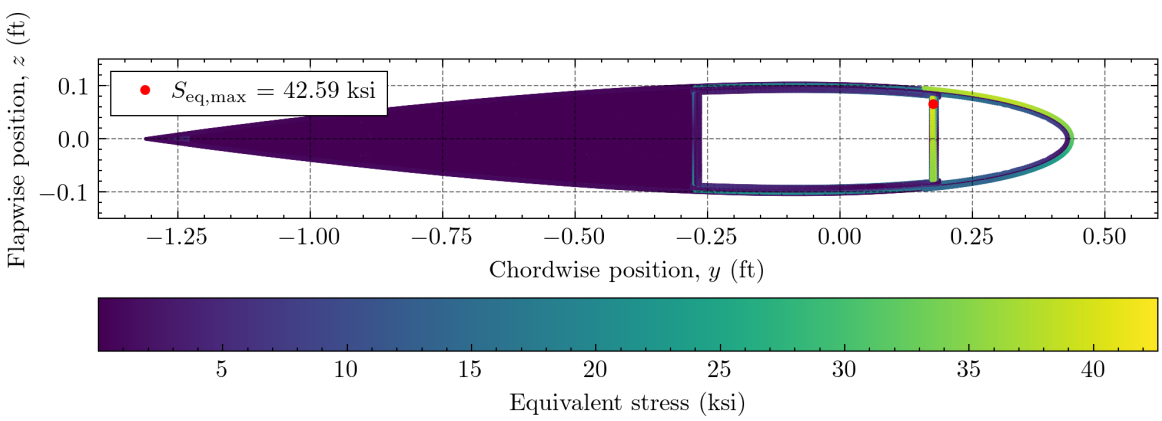


Figure 4.25: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 3.

4.4.5.5 Case 4: High Altitude

The high altitude case is similarly nearly identical to the baseline case. Figure 4.26 plots the rotor blade loads predicted for this case. The shear force magnitudes are slightly lower than the baseline case, but the magntitudes and cyclic variations in the pitching, chordwise bending, and flapwise bending moments are very nearly identical to the baseline case.

The spanwise variation of the maximum equivalent stress point, presented in Figure 4.27, is also similar to the baseline case. With a maximum equivalent stress of 36.80 ksi, this case is slightly less damaging than the baseline case. This is likely due to the lower air density at

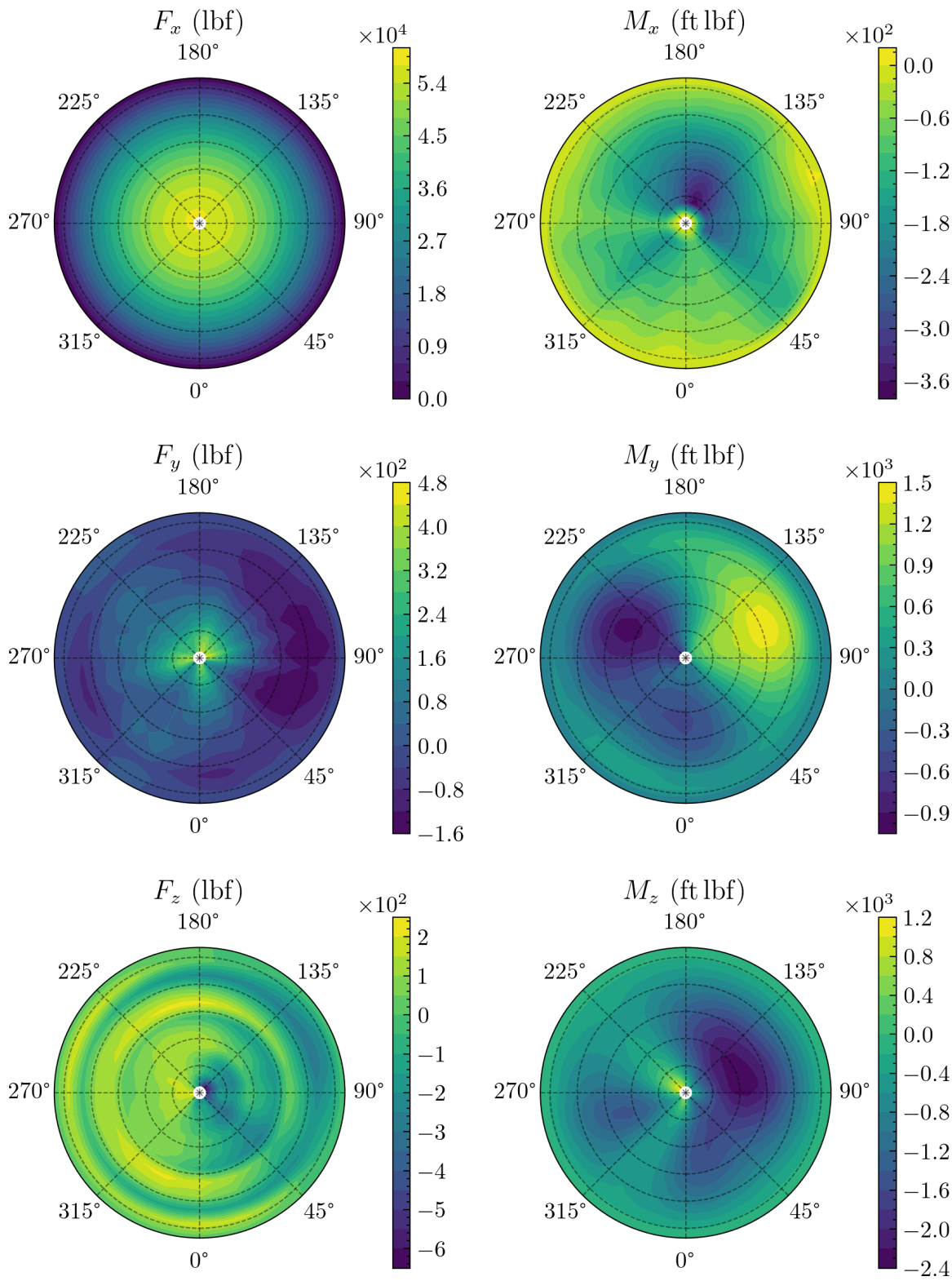


Figure 4.26: Rotor blade forces and moments for Experiment 1a, case 4.

higher altitudes, which decreases the overall aerodynamic loading on the rotor blade.

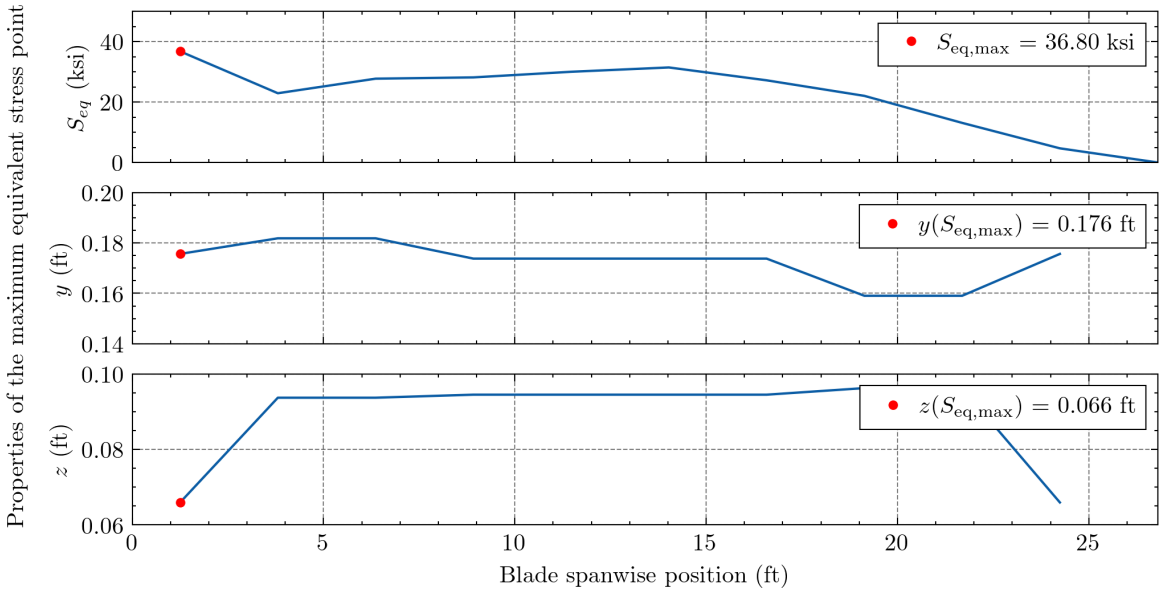


Figure 4.27: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 4.

Figure 4.28 displays the distribution of equivalent stress over the cross section. Again, other than the difference in magnitude, this stress field is very similar to the baseline case.

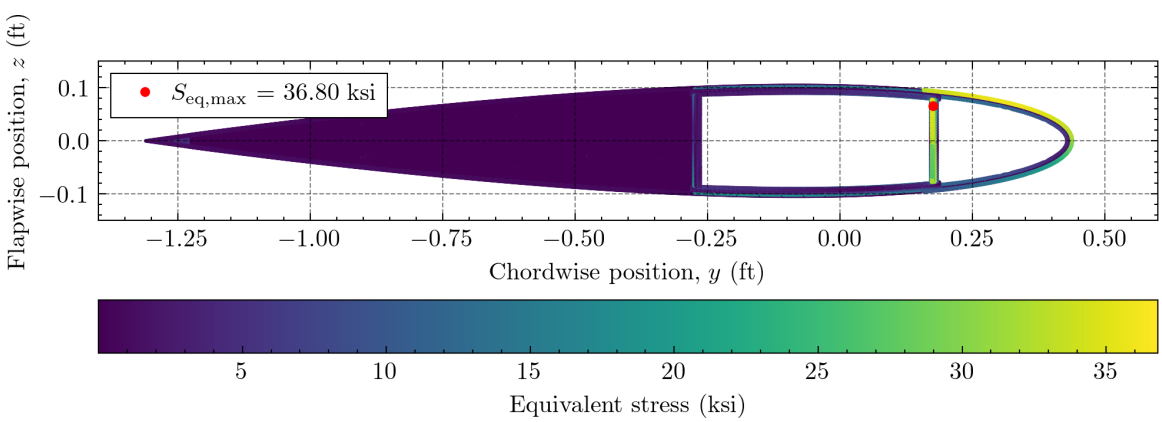


Figure 4.28: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 4.

4.4.5.6 Case 5: Low Weight

The rotor blade forces and moments for the low weight case are plotted in Figure 4.29. Compared to the baseline case, the chordwise shear force has less extreme negative peaks and the flapwise shear force have a more extreme positive peak at the blade root on the

retreating blade. This appears to be due to changes in pitch link forces required to maintain this flight condition.

The twisting moment has a more extreme pitch up (positive) moment at the root on the retreating blade. The flapwise bending moment has a greater negative moment at the root and midspan of the retreating blade, where $225° < \psi < 270°$. The chordwise bending moment has a lower magnitude overall than the baseline case. Figure 4.30 presents the spanwise variation of $S_{\mathrm{eq,max}}$.

Interestingly, despite the lower weight of the vehicle in this flight condition, the maximum equivalent stress, which is located at the root, is higher than the baseline case, at 47.49 ksi. This may be related to the higher peak in the flapwise shear force seen in Figure 4.29. However, it is important to remember that, due to the highly nonlinear nature of stress in composite materials, the stress field is somewhat abstracted from the rotor load distributions. For example, load components can interact in ways that are not intuitive: an increase in flapwise bending moment could actually decrease stress on the low pressure surface of the rotor blade because its compression counteracts the tension produced by centrifugal loading. In this case, decreases in internal blade loads due to the lower weight of the vehicle could actually contribute to an increase in stress at certain locations on the cross section.

Figure 4.31 plots the variation of S_{eq} across the cross section at the blade root. Although equivalent stress has increased on the auxiliary web of the box spar, the stress on the erosion strip appears to be slightly lower than the baseline case.

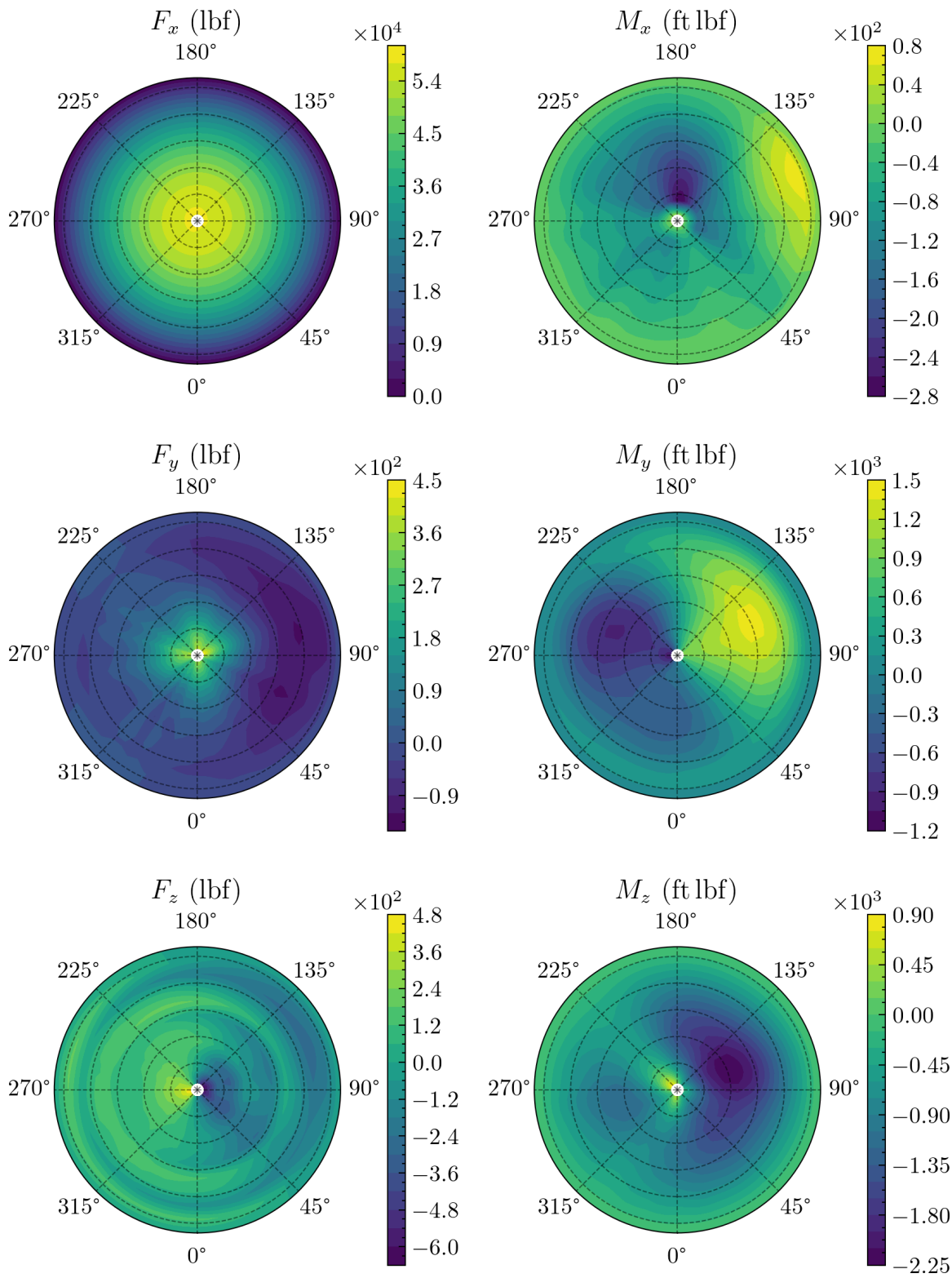


Figure 4.29: Rotor blade forces and moments for Experiment 1a, case 5.

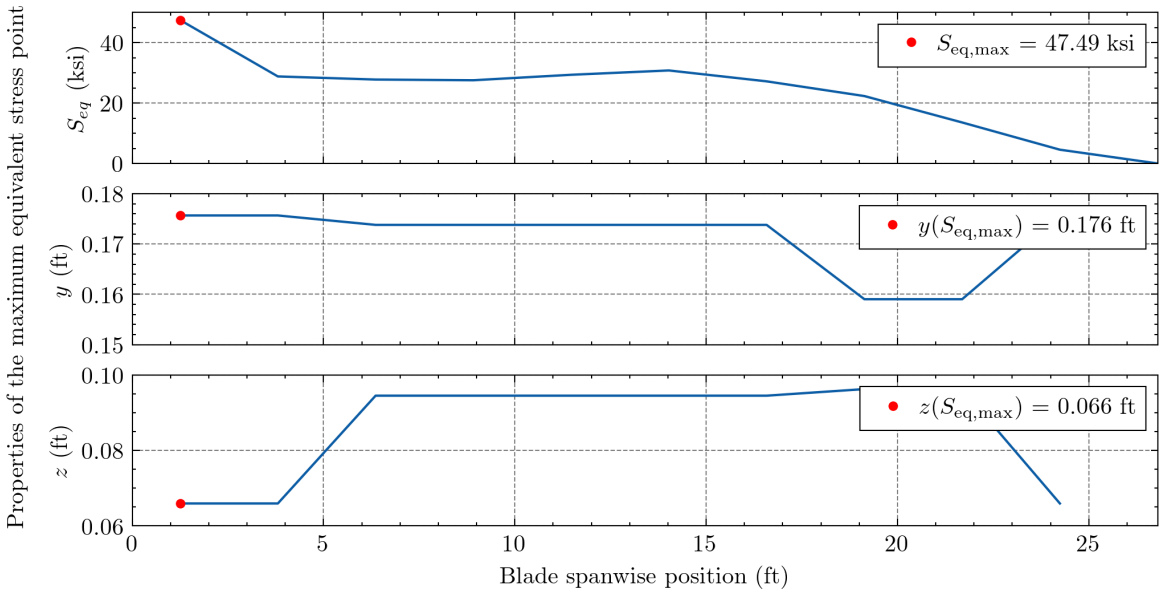


Figure 4.30: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 5.

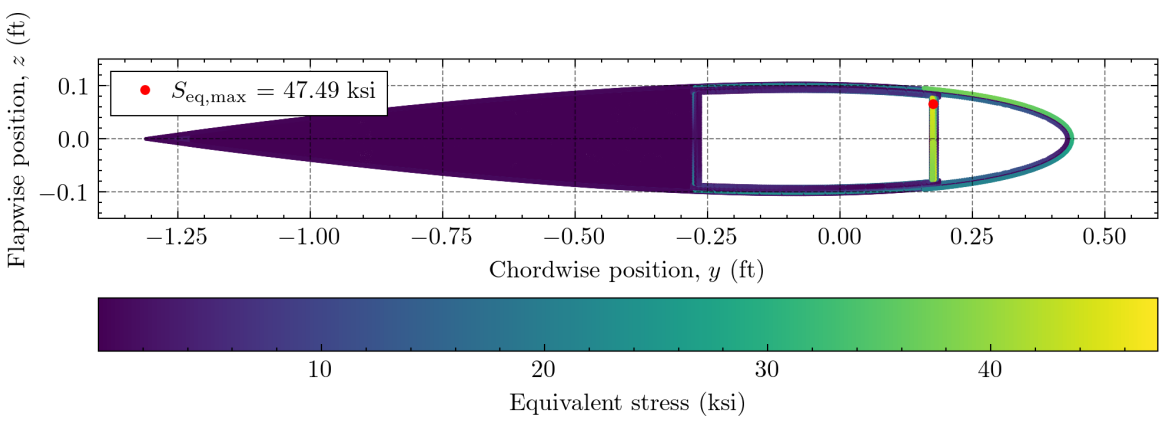


Figure 4.31: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 5.

4.4.5.7 Case 6: High Weight

Figure 4.32 plots the blade loads for the high weight case. Compared to the baseline case, the chordwise shear force has higher magnitude. There is also a significant difference in each of the moments.

The twisting moments have higher peak-to-peak amplitudes than the baseline case. In addition, there is a new region of strong pitch-down moment on the midspan of the retreating blade, where $270° < \psi < 315°$. The patterns of the flapwise bending moment are similar, but the retreating blade experiences are more negative moment where $225° < \psi < 270°$. In

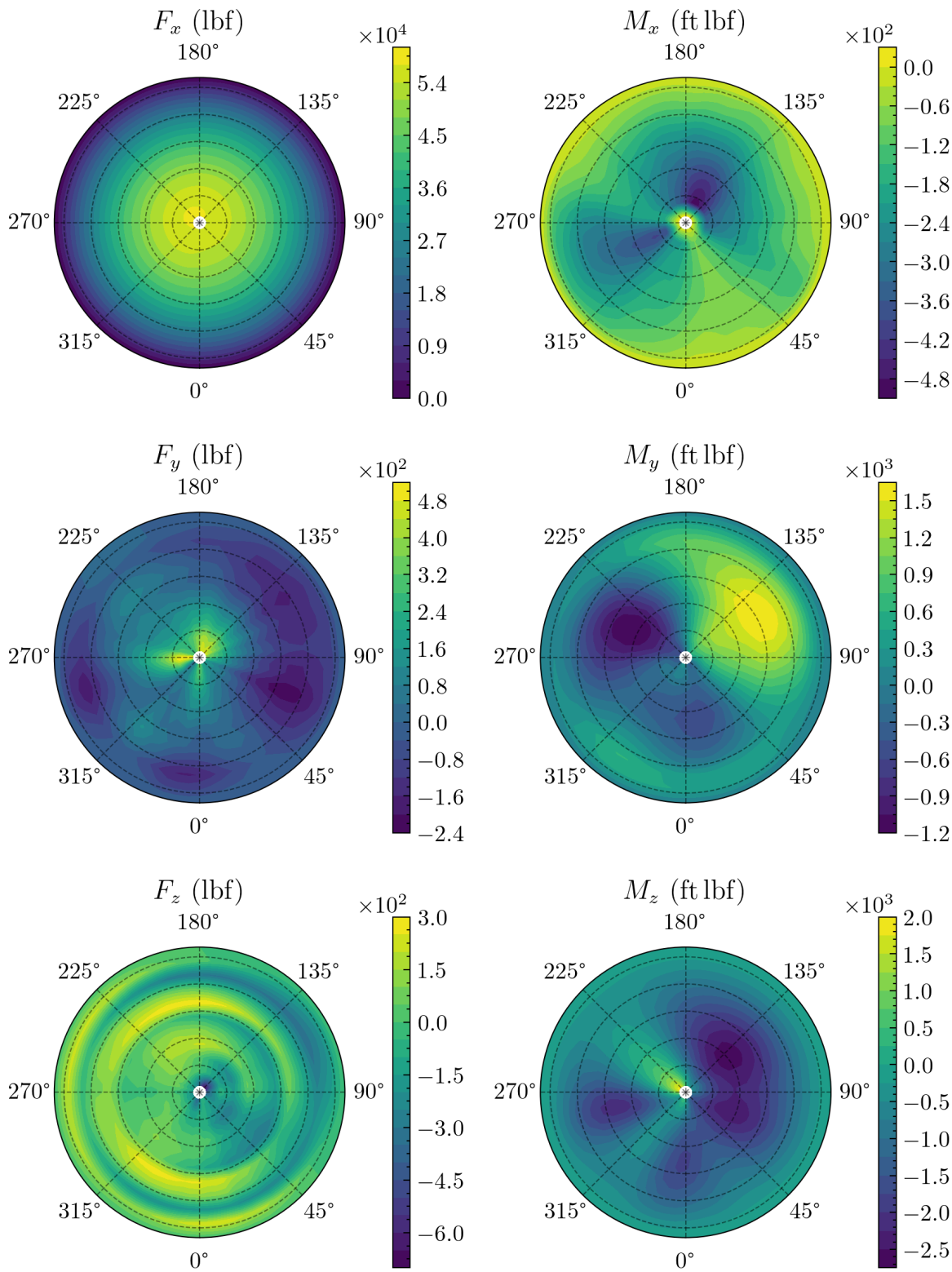


Figure 4.32: Rotor blade forces and moments for Experiment 1a, case 6.

the chordwise bending moment, Figure 4.32 shows a more extreme positive moment on the root of the retreating blade, and more extreme negative moments on the retreating blade midspan, particularly where $270° < \psi < 315°$. These patterns are consistent with a rotor system that must produce more thrust to lift the increased weight of the vehicle.

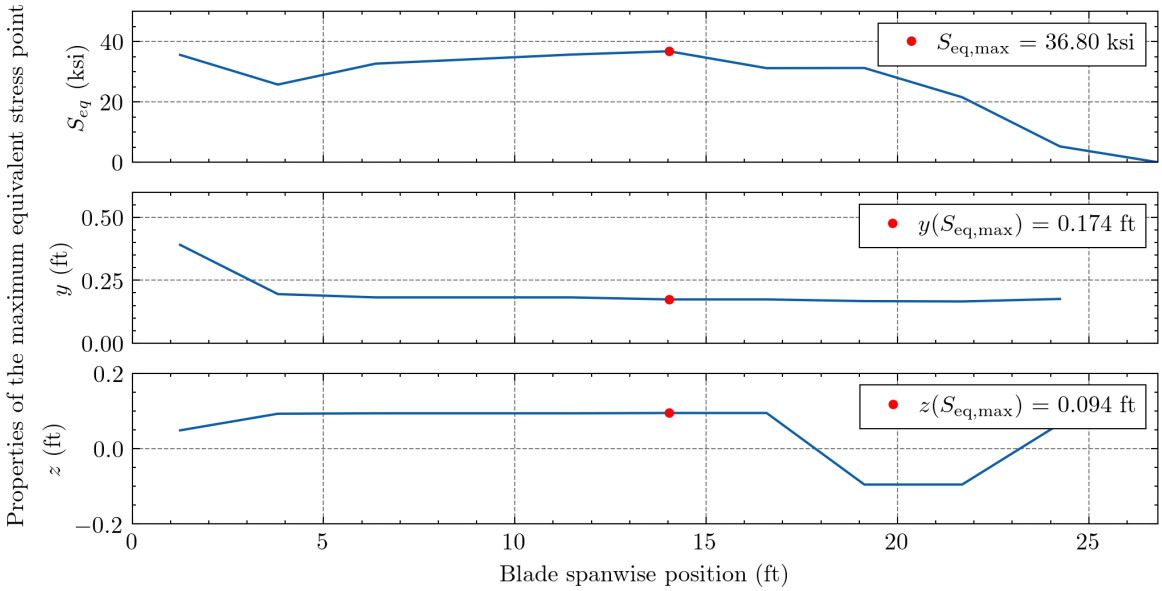


Figure 4.33: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 6.

Figure 4.33 plots the variation of $S_{\mathrm{eq,max}}$ across the rotor blade. Although the root stress remains high, the midspan of the blade actually experiences a slightly higher maximum equivalent stress than the root. However, at 36.80 ksi, this maximum stress is lower than the baseline case. Figure 4.34 presents the equivalent stress field at station 6 of the rotor blade, which is located 14.025 ft outboard of the center of the rotor shaft.

Figure 4.34 indicates a previously-unseen candidate for the critical fatigue point: the area of the low pressure surface directly above the auxiliary web of the box spar. This location corresponds to the steel erosion strip. There is a corresponding region of high stress on the high pressure surface directly below the auxiliary web. Interestingly, the box spar itself does not carry much stress. Because the centrifugal force and twisting moments are lower at the midspan, the box spar is not required to carry as much load.

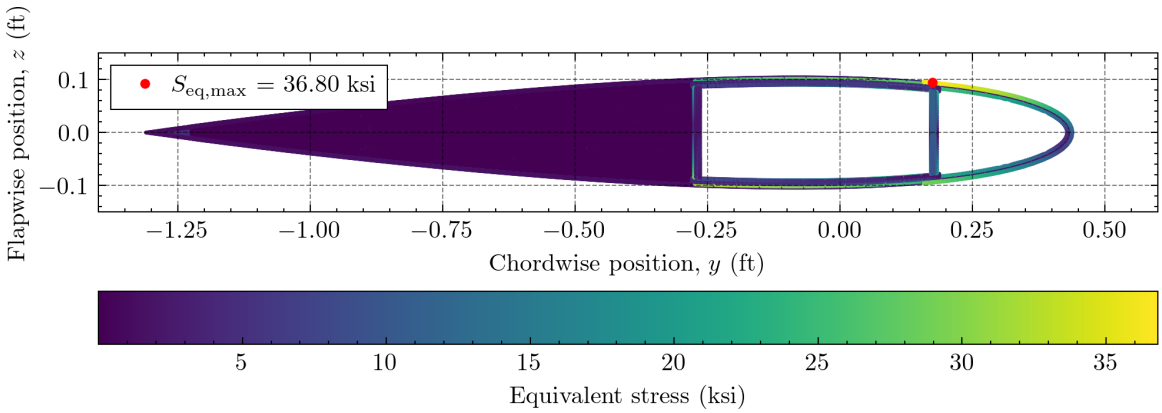


Figure 4.34: S_{eq} field at blade station 6 ($x = 14.025\,\text{ft}$) for Experiment 1a, case 6.

4.4.5.8 Case 7: Descent

The rotor blade forces and moments for the descent case are presented in Figure 4.35. In general, the flapwise and chordwise shear forces are similar to the baseline case but the peak-to-peak amplitudes are slightly lower. The advancing blade sees a higher pitch up moment on the midspan and tip of the advancing blade, specifically where $90° < \psi < 135°$.

The flapwise and chordwise bending moments are largely indistinguishable from the baseline case, except for less extreme positive chordwise moments on the root of the blade. The spanwise variations in the maximum equivalent stress point, presented in Figure 4.36, show a similar pattern to the high weight case, except $S_{\text{eq,max}}$ is much lower overall. The peak stress occurs at the midspan and is only 32.60 ksi.

Figure 4.37 shows a cross-sectional S_{eq} distribution that is also similar to the gross weight case, albeit at a lower magnitude. These results are unexpected because literature review (see Section 1.5) suggests that descent is one of the more damaging conditions in terms of material fatigue. Because the RCAS model uses BEMT inflow and does not include a wake model, the unique interactions between the main rotor wake and the rotor blades typically seen at negative rates of climb are not modeled. Thus, the current implementation of the MDA will not produce completely accurate results in descent.

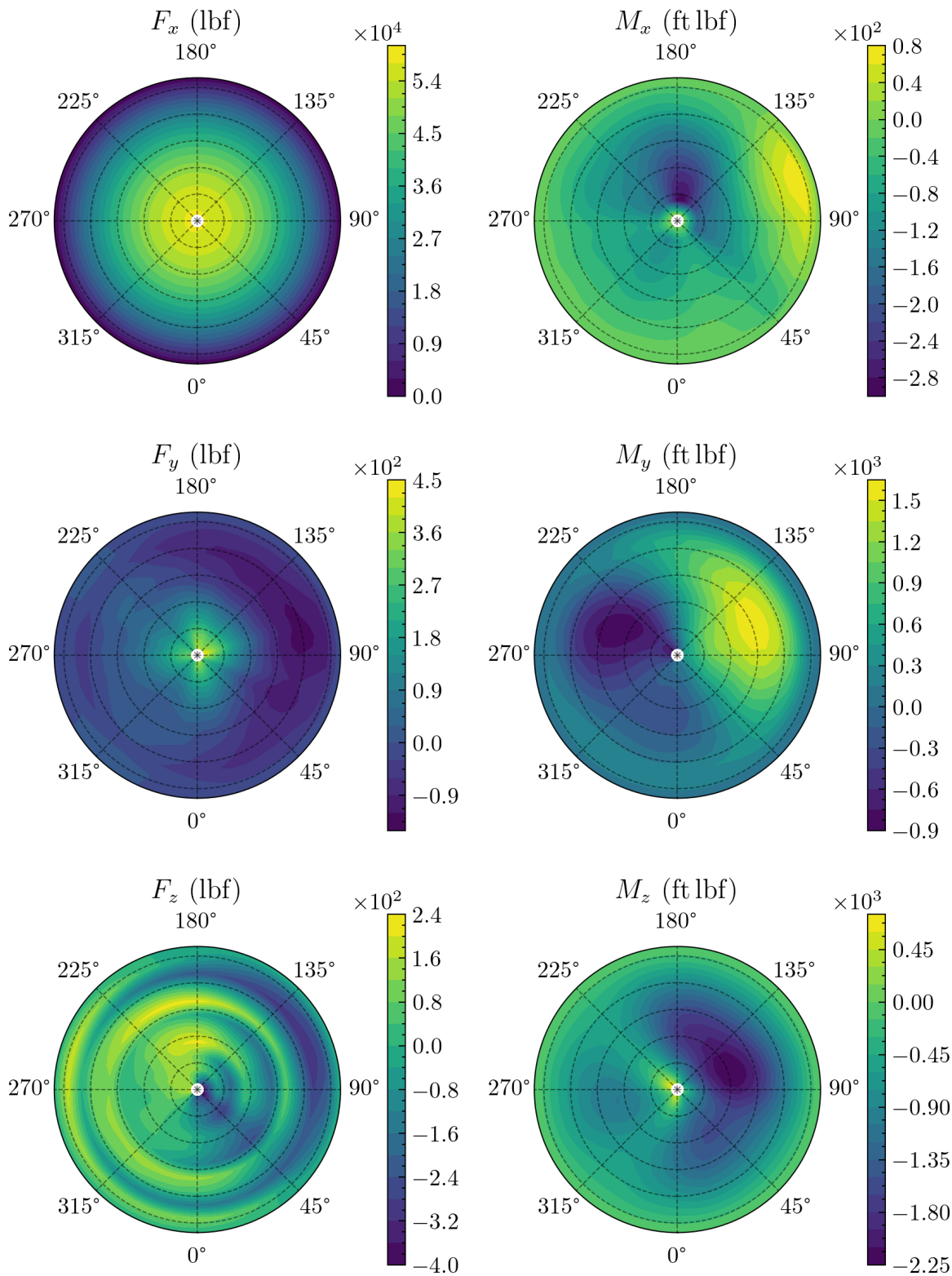


Figure 4.35: Rotor blade forces and moments for Experiment 1a, case 7.

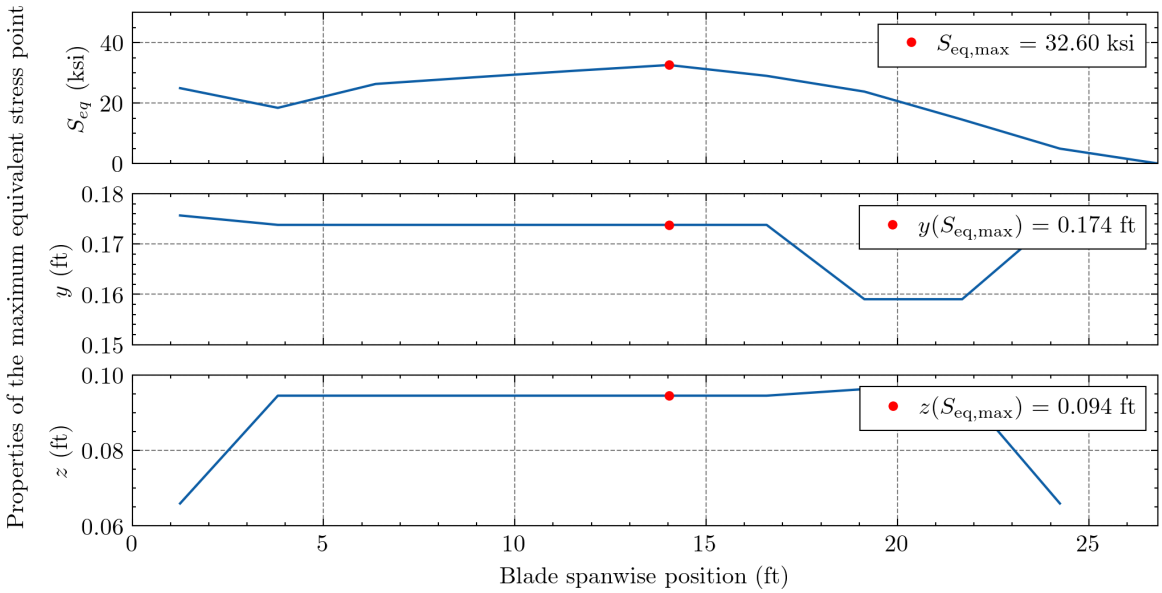


Figure 4.36: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 7.

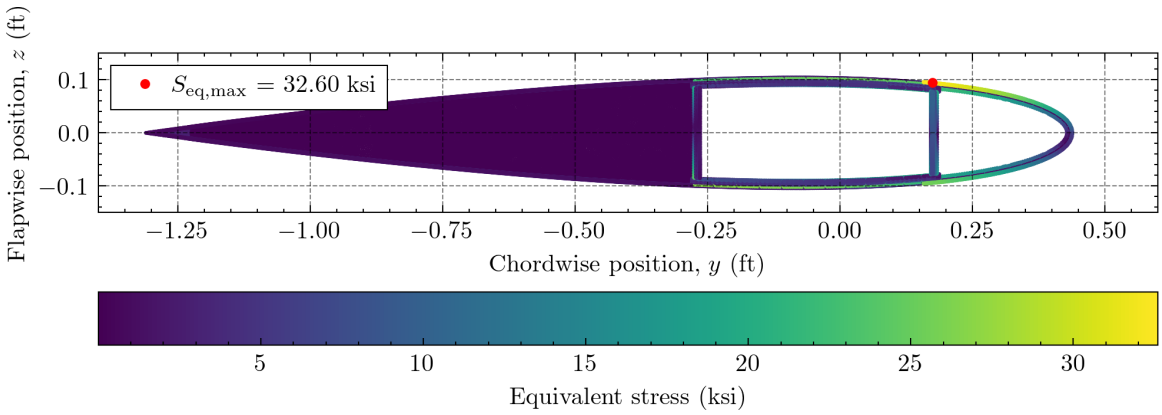


Figure 4.37: S_{eq} field at blade station 6 ($x = 14.025$ ft) for Experiment 1a, case 7.

4.4.5.9 Case 8: Climb

Figure 4.38 presents rotor blade loads for the climbing case. These loads are significantly different from the baseline case. Although the chordwise shear is only slightly higher, the negative flapwise shear at the blade root is much more significant than the baseline case. This is likely due to the forces carried in the pitch link. Additionally, the negative peaks of the pitching and chordwise bending moments are more extreme than the baseline case.

As can be seen in Figure 4.39, $S_{\mathrm{eq,max}}$ at the root is much higher than the baseline case with a value of 58.21 ksi. This appears to reflect the impact of the flapwise shear differences

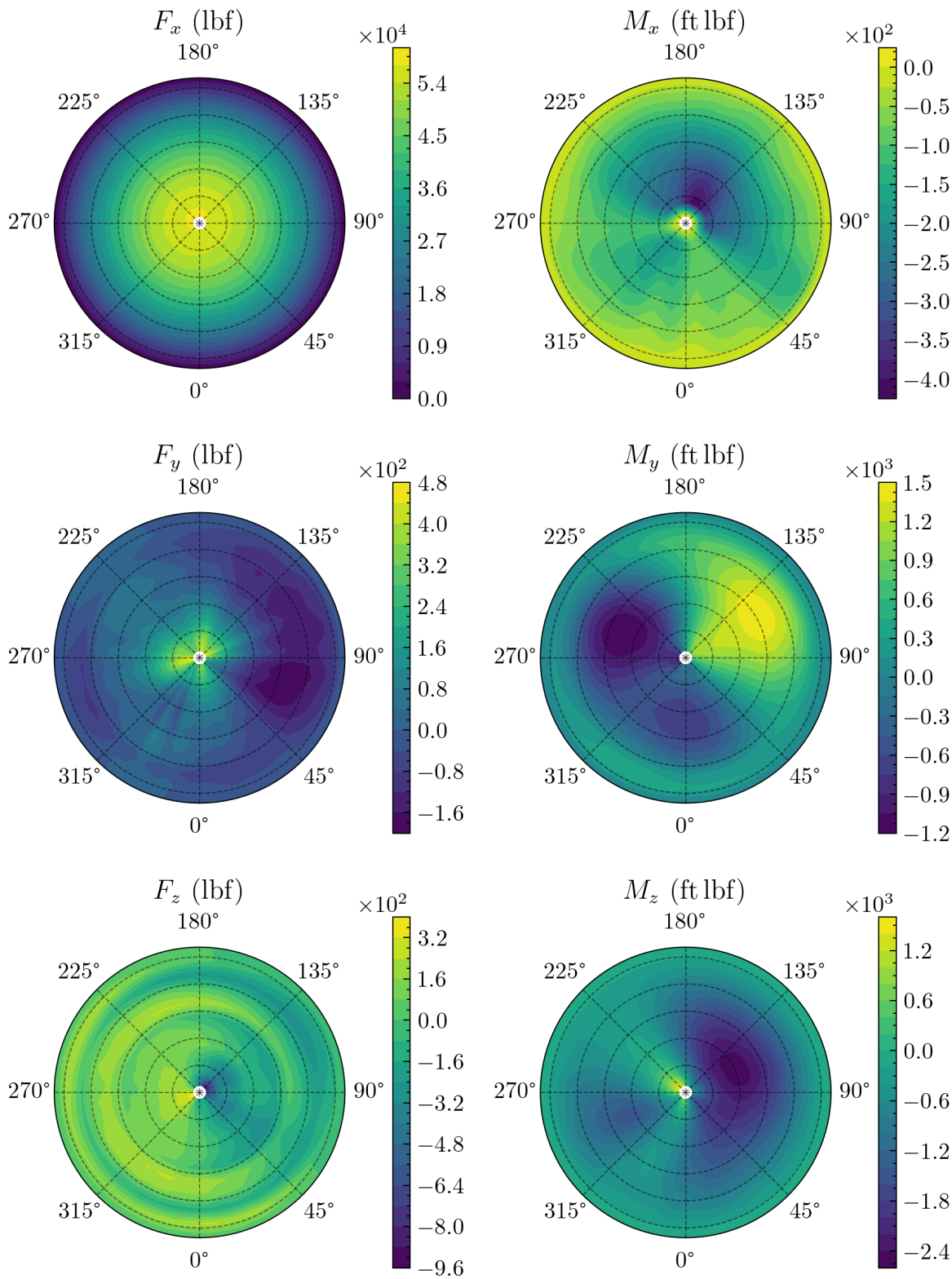


Figure 4.38: Rotor blade forces and moments for Experiment 1a, case 8.

seen in Figure 4.38. The maximum stress is similar to the baseline case at the midspan and tip locations.

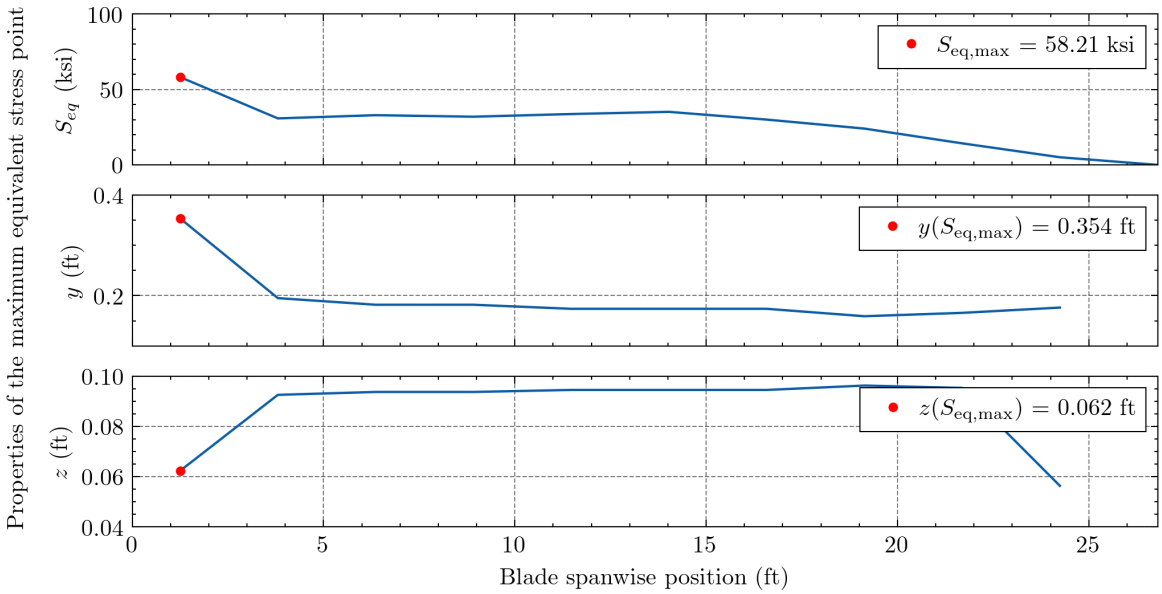


Figure 4.39: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 8.

Figure 4.40 shows that the location of the maximum stress point at the blade root is the same as in the high speed case. However, note that the equivalent stress on the box spar auxiliary web is also significantly greater than in the baseline case.

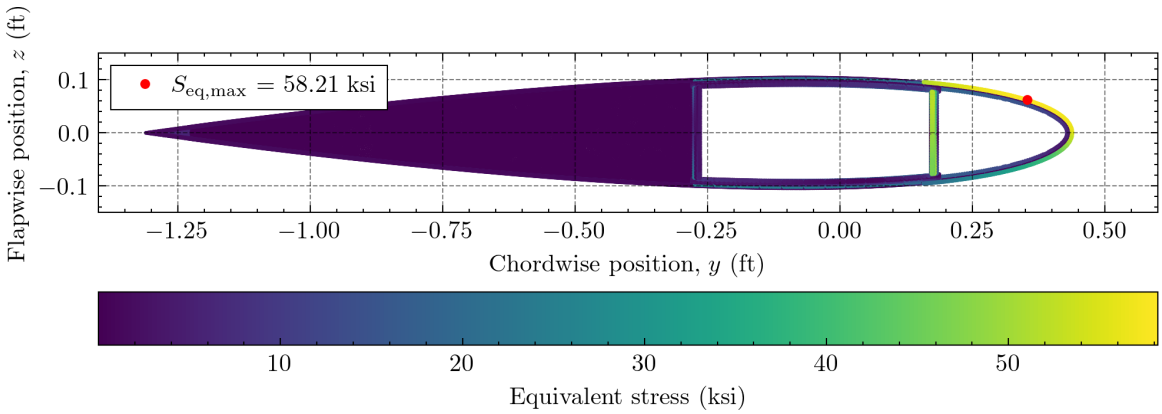


Figure 4.40: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 8.

In the climbing case, the main rotor produces thrust equal to the baseline case. However the amount of power produced by the rotor is higher. This correlates with an increase in drag on the rotor blade, which is likely responsible for the more negative chordwise bending

moment. The changes in flapwise shear appear to be correlated with changes in pitch link forces required by the trim position of the rotor system.

4.4.5.10 Case 9: Left Turn

The blade forces and moments for the left turn case are presented in Figure 4.41. The shear force plots are very similar to the baseline case, except that the chordwise shear force has a slightly smaller peak-to-peak amplitude and the flapwise shear force has a slightly wider peak-to-peak amplitude.

However, the plot of twisting moment shows a significant increase in pitch down (negative) moment on the root of the advancing blade, where $90° < \psi < 180$. The negative flapwise bending moments where $225° < \psi < 270°$ are also more significant than in the baseline case. The chordwise bending moment is very similar to the baseline case. These results are consistent with a rotor that is producing more lift on its advancing than retreating blade, resulting in a net horizontal force pointed to the port side of the vehicle.

Figure 4.42 shows a similar spanwise variation in $S_{\mathrm{eq,max}}$ to the baseline case. At 42.78 ksi, the peak equivalent stress at the root of the blade is just slightly above the baseline case. Figure 4.43 indicates that the peak stress location is in the same position on the steel erosion strip as the high speed case.

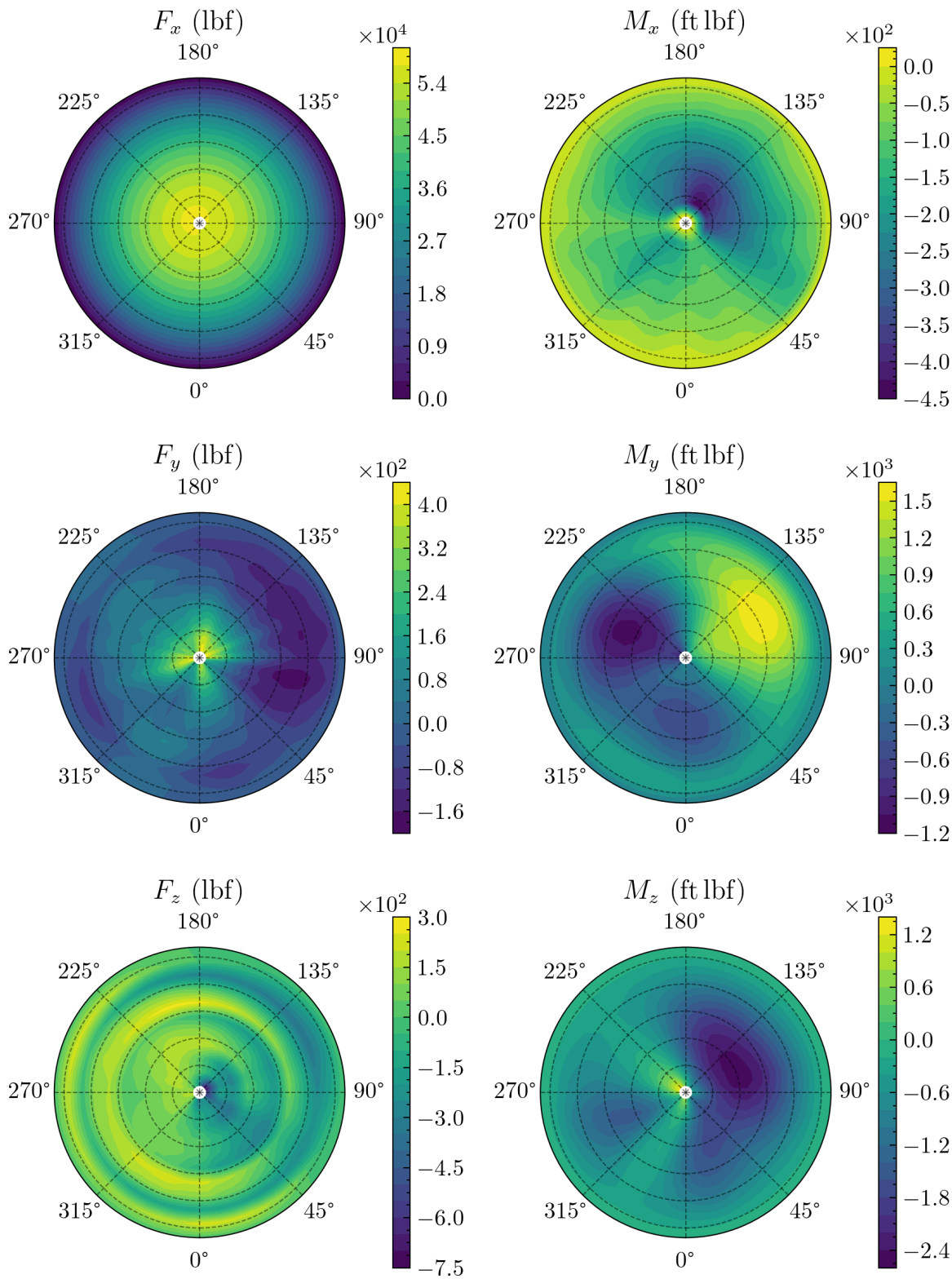


Figure 4.41: Rotor blade forces and moments for Experiment 1a, case 9.

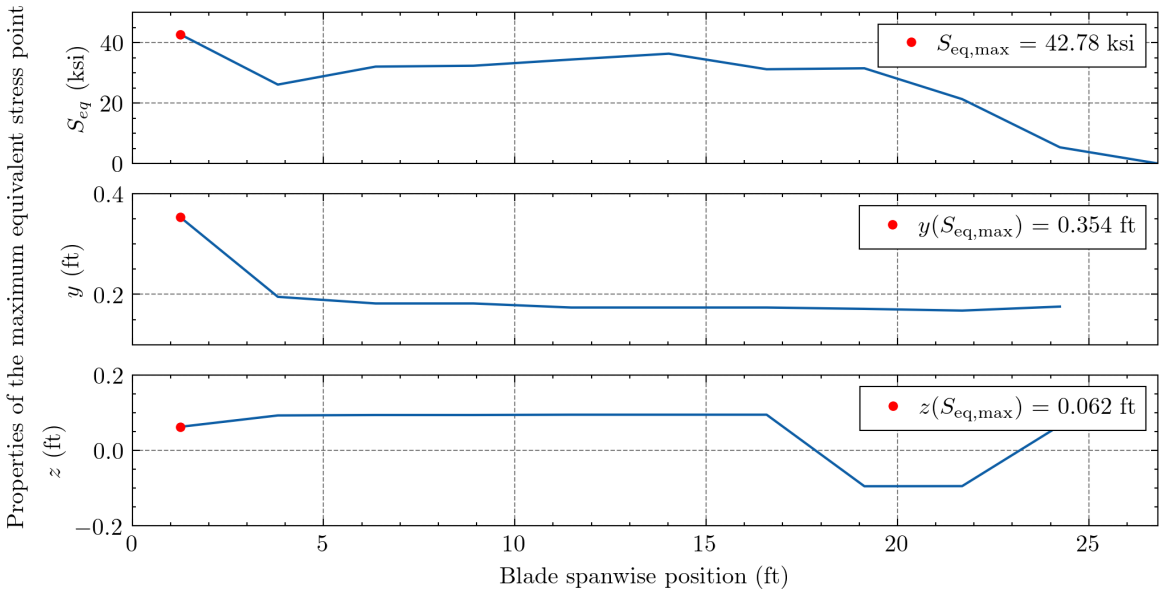


Figure 4.42: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 9.

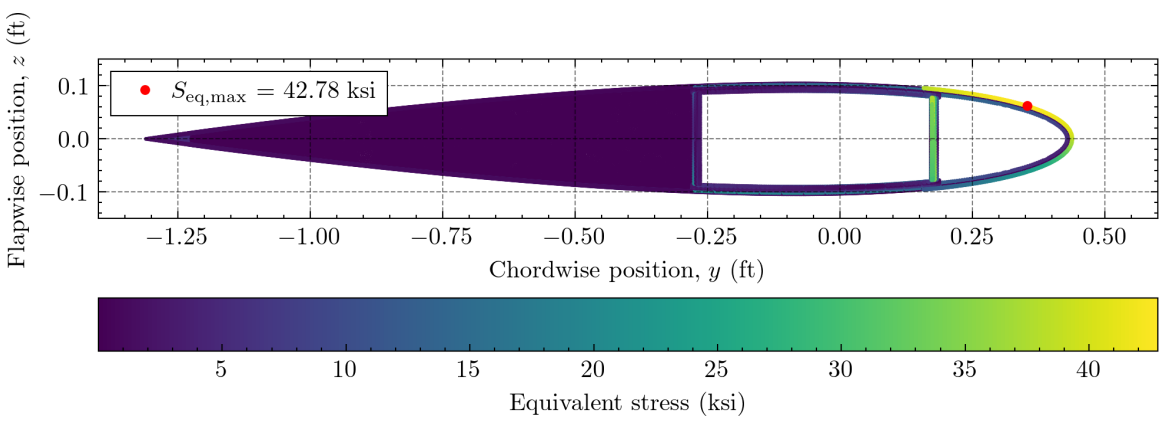


Figure 4.43: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 9.

4.4.5.11 Case 10: Right Turn

Figure 4.44 plots the rotor force and moment distribution in the right turn case, which is effectively indistinguishable from the left turn case. Any differences are due to the fact that, to produce a net horizontal force to the starboard side of the aircraft, the rotor must produce more lift on its retreating blade than its advancing blade. This is a consequence of the counterclockwise rotation direction employed on nearly all American helicopters.

Similarly, Figures 4.45 and 4.46 are also very similar to the left turn case. Although the peak equivalent stress at the root is slightly lower at 41.89 ksi, this is likely within the

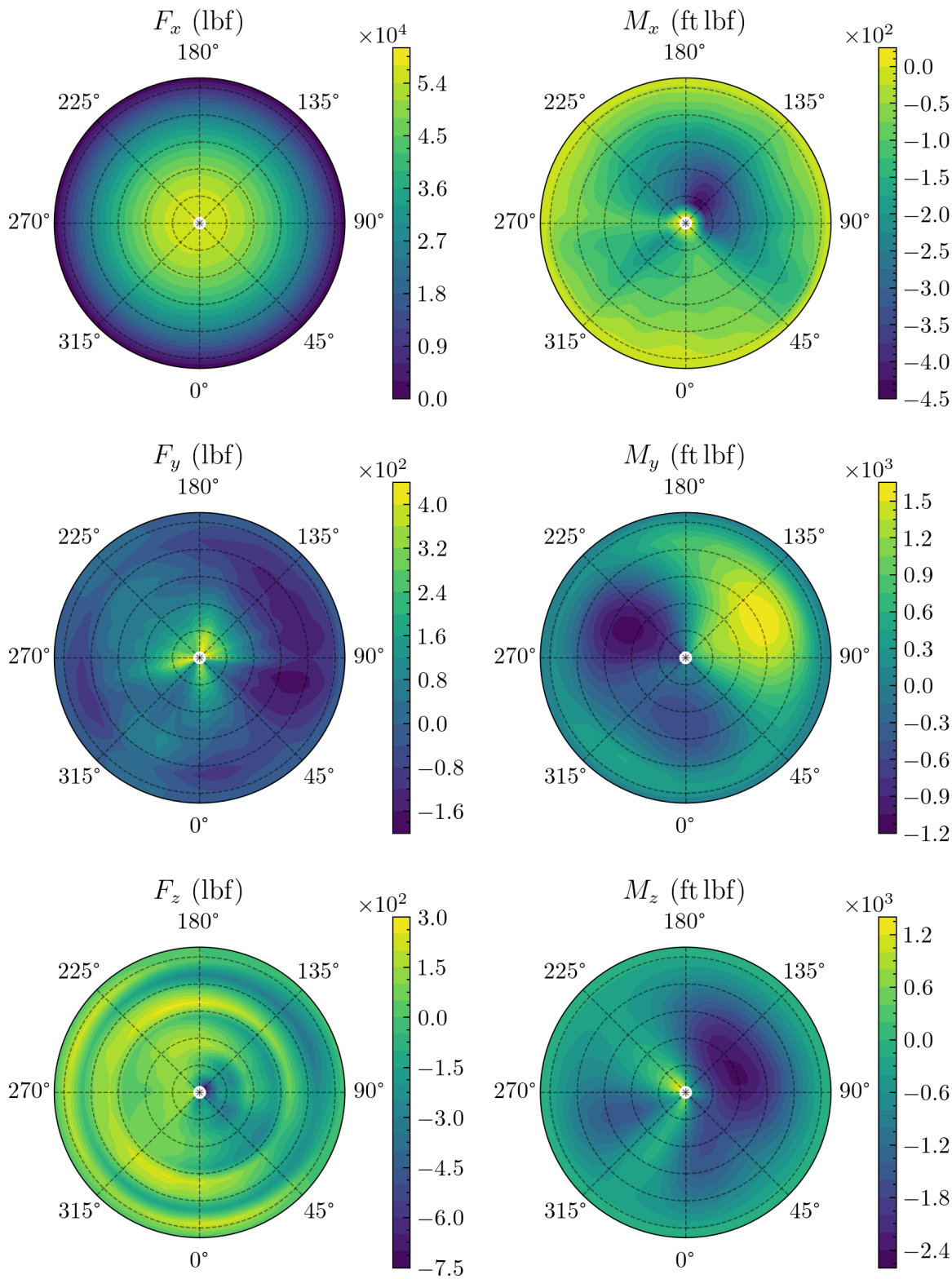


Figure 4.44: Rotor blade forces and moments for Experiment 1a, case 10.

margin of error of RCAS's trim solution.

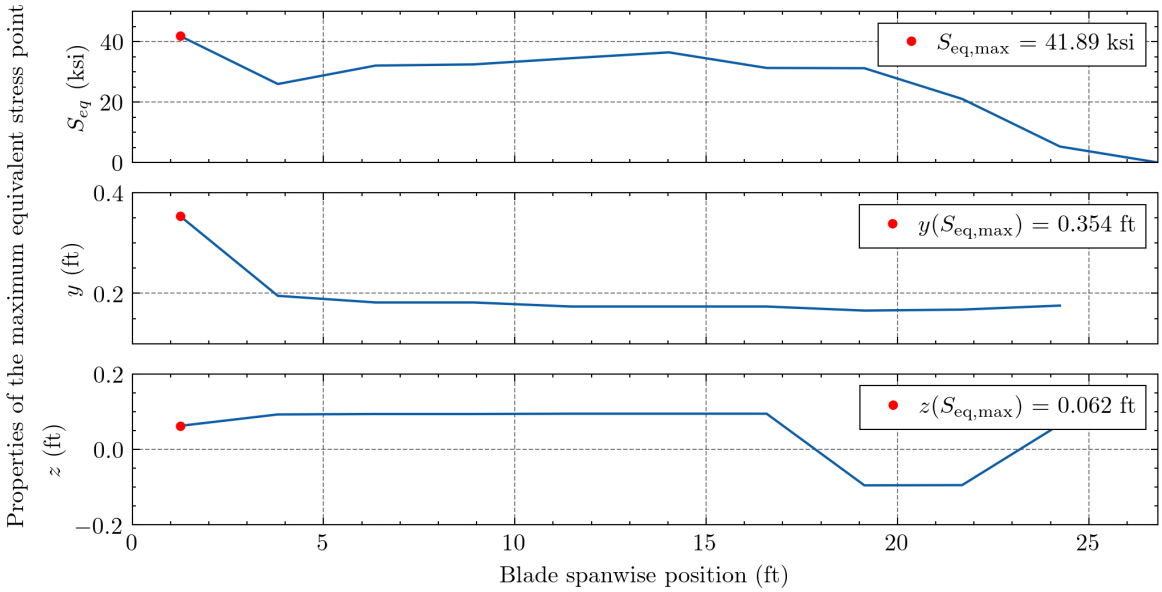


Figure 4.45: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 10.

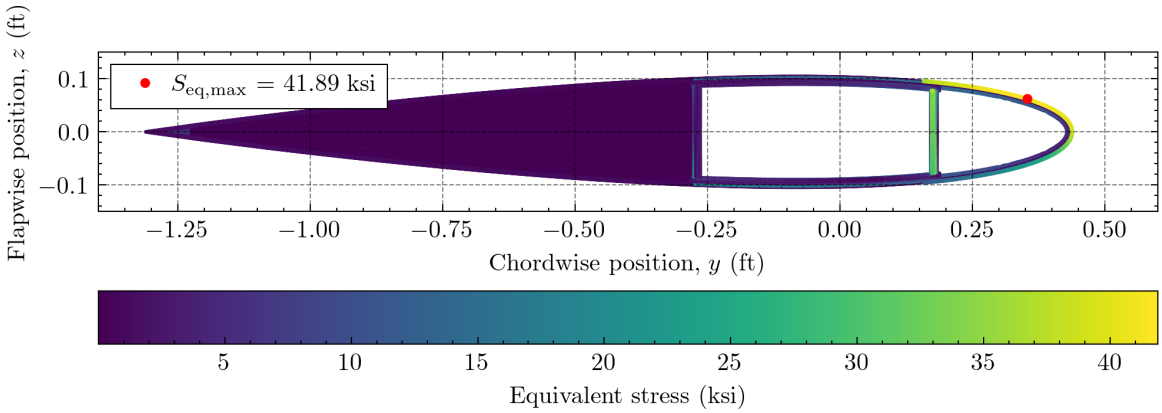


Figure 4.46: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 10.

4.4.5.12 Case 11: Forward CG

The rotor blade loads for the forward CG case are plotted in Figure 4.47. In this case, the shear forces are significantly lower than the baseline case. For the flapwise shear force in particular, the peak-to-peak amplitudes near the blade root are much less significant.

The chordwise bending moment also has lower peak-to-peak amplitudes than the baseline case, although the twisting and flapwise bending moments do not differ significantly from

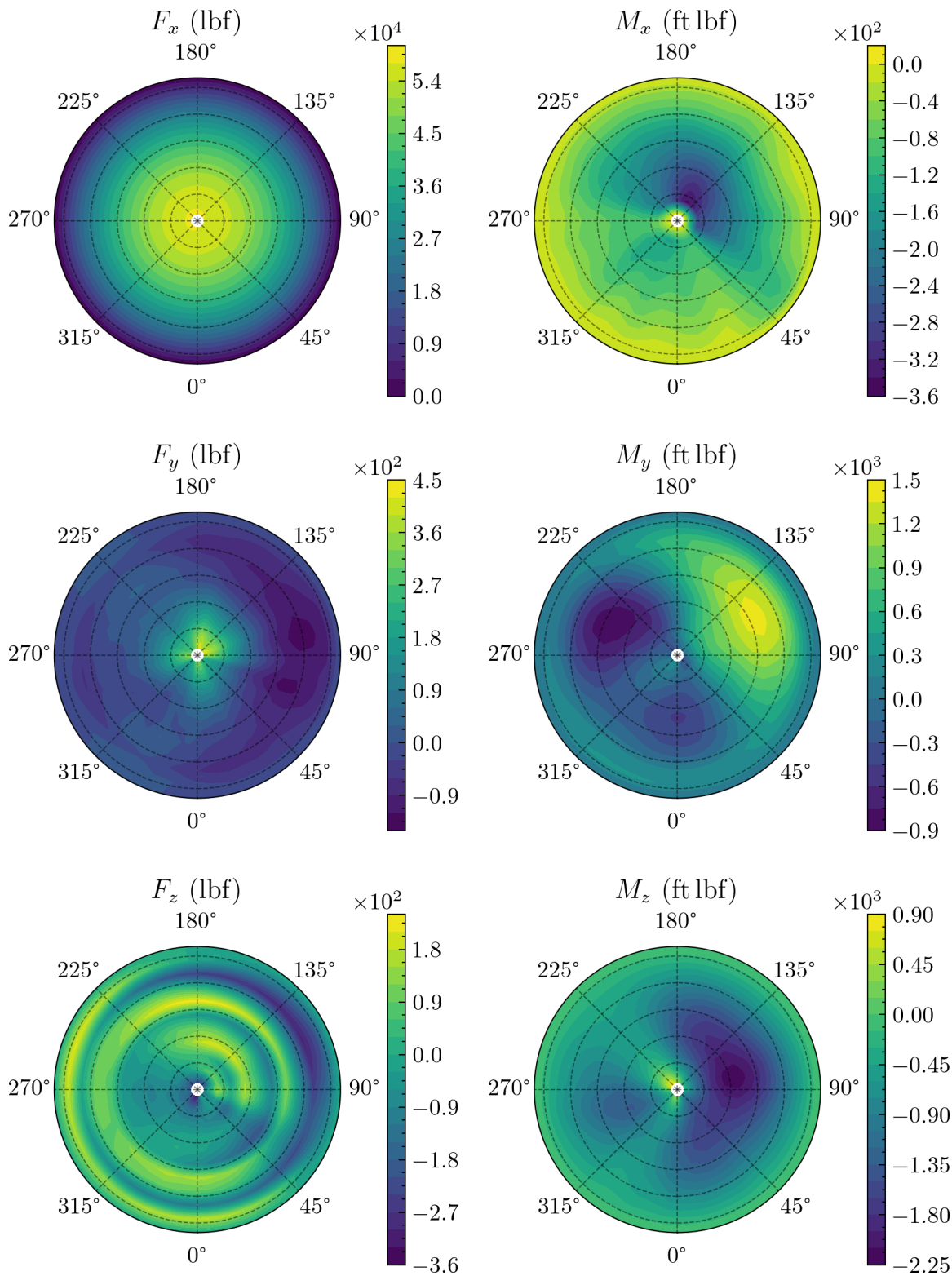


Figure 4.47: Rotor blade forces and moments for Experiment 1a, case 11.

the baseline. Significant differences in the spanwise variation of $S_{\text{eq,max}}$ can be seen in Figure 4.48.

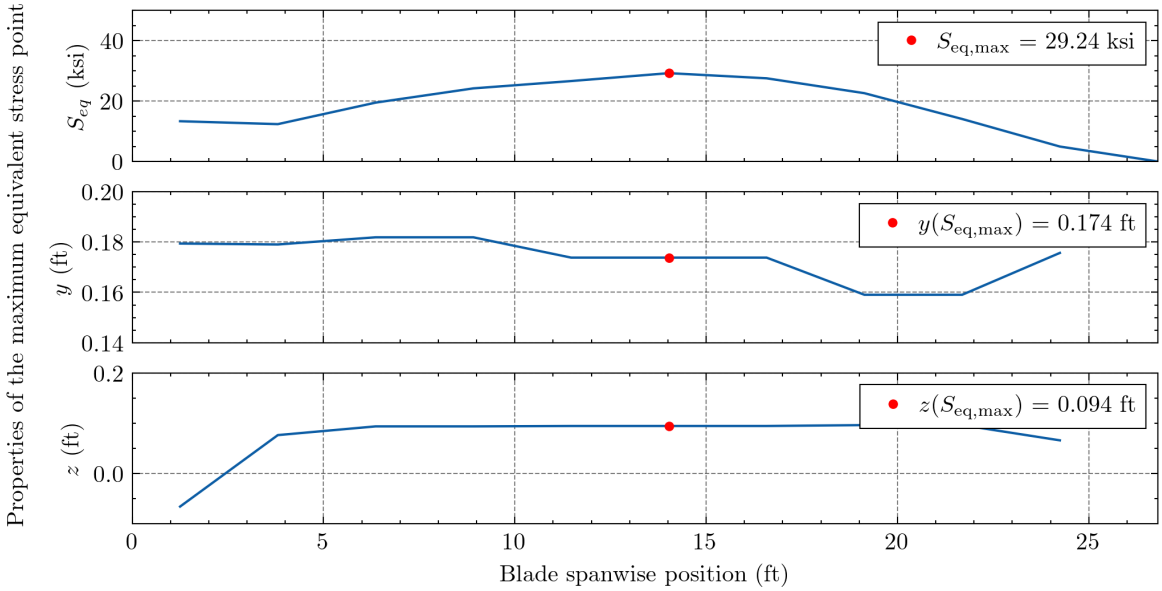


Figure 4.48: $S_{\text{eq,max}}$ and its location for Experiment 1a, case 11.

The high root stress that was present in most of the previous cases has reduced to approximately 15 ksi, and the maximum stress, which is located at the mid span, is only 29.24 ksi. Figure 4.49 shows a cross-sectional S_{eq} field that is similar to the descent case. The maximum stress location is on the steel erosion strip just above the auxiliary web, with a corresponding region of high stress on the high pressure surface.

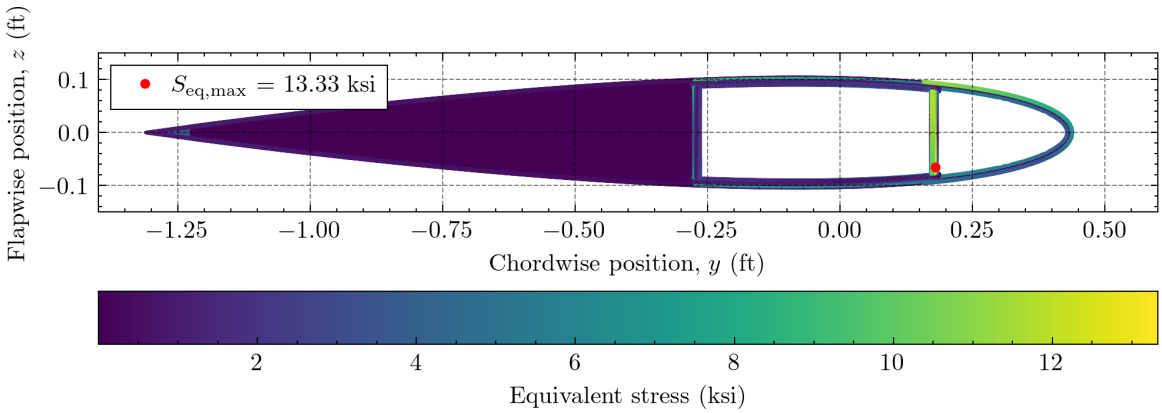


Figure 4.49: S_{eq} field at blade station 6 ($x = 14.025$ ft) for Experiment 1a, case 11.

The forward CG case is less stressful than the baseline case because it is "easier" for

the rotor. In forward flight, the vehicle must pitch down, but this is counteracted by the tail-heavy nature of the helicopter design. In the forward CG case, the CG is located just ahead of the main rotor shaft, so the pitch down moment required from the rotor system is lower. This results in lower loads and stresses on the rotor blade.

4.4.5.13 Case 12: Aft CG

The aft CG is perhaps the most interesting case surveyed in this experiment, as demonstrated by Figure 4.50. Although the chordwise shear force has a smaller peak-to-peak amplitude, the flapwise shear force is much more extreme. Particularly, the peak-to-peak amplitude of the oscillating flapwise shear force near the rotor blade root is much higher, which obscures the characteristic spanwise oscillations seen in previous cases. It also appears that the mean flapwise shear is higher than in previous cases.

The pitch up (positive) twisting moments on the tip of the advancing blade, in the region where $90° < \psi < 135°$, are higher than the baseline. The flapwise bending moment shows a similar pattern to the baseline case, but the peak-to-peak amplitude is higher. The same can be said of the chordwise bending moment plot.

Figure 4.51 reveals a significantly higher maximum equivalent stress at the root of the rotor blade. At 94.25 ksi, it is more than twice the corresponding value of the baseline case. As can be seen in Figure 4.52, this maximum stress point is at the same location near the top of the auxiliary box spar web.

The aft CG case is significant in that it is the most damaging case from a fatigue life perspective. The aftward CG shift increases the pitch up moment on the helicopter; this must be countered by the main rotor to achieve the proper pitch down orientation for trimmed forward flight. High overall pitch down moment exerts proportionally more stress on the rotor blade.

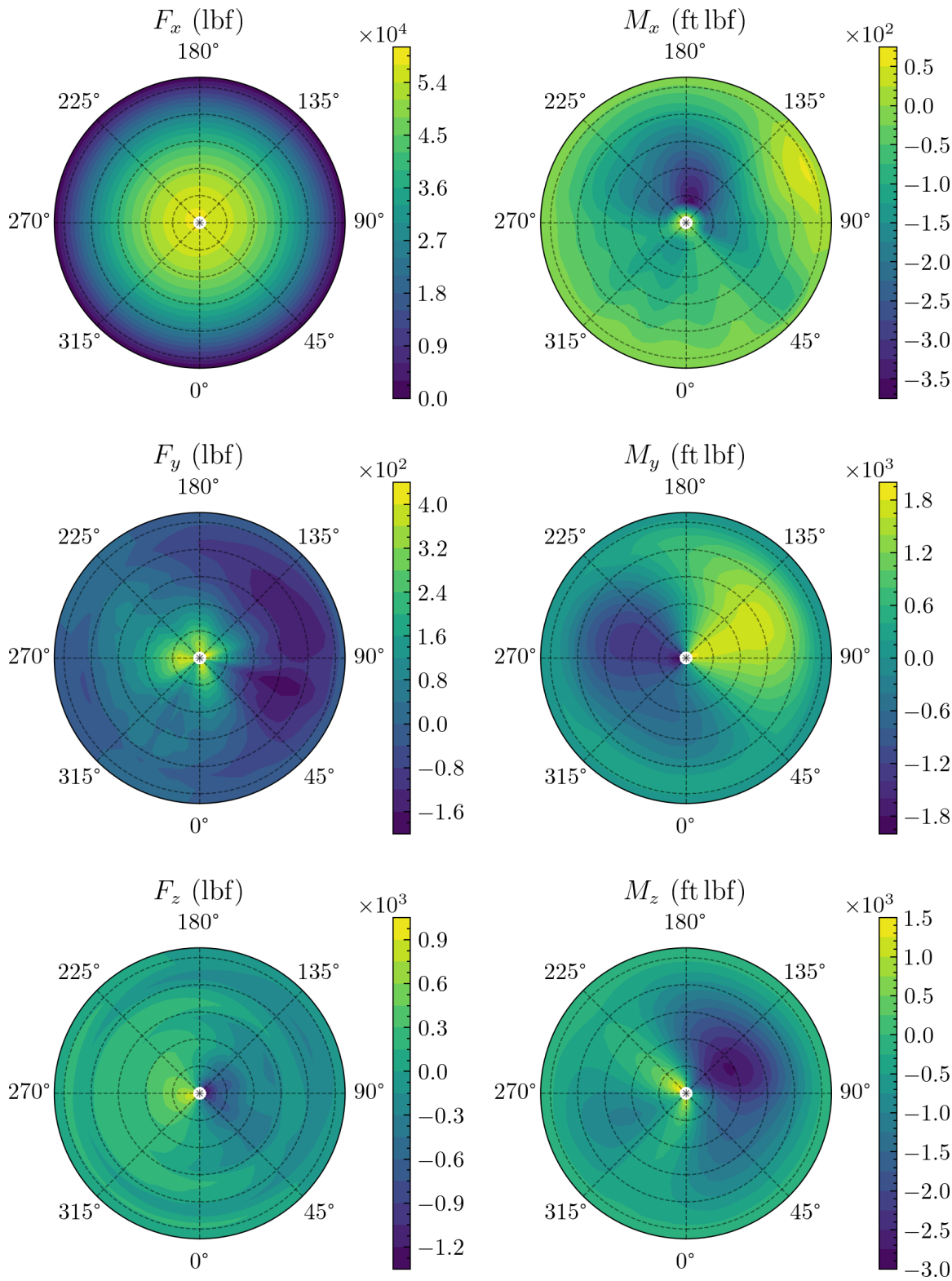


Figure 4.50: Rotor blade forces and moments for Experiment 1a, case 12.

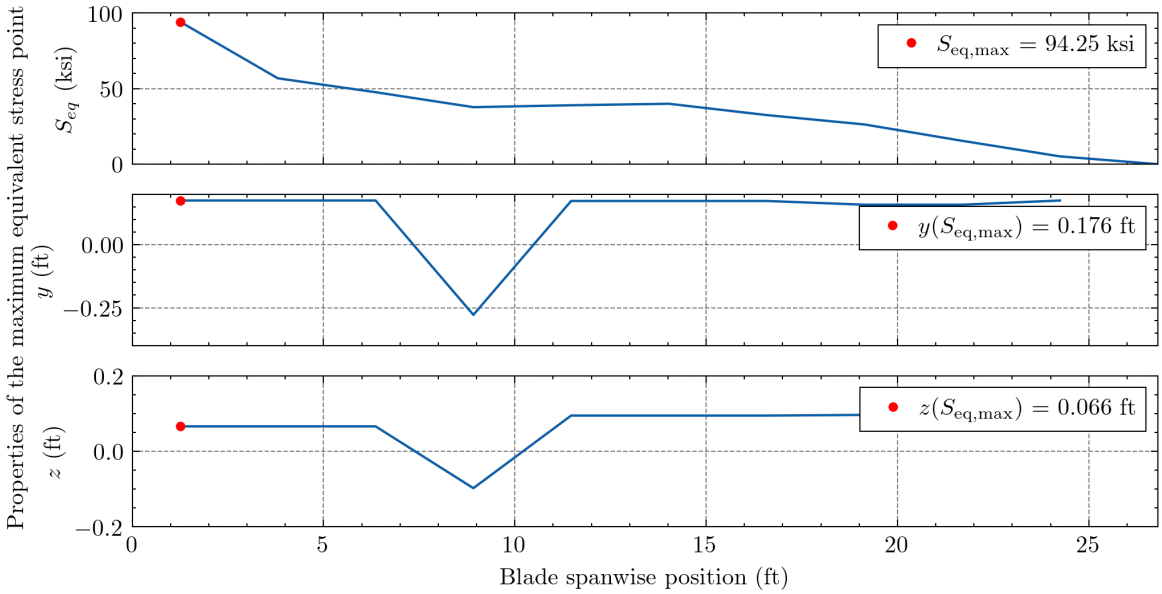


Figure 4.51: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 12.

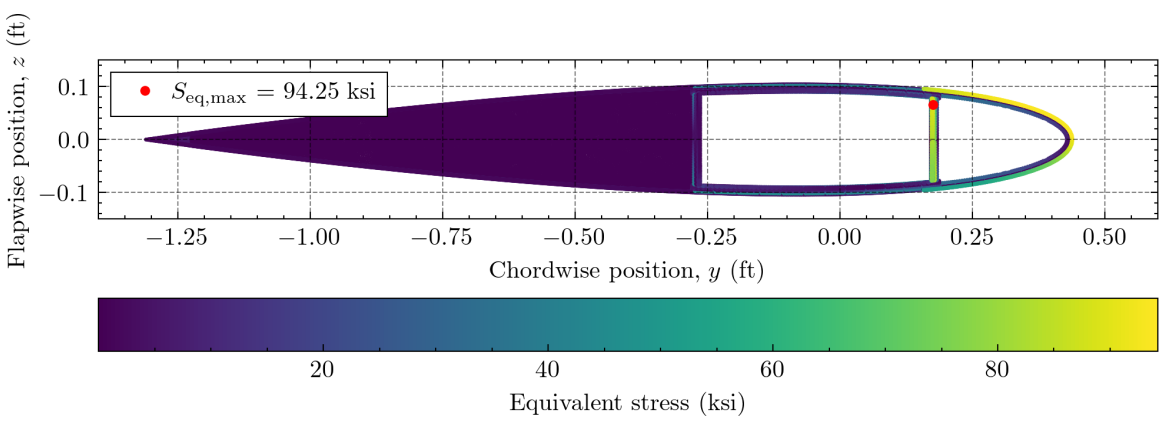


Figure 4.52: S_{eq} field at blade station 1 ($x = 1.25\,\mathrm{ft}$) for Experiment 1a, case 12.

4.4.5.14 Case 13: Reverse

The reverse flight case is very similar to the hover case. Figure 4.53 shows that the blade load distributions for this case. Although the flapwise and chordwise shear forces have slightly higher peak-to-peak amplitudes than the baseline case, the patterns are effectively identical.

The peak-to-peak amplitude of the twisting moment is almost exactly the same as the hover case, but the location of the negative peak has been shifted towards the $\psi = 315°$ azimuth position. The chordwise bending moment has a slightly larger magnitude than the

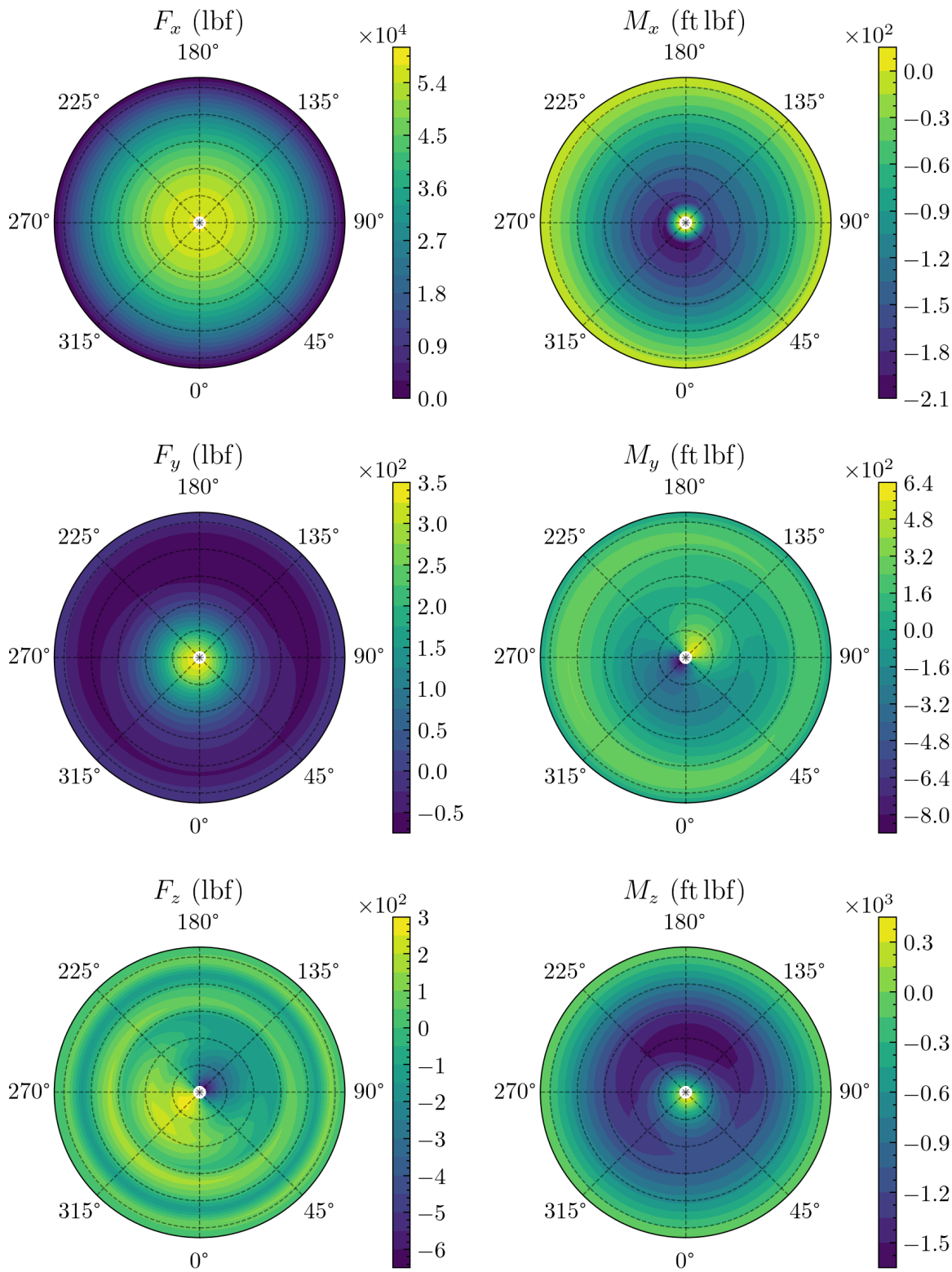


Figure 4.53: Rotor blade forces and moments for Experiment 1a, case 13.

hover case, but the negative peak is shifted towards the $\psi = 180°$ azimuth. These results are consistent with a rotor system that is generating some net rearward horizontal force.

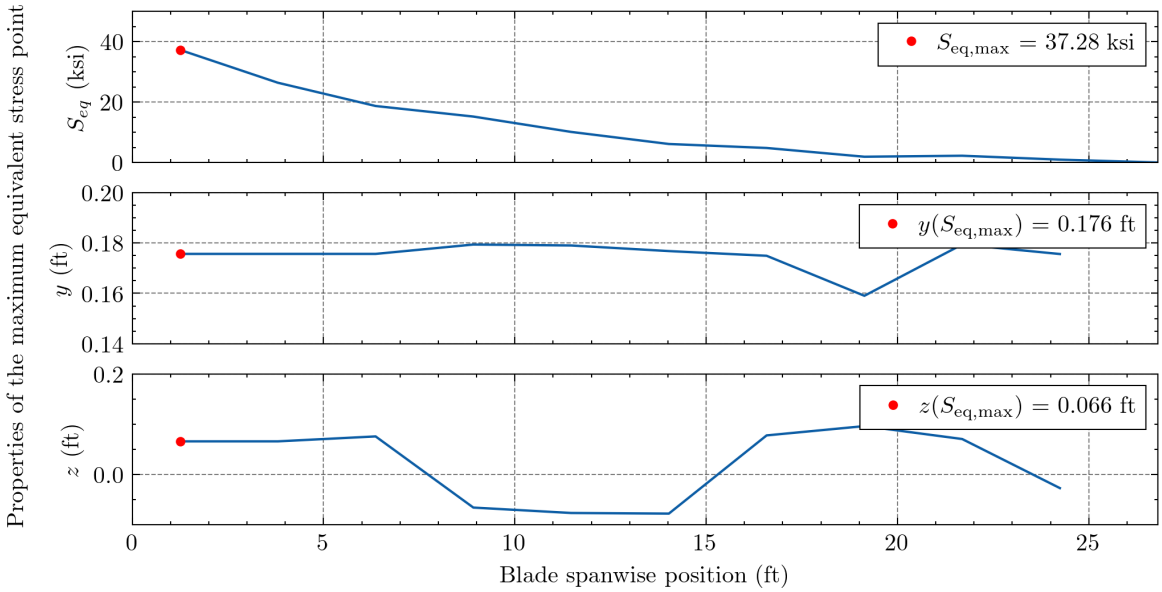


Figure 4.54: $S_{\mathrm{eq,max}}$ and its location for Experiment 1a, case 13.

Figures 4.54 and 4.55 reinforce that the reverse flight case is nearly identical to the hover case, except the maximum equivalent stress is slightly higher at 37.28 ksi.

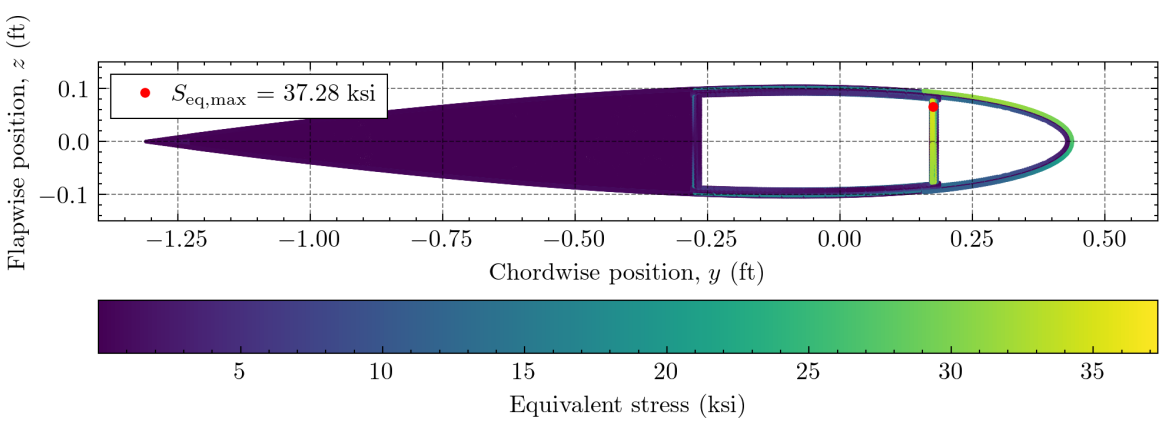


Figure 4.55: S_{eq} field at blade station 1 ($x = 1.25$ ft) for Experiment 1a, case 13.

Note that, because aerodynamic interaction between the main and tail rotors were neglected in the RCAS model, these results may be unrealistic. If the wake of the tail rotor was modeled, it is possible that it would be ingested by the main rotor in reverse flight, resulting in a more unique load environment.

4.4.6 Summary

The results of the extreme flight condition survey are summarized in Table 4.8. In general, the cases which varied the CG location of the helicopter showed the most significant differences compared to the baseline case. Variation in airspeed, gross weight, and rate of climb were also significant. Changes in altitude and turn rate did not have a very strong effect; the effect of turn direction was almost negligible. The reverse flight case was not significantly different from the hover case.

Ultimately, there are three candidates for the definition of the critical fatigue point. The point on the auxiliary web of the box spar near the low pressure surface occured in 7 of the 14 surveyed positions. Additionally, this point experiences the highest overall equivalent stress in the aft CG case. Two points on the low pressure surface, one near the leading edge and one just above the auxiliary web, occurred in a total of 4 and 3 cases, respectively.

When selecting the critical fatigue point, the consequences of a failure at that point must be considered. A failure of the auxiliary web of the box spar constitutes a serious structural failure of the rotor blade. Loss of this structural member would significantly impact the ability of the rotor blade to carry loads. The decreased structural strength at that location may lead to a complete failure of the rotor blade, which would almost certainly result in a fatal accident.

Conversely, the initiation of a fatigue crack on the steel erosion strip would have less drastic consequences. Although a crack would impair the ability of the erosion strip to protect the blade from pitting, it likely would not significantly alter the structural characteristics of the rotor blade and would be discovered at the next inspection. A portion of the erosion strip could be replaced by a rotor blade repair specialist, or a new rotor blade could be fitted. Either way, it is unlikely that this failure mode would lead to an accident.

For these reasons, the point on the auxiliary web of the box spar was selected to serve as the critical fatigue point for all subsequent experiments. This point is defined by a chordwise (y) position of 0.176 ft and a flapwise (z) position of 0.066 ft. Note that

Table 4.8: Value and position of the peak $S_{\mathrm{eq,max}}$ point and its location in the extreme flight condition survey.

Case	Name	$S_{\mathrm{eq,max}}$ (ksi)	Station	Location	Notes
0	Baseline	40.04	1	Box spar aux. web	Typical cruise condition
1	Hover	35.44	1	Box spar aux. web	
2	High speed	49.73	1	Low pressure surface near leading edge	On steel erosion strip
3	Low altitude	42.59	1	Box spar aux. web	
4	High altitude	36.80	1	Box spar aux. web	
5	Low weight	47.49	1	Box spar aux. web	
6	High weight	36.80	6	Low pressure surface above aux. web	On steel erosion strip
7	Descent	32.60	6	Low pressure surface above aux. web	
8	Climb	58.21	1	Low pressure surface near leading edge	
9	Left turn	42.78	1	Low pressure surface near leading edge	
10	Right turn	41.89	1	Low pressure surface near leading edge	
11	Forward CG	29.24	6	Low pressure surface above aux. web	Lowest overall
12	Aft CG	94.25	1	Box spar aux. web	Highest overall
13	Reverse	37.28	1	Box spar aux. web	Similar to hover

although Experiment 1a considered specific extreme flight conditions in isolation, subsequent experiments will consider combinations of these cases, such as a high gross weight *and* aft CG flight condition, which is expected to be even more damaging. Additionally, in Experiment 2b, these flight conditions will be combined into a complex mission profile to better model the operations of a realistic transport helicopter.

4.5 Experiment 1b

Experiment 1a identified the critical fatigue point on the main rotor blade of the generic SMR helicopter. Experiment 1b concerns training scalar surrogate models to predict the mean stress and stress amplitude responses at that point. Figure 4.56 provides an overview of Experiment 1b.

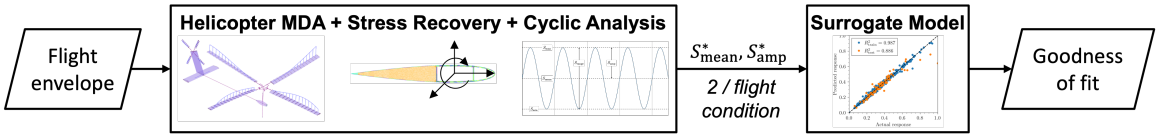


Figure 4.56: Overview of Experiment 1b.

4.5.1 Experimental Design

First, the flight envelope defined by the six variables studied in Experiment 1a will be sampled using DOEs. The complete analysis environment developed previously, consisting of the MDA, stress recovery, von Mises stress calculator, and cyclic stress analyzer, is executed at each point. To reduce computational expense, the stresses at only the first blade station are recovered and the responses at only the critical stress point are stored. The responses are the mean equivalent stress, S^*_{mean}, and the equivalent stress amplitude, S^*_{amp}, at the previously-defined critical fatigue point.

Next, a number of surrogate models will be fit to the data and trained to predict both responses. Each of the surrogate models surveyed in Section 4.1 will be tested. The architectures of each model will also be varied to determine if certain modeling choices can improve accuracy. The methods and their variations are presented in Table 4.9.

Each surrogate model will be trained while varying the quantity of training data available. This type of analysis is used to understand a model's *learning curve*. Since one of the requirements of the preliminary fatigue design methodology is to minimize cycle time, it is desirable that the selected surrogate modeling method performs well with as few training points as possible; this will minimize the runtime of the MDA sampling process.

Table 4.9: Surrogate modeling methods and architectural choices for Experiment 1b.

Method	Architecture parameter	Options
Response surface equations	Polynomial order	1st-order 2nd-order 3rd-order
Shallow neural networks	Hidden layer nodes[a]	50 100 200
Deep neural networks	Network topology	(20, 20)[b] (20, 20, 20) (50, 50) (50, 50, 50)
Gaussian process models	Covariance function	Squared exponential Matérn ($\nu = 3/2$) Matérn ($\nu = 5/2$) Rational quadratic

[a] Shallow neural networks have a single hidden layer between the input and output layers.
[b] This nomenclature describes two 20-node hidden layers between the input and output layers.

The performance of the surrogate models will be compared using two of the surrogate model performance metrics reviewed in Section 4.1. The coefficient of determination, R^2, provides a simple metric to quantify overall performance. Higher values of R^2 indicate a better fit. The ideal is $R^2 = 1$, which indicates perfect performance. The standard deviation of prediction error, σ, referred to as the model fit/representation error by Mavris [96], provides an estimation of the "spread" in the predictions. Lower values of σ indicate better performance; the ideal would be $\sigma = 0$. This performance metric is particularly relevant in the context of Experiment 2, where the uncertainty associated with surrogate model predictions must be incorporated. Both metrics will be applied to the training set, which is used to fit the model, and the testing set, which is held out during the training process and used to provide an unbiased assessment of model performance.

The surrogate modeling method and architecture that provides the best performance as

the lowest number of sample points will be selected for use in subsequent experiments. In this experiment, a good fit will be indicated by a value of R^2 near or above 0.9, and a value of σ near or below 0.05. The next sections will discuss the implementation of the flight envelope DOEs and the specific parameters used to train the surrogate models.

4.5.2 Flight Envelope Sampling

The flight envelope is sampled using the six variables explored in Experiment 1a. To enable the learning curve analysis, a series of sequential DOEs was developed. Each DOE provides a complete sampling of the flight envelope space, and additional DOEs can be added to increase the infill of sampling points.

The first DOE is a central composite design (CCD). The CCD architecture was chosen to completely cover the flight envelope with as few points as possible. An ideal CCD DOE would require only 77 points to sample a six-dimensional space. However, in this application, certain corners of the design space are physically meaningless. If airspeed, V, is zero, then a coordinated turn is impossible and the turn rate, ω, will have no effect.

Thus, the first DOE was split into a *baseline* set, where a minimum airspeed of 10 kt is maintained, and an additional *hover/axial climb* set, where V and ω are fixed at 0 kt and 0 rad/s, respectively. This four-dimensional space requires 25 points for full coverage. Together, these two DOEs make make up a 102-point initial set.

The variables and ranges for the baseline set and the hover/axial climb set are presented in Tables 4.10 and 4.11, respectively. Note that the variable ranges have been contracted slightly compared to Experiment 1a because the combination of certain points, like maximum GW and maximum ROC, were found to consistently cause the NDARC or RCAS solutions to fail. These DOEs were constructed in JMP, a statistical analysis program with a strong emphasis on surrogate modeling features.

Next, a number of *space-filling augmentation* DOEs were created in order to provide infill in regions not covered by the CCD DOE. These DOEs were defined sequentially using

Table 4.10: Baseline DOE for Experiment 1a.

Variable	Units	Minimum	Maximum
V	kt	10	160
h_d	ft	0	10,000
GW	lb	11,500	20,000
ROC	ft/min	−1200	1200
ω	deg/s	−6	6
CG	ft	−0.3	1.6

Table 4.11: Hover/axial climb DOE for Experiment 1a.

Variable	Units	Minimum	Maximum
h_d	ft	0	10,000
GW	lb	11,500	20,000
ROC	ft/min	−1200	1200
CG	ft	−0.3	1.6

[a] V and ω are fixed at 0 in this DOE.

a space-filling algorithm within JMP that considers pre-existing points when building a new DOE. Thus, the points from the augmentation DOEs will not be too close to the CCD DOEs, or to each other, which maximizes flight envelope coverage and sampling efficiency. The space-filling augmentation set expands the minimum value of V to 0 kt, but also implements a rule to prevent non-zero values of ω at any point where $V = 0\,\mathrm{kt}$. Five augmentation DOEs were created with 100 sample points each.

Finally, a fully independent *test set* DOE was created. This DOE uses the same parameters and space-filling algorithm as the augmentation DOEs, but the sampling does not consider the positions of any other sample points. In this application, a common test set is desirable because it allows the evaluation of all surrogate models on equal footing. The variables and ranges used in the space-filling augmentation and test set DOEs are described in Table 4.12.

The S^*_{mean} and S^*_{amp} responses were evaluated at all sample points using the process described in Experiment 1b. Running in parallel on eight processor cores, each case took an average 1.9 min of wall-clock time to complete, although runtime varies significantly

Table 4.12: Space-filling augmentation and test set DOEs for Experiment 1a.

Variable	Units	Minimum	Maximum
V	kt	0	160
h_d	ft	0	10,000
GW	lb	11,500	20,000
ROC	ft/min	−1200	1200
ω	deg/s	−6	6
CG	ft	−0.3	1.6

depending on the length of the RCAS trim solution.

A small number of cases failed to converge, either during the NDARC or RCAS solutions. These cases were excluded from the surrogate model training process. The input space is visualized as a scatter plot matrix in Figure 4.57. Sample points from all DOEs are included. Converged sample points are plotted with a blue dot, and non-converged sample points are plotted with an orange $\times$ symbol.

First, note that there is a greater concentration of non-converged cases in the high altitude, high GW, and high speed region. This region is one in which the RCAS model occasionally fails to converge, either because the maximum number of trim solution iterations is reached before trim is achieved, or a numerical error is encountered in the periodic solution subroutine. However, there is also a fair number of converged sample points in this region.

Also note that the concentration of sample points is greater in the corners and the center of the flight envelope space than any other regions. This is due to the use of central composite designs to define the first DOE. Since the corners of the design space correspond with the most extreme, and most damaging, flight conditions, extra data in these regions is not unappreciated.

The corresponding response space is visualized in Figure 4.58. A natural logarithm transform has been applied to the S^*_{mean} response to smooth out its distribution. This should improve the fit quality for simpler surrogate models like RSM.

Note the distinct "gulf" or gap in sample points in the response space. This is an artifact

Figure 4.57: Input space for surrogate model training in Experiment 1a.

of the signed von Mises stress calculation (see Equations (4.54) to (4.56)). Consider a time history of signed von Mises stress, as presented in Figure 4.59.

In Figure 4.59, the line labeled $S_{\text{mean,max}}$ describes the stress history at the point of maximum mean stress, and likewise for the lines labeled $S_{\text{amp,max}}$ and $S_{\text{eq,max}}$. At any point, there is a distinct discontinuity around $S_{\text{vm},s} = 0$, when $S_{\text{vm},s}$ crosses from positive to negative, or vice versa. This is because signed von Mises stress becomes negative whenever the hydrostatic stress is negative, without regard for the value of von Mises stress itself. Thus as the amplitude of one of the stress history traces increases, it may eventually cross zero, where the discontinuity artificially increases the amplitude further. This effect is responsible

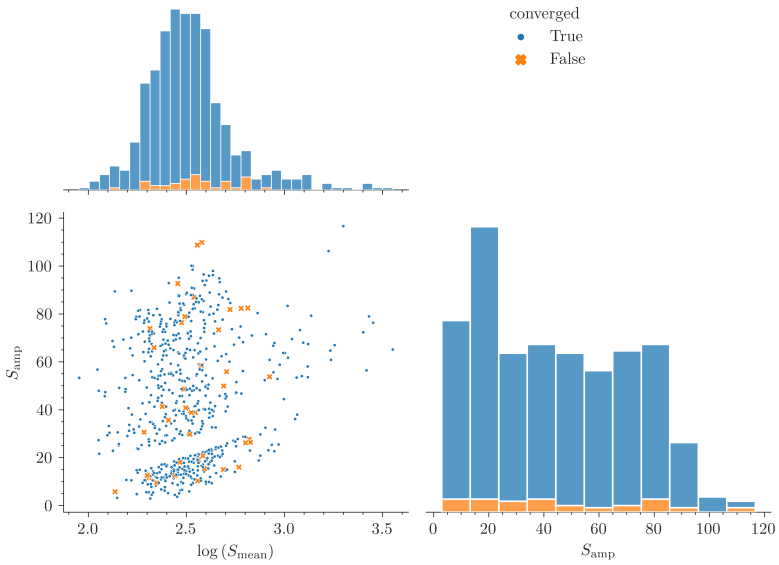


Figure 4.58: Response space for surrogate model training in Experiment 1a.

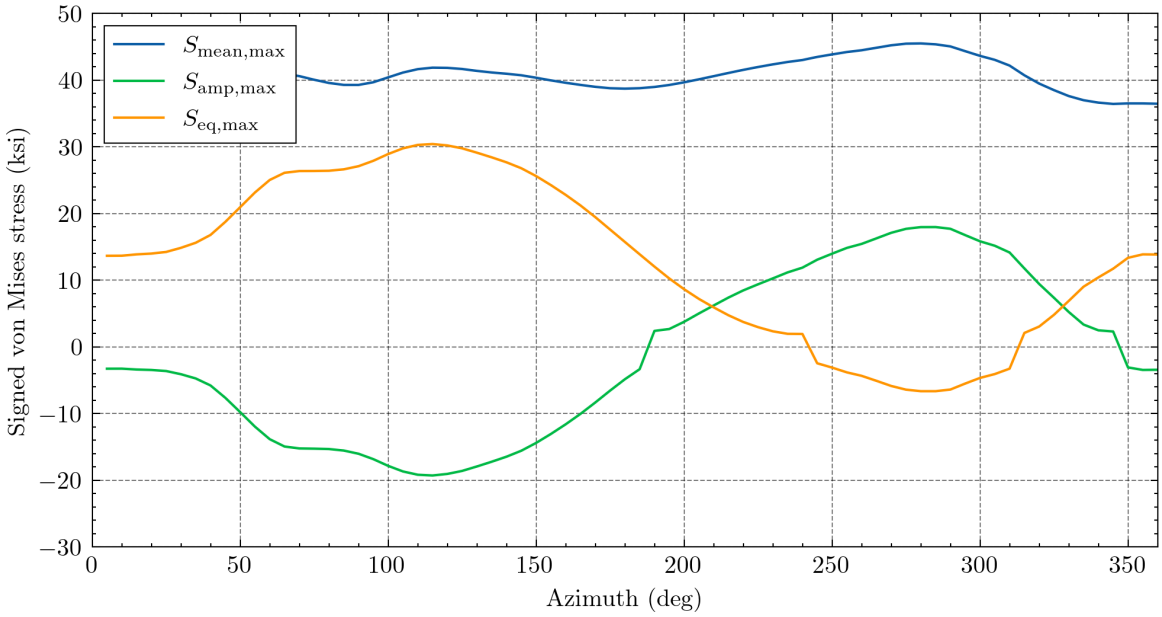


Figure 4.59: Time history of signed von Mises stress at three points demonstrating discontinuities around $S_{\text{vm},s} = 0$.

for the gulf seen in Figure 4.58.

This is a notable weakness of the signed von Mises stress formulation in this application. The additional nonlinearity in the response data will negatively impact the performance of all surrogate modeling methods. However, other stress invariants have their own unique weaknesses, and signed von Mises stress is still the most applicable for this research.

4.5.3 Surrogate Modeling Methods

Each surrogate modeling method described in Table 4.9 was implemented using the scikit-learn package [119]. First, all the data points were scaled to the $[0, 1]$ range based on the minimum and maximum values of each input variable and response. This prevents the scale of one variable overwhelming any of the others.

The RSM surrogate models were implemented using scikit-learn's `LinearRegression` class. The un-transformed input variables describe only a first-order polynomial; second- and third-order polynomials were created by transforming the input variable array using the `PolynomialFeatures` class.

The GPM surrogate models were implemented using the `GaussianProcessRegressor` class. The squared exponential kernel is modeled using the `RBF` class, the Matérn kernel uses the `Matern` class, and the rational quadratic kernel uses the `RationalQuadratic` class. Each kernel was multiplied by a `ConstantKernel`, added to a second `ConstantKernel`, and added to a `WhiteKernel`. The two constant kernels help the GPM adapt to the specific mean and scale of the response variables. The white noise kernel allows the GPM to account for noise by not requiring a perfect fit through every training point. Noise is produced by minor variations in the NDARC and RCAS trim solutions due to the tolerance in the trim solution convergence criteria. The Gaussian process models were trained using five restarts of the L-BFGS-B hyperparameter optimizer from random initial conditions to increase the chances of finding the global optimum.

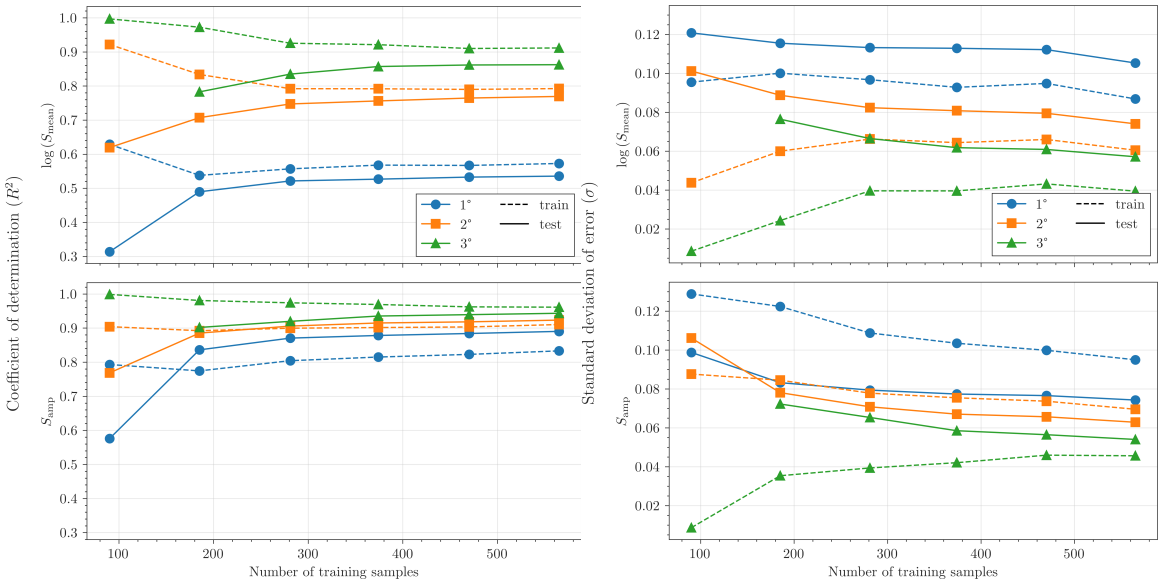
The shallow and deep artificial neural networks were implemented using the `MPLRegressor` class. The training process was configured using the L-BFGS solver, which works well on small training sets and has far fewer hyperparameters to tune than other solvers. The regularization parameter was held constant at 0.01 to minimize differences between the trained models.

Each variation of each model was trained six different times: first, with only the first 102-case CCD DOE, and then with each of the five 100-case augmentation DOEs appended

to the training set.

4.5.4 Results and Analysis

Figure 4.60 presents the learning curve for all studied RSM surrogate models. The results using the coefficient of determination performance metric are plotted in Figure 4.60a; the standard deviation of error metric is plotted in Figure 4.60b. Recall that higher R^2 indicates better performance and lower σ indicates better performance. In Figure 4.60, each score is calculated independently on the training and test set. The training set data is indicated with dashed lines, and the test set data is indicated with solid lines. Because the test set is held out during the training process, it provides a better assessment of how the model will perform with data that it has not "seen" before.



(a) Coefficient of determination (b) Standard deviation of error

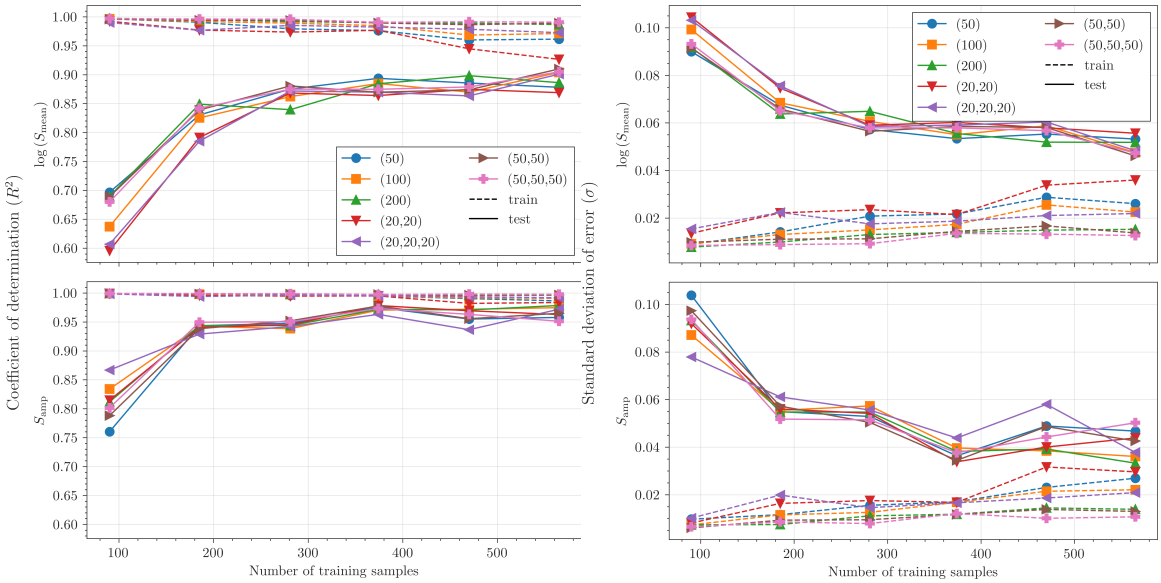
Figure 4.60: Learning curves for RSM surrogate models.

Figure 4.60 demonstrates the overall poor performance of RSM in this application. The first- and second-order RSEs do not achieve good performance by either metric, especially in the case of the mean stress response. Even with the logarithmic transform, S_{mean} is distributed less uniformly than S_{amp}, which complicates training for linear models.

These models also show evidence of overfitting with a lower amount of training data, as indicated by training scores which are much higher than corresponding testing scores. Although these models fit the training data very well, they will not be generalizable to new data. In the case of the third-order RSE, the R^2 and σ scores for the test set with only the first DOE included in the training data are so poor they are beyond the limits of the ordinate axes.

In general, as the amount of training data increases, the fit improves, although returns diminish rapidly. When all available training data is included, third-order RSE performs respectably when predicting the S_{amp} response, but does not quite reach the objectives for the S_{mean} response.

Figure 4.61 plots the learning curve for all shallow and deep ANN models described in Table 4.9. Shallow models are indicated by architectures with a single hidden node layer; deep models have multiple hidden layers between the input and output layers.



(a) Coefficient of determination (b) Standard deviation of error

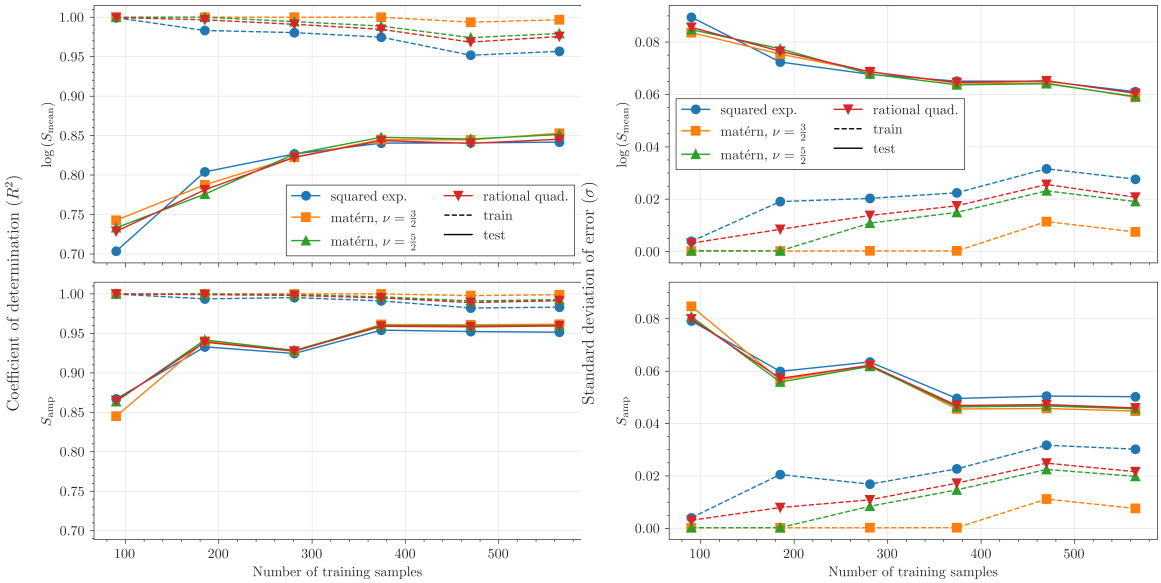
Figure 4.61: Learning curves for ANN surrogate models.

The results for the ANN surrogate models are more difficult to distinguish than those of the RSM models. There are not any obvious clear best performers across all training

data amounts, and the performance does not monotonically improve as more training data is added. There are two possible reasons for these phenomena. First, there is inherent stochasticity in the ANN training network, which adds an element of randomness to the results.

Second, some of the training DOEs may include certain outlying points that could negatively impact the performance of the model. Specifically, the second and fourth augmentation DOEs seem to slightly degrade model performance. This effect appears more prominent in the deep neural networks, which are more prone to overfitting due to the large number of parameters that must be tuned. Overall, the ANN models seem to be more sensitive to training data outliers than the RSM models.

Nevertheless, the overall performance of ANN models is much better than RSM models, especially for the S_{mean} response. Specifically, the (200) shallow network and the (50, 50) deep network show the best performance. The performance of these models plateaus after the third augmentation set is added, unlike the RSM models which continue to improve with the addition of more training data.



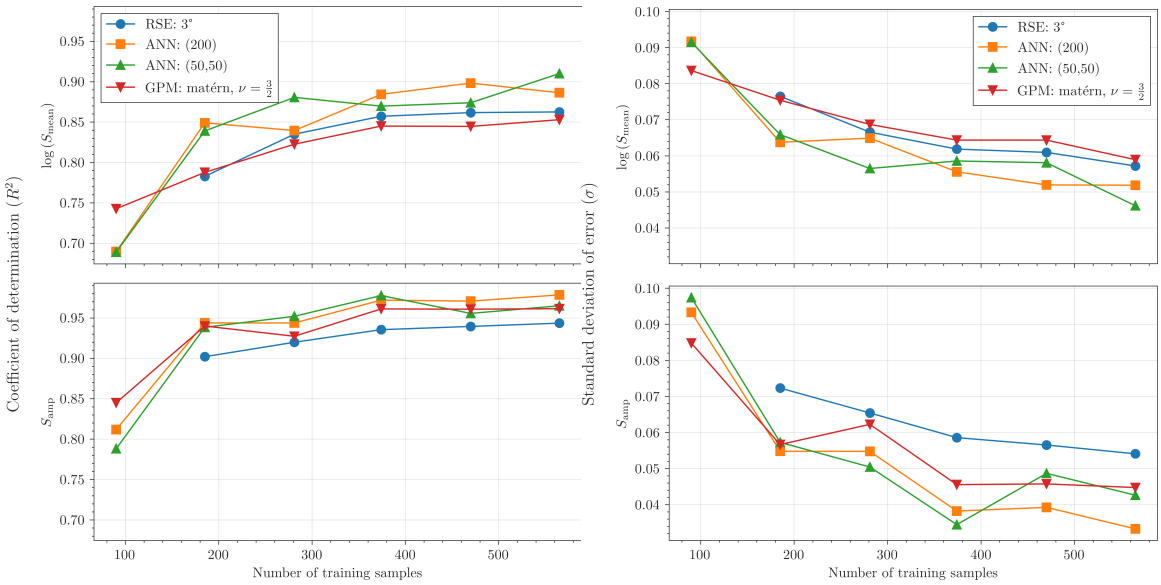
(a) Coefficient of determination (b) Standard deviation of error

Figure 4.62: Learning curves for GPM surrogate models.

The learning curves for the GPM surrogate models are plotted in Figure 4.62. The performance of all the models is very similar. The GPM models experience similar overfitting issues to the RSM models with small amounts of training data, especially in the S_{mean} response. Similarly to ANN, they also show performance degradation in response to the addition of the second augmentation DOE. The overall performance for the S_{amp} response is acceptable, but the performance for the S_{mean} response is significantly worse than ANN.

In particular, the squared exponential covariance functions shows the worst performance, which is especially visible in Figure 4.62b. The Matérn ($\nu = 3/2$) covariance function performs slightly better than all others in both responses according to both scoring metrics. Like the (200) and (50, 50) ANN models, the performance of these models flattens after the third augmentation DOE is introduced.

Figure 4.63 plots the same learning curves for the best performing model of each architecture. Specifically, the third-degree RSM, 200-node shallow ANN, (50,50) deep ANN, and Matérn ($\nu = \frac{3}{2}$) GPM models are included.



(a) Coefficient of determination (b) Standard deviation of error

Figure 4.63: Learning curves for best-performing models of each method.

The ANN models show a clear score advantage on the S_{mean} response, where both RSM

and GPM struggle. The ANN and GPM models show similar performance when fitting the S_{amp} response, while the RSM model lags behind.

The deep ANN model shows somewhat inconsistent performance compared to the shallow ANN model. The performance oscillates as the amount of training data increases, while the performance of the shallow model plateaus. Because the deep neural network has many more parameters that must be tuned during the training process, it is more susceptible to overfitting than the shallow ANN, especially since the overall number of training samples used is relatively small. Thus, the shallow ANN model is preferred.

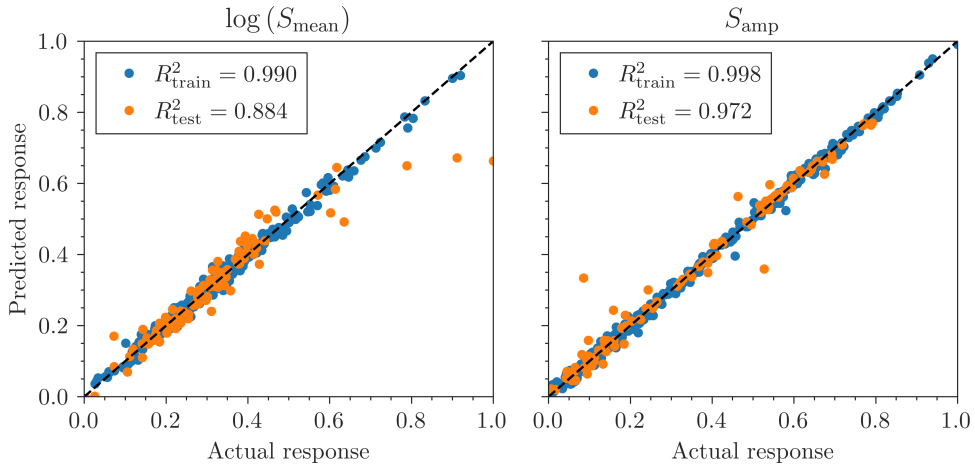
4.5.5 Summary

Overall, the shallow 200-node ANN demonstrated the best fit to both responses using both performance metrics. The performance of this model starts to plateau as the third augmentation DOE is added, which corresponds to a total of 402 training points before non-converged points were removed. Including the 100 points of the test set, about 15.9 h of runtime was required to produce the data to train and validate this model. Detailed goodness-of-fit plots for this model are included in Figure 4.64.

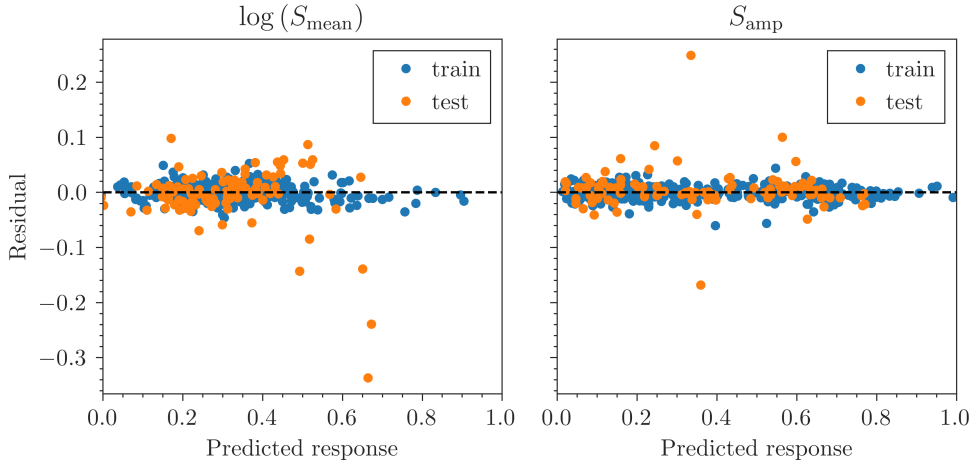
It is also worth considering the best-performing GPM surrogate model, even though it did not fit the data as well as ANN. In this research, GPM has a particular advantage when applied to probabilistic modeling. In Experiment 2b, it will be necessary to quantify the uncertainty in each surrogate model prediction. When making a prediction with a Gaussian process model, the model returns estimates of the mean response and the standard deviation of error at that point (see Section 4.1.3). Thus, if the uncertainty in the response is assumed to be normally-distributed, it can be modeled as in Equations (4.59) and (4.60):

$$S_{\text{mean}} \sim \mathcal{L}\left(\mu_\ell = c_m + k_m \mu_{S_{\text{mean}}}, \sigma_\ell = k_m \sigma_{S_{\text{mean}}}\right) \tag{4.59}$$

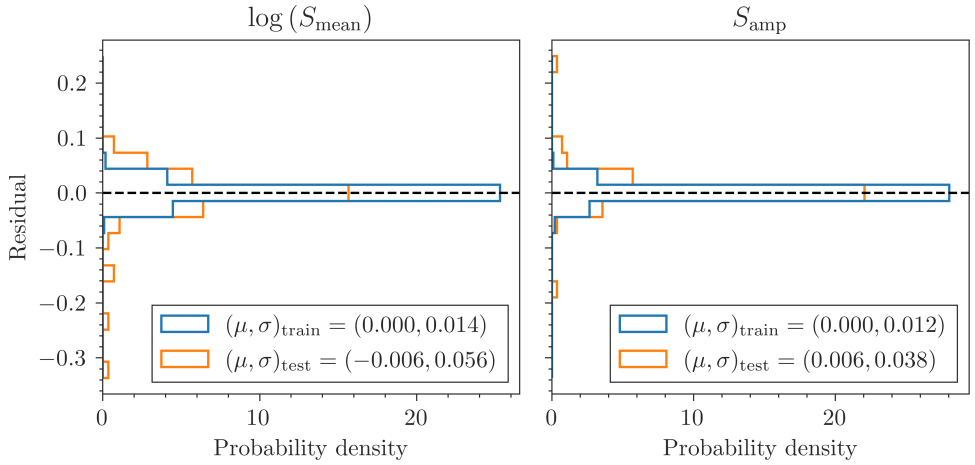
$$S_{\text{amp}} \sim \mathcal{N}\left(\mu = c_a + k_a \mu_{S_{\text{amp}}}, \sigma = k_a \sigma_{S_{\text{amp}}}\right) \tag{4.60}$$



(a) Actual vs. predicted



(b) Residual vs. predicted



(c) Model fit error (left) and model representation error (right)

Figure 4.64: Goodness-of-fit plots for the 200-node shallow ANN model.

where $\mathcal{L}$ is the lognormal distribution; μ_ℓ and σ_ℓ are the mean and standard deviation, respectively, of the underlying normal distribution; c and k are constants and scaling factors, respectively, associated with the scaling of the response to the $[0, 1]$ range; and $\mu_{S_{\text{mean}}}$, $\sigma_{S_{\text{mean}}}$, $\mu_{S_{\text{amp}}}$, and $\sigma_{S_{\text{amp}}}$ are the mean and standard deviation of the predictions returned by the GPM at a specific point. Note that S_{mean} is modeled using a lognormal distribution because that response was logarithmically transformed to improve the surrogate model fit.

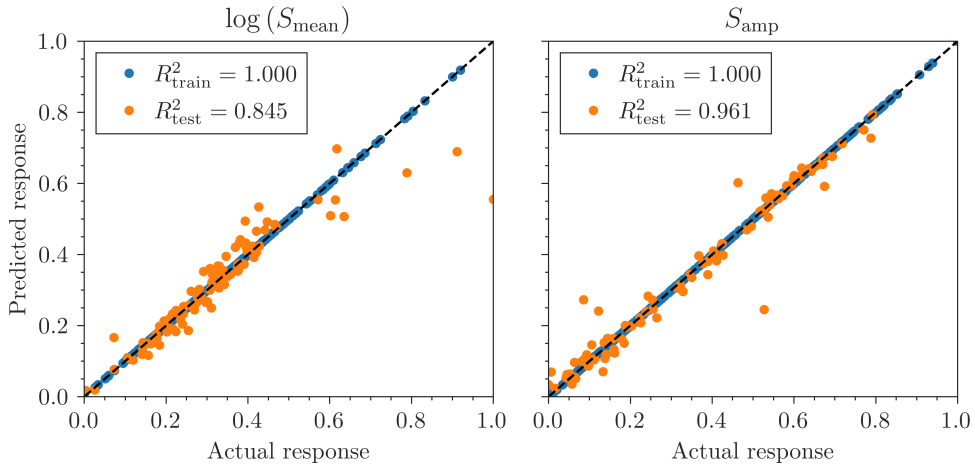
The uncertainty associated with a GPM prediction can be quantified *locally*: σ may be higher or lower depending on how confident the model is in that region of the flight envelope space. Comparatively, the uncertainty associated with an ANN prediction can only be quantified *globally*: uncertainty is estimated using the standard deviation of the test set prediction error and applied equally to all predictions, as in Equations (4.61) and (4.62):

$$S_{\text{mean}} \sim \mathcal{L}\left(\mu_\ell = c_m + k_m \mu_{S_{\text{mean}}}, \sigma_\ell = 0.056 k_m\right) \tag{4.61}$$

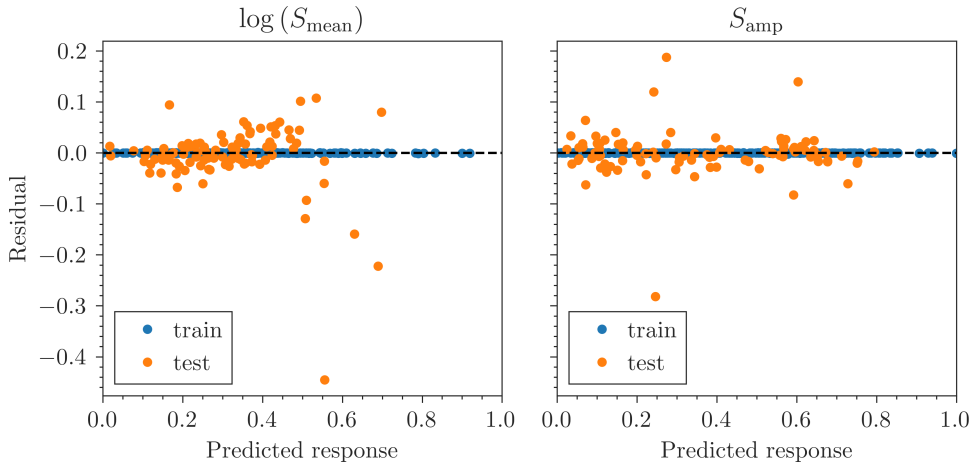
$$S_{\text{amp}} \sim \mathcal{N}\left(\mu = c_a + k_a \mu_{S_{\text{amp}}}, \sigma = 0.038 k_a\right) \tag{4.62}$$

Note that the coefficients on σ_ℓ and σ correspond to the standard deviation of model representation error presented in Figure 4.64c. If σ_ℓ and σ are consistently larger in the ANN model than the GPM model, then any probabilistic results produced using the ANN model may be overly conservative.

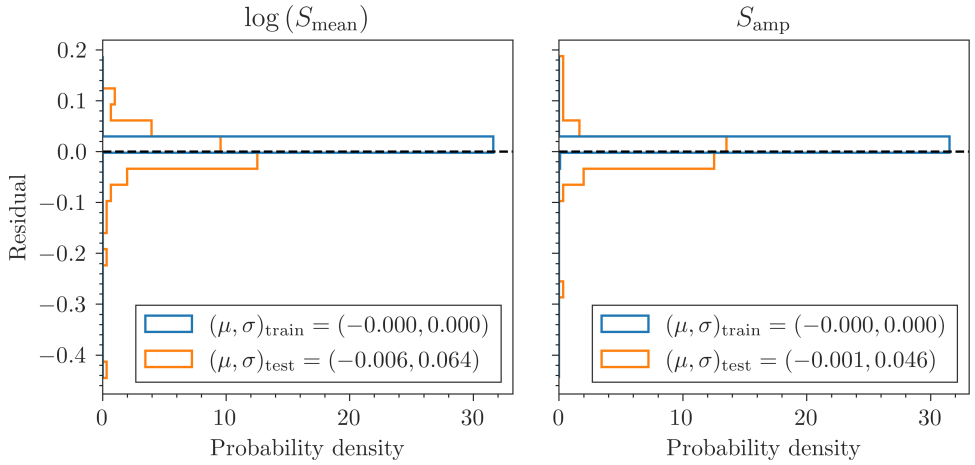
Thus, it is worthwhile to also retain the best-performing GPM surrogate model and compare the results further in subsequent experiments. Detailed goodness-of-fit plots for the Matérn ($\nu = {}^3\!/_2$) model are presented in Figure 4.65.



(a) Actual vs. predicted



(b) Residual vs. predicted



(c) Model fit error (left) and model representation error (right)

Figure 4.65: Goodness-of-fit plots for the Matérn ($\nu = 3/2$) GPM model.

4.6 Conclusions

Experiment 1 examined the possibility of using surrogate modeling to predict fatigue load components in arbitrary flight conditions, with the goal of improving upon the slow execution speed of complex comprehensive codes. Experiment 1a described the development of a rotorcraft multidisciplinary analysis environment, the implementation of a generic single-main-rotor helicopter model in that environment, and an extreme flight condition survey which was used to locate the critical fatigue point on the rotor blade. The extreme flight condition survey also served as a rudimentary sensitivity analysis for six flight envelope variables. Experiment 1b compared a large number of different surrogate modeling methods and architectures to determine which was the most appropriate for predicting fatigue load components across the entire flight envelope.

Recall Hypothesis 1:

Hypothesis 1

At least one of the surveyed surrogate modeling methods can be used to derive a complete load spectrum from comprehensive analysis results, enabling rapid design space exploration.

Ultimately, Hypothesis 1 was *supported* by the results of Experiment 1. Two suitable surrogate modeling methods were identified and retained for further comparison in subsequent experiments. Moreover, ANN surrogate models achieved satisfactory performance with only 502 load samples in total. Because probabilistic analyses can require hundreds of thousands or even millions of load samples, it is likely that a significant speed advantage has been realized. This advantage becomes even more notable if more accurate and costly comprehensive analyses, or coupled CFD–CSD analyses, are used in the MDA environment.

Thus, the preliminary fatigue design methodology, previously presented in Figure 3.1, can now be updated based on the results of Experiment 1.

Figure 4.66 reflects that ANN or GPM surrogate modeling can be used to effectively

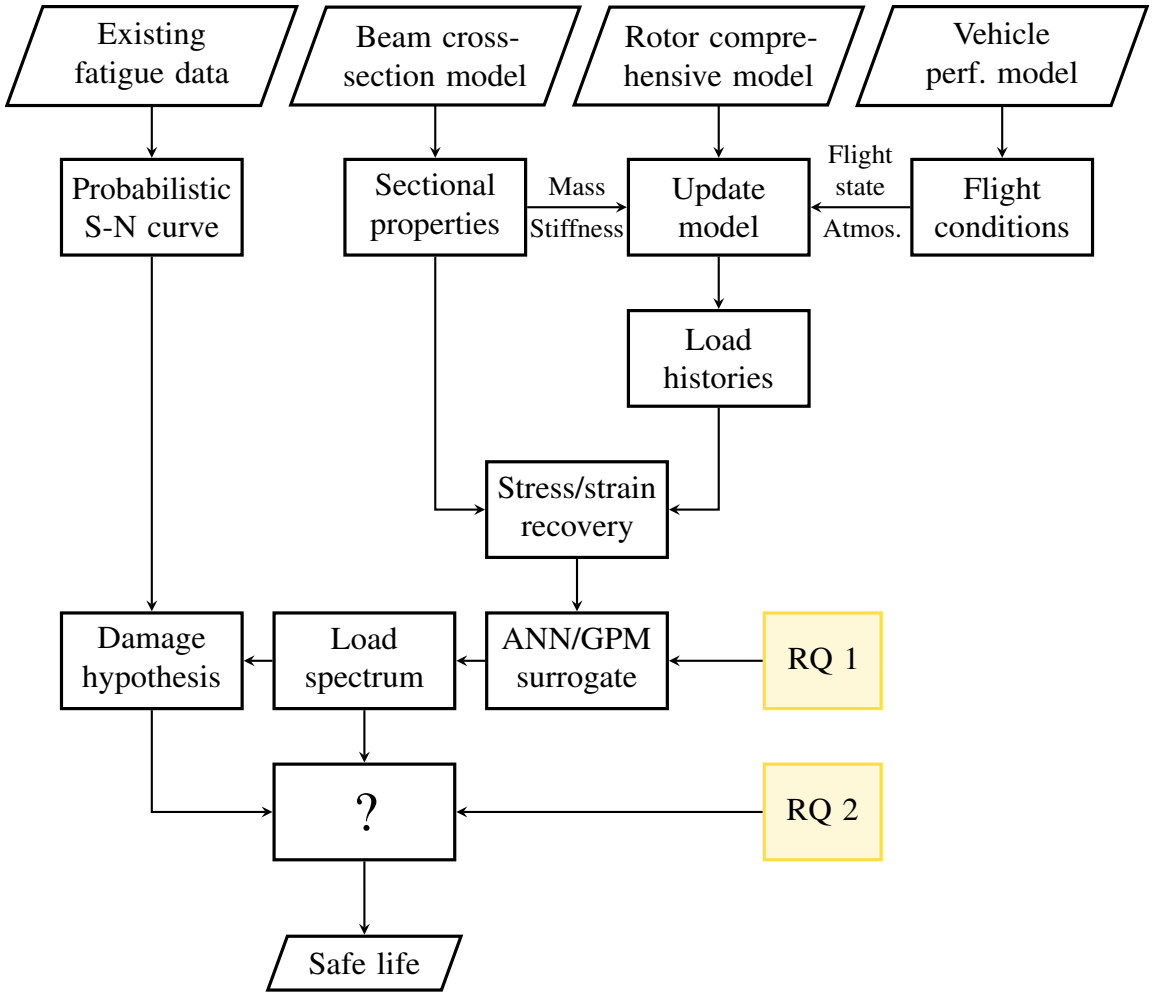


Figure 4.66: Updated flowchart of the preliminary fatigue design methodology after Experiment 1.

capture the knowledge of the MDA with regards to fatigue loads throughout the flight envelope.

CHAPTER 5

STRUCTURAL RELIABILITY SOLUTIONS TO THE FATIGUE LIFE PROBLEM

Research Question 2 (see Section 3.3) asks if it is possible to remove the need for reductions and safety factors by shifting the traditional safe life methodology from a deterministic to a probabilistic process. For reference, Research Question 2 is reprinted below.

Research Question 2

How can probabilistic methods be applied to efficiently remove the dependence of traditional fatigue design methodologies on reductions and safety factors?

Wind turbine designers, whose work was discussed previously in Section 1.6, make use of *structural reliability* methods when specifying rotor blade service lives. The discipline of structural reliability focuses on utilizing known or estimated uncertainty in loads, material strength, and modeling accuracy to analytically or numerically predict the reliability of a structural component against ultimate failure or fatigue failure.

Although some authors [78, 79] have applied simple structural reliability techniques such as Monte Carlo simulation to rotorcraft fatigue life predictions, it appears that these methods are not commonly used in the fatigue design process itself [4, 56]. Additionally, Monte Carlo simulation is a crude method of calculating reliability, and may fail when especially low probabilities of failure are necessary [120]. As such, original research is required to determine the applicability of more advanced structural reliability methods to a preliminary fatigue design methodology. Specifically, methods for determining service life for exceptionally low probabilities of failure should be investigated and tested.

This section begins with a brief review of structural reliability methods and solution techniques. Then, a hypothesis to Research Question 2 is formed. Finally, Experiment 2 is conducted to test Hypothesis 2.

5.1 Review of Structural Reliability Methods

The discipline of structural reliability seeks to improve the safety of structures by modeling uncertainty and variability in loads, strength, and geometry inherent to real structures. If treated appropriately, these uncertainties can be used to quantify the reliability of a structure, or the probability that it will survive throughout its intended life without failure. The reliability of a structure is quantified using the probability of survival, P_s, or the probability of failure, P_f. Since the failure state of a structure is typically considered to be binary, $P_f = 1 - P_s$. For example, the six nines reliability requirements specifies $P_s = 0.999999$. Thus, $P_f = 1 - 0.999999 = 1 \times 10^{-6}$. This is an extremely weak probability of failure; the implications of this fact will be discussed later in this section.

This section provides a basic discussion of structural reliability methods and their potential application to safe life prediction. It is not intended to be an exhaustive description of the field. For more detail, the reader is encouraged to review the texts by Ditlevsen and Madsen [121], Lemaire [122], and Mansour [123].

5.1.1 Basic Concepts

In the simplest case, the failure of a structure occurs when the internal stresses, S, in the structure exceed the inherent resistance, R, provided by the material [124]. For example, stresses may be produced due to applied loads, heat, or displacement; the resistance can be represented using the material's yield strength or ultimate strength. This can be represented using the *limit state equation*:

$$G(R, S) = R - S = 0 \tag{5.1}$$

where $G(R, S)$ is the *performance function*. In Equation (5.1), R and S are random variables; a specific outcome of these variables is denoted using lower case letters as r and s, respectively. The distributions of R and S are described by their joint probability density

function, $f_{R,S}(r, s)$. The failure domain, $\mathcal{D}_f$, is given by $G \leq 0$ and the survival domain, $\mathcal{D}_s$, is given by $G > 0$. The limit state surface, $G = 0$, separates the two domains. The objective of the problem is to determine the volume of probability encompassed by the failure domain, P_f. This requires exact or approximate knowledge of the location and shape of the limit state surface.

For example, Figure 5.1 presents a hypothetical simple structural reliability problem. The colored lines are contours of $f_{R,S}(r, s)$; in this case, the joint probability density function is simply a bivariate normal distribution with some correlation between the variables. The failure domain is represented by the grey shaded area below the limit state equation, $G(R, S) = R - S = 0$.

The probability of failure can be calculated by integrating the joint probability density function over the failure domain:

$$P_f = \Pr\left(R - S \leq 0\right) = \int_{r-s\leq 0} f_{R,S}(r, s)\,\mathrm{d}r\,\mathrm{d}s \tag{5.2}$$

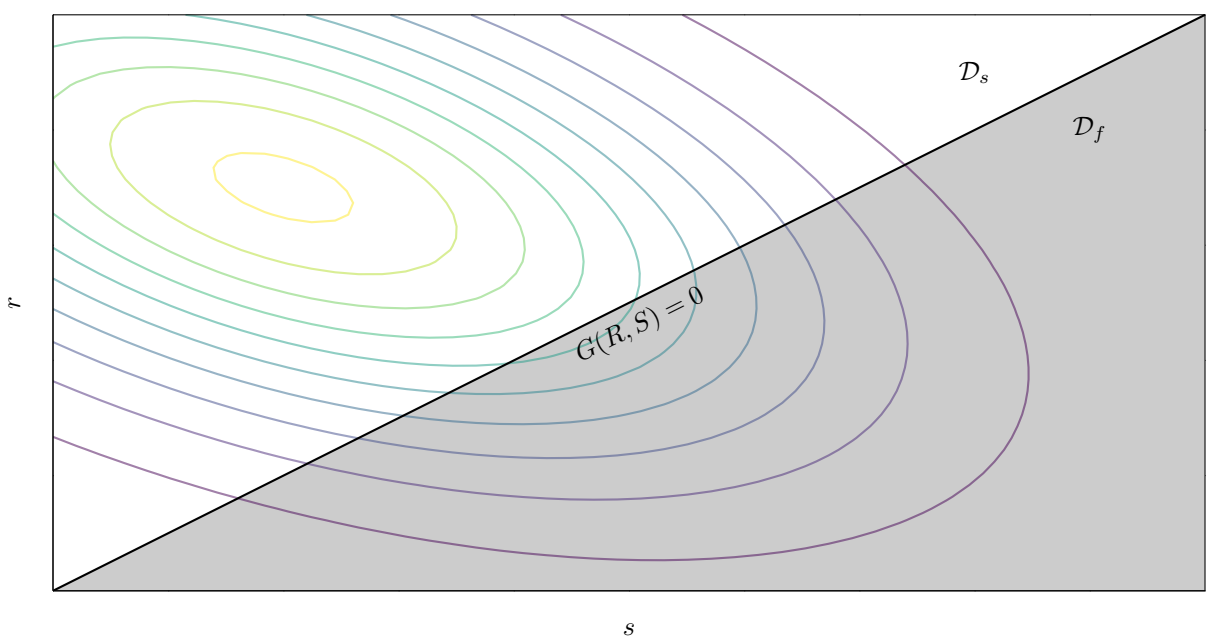


Figure 5.1: A hypothetical simple structural reliability problem. The colored lines are contours of $f_{R,S}(r, s)$ and the grey shaded area is the failure domain.

This equation is difficult to integrate analytically except in extremely simple cases, such as when R and S are independent or when one of the variables is deterministic. For more complex problems, Equation (5.2) can be generalized to a vector of random variables, $\boldsymbol{X} = [X_1, X_2, \ldots, X_n]^\top$. Then,

$$P_f = \Pr\left(G(\boldsymbol{X}) \leq 0\right) = \int_{G(\boldsymbol{x}) \leq 0} f_{\boldsymbol{X}}(\boldsymbol{x})\, \mathrm{d}x_1\, \mathrm{d}x_2 \cdots \mathrm{d}x_n \tag{5.3}$$

Solving Equation (5.3) is the primary focus of structural reliability analyses. This task is complicated by the following factors:

1. The complete joint probability distribution for $\boldsymbol{X}$ is rarely known. Instead, knowledge may be limited to the marginal distributions and the covariance matrix. In the worst case, knowledge is limited to only the first and second moments (mean and standard deviation) of each random variable distribution.
2. The analytical form of performance function is rarely known. Instead, $G(\boldsymbol{X})$ may be a mechanistic model, such as a FEA code.
3. Numerical integration of Equation (5.3) will likely produce errors on the same order as P_f, especially for very weak probabilities.

This drives the development of approximate solution methods and sampling methods, which will be discussed further later in this section.

These solution methods rely on the calculation of a quantity known as the *reliability index*, represented by β. Determining β first requires the transformation of each random variable X_i into independent standard normal variables, denoted U_i. This is known as an *isoprobabilistic* transformation, T_{IP}. In the case of normal variables, this can be accomplished using

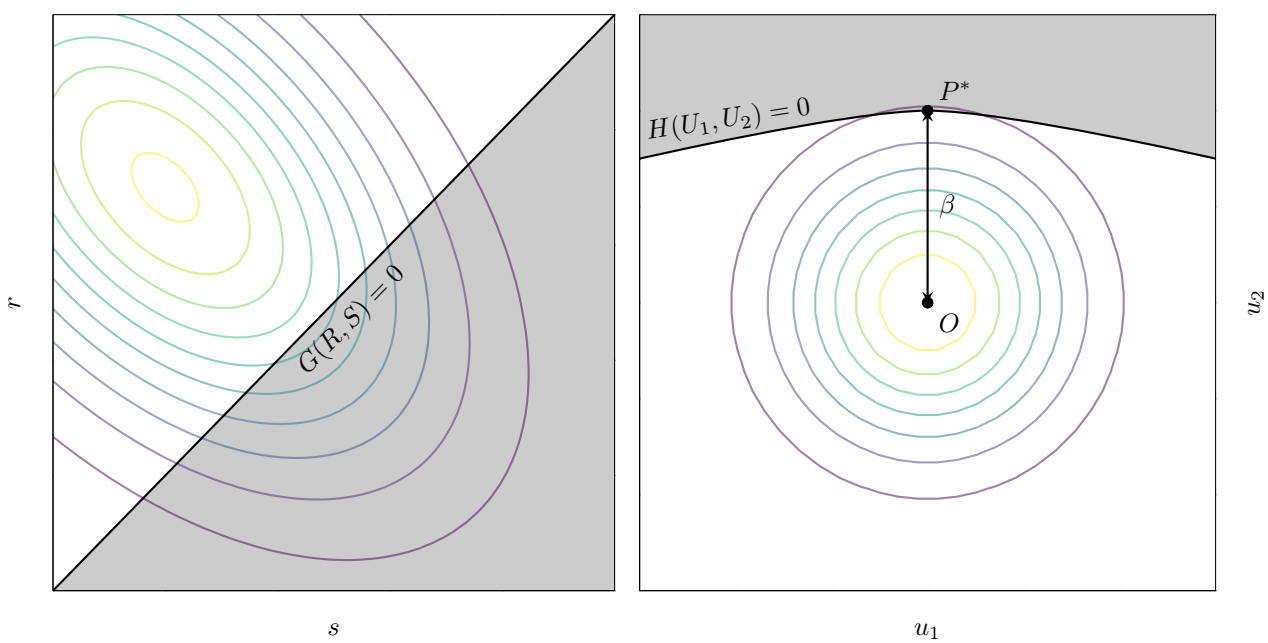
$$T_{\mathrm{IP}}(x_i) = u_i = \frac{x_i - \mathrm{E}[X_i]}{\mathrm{Std}(X_i)} \tag{5.4}$$

For other distributions, one may use the inverse transformation theorem as follows:

$$T_{\mathrm{IP}}(x_i) = u_i = \Phi^{-1}\left(F_{X_i}(x_i)\right) \tag{5.5}$$

where Φ is the cumulative distribution function of the multivariate standard normal. Of course, this transformation is only possible if the cumulative distribution function, $F_{X_i}(x_i)$, is known.

In other cases, a number of approximate isoprobabilistic transformations are available. Depending on the level of knowledge regarding the distributions of X_i, one may use the Rosenblatt, Nataf, or Hermite transformations. If the variables are dependent, one may decorrelate the variables using information from the correlation matrix. These methods are not discussed at length here; more detail is given by Lemaire [125]. A notional isoprobabilistic transformation of the system depicted in Figure 5.1 is presented in Figure 5.2.



(a) Before transformation

(b) After transformation

Figure 5.2: Notional isoprobabilistic transformation.

Note that after the isoprobabilistic transformation is applied, the limit state surface is

represented as $H(\boldsymbol{U}) = 0$ rather than $G(\boldsymbol{X}) = 0$. Equation (5.3) becomes

$$P_f = \Pr\big(H(\boldsymbol{U}) \leq 0\big) = \int_{H(\boldsymbol{u})\leq 0} \phi(\boldsymbol{u})\,\mathrm{d}u_1\,\mathrm{d}u_2 \cdots \mathrm{d}u_n \tag{5.6}$$

where ϕ is the probability density function of the multivariate standard normal. In cases where the standardized limit state surface is linear, this equation can be evaluated analytically.

In general, the probability of failure must be calculated by first determining the reliability index. The reliability index is defined as the shortest Euclidean distance between the origin and the limit state surface in the $\boldsymbol{u}$-space. The nearest point on the limit state surface to the origin is known as the *most-probable failure point* or the *standard design point* and is denoted P^*. Thus, finding the reliability index is a constrained optimization problem that can be stated as

$$\begin{aligned} \beta = \min_{\boldsymbol{u}} \quad & \sqrt{\boldsymbol{u}^\top \boldsymbol{u}} \\ \text{s.t.} \quad & H(\boldsymbol{u}) = 0 \end{aligned} \tag{5.7}$$

Standard constrained optimization techniques are applicable to this problem.

For example, in the simple $R-S$ case discussed previously, if both variables are assumed to be normally distributed and independent, the standardized variables, u_R and u_S, can be found using Equation (5.2). Then, the limit state equation becomes

$$u_R\,\mathrm{Std}(R) - u_S\,\mathrm{Std}(S) + \mathrm{E}[R] - \mathrm{E}[S] = 0 \tag{5.8}$$

The shortest distance from the limit state surface to the origin is found analytically using

$$\beta = \frac{\mathrm{E}[R] - \mathrm{E}[S]}{\sqrt{[\mathrm{Std}(R)]^2 + [\mathrm{Std}(S)]^2}} \tag{5.9}$$

In this case, the limit state surface is linear and the isoprobabilistic transformation allows

Equation (5.3) to be integrated analytically to find

$$P_f = \Phi(-\beta) \tag{5.10}$$

Note that this equation is not generally applicable, although it does provide a first-order estimate of P_f in some of the methods to be discussed subsequently.

5.1.2 Approximate/Analytical Methods

The previously-described example is one of the few cases where the exact probability of failure can be determined analytically. In other cases, P_f can be analytically approximated by simplifying the form of the problem formulation. These methods require that the isoprobabilistic transformation can be applied and the reliability index can be found.

5.1.2.1 First-Order Reliability Method

The first-order reliability method (FORM) approximates the limit state surface using a hyperplane tangential to the actual surface at P^* [126]. The approximate limit state surface, $\widehat{H}(\boldsymbol{u})$, is constructed as

$$\widehat{H}(\boldsymbol{u}) = \boldsymbol{\alpha}^\top \boldsymbol{u} + \beta \tag{5.11}$$

where $\boldsymbol{\alpha}$ is the vector of direction cosines between P^* and the origin of the standardized space. Then, the probability of failure can be approximated using

$$\widehat{P_f} = \Phi(-\beta) \tag{5.12}$$

which is exact if the true limit state surface is linear.

Note that the accuracy of the FORM result depends on the curvature of the limit state surface. If the limit state surface curves away from the origin of the standardized space, the FORM result will likely overestimate the true probability of failure. However, if the surface

curves towards to the origin, the approximate result will underestimate the true probability of failure, sometimes by several orders of magnitude in special cases. Lemaire states that FORM typically provides an estimate of P_f on the same order as the true value for simple problems. However, the result should be validated using another method if possible.

5.1.2.2 Second-Order Reliability Method

The second-order reliability method (SORM) approximates the limit state surface using a quadratic surface that osculates the true limit state surface at P^* [126]. This requires knowledge of the curvature of the limit state surface at the most-probable failure point, which generally must be approximated numerically. SORM approximations typically take the form of corrections to the FORM result, and a number of different forms exist.

A commonly used SORM correction is the Breitung formula:

$$\widehat{P_f} = \Phi(-\beta) \left(\prod_{i=1}^{n-1} \frac{1}{\sqrt{1+\beta\kappa_i}} \right) \tag{5.13}$$

where κ_i are the approximate principal curvatures of the limit state surface at P^*. Note that Equation (5.13) is essentially the same as Equation (5.12) with a correcting coefficient based on the curvature. Other forms of SORM include the Hohenbichler approximation and the Tvedt approximation, which are described in detail by Lemaire [126]. Although SORM generally improves upon FORM results, Lemaire notes that in special cases SORM can be less accurate, so care must be taken in its application.

5.1.3 Sampling/Simulation Methods

Sampling methods, or simulation methods, are those that rely on sampling from the input random variable distributions and calculating the resulting value of the performance function [120]. The probability of failure is estimated from the proportion of cases for which $G(\boldsymbol{x}) \leq 0$. This techniques are useful in cases where constructing the isoprobabilistic trans-

formation, finding the reliability index, or applying the FORM and SORM approximations is difficult or impossible, but are generally considered to be more expensive and less accurate.

Some sampling methods incorporate reliability information from the approximate solution methods to speed their convergence. In this section, a selection of sampling methods are discussed in order from least to most reliability information required.

5.1.3.1 Monte Carlo Simulation

Monte Carlo simulation (MC) is perhaps the simplest and most widely-applicable sampling method available [120]. As discussed in Section 2.3.4, this technique has been applied to assess the fatigue life reliability produced by the traditional safe life methodology [79]. As a reliability solution method, MC is characterized by its minimal preparation requirements. One only needs to be able to evaluate the performance function, $G(\boldsymbol{X})$, and sample from the random variable distributions. These samples are termed $\boldsymbol{x}$. It is not necessary to perform the isoprobabilistic transformation or find the most-probable failure point.

In Monte Carlo simulations, Equation (5.3) is approximated by introducing a failure indicator function, $\boldsymbol{I}_f$, such that

$$\boldsymbol{I}_f(\boldsymbol{x}) = \begin{cases} 1, & \text{if } G(\boldsymbol{x}) \leq 0 \\ 0, & \text{if } G(\boldsymbol{x}) > 0 \end{cases} \tag{5.14}$$

Essentially, the indicator function counts the number of cases in a simulation that exist within the failure domain. Then, for N samples, the probability of failure can be approximated using

$$P_f = \int_{\mathbb{R}_n} \boldsymbol{I}_f(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) \, \mathrm{d}x_1 \, \mathrm{d}x_2 \cdots \mathrm{d}x_n = \mathrm{E}[\boldsymbol{I}_f] \tag{5.15}$$

$$\mathrm{E}[\boldsymbol{I}_f] \approx \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{I}_f(\boldsymbol{x}_i) = \widehat{P_f} \tag{5.16}$$

An example MC is presented in Figure 5.3. The reliability of the system presented previously in Figure 5.1 was estimating by producing 1000 samples of the bivariate normal distribution and determining failure by evaluating the performance function at each point. In total, 155 of 1000 samples fall in the failure domain; thus, $\widehat{P_f} = 0.155$.

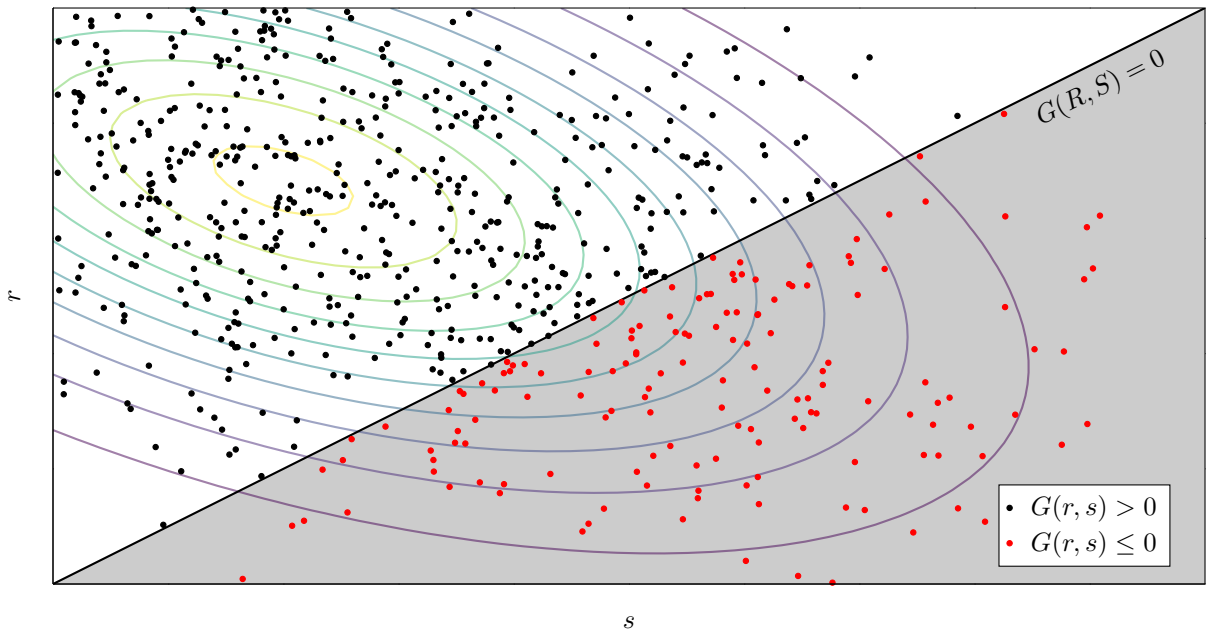


Figure 5.3: Demonstration of reliability calculations using Monte Carlo simulation.

The variance of $\widehat{P_f}$ can be estimated using

$$\mathrm{Var}(\widehat{P_f}) \approx \frac{1}{N}\widehat{P_f}(1 - \widehat{P_f}) \tag{5.17}$$

leading to an approximate coefficient of variation (COV) of

$$\text{COV} \approx \sqrt{\frac{1 - \widehat{P_f}}{N\widehat{P_f}}} \tag{5.18}$$

For example, the MC depicted in Figure 5.3 has a COV of approximately 0.0738.

Knowledge of the COV is valuable because it is used to quantify the confidence in the Monte Carlo estimate of $\widehat{P_f}$. Typically, a value of COV as low as 0.05 to 0.10 is desirable. This dictates the required number of samples. Equation (5.18) indicates that estimating

a probability of failure near 10^{-n} with a COV of 0.1 requires approximately $N = 10^{n+2}$ samples. This in turn drives the runtime of the Monte Carlo simulation.

5.1.3.2 Directional Simulation

Directional simulations (DS) improve upon MC by reducing the number of samples required to reach a specified COV [120]. This technique requires only that the isoprobabilistic transformation be performed prior to its application.

Although their are several varieties of directional simulation, they share many commonalities [127, 128]. Directional simulation begins by representing the vector of standardized random variables, $\boldsymbol{U}$, as the product of a scalar radius and multidimensional direction:

$$\boldsymbol{U} = R\boldsymbol{\Delta} \tag{5.19}$$

where R^2 is a χ^2_n-distributed random variable and $\boldsymbol{\Delta}$ is uniformly distributed on the unit n-sphere centered on the origin of the standardized space. The basic process for the ith DS sample can be summarized as follows:

1. Generate a radial direction $\boldsymbol{\delta}_i$ by sampling $\boldsymbol{\Delta}$.
2. Search for the solution to the limit state equation, $H(\boldsymbol{U}) = 0$, along $\boldsymbol{\delta}_i$ using a line search method. The solution is found at the radius r_i.
3. Calculate the conditional probability of a sample along this line falling in the failure domain using

$$P_{f,i} = \Pr\left(H(R\boldsymbol{\delta}_i) \leq 0\right) = \Pr(R > r_i) = \Pr\left(1 - \chi^2_n(r_i^2)\right) \tag{5.20}$$

The probability of failure is then estimated using

$$\widehat{P_f} = \frac{1}{N}\sum_{i=1}^{N} P_{f,i} \tag{5.21}$$

and the variance of $\widehat{P_f}$ may then be estimated as

$$\mathrm{Var}(\widehat{P_f}) \approx \frac{1}{N(N-1)} \sum_{i=1}^{N} \left(P_{f,i} - \widehat{P_f} \right)^2 \tag{5.22}$$

The DS method improves upon the Monte Carlo simulation economically because Equation (5.20) can be evaluated analytically. Directional simulation is especially powerful if the limit state surface is near-spherical in the standardized design space. In most cases, directional simulation achieves a certain COV in far fewer samples than Monte Carlo simulation.

5.1.3.3 Importance Sampling

The importance sampling method is similar to Monte Carlo simulation but the sampling process is conditioned on the location of the most-probable failure point [120, 128]. Of course, this requires locating P^* and thus determining the reliability index, so at the very least the FORM solution must be known.

Importance sampling assumes that the majority of the failure probability is located near P^*, so it is more efficient to sample near this point rather than from the complete joint distribution of the random variable vector. The new sampling distribution is termed $\psi(\boldsymbol{u})$. For example, one may sample from the multivariate standard normal centered on P^*. Then, the probability of failure becomes

$$P_f = \int_{\mathbb{R}_n} \boldsymbol{I}_f(\boldsymbol{u}) \frac{\phi(\boldsymbol{u})}{\psi(\boldsymbol{u})} \psi(\boldsymbol{u}) \, \mathrm{d}u_1 \, \mathrm{d}u_2 \cdots \mathrm{d}u_n \tag{5.23}$$

which can be estimated using

$$\widehat{P_f} = \sum_{i=1}^{n} \boldsymbol{I}_f(\tilde{\boldsymbol{u}}_i) \frac{\phi(\tilde{\boldsymbol{u}}_i)}{\psi(\tilde{\boldsymbol{u}}_i)} \tag{5.24}$$

where $\tilde{\boldsymbol{u}}_i$ are samples of the ψ distribution. The variance can be estimated using

$$\mathrm{Var}(\widehat{P_f}) = \sum_{i=1}^{n} \left[\boldsymbol{I}_f(\tilde{\boldsymbol{u}}_i) \frac{\phi(\tilde{\boldsymbol{u}}_i)}{\psi(\tilde{\boldsymbol{u}}_i)} - \widehat{P_f} \right]^2 \tag{5.25}$$

A graphical representation of the importance sampling process is depicted in Figure 5.4. Note that this method appears similar to the Monte Carlo simulation depicted in Figure 5.3, but the samples are drawn from a distribution near the most-probable failure point rather than the joint distribution of the input random variable vector.

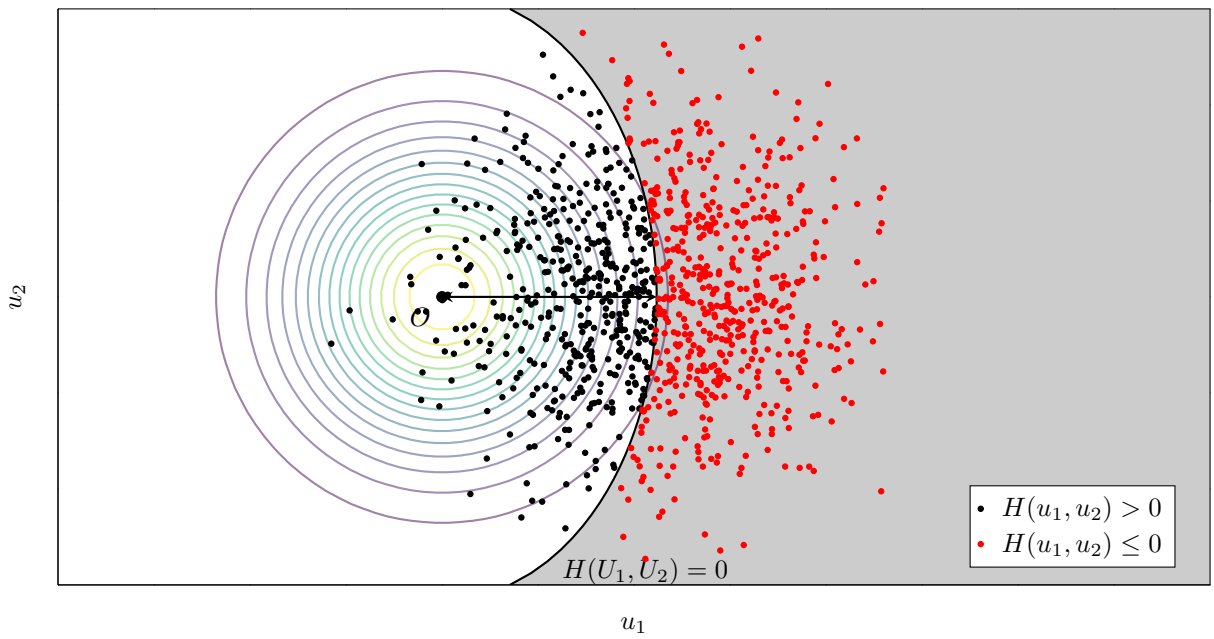


Figure 5.4: Demonstration of reliability calculations using importance sampling.

It is easy to see the advantage of importance sampling from Figure 5.4. Because the $\psi(\boldsymbol{u})$ distribution is located much nearer to the limit state surface, many of the samples fall in the failure domain. This allows one to predict extremely low probabilities of failure with low COV using far fewer samples than Monte Carlo simulation, and possibly even fewer samples than directional simulation. However, application of this technique may be difficult or impossible if the most-probable failure point cannot be readily found.

There are a number of similar methods that involve sampling the standardized space near

the limit state surface, such as conditional sampling, adaptive sampling, and conditional importance sampling. These methods, among others, are described in more detail by Lemaire [120].

5.1.4 Comparison

The methods discussed previously are equally capable of providing an estimate of the probability of failure of a structure. However, the quality of this estimate and the amount of effort required to obtain it varies significantly.

Lemaire conducted a comparison of the approximate and sampling methods using the simple system described in Equation (5.1) with uniformly distributed random variables. The author found that the SORM approximation came incredibly close to the true solution, and the FORM approximation was within the same order of magnitudue. Monte Carlo simulation required over 3000 samples to estimate the probability of failure with a COV of 0.1. Direction sampling required approximately 600 samples, and importance sampling required approximately 400 samples. These numbers do not include the effort required to execute the isoprobabilistic transformation and find P^*.

Of course, these results are applicable only to the specific case investigated by Lemaire, which has a strong, easily-identifiable most-probable failure point. In general, it is difficult to determine which method is best to apply for a particular problem. FORM and SORM results are probably the fastest for simple, low-order systems in which P^* is easily found. More complex systems with high-dimensional, highly-nonlinear performance functions may benefit from the application of sampling methods as it will be too costly to locate the most-probable failure point. However, if the performance function is defined by an expensive external analysis program, or if the probability of failure is especially weak, then crude sampling methods such as Monte Carlo simulation should be avoided due to the high sampling requirement.

In addition to the MC and DS methods described previously, there are a number of

other sampling methods that do not require any analytical reliability information. Latin hypercube simulation (LHS) uses a stratified sampling strategy which covers the random variable ranges more efficiently than crude MC. However, it is only applicable to systems with independent random variable inputs. The quasi-Monte Carlo (QMC) method uses quasi-random low-discrepancy sequences to sample the input random variables, rather than the pseudorandom approach used by crude Monte Carlo. This provides for a faster convergence rate. The details of these methods are not discussed in detail here, but are described at length in many structural reliability resources [121, 128].

In the next section, the application of the aforementioned structural reliability solution techniques to the fatigue life prediction problem will be discussed.

5.2 Hypothesis to Research Question 2

As discussed in Sections 2.2 and 3.4, Miner's sum is the preferred damage hypothesis for the preliminary fatigue design methodology due to its flexibility, relevance, and the minimal amount of experimental data required. Miner's sum is given in Equation (2.5). This equation can be reformulated as a performance function (with a slight change in notation) as follows:

$$G(\boldsymbol{n}, \boldsymbol{N}, \boldsymbol{S}_{\mathbf{eq}}) = 1 - \sum_{i=1}^{k} \frac{n_i}{N_i(S_{\mathrm{eq},i})} \tag{5.26}$$

where

$$\boldsymbol{n} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_i \\ \vdots \\ n_k \end{bmatrix}, \quad \boldsymbol{N} = \begin{bmatrix} N_1(S_{\mathrm{eq},1}) \\ N_2(S_{\mathrm{eq},2}) \\ \vdots \\ N_i(S_{\mathrm{eq},i}) \\ \vdots \\ N_k(S_{\mathrm{eq},k}) \end{bmatrix}, \quad \text{and } \boldsymbol{S}_{\mathbf{eq}} = \begin{bmatrix} S_{\mathrm{eq},1} \\ S_{\mathrm{eq},2} \\ \vdots \\ S_{\mathrm{eq},i} \\ \vdots \\ S_{\mathrm{eq},k} \end{bmatrix}$$

where k is the total number of load spectrum segments, $S_{\mathrm{eq},i}$ is the equivalent load for the ith segment, n_i is the length of the ith segment in cycles, and N_i is the number of cycles to failure at $S_{\mathrm{eq},i}$. Note that the functional relationship between N_i and $S_{\mathrm{eq},i}$ is described by the S-N curve.

In Equation (5.26), $\boldsymbol{n}$, $\boldsymbol{N}$, and $\boldsymbol{S}_{\mathbf{eq}}$ are all vectors of random variables of length k. Thus, the structural reliability problem is $3k$-dimensional. The mean of n_i represents the expected lengths of the ith segment in the load spectrum; its variance represents uncertainty in the segment length brought about by differences in usage between operators. The mean of $N_i(S_{\mathrm{eq},i})$ represents the expected cycles to failure at the ith load level as described by the mean S-N curve; its variance represents uncertainty in the number of cycles to failure due to scatter in the S-N curve. Finally, the mean of $S_{\mathrm{eq},i}$ represents the equivalent load of the ith segment in the load spectrum; its variance represents uncertainty in the load due to scatter in the flight load measurements if an actual loads survey is used, or surrogate model error if a regression model is used as described in Chapter 4. Variance in $S_{\mathrm{eq},i}$ could also represent the degree of uncertainty in comprehensive analysis load predictions if the model has been calibrated to test data and the level error is known.

In the traditional safe life methodology, it is common to reduce the original safe life calculation when establishing component replacement times due to uncertainty in the fatigue life prediction produced by Miner's sum. In the structural reliability field, this is known as *modeling uncertainty*, which acknowledges the fact that any model is at best an approximation of reality and does not produce completely reliable results [121]. Modeling uncertainty can be incorporated into Equation (5.26) by replacing the first term in the performance function by another random variable:

$$G(\boldsymbol{n}, \boldsymbol{N}, \boldsymbol{S}_{\mathbf{eq}}, C) = C - \sum_{i=1}^{k} \frac{n_i}{N_i(S_{\mathrm{eq},i})} \tag{5.27}$$

Here, C is a scalar random variable with an expected value of unity; its variance represents

the level of uncertainty in the Miner's sum prediction.

It may be necessary to repeat the load spectrum a number of times. For example, if the load spectrum represents a single flight with known duration, then the summation term in Equation (5.27) could be multiplied by a deterministic coefficient, r, to represent the desired service lifetime of the component:

$$G(\boldsymbol{n}, \boldsymbol{N}, \boldsymbol{S}_{\mathbf{eq}}, C, r) = C - r \sum_{i=1}^{k} \frac{n_i}{N_i(S_{\mathrm{eq},i})} \tag{5.28}$$

Note that this is only possible due the linear nature of Miner's sum and would not be applicable to fatigue life prediction models in which the order of load cycle application is non-trivial.

The performance function described by Equation (5.28) is much more complex than the simple systems described in Section 5.1. It contains $3k + 1$ random variables, which greatly increases the expense of numerically calculating gradients (and possibly Hessian matrices) required by the optimization problem described by Equation (5.7). Additionally, the random variable N_i is a function of another random variable, $S_{\mathrm{eq},i}$. Further, in the rotorcraft fatigue life application, $S_{\mathrm{eq},i}$ is itself a function of the variables used to define each segment of the load spectrum. Lemaire terms these *compound variables* and notes that they are, in effect, random variables with random distribution parameters, forming a compound distribution [129]. Compound distributions rarely have analytical expressions and instead are implemented using numerical methods such as the previously-discussed sampling methods [130].

Due to the complexity of the Miner's sum performance function, it is possible that approximate methods dependent on finding the most-probable failure point will not be practical. Thus, evaluating the probability of fatigue failure may require the use of sampling methods such as Monte Carlo simulation and directional simulation. Fortunately, the calculations required by Equation (5.28) are relatively simple to accomplish numerically,

especially if surrogate models are used to describe the load spectrum, removing the need for an interface to an external analysis program. This discussion leads to a hypothesis to Research Question 2:

Hypothesis 2

At least one of the surveyed structural reliability sampling methods can be used to efficiently remove the dependence on reductions and safety factors by quantifying the reliability of fatigue life predictions using Miner's sum.

The null hypothesis to Hypothesis 2 can be stated as follows:

> The surveyed structural reliability sampling methods cannot be used to efficiently remove the dependence on reductions and safety factors.

The null hypothesis represents the case in which the traditional safe life methodology performs equivalently to or better than the structural reliability methods discussed previously. This will be assessed by comparing results attained using traditional deterministic methods to those obtained using probabilistic methods. This process will be discussed further in Section 5.3.

5.3 Experiment 2 Overview

In this experiment, the capabilities of different structural reliability techniques will be compared methodically. As discussed previously, Miner's sum is expected to be an especially challenging case due to its high dimensionality, extreme nonlinearity, and compound variables. Thus, selecting an appropriate structural reliability method is critical to the success of the preliminary fatigue design methodology.

Experiment 2 is a multi-part experiment. Experiment 2a will examine the performance characteristics of different structural reliability methods against varying levels of complexity using a notional fatigue reliability problem. In Experiment 2b, results from Experiment 2a will be verified using data and models from the generic SMR helicopter implemented in Experiment 1.

5.4 Experiment 2a

Figure 5.5 provides an overview of Experiment 2a. The following sections detail the design of the experiment, implementation of the notional fatigue reliability problem and structural reliability solution methods, results, and conclusions.

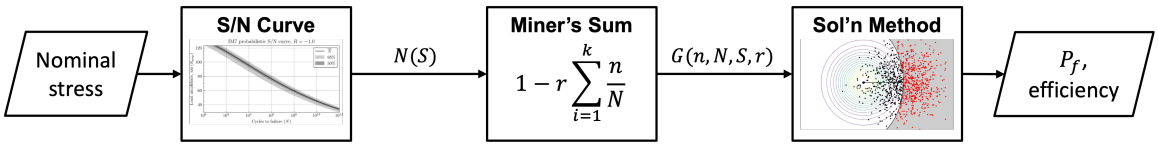


Figure 5.5: Overview of Experiment 2a.

5.4.1 Experimental Design

Experiment 2a is concerned with determining the capability of different structural reliability solution methods to calculate results for increasingly complex problems. The methods under consideration are the MC, DS, LHS, QMC, FORM, SORM, and FORM-IS techniques described in Section 5.1. The experimental environment described in Section 4.4.2 will not be used. Instead, a simplified flexible testing environment with a notional fatigue life problem will be used to reduce runtime while allowing more control over the variables. The Miner's sum performance function from Equation (5.28) is simplified by setting $C = 1$ and $n = 1$, resulting in Equation (5.29):

$$G(\boldsymbol{N}, \boldsymbol{S}_{\mathbf{eq}}, r) = 1 - r\sum_{i=1}^{k}\frac{1}{N_i(S_{\mathrm{eq},i})} \tag{5.29}$$

In Equation (5.29), the *dimensionality*, or the number of random variables in the problem, is k. Increasing dimensionality increases the difficulty of the problem for both analytical and sampling solution methods. In analytical methods, dimensionality dictates the size of the space that is subject to isoprobabilistic transformation and numerical optimization. In sampling methods, dimensionality is related to the number of individual distributions that

must be sampled at each step of the algorithm. Some sampling algorithms additionally require transformations of each distribution to find the next sampling point.

The total number of load cycles in Equation (5.29) is rk since each load is applied for only one cycle. Problems with fewer numbers of load cycles will be less likely to fail. This is expected to make the problem more difficult to solve, especially for crude sampling methods like MC. Because sampling problems estimate P_f by counting the number of iterations that fail and comparing that to the number of iterations that survive, low P_f problems will be more difficult to solve with an appropriate level of confidence because many more iterations are required to produce a satisfactory number of failed iterations.

Table 5.1 describes the different values of the k and rk parameters that will be tested in Experiment 2a. Each parameter is varied between its minimum and maximum values in a logarithmic fashion. Eight steps are used for k and seven steps are used for rk, resulting in a total of 56 structural reliability problems that must be solved for each of the seven solution methods identified previously.

Table 5.1: Structural reliability problem parameters for Experiment 2a.

Parameter	Symbol	Min.	Max.	Steps	Notes
Dimensionality	k	1	200	8	$k = 1$ represents constant amplitude loading, $k = 200$ represents a complex load spectrum
Number of cycles	rk	10^6	10^9	7	$rk = 10^6$ corresponds to $P_f \approx 10^{-5}$, $rk = 10^9$ corresponds to $P_f \approx 5 \times 10^{-1}$

Note that the minimum and maximum values for rk were determined after the notional reliability problem was implemented; this will be described further in Section 5.4.2.

A number of performance metrics will be used to assess each solution. First, the results will be compared to the exact solution in simpler cases where solving the exact solution is feasible. This process will be described further in Section 5.4.3. The expense of each structural reliability method will be determined by recording the total number of calls of

the performance function, the total number of samples of the notional stress distribution, and the wall-clock runtime required to produce a converged solution. These performance metrics are summarized in Table 5.2.

Table 5.2: Structural reliability solution performance metrics for Experiment 2a.

Metric	Objective	Notes
Accuracy	Maximize	Compared to exact solution; only possible in simple cases
Performance function calls	Minimize	Related to iterations of solver
Stress distribution samples	Minimize	Performance function calls multiplied by k
Runtime	Minimize	Wall-clock time for each solution

Of particular interest is the values of these performance metrics in the weak P_f and high dimensionality region. Because extremely low probabilities of failure are required for civil and military rotorcraft designs, determining which solution method can accurately and efficiently solve this type of problem is the primary objective of this experiment. The best-performing structural reliability method(s) will be validated in Experiment 2b and used in subsequent experiments.

5.4.2 Notional Fatigue Reliability Problem

The notional fatigue reliability problem defined by Equation (5.29) requires a S-N curve to define the relationship between $S_{\text{eq},i}$ and N_i. As described in Section 3.4, this research will use a probabilistic S-N curve to capture uncertainty in the material's fatigue resistance. A probabilistic S-N curve formulation described by Freire Júnior and Belísio [131] was used in this research.

Freire Júnior and Belísio define the *mean* S-N curve using a standard log-log relationship, as in Equation (5.30):

$$\log S = A - B\left(log\overline{N}\right)^C \tag{5.30}$$

where A, B, and C are constants related to a specific material that can be derived experimentally, S is the fatigue load amplitude, and $\overline{N}$ is the mean fatigue life corresponding to S.

Note that S can be a deterministic scalar or a random variable, and that $\log$ is the base-10 logarithm.

Equation (5.30) can be rearranged to describe the distribution of $\log \overline{N}$ given a random variable S, as in Equation (5.31):

$$\log \overline{N} \sim \left(\frac{A - \log S}{B} \right)^{\frac{1}{C}} \tag{5.31}$$

Next, the fatigue life distribution given uncertainty in material strength, N, is defined using

$$\frac{N}{\overline{N}} \sim \mathcal{W}\left(\alpha, \beta, \gamma\right) \tag{5.32}$$

where $\mathcal{W}$ is a Weibull distribution with shape parameter α, scale parameter β, and location parameter γ.

Because a number of different distributions are referred to by the common name "Weibull", Equation (5.33) defines the probability density function (PDF) of the specific distribution used in this research:

$$f_X(x) = \frac{\alpha}{\beta} \left(\frac{x - \gamma}{\beta} \right)^{\alpha - 1} \exp\left[- \left(\frac{x - \gamma}{\beta} \right)^{\alpha} \right], \quad x \in [\gamma, +\infty) \tag{5.33}$$

This model was implemented using material data derived from fatigue loading experiments of IM7 coupons. Data for the constants in Equations (5.30) and (5.32) were provided by Freire Júnior and Belísio [131] and Harris, Gathercole, Lee, et al. [132]. For a stress ratio of $R = -1$ (fully reversed loading), $A = 2.95$, $B = 0.020$, $C = 1.36$, $\alpha = 1.41$, $\beta = 1.23$, and $\gamma = 0.05$. Note that these values assume S is provided in MPa. Because the surrogate models derived in Experiment 1b predict stress in units of ksi, it is necessary to convert to MPa prior to applying Equation (5.31).

Figure 5.6 plots the probabilistic S-N curve created using these parameters. The dark grey and light grey regions represent 50% and 95% confidence intervals around $\overline{N}$. The

widths of these regions are proportional to the amount of uncertainty introduced into the material definition by Equation (5.32).

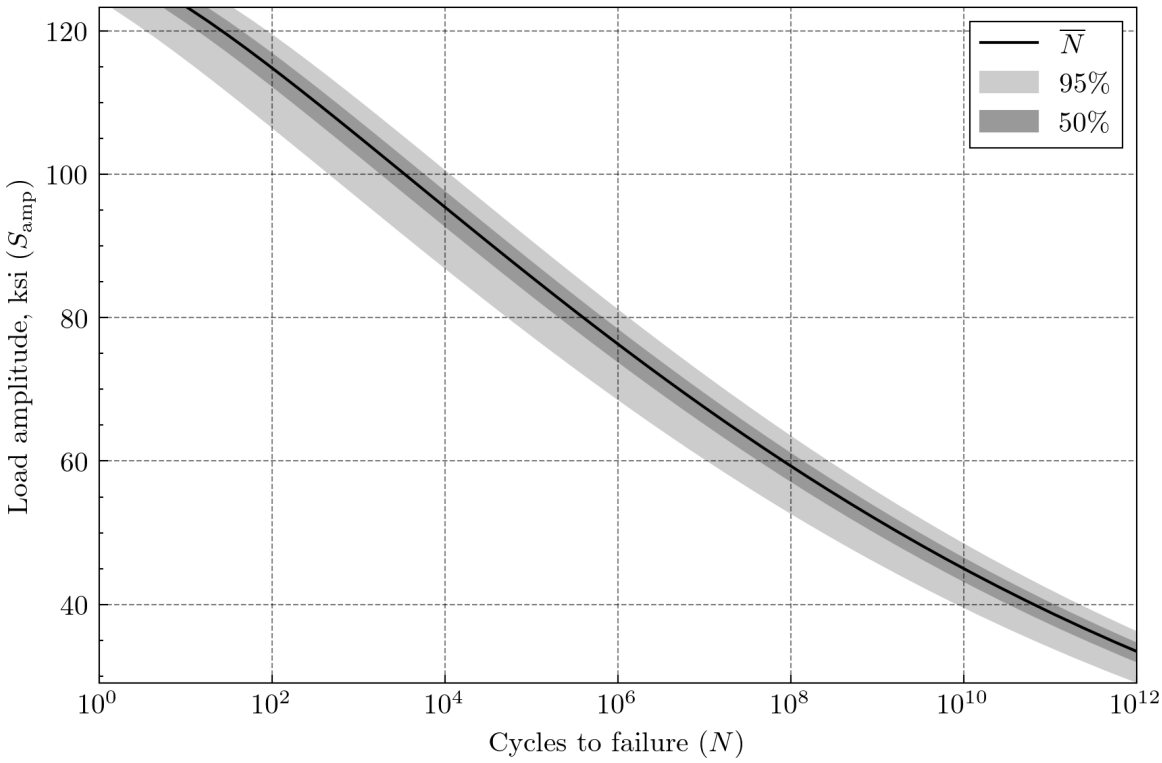


Figure 5.6: Probabilistic S-N curve used in Experiment 2a.

This experiment uses a notional load distribution of $S \sim \mathcal{N}(50, 5^2)$ to model fatigue loading. This distribution was chosen because it is similar to the predicted S_{eq} responses observed throughout Experiment 1. In Experiment 2b, the notional load distribution will be replaced with a much more realistic load spectrum derived from the generic SMR helicopter model and the surrogate models developed in Experiment 1b.

5.4.3 Structural Reliability Solutions

All of the structural reliability solution methods used in Experiment 2 were implemented using the the OpenTURNS package [128]. OpenTURNS provides a large number of tools to solve structural reliability problems, including distribution modeling, distribution transformation, and solution algorithms. This section describes the details of the OpenTURNS

implementation for this experiment.

The performance function G (see Equation (5.29)) is implemented symbolically using OpenTURNS' `SymbolicFunction` class. The symbolic implementation allows OpenTURNS to automatically calculate gradients and Hessians as needed for the analytical solution methods. The inputs to the symbolic function are each stress distribution, S_i, and the Weibull distribution, $N/\overline{N}$. For a given iteration, if $G \leq 0$, the structure has failed; if $G > 0$, the structure has survived.

In extremely simple cases, where $k = 1$ and P_f is high, the problem can be solved exactly. For this exercise, Equation (5.31) was implemented using the `SymbolicFunction` class, and the `CompositeDistribution` class was used to define the $\overline{N}$ distribution. Then, the N distribution is found using Equation (5.34):

$$N \sim \frac{N}{\overline{N}} \times \overline{N} \tag{5.34}$$

and the performance function can be reduced to a single random variable distribution using Equation (5.35):

$$G \sim 1 - r\left(\frac{1}{N}\right) \tag{5.35}$$

Then, the probability of failure after r load cycles can be found using Equation (5.36):

$$P_f = \Pr\left(G \leq 0\right) = \int_{G \leq 0} g_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \tag{5.36}$$

where $g_{\boldsymbol{X}}(\boldsymbol{x})$ is the PDF of G. Note that as P_f decreases, numerical errors in the integration algorithm approach the order of P_f, so the results cannot be considered accurate beyond a certain point. Nevertheless, solving the exact solution provides a measure of confidence in the solutions produced by the analytical and sampling methods.

The structural reliability solution methods themselves are implemented using classes in the OpenMDAO library. MC, QMC, and LHS are implemented using the `Probabil-`

itySimulationAlgorithm class configured with the MonteCarloExperiment, LowDiscrepancyExperiment, and LHSExperiment sampling classes, respectively. QMC uses the SobolSequence class to define the low-discrepancy experiment.

The DS implementation uses the DirectionSampling class. The simulation was configured using the default MediumSafe root-finding strategy with the Brent solver and the OrthogonalDirection sampling strategy.

FORM and SORM make use of the FORM and SORM classes, each using the Cobyla numerical optimizer. The SORM implementation is configured to use the Breitung correction to estimate P_f. The numerical optimizer used by FORM and SORM is initialized from the mean of the multivariate distribution $\boldsymbol{X} = \left[{}^{N}/_{\overline{N}}, S_0, S_1, S_2, \ldots, S_k\right]$.

Finally, the FORM-IS implementation is implemented using the ProbabilitySimulationAlgorithm class configured with the ImportanceSamplingExperiment sampling class. Because the importance sampling simulation initializes around a FORM design point, the FORM solution must be completed first then passed to the FORM-IS solution.

Each structural reliability solution runs until convergence is achieved. For sampling methods, convergence is assumed when the coefficient of variation (COV) is less than 0.1. COV is estimated after each iteration using Equation (5.18). For analytical methods, convergence is assumed when the absolute, relative, residual, and constraint errors are all less than 10^{-10}. After the run concludes, the estimate of P_f and its 95% confidence interval, where applicable, are returned. Additional data necessary to populate the metrics in Table 5.2 is also collected from each run.

5.4.4 Results and Analysis

5.4.4.1 Accuracy

The predicted P_f is plotted against the number of load cycles, rk, at $k = 1$ for each solution method in Figure 5.7. The exact solutions are plotted with single points, the analytical solutions are plotted with marked lines, and the sampling solutions are plotted with a filled

area that spans the 95% confidence interval.

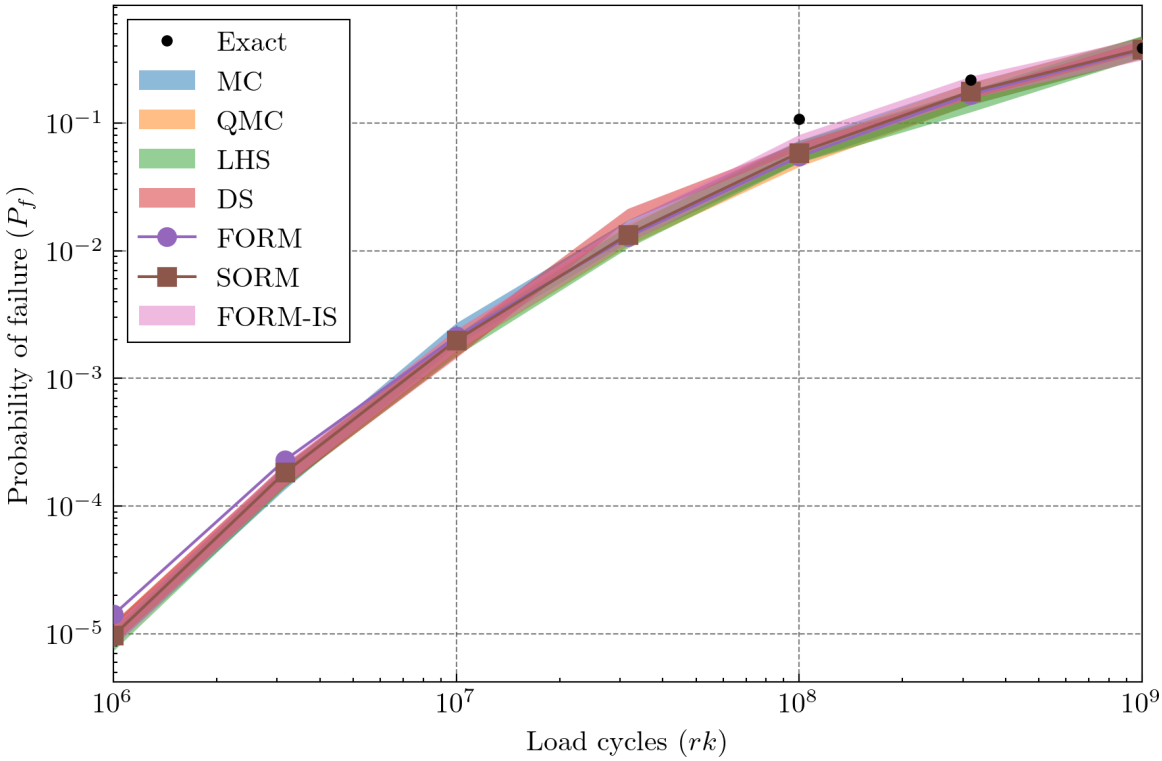


Figure 5.7: P_f predictions from all solution methods at $k = 1$.

As expected, P_f decreases as rk decreases. At 10^9 load cycles, P_f is almost 0.5 or 50%, but at 10^6 cycles, P_f is as low as 10^{-5}. All of the solutions are in general agreement, although the FORM solution appears to slightly overpredict P_f at low values of rk.

The predictions align with the exact solution at $rk = 10^9$, but are lower than the exact solution at $rk = 10^{8.5}$ and $rk = 10^8$. As described previously, the exact solution is inaccurate at lower P_f because of compounding errors associated with numerical integration. Because the simulation and analytical solution methods agree at $rk = 10^8$, it is likely that they are more accurate than the "exact" solution.

Figure 5.8 examines the effect of increasing dimensionality on prediction accuracy. It is expected that as dimensionality increases, P_f remains mostly constant, since each additional stress distribution is identical. In Figure 5.8, P_f is plotted against k at at $rk = 10^9$, $rk = 10^{8.5}$, and $rk = 10^8$. The different solution methods are represented in the same

manner as in Figure 5.7.

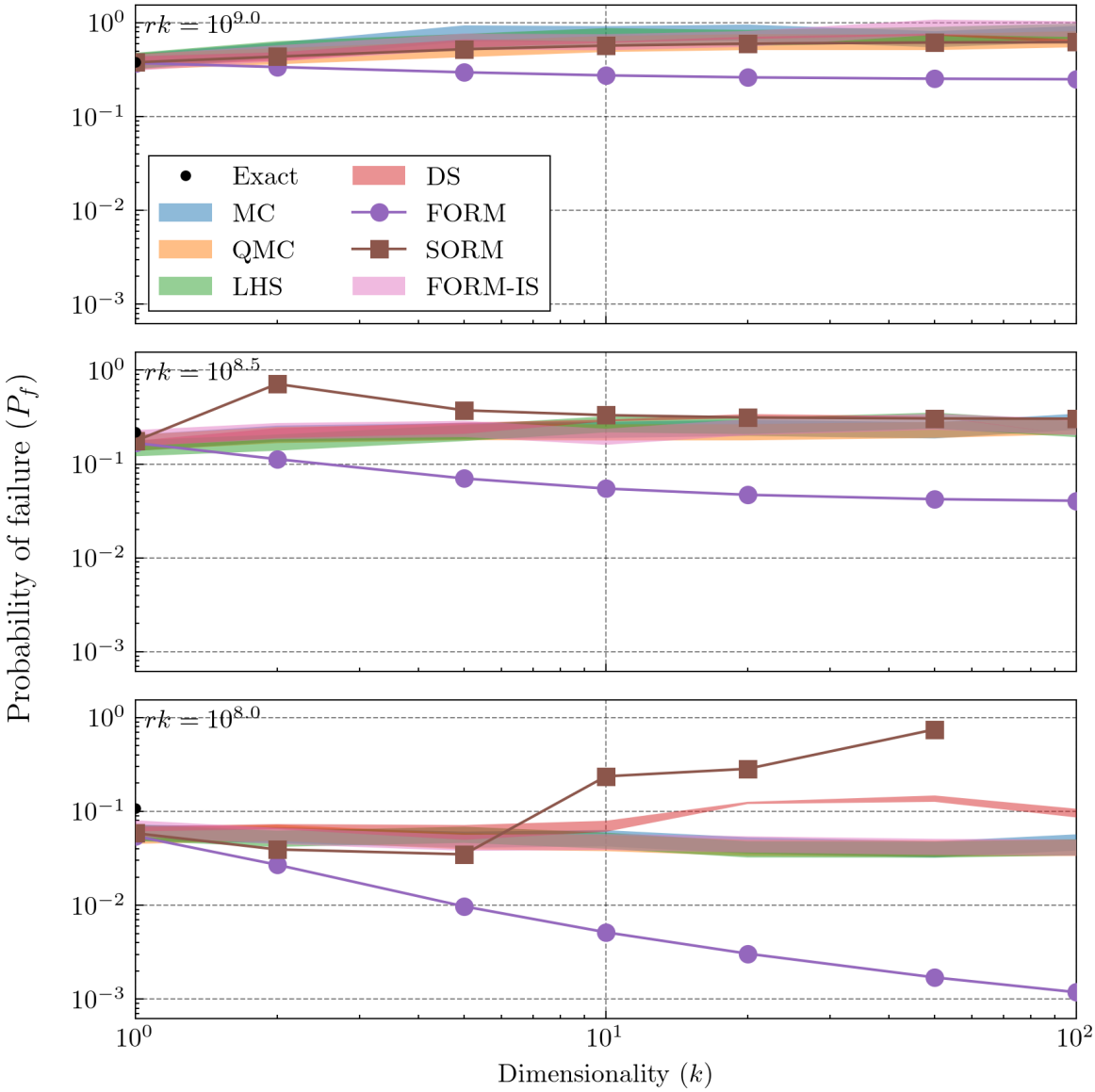


Figure 5.8: P_f predictions from all solution methods at $rk = 10^9$, $rk = 10^{8.5}$, and $rk = 10^8$.

At $rk = 10^9$, all of the solution methods except FORM satisfy the expectation that P_f should remain mostly constant as k increases. FORM consistently underpredicts P_f at higher values of k. This is likely due to FORM's linear approximation of the limit state surface, which, depending on the convexity of the surface, can cause the method to consistently underpredict or overpredict the true solution. Note that SORM, which models the limit state surface as a quadratic surface, does not suffer from the same inaccuracy in this case.

Also note that FORM-IS corrects the FORM solution: the importance sampling process is conditioned on the FORM solution, but it can more accurately capture the shape of the limit state surface than FORM alone.

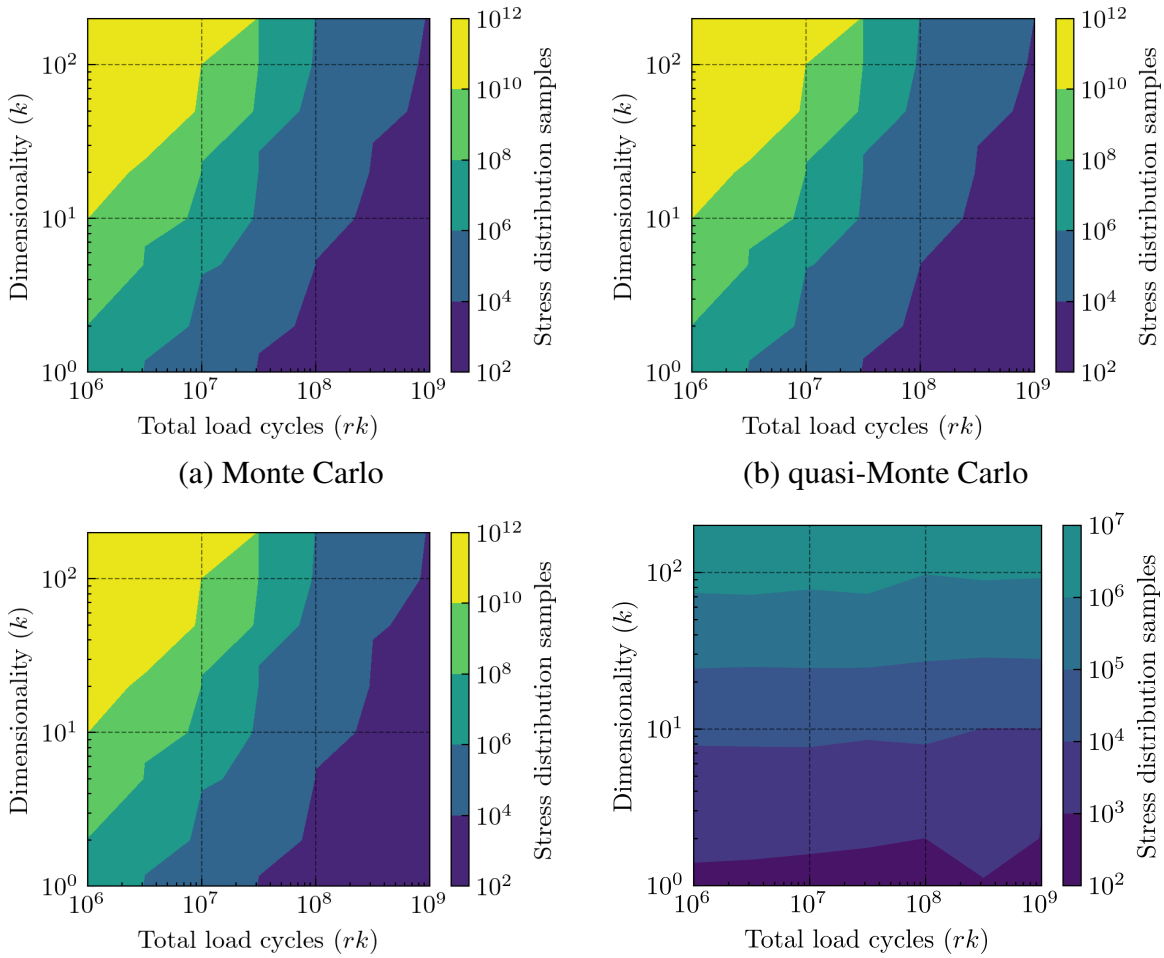
At $rk = 10^{8.5}$, the SORM solution begins to break down. SORM shows significant overprediction of P_f at $k = 2$, $k = 5$, and $k = 10$. The underprediction of the FORM solution also increases at this level. It appears that the shape of the limit state surface is becoming more complex at lower numbers of cycles, which degrades the performance of the analytical solutions.

At $rk = 10^8$, FORM and SORM degrade further. Now, SORM overpredicts P_f at $k = 10$, $k = 20$, and $k = 50$, and fails entirely at $k = 100$. FORM continues to underpredict P_f by an even greater amount. Additionally, the DS solution exhibits a similar overprediction pattern to SORM. Seemingly, the solution methods which involve estimating the shape of the limit state surface break down at lower numbers of load cycles, while the solution methods that rely on sampling alone continue to perform well. The FORM, SORM, and DS methods were removed from consideration due to accuracy issues.

5.4.4.2 Performance

Figure 5.9 plots the number of stress distribution samples required to produce a converged solution. The MC, QMC, LHS, and FORM-IS methods are plotted individually in Figures 5.9a to 5.9d.

The MC, QMC, and LHS plots all show a similar pattern: the number of stress distribution samples is low at high rk (and thus high P_f) and increase either as rk decreases or k increases. This is consistent with the expectation that low P_f problems are more difficult for sampling algorithms to solve. Despite QMC and LHS using more intelligent sampling algorithms, their performance is not significantly improved over the MC solution. If one examines the data behind Figure 5.9 directly, a slight improvement for QMC and LHS is apparently, but it is not significant enough to appear in the contour plots.



(a) Monte Carlo

(b) quasi-Monte Carlo

(c) Latin hypercube sampling

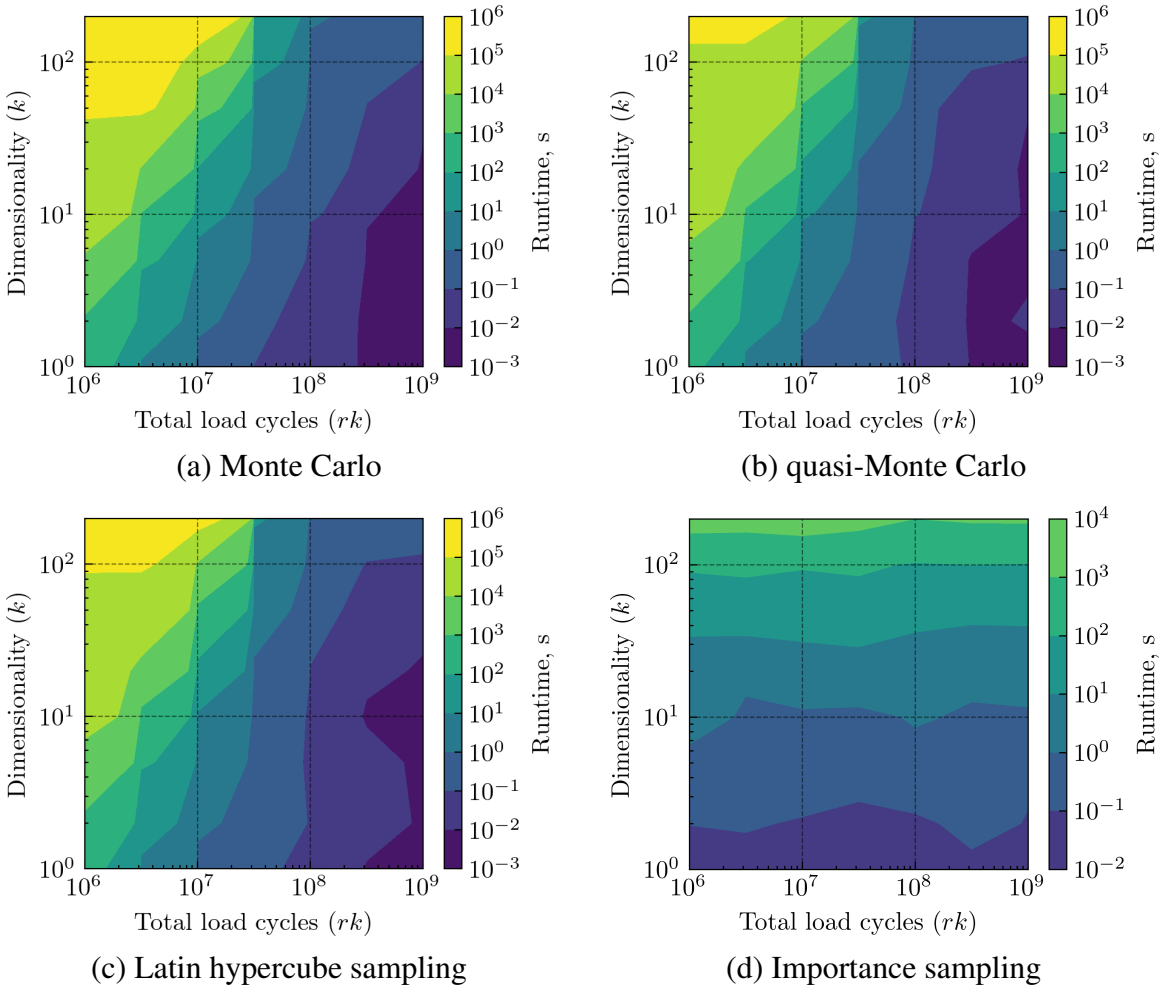
(d) Importance sampling

Figure 5.9: Number of stress distribution samples per solution.

Conversely, the FORM-IS plot is starkly different from the other three solution methods. FORM-IS requires far fewer stress distribution samples overall, and is largely insensitive to rk, and thus P_f. This is because FORM-IS makes use of the FORM solution to effectively "skip" to a location near the most probable failure point, P^* (see Figures 5.2 and 5.4). Despite FORM's accuracy issues, it is able to find a solution much faster than any of the sampling methods. While MC, QMC, and LHS spend a large amount of time sampling and evaluating points throughout the problem space, FORM-IS samples points only near the limit state surface and is thus able to produce an accurate, converged solution in far fewer samples than MC, QMC, or LHS.

Figure 5.10 plots the wall-clock runtime for the same set of solutions. For the MC,

QMC, and LHS solutions, runtime at high P_f is as low as 10^{-3} s, but for low P_f and high dimensionality, runtimes are as long as 10^6 s, 278 h, or 11.6 d. These solutions were executed in parallel on eight processor cores over a period of several weeks.



(a) Monte Carlo

(b) quasi-Monte Carlo

(c) Latin hypercube sampling

(d) Importance sampling

Figure 5.10: Wall-clock runtime per solution.

Figure 5.10 does make the advantages of QMC and LHS more apparent: the region of extremely high runtime near the top-left of each plot is not as large for either solution method as it is for MC. However, FORM-IS continues to show the strongest performance, with maximum runtimes of approximately 10^4 s, or 2.78 h. The patterns are similar to Figure 5.9, with MC, QMC, and LHS showing strong sensitivity to both rk and k and IS showing strong sensitivity to k but only weak sensitivity to rk.

5.4.5 Summary

Experiment 2a described the development of a notional fatigue problem to compare the performance of different solution methods under varying conditions. The results showed that MC, QMC, LHS, and FORM-IS are all capable of producing accurate solutions at low probability of failure and high dimensionality. FORM, SORM, and DS all exhibited accuracy issues that prevent their application at low probabilities of failure.

From a performance perspective, FORM-IS is a clear improvement over MC, QMC, and LHS, requiring two orders-of-magnitude less runtime to solve the most difficult problems. The accuracy and efficiency of this algorithm lends initial support to Hypothesis 2. However, these are only preliminary results based off a simple notional test case with a normally-distributed stress distribution. These results must be validated with a more complex problem before definitive conclusions can be drawn. Experiment 2b will adapt the processes and methods developed and tested in Experiment 2a to a complete helicopter fatigue life quantification problem.

5.5 Experiment 2b

Figure 5.11 provides an overview of Experiment 2b. The following sections with describe the experimental design, implementation of the mission spectrum, transformation of the mission spectrum into a load spectrum, results, and analysis.

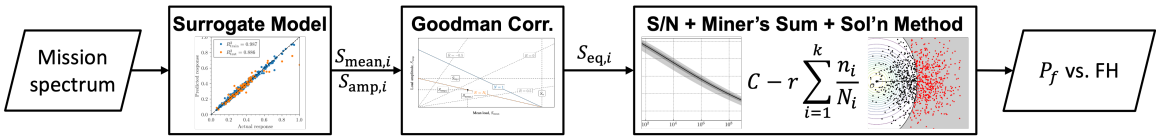


Figure 5.11: Overview of Experiment 2b.

5.5.1 Experimental Design

Experiment 2b is intended to validate the results of Experiment 2a by testing the most successful structural reliability solution methods (MC, QMC, LHS, and FORM-IS), using a realistic rotor blade fatigue life estimation problem.

First, a mission spectrum based on a common transport mission is developed. Rather than prorating the mission spectrum to account for changes in vehicle configuration, altitude, and airspeed as described in Section 2.2.2.2, random variable distributions are used to capture usage variation. Using the six different flight condition variables defined in Experiment 1, a wide variety of operations can be captured in a single mission definition. For example, the same basic mission profile can be used to define a high-weight cargo transport mission or a low-weight ferry mission.

Next, the fatigue load surrogate models trained in Experiment 1b are utilized to convert the mission spectrum into a load spectrum. The end result of this process is a series of equivalent stress distributions, one for each segment of the mission, that capture all possible variability in the fatigue loads for that segment. Sources of variability include surrogate model uncertainty and the aforementioned stochasticity in the mission spectrum definition.

Finally, the structural reliability solution process developed in Experiment 2a is executed using the newly-populated load spectrum. Predictions of P_f at different specified service lives are used to compare the different solution methods using the performance metrics defined in Table 5.2, and to further study the differences between the ANN and GPM surrogate models. This step also includes a comparison to traditional deterministic fatigue life prediction methods, described in Section 2.2.2.

5.5.2 Mission Spectrum

A critical component of Experiment 2b is the mission spectrum. The mission spectrum describes a set of mission profiles that are used to evaluate the fatigue life of the rotor blade. It must be flexible enough to cover a wide variety of potential use cases, yet focused enough

to support meaningful analysis.

For this experiment, the mission spectrum was adapted from a UAM design mission profile published by Silva, Johnson, Solis, et al. [133]. The authors defined a short-range passenger transport mission intended to model a typical UAM concept of operations. For this research, the range of the design mission was increased to reflect the greater capabilities of a turbine-powered conventional helicopter over electrically-powered VTOL aircraft. Several variations of this mission were implemented in the NDARC model of the generic SMR helicopter to observe its performance and fuel burn rate throughout the mission. Mission variations included standard weight, high gross weight, low gross weight, high cruise speed, and high altitude. Results from this study are not included here, but influenced the ultimate design of the probabilistic mission spectrum.

Figure 5.12 diagrams the mission profile. The mission is constructed in 16 segments or phases.

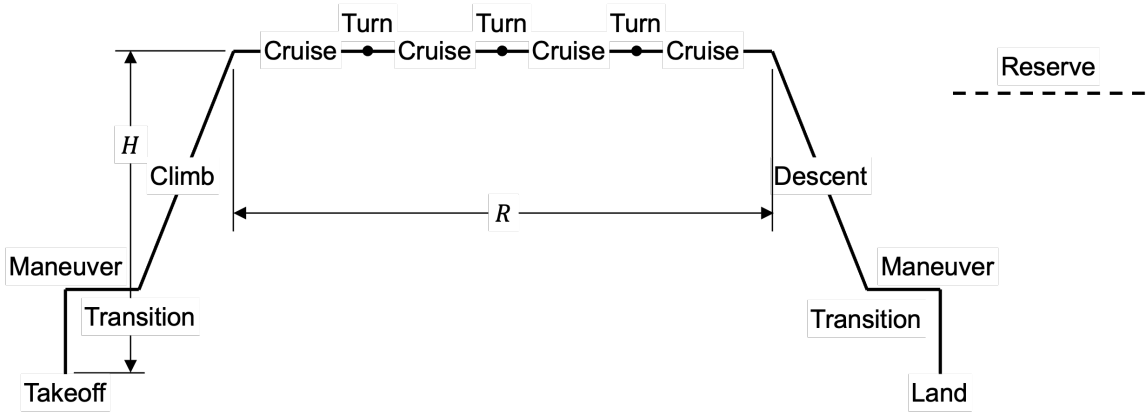


Figure 5.12: Diagram of the mission profile used in Experiment 2b.

First, the vehicle takes off, maneuvers around the vertiport, and transitions to forward flight. Then, it climbs to cruising altitude. Cruise is divided into four segments in order to more accurately model decreasing weight throughout the mission. Interspersed between the four cruise segments are three turning segments to account for changes in the direction of travel during cruise. Cruise is followed by descent to the landing site. After the descent stage, the helicopter transitions to hovering flight, maneuvers to the helipad, and settles to

the ground. A reserve segment is also included in the mission definition.

As described previously, each segment of the mission is defined by a set of random variables to account for usage variability. In the absence of historical usage data, a large number of assumptions concerning these distributions were required. To minimize the total amount of assumptions, four very simple distributions were used throughout the mission spectrum. All distributions were constructed using OpenTURNS.

First, the Dirac distribution, $\mathcal{D}(a)$, is used to model deterministic parameters. The PDF of the Dirac distribution is 1 at some value $x = a$ and 0 otherwise (see Equation (5.37)).

$$f(x) = \begin{cases} 1, & x = a \\ 0, & \text{otherwise} \end{cases} \tag{5.37}$$

The uniform distribution, $\mathcal{U}(a, b)$, is used to model a uniform probability between two points $x = a$ and $x = b$. Its PDF is given by Equation (5.38):

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{5.38}$$

A triangular distribution, $\Delta(a, b, c)$, has a minimum at $x = a$, a maximum at $x = b$, and a mode at $x = m$. The triangular distribution can be used similarly to the normal distribution or other symmetric distributions, but it is bounded and does not require any specific assumptions about standard deviation or other shape parameters. Its PDF is given by Equation (5.39):

$$f(x) = \begin{cases} \frac{2(x-a)}{(m-a)(b-a)}, & a \leq x < m \\ \frac{2(b-x)}{(b-m)(b-a)}, & m \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{5.39}$$

Finally, the trapezoidal distribution, $\mathcal{T}(a, b, c, d)$ is used for cases that require more

complexity than any of the previous distributions can model. The trapezoidal distribution has a minimum at $x = a$, two vertices at $x = b$ and $x = c$, and a maximum at $x = d$. Its PDF is given by Equation (5.40):

$$f(x) = \begin{cases} \left(\frac{2}{d+c-a-b}\right)\frac{x-a}{b-a}, & a \leq x < b \\ \frac{2}{d+c-a-b}, & b \leq x < c \\ \left(\frac{2}{d+c-a-b}\right)\frac{d-x}{b-c}, & c \leq x \leq d \\ 0, & \text{otherwise} \end{cases} \tag{5.40}$$

The complete definition of the mission spectrum is included in Table 5.3. For each segment, distributions are defined for the six flight condition variables: V, h, GW, ROC, ω, and CG. Additional distributions are defined for segment distance, D, and time, t. The amount of fuel burned in each segment, $\Delta\mathrm{GW}$, was defined deterministically because of computational issues related to adding and subtracting distributions in OpenTURNS. Deterministic fuel burn estimates were derived by studying the NDARC mission results.

Broadly, the mission is defined by a few key distributions. Cruise speed, V_5, is defined by a triangular distribution, $\Delta(100, 130, 160)$ kt, which captures low- and high-speed cruise. Gross weight at takeoff is defined by a trapezoidal distribution, $\mathcal{T}(12, 15, 17, 20) \times 10^3$ lb, which captures low-, medium-, and high-weight missions.

Similarly, rate of climb and rate of descent are defined by two triangular distributions, $\Delta(600, 900, 1200)$ ft/min and $\Delta(-1200, -900, -600)$ ft/min, respectively, which models different levels of expediency that may be required by operational constraints. Finally, the total range of the cruise segments is given by $R \sim \mathcal{T}(25, 50, 75, 100)$ NM to accommodate short- and long-range missions, and the CG position is given by $X_{\mathrm{CG}} \sim \mathcal{T}(-0.3, 0, 0.3, 0.6)$ to account for different payload configurations.

Table 5.3: Probabilistic mission spectrum developed for Experiment 2b.

Seg.	Label	V kt	h ft	GW lb	ROC ft/min	ω deg/s	CG ft	D NM	t s	ΔGW lb
1	Takeoff	$\mathcal{T}(0,0,5,10)$	$\mathcal{T}(0,0,2,6)\times 10^3$	$\mathcal{T}(12,15,17,20)\times 10^3$	$\mathcal{U}(100,500)$	$\mathcal{D}(0)$	$X_{\mathrm{CG}}{}^{*}$	$\mathcal{D}(0)$	$60\frac{50}{\mathrm{ROC}_1}$	10
2	Maneuver 1	$\mathcal{U}(10,30)$	h_1+50	$\mathrm{GW}_1-\Delta\mathrm{GW}_1$	$\Delta(0,0,500)$	$\Delta(-3,0,3)$	CG_1	$\frac{V_1t_1}{3600}$	$60\mathcal{U}(0,1)$	10
3	Transition 1	$V_2+\frac{V_3-V_2}{2}$	h_2	$\mathrm{GW}_2-\Delta\mathrm{GW}_2$	$\mathcal{D}(0)$	$\mathcal{D}(0)$	CG_2	$\frac{V_2t_2}{3600}$	$60\mathcal{U}(0.25,0.75)$	10
4	Climb	$\mathcal{U}(80,130)$	$h_3+\frac{h_5-h_3}{2}$	$\mathrm{GW}_3-\Delta\mathrm{GW}_3$	$\Delta(6,9,12)\times 10^2$	$\mathcal{D}(0)$	CG_3	$\frac{V_3t_3}{3600}$	$60\frac{h_5-h_3}{\mathrm{ROC}_4}$	70
5	Cruise 1	$\Delta(100,130,160)$	$\mathcal{T}(4,4,6,10)\times 10^3$	$\mathrm{GW}_4-\Delta\mathrm{GW}_4$	$\mathcal{D}(0)$	$\mathcal{D}(0)$	CG_4	$\frac{R}{4}{}^{\dagger}$	$3600\frac{D_5}{V_5}$	120
6	Turn 1	V_5	h_5	$\mathrm{GW}_5-\Delta\mathrm{GW}_5$	ROC_5	$\mathcal{T}(-6,-3,3,6)$	CG_5	$\frac{V_6t_6}{3600}$	$60\mathcal{U}(0,1)$	10
7	Cruise 2	V_5	h_5	$\mathrm{GW}_6-\Delta\mathrm{GW}_6$	ROC_5	ω_5	CG_6	$\frac{R}{4}{}^{\dagger}$	$3600\frac{D_7}{V_7}$	120
8	Turn 2	V_5	h_5	$\mathrm{GW}_7-\Delta\mathrm{GW}_7$	ROC_5	ω_6	CG_7	$\frac{V_8t_8}{3600}$	t_6	10
9	Cruise 3	V_5	h_5	$\mathrm{GW}_8-\Delta\mathrm{GW}_8$	ROC_5	ω_5	CG_8	$\frac{R}{4}{}^{\dagger}$	$3600\frac{D_9}{V_9}$	120
10	Turn 3	V_5	h_5	$\mathrm{GW}_9-\Delta\mathrm{GW}_9$	ROC_5	ω_6	CG_9	$\frac{V_10t_10}{3600}$	t_6	10
11	Cruise 4	V_5	h_5	$\mathrm{GW}_{10}-\Delta\mathrm{GW}_{10}$	ROC_5	ω_5	CG_{10}	$\frac{R}{4}{}^{\dagger}$	$3600\frac{D_{10}}{V_{10}}$	120
12	Descent	$\mathcal{U}(80,130)$	$h_{11}-\frac{h_{11}-h_{13}}{2}$	$\mathrm{GW}_{11}-\Delta\mathrm{GW}_{11}$	$-\Delta(12,9,6)\times 10^2$	$\mathcal{D}(0)$	CG_{11}	$\frac{V_{12}t_{12}}{3600}$	$60\frac{h_{13}-h_{11}}{\mathrm{ROC}_{12}}$	40
13	Transition 2	$V_{12}+\frac{V_{14}-V_{12}}{2}$	h_3	$\mathrm{GW}_{12}-\Delta\mathrm{GW}_{12}$	ROC_3	ω_3	CG_{12}	$\frac{V_{13}t_{13}}{3600}$	t_3	10
14	Maneuver 2	V_2	h_2	$\mathrm{GW}_{13}-\Delta\mathrm{GW}_{13}$	ROC_2	ω_2	CG_{13}	$\frac{V_{14}t_{14}}{3600}$	t_2	10
15	Land	V_1	h_1	$\mathrm{GW}_{14}-\Delta\mathrm{GW}_{14}$	$\mathcal{U}(-100,-50)$	ω_1	CG_{14}	$\mathcal{D}(0)$	$60\frac{(-50)}{\mathrm{ROC}_{15}}$	10
16	Reserve	V_5	h_5	$\mathrm{GW}_{15}-\Delta\mathrm{GW}_{15}$	ROC_5	ω_5	CG_{15}	$\frac{V_{16}t_{16}}{3600}$	$60\mathcal{U}(0,20)$	250

* $X_{\mathrm{CG}}\sim\mathcal{T}(-0.3,0,0.3,0.6)$
† $R\sim\mathcal{T}(25,50,75,100)$

The mean range of the mission spectrum in 62.5 NM, although this does not include any segments other than the cruise phase. Including all segments results in a nominal wind distance of 104.5 NM. The average time required to complete the mission is 0.87 FH, resulting in an average of 13,523 load cycles.

There are some important limitations related to the mission as defined in Table 5.3. Primarily, each distribution is defined independently. In reality, distributions would be correlated within and across mission segments. For example, the amount of fuel burn in one segment is positively correlated with the airspeed and gross weight in that segment. Although the mission spectrum was made as realistic as possible by defining specific distributions as functions of other distributions, developing a complete correlated mission spectrum would require significant investment in mission simulation, sampling, and fitting distributions and correlation coefficients to the results, which is beyond the scope of this research. Additionally, several of the structural reliability solution methods reviewed in Section 5.1 require independent input distributions.

5.5.3 Load Spectrum

After defining the mission spectrum, a process was developed to convert the data in Table 5.3 to a set of equivalent stress, S_{eq}, distributions. This process makes use of methods implemented in OpenTURNS to produce S_{mean} and S_{amp} distributions using the surrogate models trained in Experiment 1b, which are later converted to S_{eq} distributions using Goodman's relation. The process is applied to each mission segment sequentially.

First, an instance of the `ComposedDistribution` class is defined. This class combines the six independent flight condition distributions into a six-dimensional multivariate distribution with an independent copula.

Next, distributions are initialized for the S_{mean} and S_{amp} responses. As discussed in Section 4.5.5, S_{mean} and S_{amp} can be modeled with lognormal and normal distributions, respectively. In OpenTURNS, the lognormal distribution, $\mathcal{L}(\mu_\ell, \sigma_\ell, \gamma)$, has three parameters:

μ_ℓ is the mean of the underlying normal distribution, σ_ℓ is the standard deviation of the underlying distribution, and γ is a location parameter that is fixed to zero in this research. The normal distribution, $\mathcal{N}(\mu, \sigma)$, has two parameters: μ is the mean and σ is the standard deviation. Equations (4.59) to (4.62) describe how the parameters of these distributions relate to the predictions and uncertainty of the GPM and ANN surrogate models.

The lognormal and normal distributions are combined into a block independent distribution using OpenTURNS' `BlockIndependentDistribution` class. A block independent distribution is similar to a composed distribution, except the parameters can easily be modified after the fact. Essentially, the block independent distribution creates a two-dimensional five-parameter distribution from the lognormal and normal distributions.

Next, the parameters of the block independent distribution are *conditioned* on the flight condition distributions using the `ConditionalDistribution` class. In OpenTURNS, a conditional distribution is defined by Equation (5.41):

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \int f_{\boldsymbol{X}|g(\boldsymbol{y})}\left(\boldsymbol{x}|g(y)\right) f_{\boldsymbol{Y}}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \tag{5.41}$$

where $f_{\boldsymbol{Y}}(\boldsymbol{y})$ is the PDF of the composed distribution, $\boldsymbol{Y}$ are the six flight condition distributions, $f_{\boldsymbol{X}}(\boldsymbol{x})$ is the PDF of the conditioned block independent distribution, $\boldsymbol{X}$ are the lognormal and normal stress response distributions, and $g(\boldsymbol{y})$ is a *link function*. The link function maps $\boldsymbol{Y}$ to the parameters of $\boldsymbol{X}$. This is accomplished by querying the surrogate models for each stress response, deriving the parameters as seen in Equations (4.59) to (4.62), and concatenating the five parameters into a vector.

Finally, the marginal distributions of the block independent distribution are extracted to recover the S_{mean} and S_{amp} distributions. Goodman's relation (see Equation (2.4)) is used to form S_{eq} from S_{mean} and S_{amp}. In this experiment, S_u was set to 1430 MPa using data from Harris, Gathercole, Lee, et al. [132].

The end result of this process is a set of 16 S_{eq} distributions, each corresponding to one

of the 16 mission segments defined in Table 5.3. Each stress distribution captures all the variability in the mission definition and the uncertainty imparted by the stress surrogate models. Because the distributions were constructed analytically, there is no loss of fidelity in the process. These distributions replace the nominal stress distribution used in Experiment 2a in the fatigue life problem definition.

5.5.4 Ground–air–ground cycle

The ground–air–ground (GAG) cycle, described in Section 1.5, is an important type of low-cycle fatigue (LCF), that can significantly impact the fatigue life of helicopter components. Even though the cycle count is low compared to high-cycle fatigue (HCF), stress amplitude is very high. In order to more realistically model the fatigue life of the generic SMR helicopter, and to demonstrate that the preliminary fatigue design methodology can account for both HCF and LCF, the GAG cycle was incorporated into the mission spectrum.

The GAG cycle is implemented as the 17th mission segment, with a cycle count of one. The mean stress and stress amplitude of the GAG cycle are derived using Equations (5.42) and (5.43):

$$S_{\text{amp}} = \frac{1}{2}\left(S_{\text{max}} - S_{\text{min}}\right) \tag{5.42}$$

$$S_{\text{mean}} = S_{\text{min}} + S_{\text{amp}} \tag{5.43}$$

where S_{max} and S_{min} are the maximum and minimum values of stress experienced throughout the mission.

S_{max} and S_{min} can be constructed using the `MaximumDistribution` class.[1] However, attempting to subtract the S_{min} distribution from the S_{max} distribution resulted in computational issues that crashed the program. This may be a bug in the current version of OpenTURNS as similar issues were experienced when constructing the mission spectrum.

[1]The minimum stress is simply the negative maximum of the negative stress distributions.

To bypass this issue, the GAG was defined simply as $S_{\text{eq}} = S_{\text{max}}$, which is derived from Equations (5.42) and (5.43) when $S_{\text{max}} = -S_{\text{min}}$. This is a conservative implementation that assumes the amplitude of the GAG cycle is higher than it would be otherwise.

5.5.5 Results and Analysis

5.5.5.1 Equivalent Stress Distributions

The equivalent stress distributions generated during Experiment 2b are plotted in Figure 5.13. Figure 5.13 is a series of violin plots, each of which represents a S_{eq} distribution for a different mission segment. Distributions produced by ANN and GPM are plotted side-by-side for comparison. Each violin plot was formed by sampling the appropriate distribution 10,000 times and fitting a distribution with kernel density estimation (KDE).

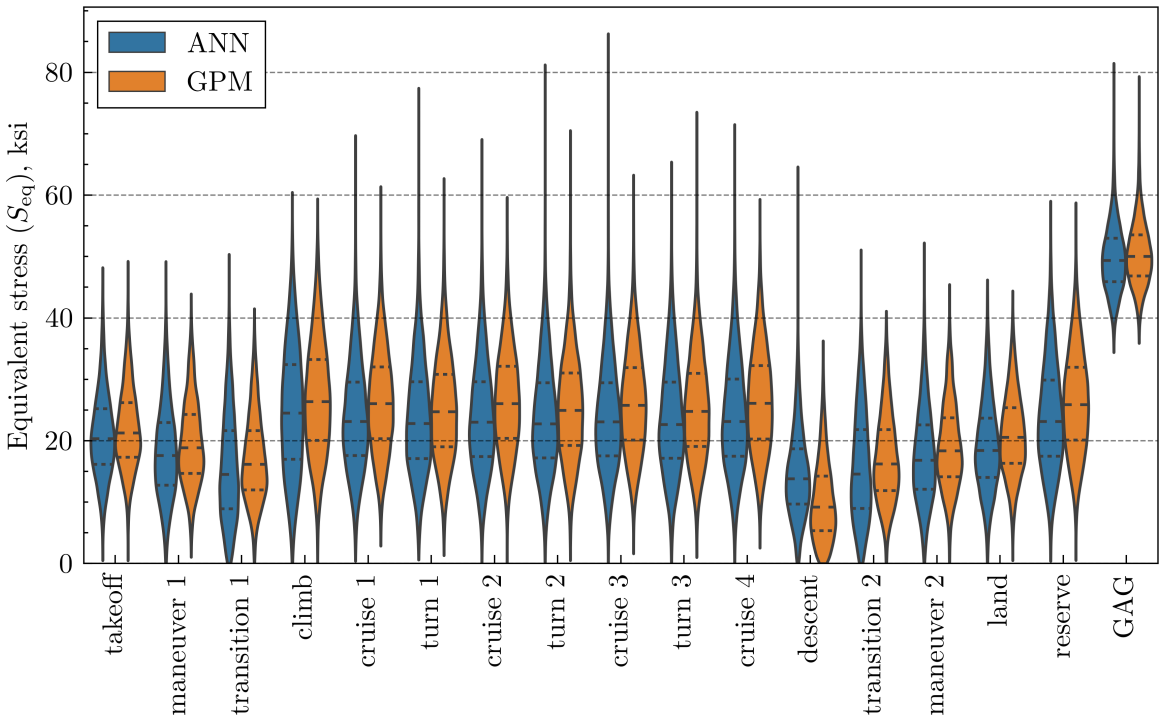


Figure 5.13: Equivalent stress distributions produced in Experiment 2b.

In general, the distributions produced by the GPM surrogate model have higher medians than those produced by ANN. Because the ANN model has better overall goodness-of-fit than the GPM model (see Figures 4.61 and 4.62), it is likely that the GPM model is

overpredicting some results. However, the ANN model produces distributions with longer tails. This is an effect of the global versus local uncertainty quantification used when defining the link functions for these two models, which was discussed previously in Section 4.5.5.

Clearly the GAG cycle has the highest overall S_{eq} distribution. This will always hold true regardless of the particular model or mission spectrum because of the way the GAG stress distribution is derived. In terms of the standard mission segments, the climb, cruise, and turning segments have the highest S_{eq} distributions. The cruise and turning segments are mostly indistinguishable, which reflects the findings of Experiment 1a. However, the climb segment has noticeably "fatter" tails than the other segments, which suggests it is likely to be more damaging than other segments.

The descent segment has the lowest equivalent stress distribution overall. As discussed in Section 4.4.5.8, this effect may be physically inaccurate and is a limitation of the inflow and wake models used in the RCAS implementation of the generic SMR helicopter. Also notice that the descent segment is the site of the largest discrepancy between the ANN and GPM surrogate models, which indicates that one of the models may have not been able to fit the negative rate of climb region well.

5.5.5.2 Probability of Failure

Next, the structural reliability solution methods that performed well in Experiment 2a were used to predict probability of failure at different service life requirements. P_f was predicted at service lives ranging from 350 FH to 200,000 FH. The full Miner's sum performance function (see Equation (5.28)) was used. Compared to Experiment 2a, random variable distributions derived from mission segment time were used to populate n_i and C was set to $\mathcal{N}(1, 0.05^2)$.

Unfortunately, the QMC, LHS, and FORM-IS methods, which showed promise in Experiment 2a, failed to produce usable results for this experiment. Although the exact cause of failure is unknown, it is most likely related to the complexities associated with the

definition of the S_{eq} distributions for Experiment 2b. QMC and LHS both must transform the input random variable distributions to determine the sampling points, and FORM-IS must initialize from a FORM solution that requires an isoprobabilistic transformation of the input variables. Although these tasks were simple in Experiment 2a when the input variables were normally distributed, they are non-trivial when applied to distributions constructed in the manner described in Section 5.5.3. The current version of OpenTURNS does not provide enough diagnostic information to solve this issue or develop a workaround.

However, crude Monte Carlo sampling does not require any transformations of the input variables. Because the S_{eq} distributions can be sampled relatively quickly, the MC simulation was the only functional solution method tested in this experiment. The results produced by the MC method are presented in Figure 5.14.

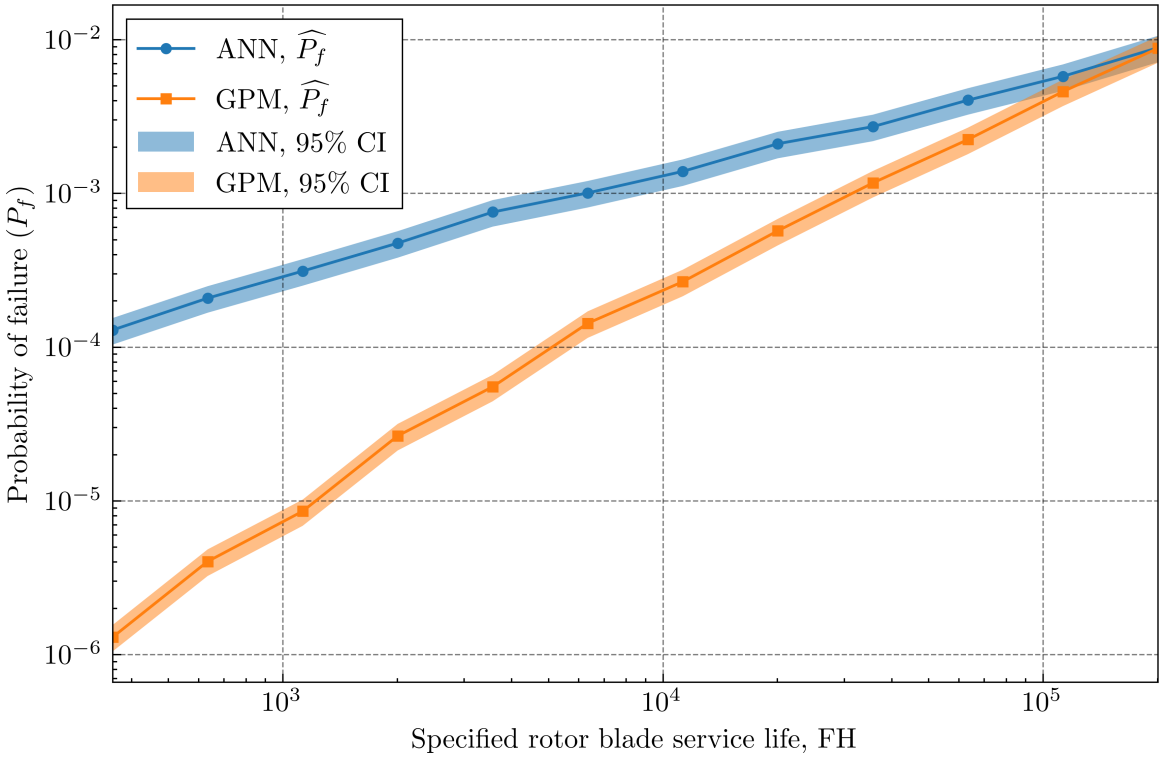


Figure 5.14: Probability of failure at different service life requirements for Experiment 2b.

For context, the fatigue life of the rotor blade of the Robinson R22, a popular two-seat general aviation helicopter, is 2200 FH. The fatigue life of the rotor blade of the Sikorsky S-76, a high-end executive transport helicopter, is 28,000 FH [3]. Recall that the target P_f

for a flight-critical component under the six-nines reliability paradigm is 10^{-6}.

Although the results produced using the ANN and GPM surrogate models are in agreement at high P_f, GPM clearly predicts below the ANN model at lower P_f. Low P_f corresponds to lower cycle counts, where most failed iterations are those that include samples from the tails of the S_{eq} distribution. Because the stress distributions produced by the ANN surrogate models have longer tails, those distributions are more likely to produce elevated levels of stress that lead to component failure. Conversely, at high cycle counts, failed iterations can include samples from the bodies or tails of the stress distributions. In the bodies of the distributions, ANN and GPM are more similar, so the predictions show better agreement.

5.5.5.3 Comparison to Deterministic Methods

In order to compare the results of the probabilistic fatigue life methodology to traditional deterministic fatigue life methodologies, reduction methods from Section 2.2.2 were applied to the data generated in Experiment 2b.

First, the probabilistic S-N curve described in Section 5.4.2 was reduced to a deterministic curve at 80% of the probabilistic curve's mean value. The equivalent stress distributions were increased to deterministic values using either the $\mu + 3\sigma$ method, where the deterministic stress is equal to the mean of the stress distribution plus three times its standard deviation, or the top-of-scatter (TOS) method, where the deterministic stress value is equal to the maximum observed sample from the stress distribution. All parameters were estimated using 10,000 samples of each distribution.

Equation (5.28) was rearranged to solve the deterministic fatigue life problem (see Equation (5.44)):

$$r = \frac{C}{\sum_{i=1}^{k} \frac{n_i}{N_i(S_{\mathrm{eq},i})}} \tag{5.44}$$

where r describes the number of mission repetitions to failure. n_i was populated using the means of each mission segment time distribution and C was set to 0.85 using the $\mu - 3\sigma$

reduction. The results of this study are presented in Table 5.4.

Table 5.4: Deterministic rotor blade fatigue life predictions.

	Fatigue life (FH)	
Surrogate model	$\mu + 3\sigma$ stress	TOS stress
ANN	63,350	155.9
GPM	58,170	2226

From Table 5.4, it is immediately obvious that deterministic results vary widely depending on the methodology used. Cross-referencing with Figure 5.14, it can be seen that predictions made using $\mu + 3\sigma$ drastically overpredict the safe life, corresponding with probabilities of failure from 2×10^{-3} to 4×10^{-3}. The probability of failure predicted by the TOS method using the GPM surrogate model corresponds to approximately 3×10^{-5}, but the prediction for the ANN model does not appear on the Figure 5.14 at all.

Although the TOS/GPM prediction has acceptable levels of reliability, the fact remains that *the reliability of these predictions cannot be quantified using deterministic methods.* This reinforces the earlier argument that deterministic fatigue life predictions derived from organization experience and historic data cannot be assumed to produce sufficient levels of reliability, especially where new rotorcraft configurations are concerned.

5.5.6 Summary

Experiment 2b validated the results of Experiment 2a by constructing a realistic probabilistic mission spectrum, producing equivalent stress distributions corresponding to that mission spectrum using the stress surrogate models, and producing new fatigue life predictions using the previously-tested structural reliability solution methods. Ultimately, it was found that, given the complex construction of the S_{eq} distributions, only Monte Carlo simulation was capable of producing predictions.

This experiment also compared the predictions produced using the ANN and GPM surrogate models from Experiment 1b. GPM surrogate models were found to consistently

predict P_f values several orders of magnitude below the ANN predictions, especially at low P_f. Because the ANN surrogate model is more conservative, faster to execute, and demonstrated better goodness-of-fit than the GPM model, the ANN model will be retained for use in subsequent experiments.

Finally, the results produced by the probabilistic fatigue life methodology were compared to deterministic results produced by traditional reductions and safety factors. The deterministic results were found to vary significantly depending on the safety factor used, which highlights the value of a probabilistic method which can predict both a fatigue life specification and the probability of failure associated with that prediction.

5.6 Conclusions

Experiment 2 examined the possibility of using structural reliability methods to probabilistically evaluate Miner's sum for fatigue life analysis, with the goal of improving upon traditional deterministic methods. Experiment 2a described the development of a notional fatigue life problem and the comparison of a number of popular structural reliability methods using that problem. Experiment 2b expanded the notional fatigue life problem into a complete helicopter fatigue analysis using a realistic mission spectrum and stress predictions from the generic SMR helicopter.

Recall Hypothesis 2:

Hypothesis 2

At least one of the surveyed structural reliability sampling methods can be used to efficiently remove the dependence on reductions and safety factors by quantifying the reliability of fatigue life predictions using Miner's sum.

The results of Experiment 2 *partially support* Hypothesis 2. High-reliability fatigue life predictions were produced with no deterministic safety factors. Comparison to deterministic methods demonstrated the value of using probabilistic methods instead of trusting safety factors and reductions which are derived from heuristics and experience. However, the

only functional solution method, Monte Carlo simulation, is not efficient at low P_f, as demonstrated by Experiment 2a. Thus, this experiment does not fully support the hypothesis it was intended to test.

The preliminary fatigue design methodology, previously presented in Figures 3.1 and 4.66, can now be updated based on the results of Experiment 2.

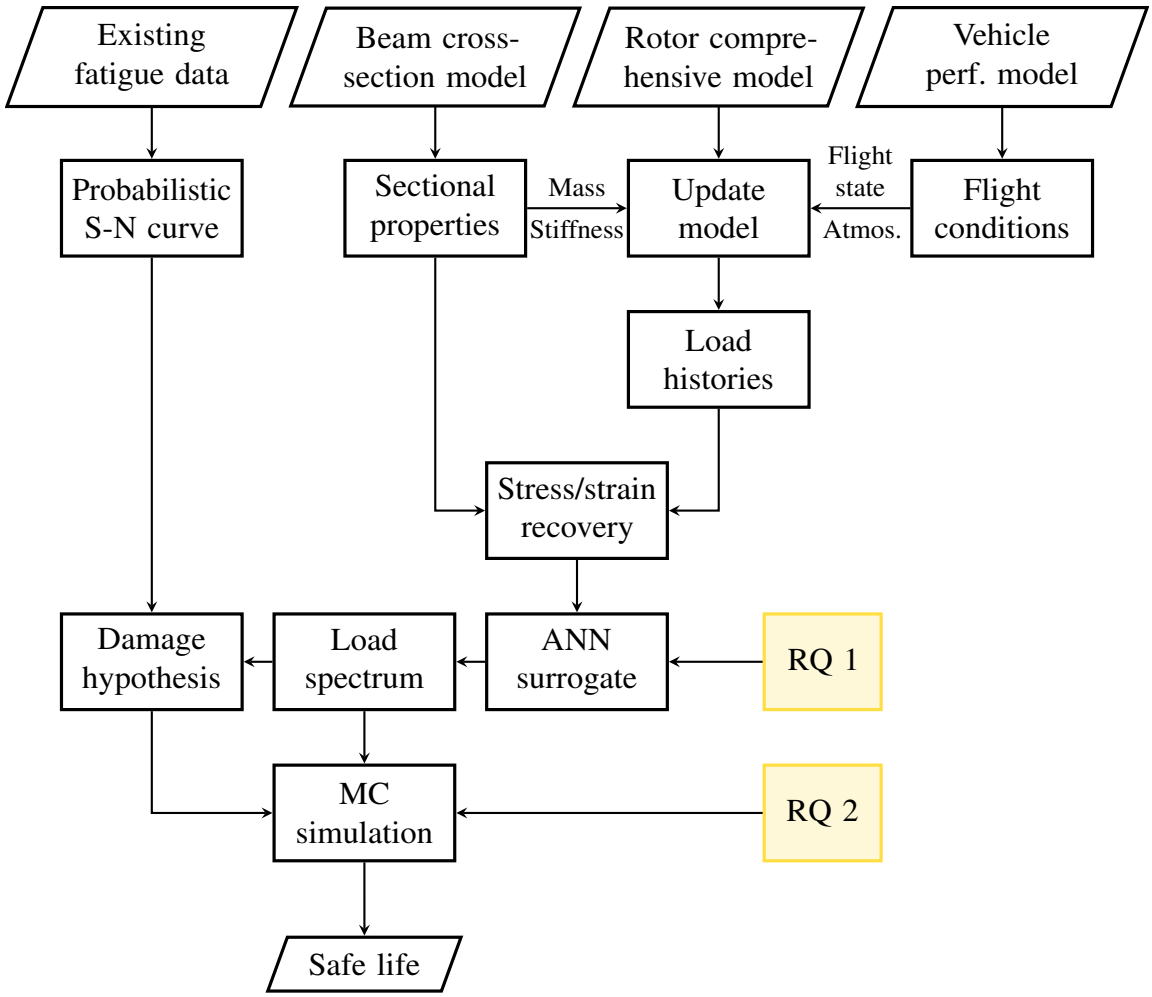


Figure 5.15: Updated flowchart of the preliminary fatigue design methodology after Experiment 2.

Figure 5.15 reflects that Monte Carlo simulation can be used to produce a probabilistic safe life and its associated probability of failure. It also reflects that ANN surrogate modeling was chosen over GPM.

With the end of Experiment 2, the preliminary fatigue design methodology is finalized.

However, its utility must be demonstrated. The final experiment of this thesis, Experiment 3, will involve applying the methodology to a hypothetical fatigue design exercise. This experiment is described further in Chapter 6.

CHAPTER 6

FATIGUE DESIGN OF A CONCEPTUAL ROTARY-WING AIRCRAFT

After Experiments 1 and 2, all critical elements of the preliminary fatigue design methodology (see Figure 5.15) have been determined. However, the aforementioned experiments are not intended to represent actual use of the methodology. In this chapter, the effectiveness of the methodology will be tested using a series of experiments designed to prove its ability to serve as an effective preliminary design tool. These experiments will also serve as instructive use cases demonstrating potential applications of the methodology to specific design problems.

Conjecture 0 requires the methodology to enable the use of rotor component fatigue life as a design driver. Thus, the methodology must be capable of predicting the impact of changes of common conceptual or preliminary design variables on the rated fatigue life, or the associated probability of failure, of a flight-critical component. This led to Research Question 3, which is reprinted below for the reader's convenience:

Research Question 3

Does the preliminary fatigue design methodology enable evaluation of the relative impact of common preliminary design variables on the probability of fatigue failure of a flight-critical component in a conceptual helicopter design?

In the next section, hypotheses to Research Question 3 will be developed. Then, a series of experiments will be designed and executed to test each hypothesis. If the experiments successfully demonstrate the ability of the preliminary fatigue design methodology to predict the impact of preliminary design variables, then Conjecture 0 will be supported.

6.1 Hypotheses to Research Question 3

Throughout this research, the multi-disciplinary nature of the helicopter fatigue life prediction problem has been emphasized. Because a component's fatigue life is dependent on its design, the overall design of the helicopter, and the characteristics of the helicopter's missions, changes in any of those variables should impact the predicted probability of fatigue failure. Specifically, the newly-developed preliminary fatigue design methodology must be capable of predicting changes in the probability of failure of the main rotor blade given modifications to the rotor blade cross section design, the vehicle design, and the design mission requirements.

6.1.1 Rotor Blade Cross Section Design

Direct changes to the rotor blade cross section may be the most effective way to improve rotor blade fatigue life. However, the rotor blade cross section is one of the most constrained elements of a helicopter. The cross section must be designed to satisfy weight, aerodynamic, ultimate failure, aeroelastic stability, maintainability, manufacturability, and cost constraints [87, 93]. There may not be a significant amount of design freedom available to design for fatigue life.

The preliminary fatigue design methodology could be integrated into a rotor blade design framework similar to that developed by Li [87], where the rotor blade is optimized for weight subject to natural frequency and fatigue life constraints (see Section 2.3.1). Alternatively, fatigue life could serve as the objective function while weight and structural dynamic concerns are relegated to constraints. In either case, the preliminary fatigue design methodology must be able to predict the impact of cross section design changes of probability of fatigue failure.

In Experiments 1 and 2, fatigue life was quantified at a critical fatigue point on the auxiliary web of the rotor blade box spar. The signed von Mises stress at that point was used

to model damage progression using Miner's sum. At a basic level, stress is proportional to the amount of force divided by the area of the surface with a normal vector in the direction of that force. Thus, if the area around the the critical fatigue design point increases, the signed von Mises stress at that point will decrease, ultimately leading to a decrease in probability of failure.

The preliminary fatigue design methodology should be sensitive enough to quantify this effect. This leads to Hypothesis 3.1:

Hypothesis 3.1

The preliminary fatigue design methodology will enable quantification of the impact of blade spar thickness on probability of rotor blade fatigue failure at a specified service life.

The null hypothesis to Hypothesis 3.1 can be stated as follows:

> The preliminary fatigue design methodology does not provide sufficient information to predict the impact of blade spar thickness on the probability of rotor blade fatigue failure.

This hypothesis will be tested by Experiment 3.1.

6.1.2 Vehicle Design

If the rotor blade cross section design is already highly-constrained, designers may instead look to the design of the rotorcraft itself to improve fatigue life. Although the vehicle layout is subject to weight, performance, handling, maneuverability, and payload constraints, its design may be more flexible than the rotor blade itself, especially in the case of advanced configurations with many available design variables.

In this case, the preliminary fatigue design methodology could be integrated into a rotorcraft vehicle design framework. The closest example that could be found in the literature is that of Arruda, Hamel, and Collins [90], although these authors tested the impact

of main rotor design, not the vehicle as a whole. For a review of this design framework, see Section 2.3.2. In this application, the preliminary fatigue design methodology would provide fatigue life predictions driven by changes in vehicle layout and configuration to enhance the design environment with fatigue-related objectives or constraints.

In Experiments 1a and 1b, the station line of the generic SMR helicopter's center of gravity was found to have a significant impact on stress at the critical fatigue point. It was hypothesized that, as the CG shifts further aft, additional pitch down moment is required from the main rotor to keep the helicopter in the appropriate orientation for forward flight. This in turns increases the aerodynamic forces on, and the stress in, the rotor blade.

Several existing helicopters, such as the UH-60 Black Hawk and the CH-53K King Stallion, have a positive *cant* angle on the tail rotor. Canting the tail rotor allows that rotor to produce anti-torque thrust *and* a vertical thrust component, described by Equations (6.1) and (6.2):

$$T_h = T \cos \epsilon \tag{6.1}$$

$$T_v = T \sin \epsilon \tag{6.2}$$

where T is the total tail rotor thrust, T_h is the horizontal (y-axis) component of thrust, T_v is the vertical (z-axis) component of thrust, and ϵ is the tail rotor cant angle.

In the case of the generic SMR helicopter, positive tail rotor cant would produce a positive vertical thrust component at the tail of the helicopter, which would result in a net pitch down moment. This could compensate for the aft-biased center of gravity without requiring increased aerodynamic loading on the main rotor, thus reducing stress on the rotor blade and improving the probability of fatigue failure.

Hypothesis 3.2 states that the preliminary fatigue design methodology is capable of predicting the impact of tail rotor cant angle on fatigue life:

Hypothesis 3.2

The preliminary fatigue design methodology will enable quantification of the impact of tail rotor cant on probability of rotor blade fatigue failure at a specified service life.

The null hypothesis to Hypothesis 3.2 can be stated as follows:

> The preliminary fatigue design methodology does not provide sufficient information to predict the impact of tail rotor cant angle on the probability of rotor blade fatigue failure.

This hypothesis will be tested by Experiment 3.2.

6.1.3 Design Mission Requirements

In the case that neither the blade cross section or vehicle design can be modified to improve fatigue life, designers may want to revisit the design mission requirements. Although mission requirements are traditionally treated as "set in stone", designers may have an opportunity to propose changes to the requirements if they are found to have an overly-detrimental effect on the product. Alternatively, the organization defining the requirements may wish to evaluate their impact on the reliability and cost of the helicopter.

Mission requirements are used to construct the mission spectrum, which directly drives the load spectrum, which is used as the basis for fatigue life prediction. Changes in mission requirements could therefore have a significant impact on the fatigue life of helicopter components. To the best of this author's knowledge, there are no works in the literature that directly assess the influence of design mission requirements on fatigue life.

In Experiment 1a, the high speed and high rate of climb cases were found to significantly increase the equivalent stress at the critical fatigue point. Experiment 2b found that the climb and cruise mission segments featured some of the highest equivalent stress distributions in the mission spectrum. In particular, the tails of the climb segment were "fatter" than those of the cruise segment, indicating that the blade is more likely to see higher stresses while climbing.

These results suggest that the probability of rotor blade fatigue failure could be reduced by reducing the rate of climb during the climb segment and/or reducing the airspeed in the cruise segment. Hypothesis 3.3 is divided into two parts: Hypothesis 3.3.1 asserts that the preliminary fatigue design methodology will be capable of modeling the impact of changes to rate of climb requirements, and Hypothesis 3.3.2 states that the methodology will predict the effects of changes to airspeed requirements.

Hypothesis 3.3

1. The preliminary fatigue design methodology will enable quantification of the impact of average rate of climb on probability of rotor blade fatigue failure at a specified service life.
2. The preliminary fatigue design methodology will enable quantification of the impact of average cruise speed on probability of rotor blade fatigue failure at a specified service life.

The null hypotheses to Hypotheses 3.3.1 and 3.3.2 can be stated as follows:

1. The preliminary fatigue design methodology does not provide sufficient information to predict the impact of average rate of climb on the probability of rotor blade fatigue failure.
2. The preliminary fatigue design methodology does not provide sufficient information to predict the impact of average cruise speed on the probability of rotor blade fatigue failure.

These hypotheses will be tested by Experiment 3.3.

6.2 Experiment 3 Overview

In Experiment 3, the predictive capabilities of the preliminary fatigue design methodology will be tested. Experiment 3 is a multi-part experiment, where each part aims to confirm the ability of the methodology to predict the impact of a different design variable. Experiment 3.1

tests the impact of rotor blade box spar thickness, Experiment 3.2 predicts the effects of tail rotor cant angle, and Experiment 3.3 studies the influence of rate of climb and cruise speed.

The following sections describe the design, implementation, and results of each part of Experiment 3. Afterwards, Hypotheses 3.1 to 3.3 will be revisited. The results of these experiments will either support or fail to support Conjecture 0.

6.3 Experiment 3.1

Figure 6.1 provides an overview of Experiment 3.1. The following sections detail the design of the experiment, implementation of the blade spar thickness design variable, results, and analysis.

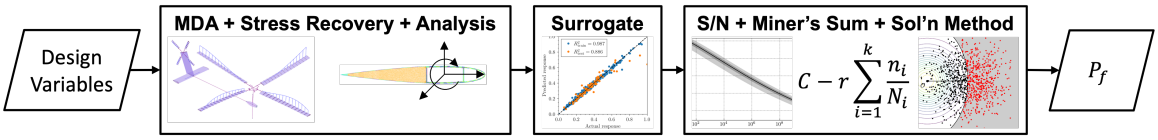


Figure 6.1: Overview of Experiment 3.1.

6.3.1 Experimental Design

Experiment 3.1 analyzes the impact of blade spar thickness on the predicted probability of rotor blade fatigue failure after 5000 FH. The results are compared against the baseline design described in Section 4.4.3.3 and quantified in Section 5.5.

The experiment is divided into five cases, including the baseline case. Each subsequent case increases the blade spar thickness by a slight amount by modifying the number of blade spar layers in the generic SMR helicopter PreVABS+VABS model. This rotor blade design variable can be easily modified using the Python MDA environment and does not require direct modification of the PreVABS input files.

In Experiment 1a, a critical fatigue point on the rotor blade cross section was identified by recovering the full equivalent stress field in a number of extreme flight conditions. In this experiment, it cannot be assumed that the critical fatigue point remains in the same location

as the thickness of the blade spar increases. A new critical fatigue point is identified for each case by recovering the full equivalent stress field for the baseline flight condition used in Experiment 1a; the complete extreme flight condition survey is not repeated for each case.

Next, the flight envelope of the generic SMR helicopter is sampled using the flight envelope DOEs developed in Experiment 1b. For each case, sampling covers the the same points in the training and testing sets used to develop the surrogate models that were applied in Experiment 2b. Based on the conclusions of Experiment 2 (see Section 5.6), an artificial neural network surrogate model is fit to each data set.

Finally, the Monte Carlo structural reliability solution tested in Experiment 2b is used to predict the probability of fatigue failure at 5000 FH. In this experiment, P_f at a fixed fatigue life is used as the independent variable since it only requires a single evaluation of the structural reliability problem. Predicting a new fatigue life would require specifying the desired P_f and running multiple iterations of the structural reliability problem with different values of r until P_f is met. The two problems are related: if a design change results in a reduction in P_f over the baseline value, it would also result in an increase in fatigue life at the baseline P_f.

Table 6.1 summarizes the cases that are tested in this experiment. The new critical fatigue points are also identified.

Table 6.1: Experimental design for Experiment 3.1.

Case	Number of spar layers	Spar thickness (ft)	Critical point location (ft)
Baseline	4	0.0125	(0.176, 0.066)
1	6	0.0200	(0.169, 0.066)
2	8	0.0274	(0.162, 0.066)
3	10	0.0349	(0.155, 0.066)
4	12	0.0423	(0.148, 0.066)

In this experiment, it is assumed that all of the cross section designs specified in Table 6.1 satisfy the structural, aeroelastic, and stability requirements of the rotor system. Additionally, it is assumed that the failure of the box spar auxiliary web remains the primary fatigue

failure mode for the rotor blade.

6.3.2 Implementation

The design variables modified in this experiment concern only the PreVABS+VABS rotor blade model, described previously in Section 4.4.3.3. Figure 4.12c provides a detailed view of the box spar of the rotor blade and Table 4.6 defines the different lamina used in the rotor blade. The main and auxiliary spar webs are constructed using a 0°/45°/–45°/90° IM7 layup.

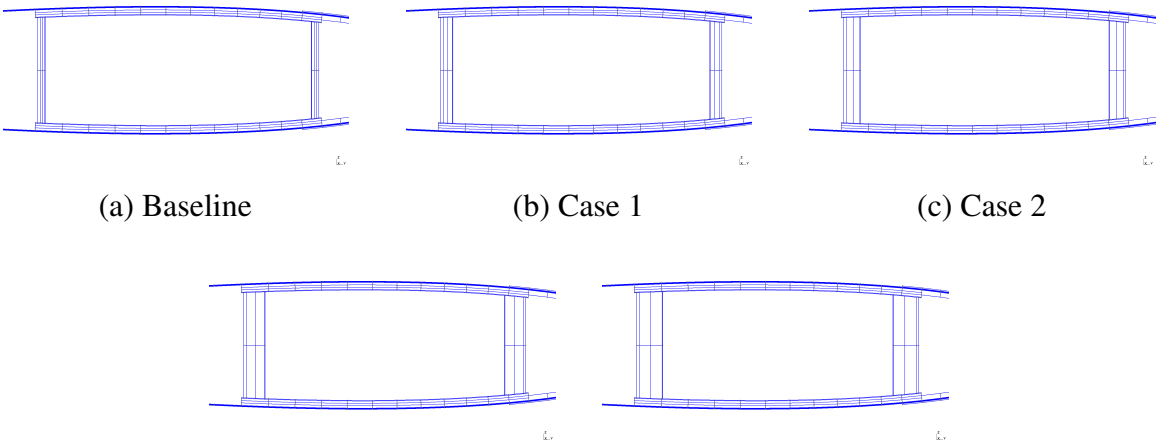
In this experiment, the 45°/–45° layers were simply repeated to increase the blade spar thickness as described in Table 6.1. For symmetry, the modifications were applied to both the main and auxiliary webs, which minimizes movement of the CG and maintains aerodynamic stability. Table B.32 includes the exact definition of the PreVABS+VABS variables used to effect this change, and describes their mapping to the OpenMDAO variables used by the MDA.

The geometries of the rotor blade cross section for each case are displayed in Figure 6.2. Because the basepoints defining the position of the main and auxiliary web spars are located on the outside of the spar box, the spars extend further inwards in each case, rather than extending towards the leading and trailing edges of the blade.

Figure 6.3 shows the resultant VABS mesh for each geometry in Figure 6.2 after the PreVABS automesher is applied. The mesh size is constant, so the number of mesh elements grows slightly for each case.

6.3.3 Results and Analysis

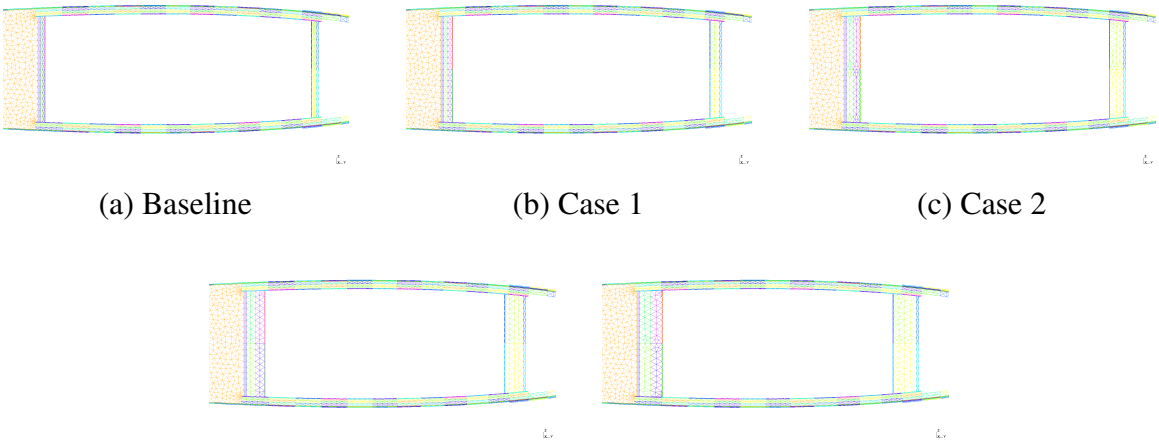
The equivalent stress distributions for each mission segment were derived in a manner identical to Experiment 2b. Figure 6.4 plots the S_{eq} distributions for each mission segment for each case listed in Table 6.1. The data is presented as a series of box plots rather than the violin plots used in Figure 5.13 because box plots provide better readability with a large



(a) Baseline (b) Case 1 (c) Case 2

(d) Case 3 (e) Case 4

Figure 6.2: Geometry of the PreVABS+VABS rotor blade model in different cases of Experiment 3.1 (box spar detail).



(a) Baseline (b) Case 1 (c) Case 2

(d) Case 3 (e) Case 4

Figure 6.3: Mesh of the PreVABS+VABS rotor blade model in different cases of Experiment 3.1 (box spar detail).

number of distributions on the same plot.

In Figure 6.4 each group of box plots applies to the same mission segment. Each individual box plot describes the results of a different blade spar thickness setting. The box plots visualize the median, interquartile range, extrema, and outliers of each S_{eq} distribution. Each box plot was created by sampling the corresponding equivalent stress distribution 10,000 times.

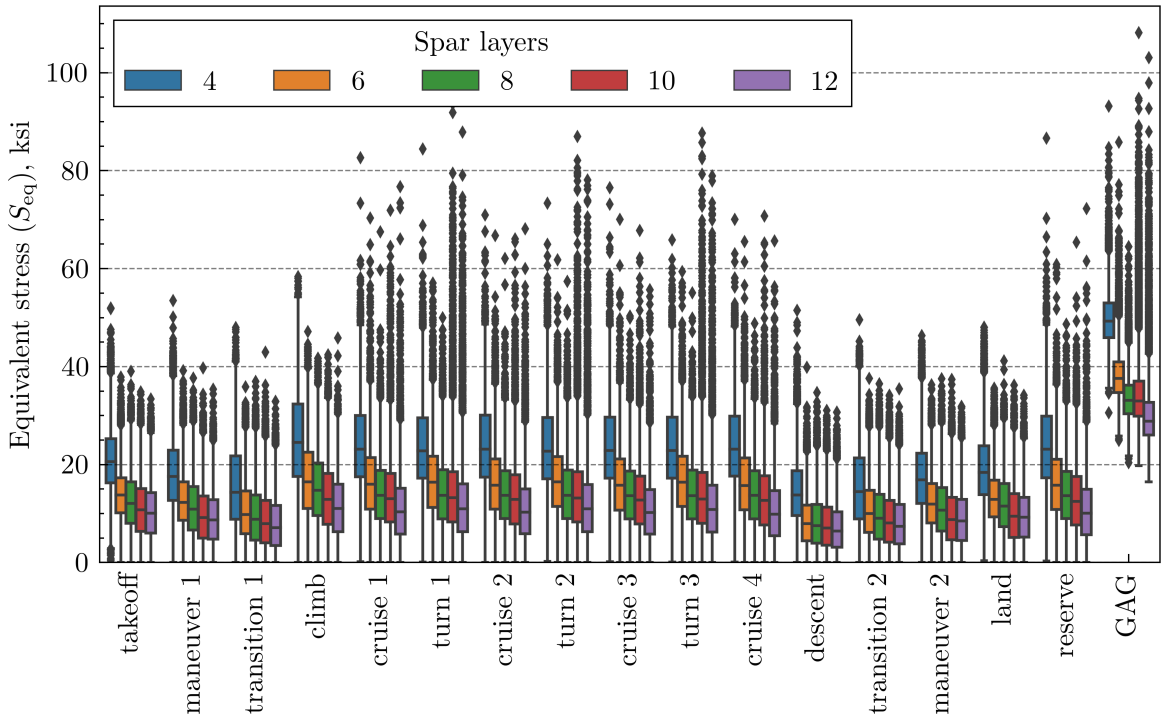


Figure 6.4: Equivalent stress distributions produced in Experiment 3.1.

From Figure 6.4, it is immediately apparently that the bodies of the stress distributions decrease as blade spar thickness increases. The medians and interquartile ranges are lowest at high numbers of spar layers for every segment. However, the most dramatic change occurs between the baseline case and the first case, with diminishing returns thereafter. This suggests that increasing the thickness of the spar web correlates with a decrease in equivalent stress for most flight conditions.

However, the tails of the distributions do not follow the same trend. In the third and fourth cases, the outliers of the S_{eq} distributions in the cruise, turn, reserve, and GAG segments extend further than the tails of the baseline, first, and second cases. The effect is strongest at 10 spar layers and decreases slightly at 12 spar layers.

Although it is difficult to determine the exact cause of this effect without a more detailed analysis of the rotor system, it is possible that modifications to the blade cross section have introduced an aeroelastic effect that cancels out the improvements in certain flight conditions. For example, if, in the third and fourth case, an aerodynamic–structural feedback

loop increases blade deformation in high speed flight, that could increase equivalent stress in those flight conditions. Regardless of the cause, these longer tails are likely to have a significant impact on probability of fatigue failure, as seen previously when comparing ANN and GPM surrogate models in Experiment 2b.

After analyzing the equivalent stress distributions, Monte Carlo simulation was used to predict the probability of fatigue failure at 5000 FH in a manner similar to Experiment 2b. These results are plotted in Figure 6.5. The height of each bar describes P_f, and the error bars illustrate the 95% confidence interval for each prediction.

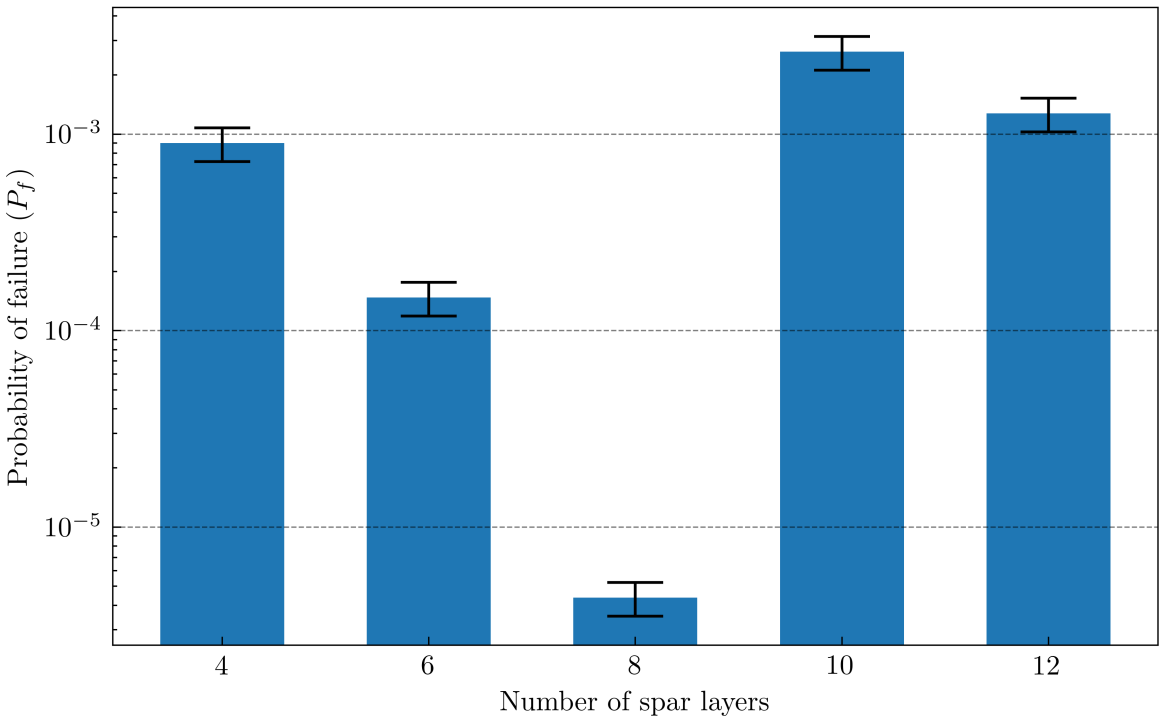


Figure 6.5: Impact of blade spar thickness on probability of fatigue failure at 5000 FH.

The results in Figure 6.5 follow logically from Figure 6.4. For the first two cases, P_f is improved significantly compared to the baseline case. For the third and fourth cases, the longer tails of the stress distributions increase P_f dramatically. In fact, each of these cases is worse than the baseline case in terms of probability of fatigue failure.

The statistical significance of these results was calculated using one-sided statistical means testing. For cases where the predicted P_f is lower than the baseline, the probability

of that case's P_f distribution being *greater* than the baseline is calculated, and vice versa for cases where P_f is higher than the baseline. If the calculated probability, p, is less than 0.01, the results are considered to be statistically significant. In Experiment 3.1, all cases were found to be statistically significant. For cases 1, 2, and 3, $p \approx 0$, and for case 4, $p = 0.008$.

Notably, this experiment disproved the heuristic proposed in Section 6.1.1, where it was suggested that increasing spar thickness would monotonically decrease P_f. Instead, the probabilistic fatigue design methodology demonstrated the existence of higher-order effects leading to negative P_f improvement that may not be obvious to a designer basing their decisions on rules of thumb and experience.

6.4 Experiment 3.2

Figure 6.6 provides an overview of Experiment 3.2. The following sections detail the design of the experiment, implementation of the tail rotor cant design variable, results, and analysis.

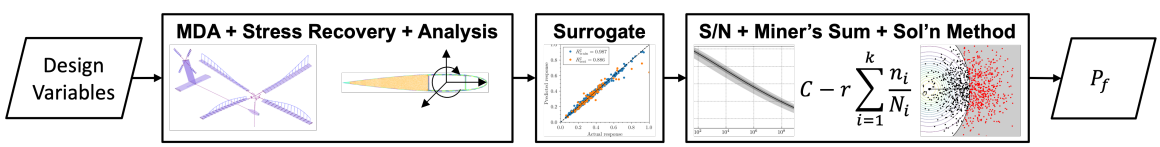


Figure 6.6: Overview of Experiment 3.2.

6.4.1 Experimental Design

Experiment 3.2 examines the impact of tail rotor cant angle on the probability of main rotor blade fatigue failure at 5000 FH. This experiment, which is conducted similarly to Experiment 3.1, is intended to demonstrate the ability of the probabilistic fatigue design methodology to predict the fatigue life impact of vehicle layout changes.

This experiment is divided into five cases, including the baseline case. Each subsequent case increases the tail rotor cant angle, ϵ, by 10°. The tail rotor cant angle MDA variable is connected to the NDARC and RCAS models.

Unlike Experiment 3.1, it is not necessary to relocate the critical fatigue point in this experiment because the baseline rotor blade cross section design is used for all cases. However, the flight envelope sampling and ANN training processes must be complete for each case. After new ANN surrogate models are trained, the Monte Carlo structural reliability solution is used to find the probability of main rotor blade fatigue failure as in Experiment 2b and Experiment 3.1.

Table 6.2 describes each case tested in this experiment. For reference, the horizontal and vertical coefficients of tail rotor thrust, $\cos\epsilon$ and $\sin\epsilon$, respectively, are also provided for each value of ϵ.

Table 6.2: Experimental design for Experiment 3.2.

Case	Tail rotor cant, ϵ (deg)	T_v coefficient	T_h coefficient
Baseline	0	0	1
1	10	0.174	0.985
2	20	0.342	0.940
3	30	0.500	0.866
4	40	0.643	0.766

In this experiment, it is assumed that modifications to the tail rotor cant angle do not negatively impact the performance, handling qualities, stability, or controllability of the generic SMR design. That is, changes in ϵ will not produce a helicopter design that fails to meet any of its original requirements.

6.4.2 Implementation

The design variables in this experiment concern the NDARC and RCAS models, described previously in Section 4.4.3.1 and Figure 4.11. In NDARC, the tail rotor cant angle can be set directly. A positive tail rotor cant angle corresponds to a counterclockwise rotation of the tail rotor hub, when viewed from behind the helicopter. The mapping between the OpenMDAO tail rotor cant variable and the NDARC tail rotor cant variable is included in Table B.2.

In the RCAS model, the orientation of the structural and aerodynamic tail rotor subsystems must both be modified. Additionally, because the baseline RCAS model is defined in a x-forward, y-right, z-down coordinate system, the tail rotor subsystems are rotated by 180° about the y axis, then –90° about the x axis to achieve a proper orientation. A new module was added to the OpenMDAO model to convert the tail rotor cant angle to RCAS x axis rotations. This transformation is described by Equation (6.3):

$$\epsilon_{\text{RCAS}} = -90^\circ + \epsilon \tag{6.3}$$

This module is positioned just before the RCAS wrapper in the execution order. The mapping between the OpenMDAO tail rotor cant variable and the RCAS tail rotor cant variable is included in Table B.30.

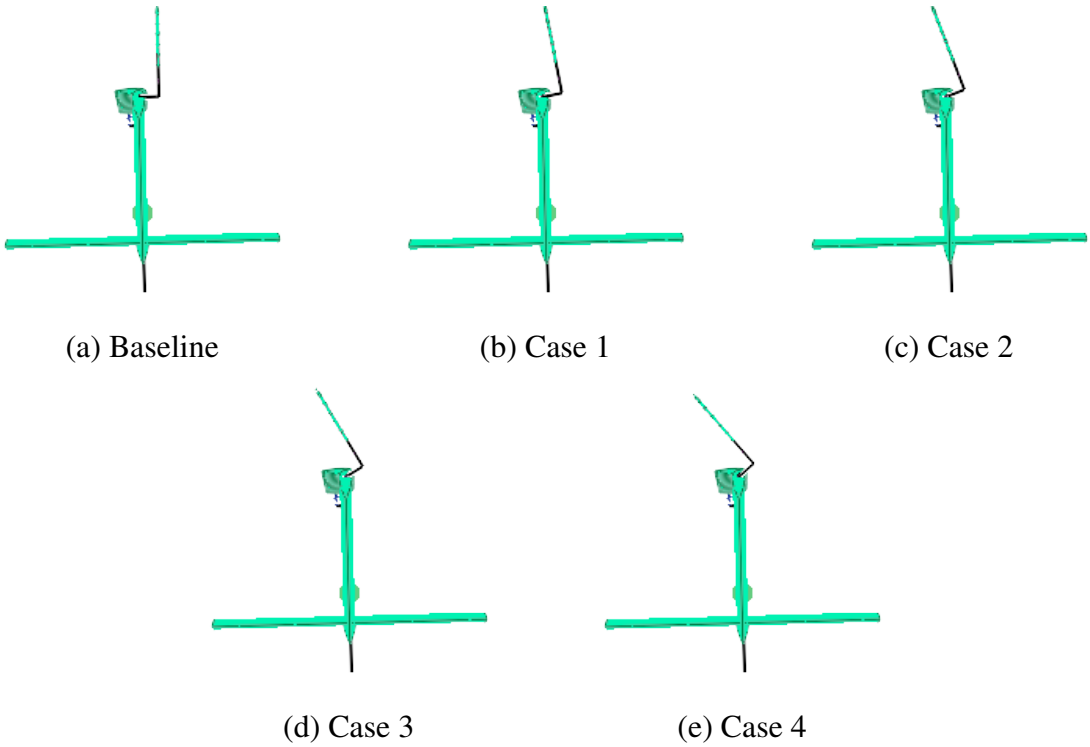
The impact of tail rotor cant angle on the RCAS model is presented in Figure 6.7. The view is from behind the helicopter. Note that the horizontal and vertical stabilizers also have a slight counterclockwise rotation from horizontal because the model is oriented to the attitude it would take in trimmed flight at cruise speed.

6.4.3 Results and Analysis

Figure 6.8 plots the equivalent stress distributions for each case and each mission segment. The data in this figure largely rejects the assumptions of Section 6.1.2, which posited that increasing the tail rotor cant angle would reduce S_{eq} on the main rotor blade.

Instead, despite slight improvements in S_{eq} in the first case, cases 2 to 4 show elevated stress in both the bodies and tails of the distributions. It appears that the change in tail rotor cant has increased the magnitude of aerodynamic moments required of the main rotor to maintain trimmed flight.

This may be due to a a reduction in anti-torque effectiveness: as the horizontal component of tail rotor thrust decreases (see Table 6.2), the tail rotor collective must increase to



(a) Baseline (b) Case 1 (c) Case 2

(d) Case 3 (e) Case 4

Figure 6.7: Varying tail rotor cant angles in RCAS model for Experiment 3.1 (view from behind helicopter).

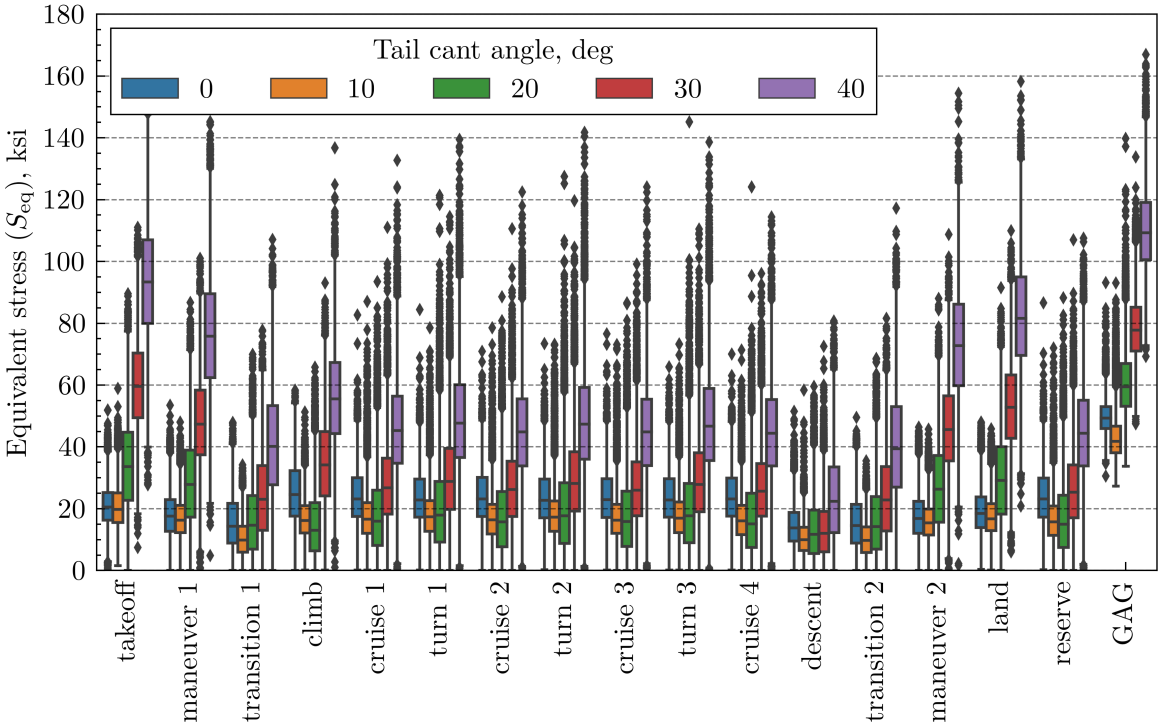


Figure 6.8: Equivalent stress distributions produced in Experiment 3.2.

counteract the torque from the main rotor. If the tail rotor collective reaches a point where the blades begin to stall, it will be less effective and more control authority will be required from the main rotor. Alternatively, the increase in tail rotor cant may have upset the stability of the helicopter in forward flight conditions, with larger pilot control inputs required to compensate.

The resultant impact on the predicted main rotor blade fatigue life is presented in Figure 6.9. As expected from Figure 6.8, increasing tail rotor cant increases P_f at 5000 FH.

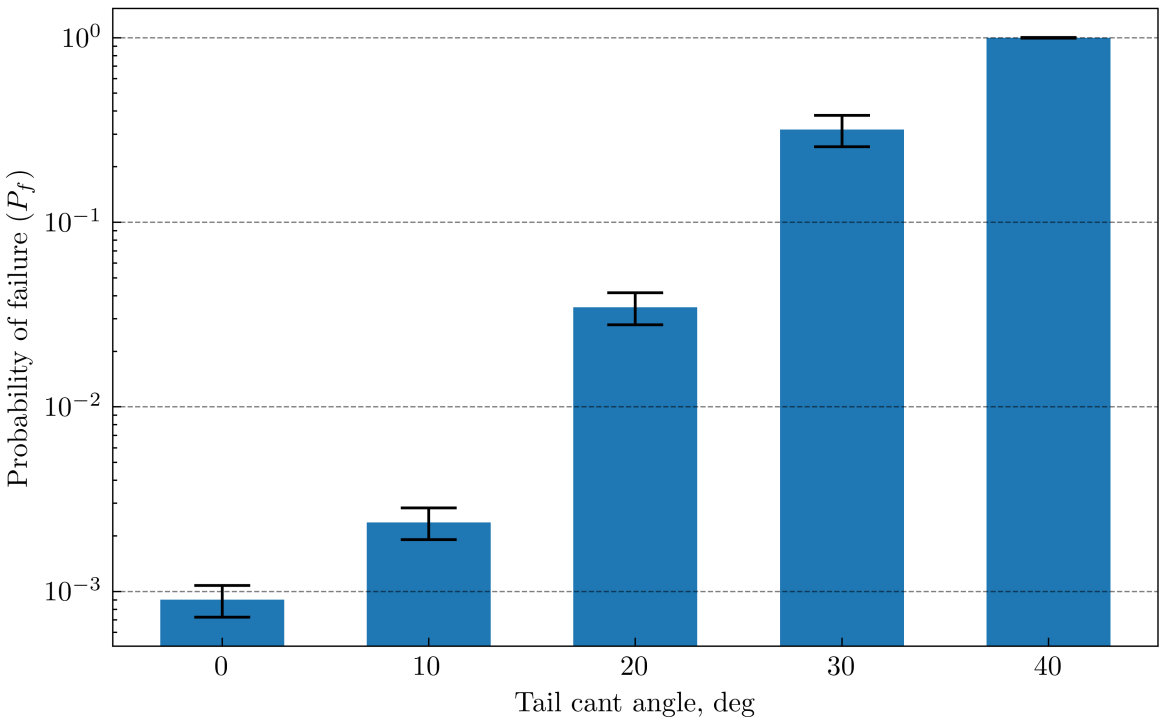


Figure 6.9: Impact of tail rotor cant angle on probability of fatigue failure at 5000 FH.

By the fourth case, $P_f \approx 1$: the main rotor blade is nearly guaranteed to fail at or before 5000 FH. Each case is statistically significant at $p \approx 0$.

Recall from Table 5.3 that each segment in the mission spectrum uses a CG distribution described by Equation (6.4):

$$X_{\mathrm{CG}} \sim \mathcal{T}(-0.3, 0, 0.3, 0.6)\,\mathrm{ft} \tag{6.4}$$

where X_{CG} is a random variable describing the x-axis shift of the CG location compared to the main rotor hub station line.

This distribution was selected to model typical loading conditions experienced by a transport helicopter, but does not cover the complete CG range tested in Experiment 1b, which spanned from –0.3 ft to 1.6 ft. To further test the impact of tail rotor cant, the baseline CG distribution from Equation (6.4) was replaced by an aft-shifted CG distribution described by Equation (6.5):

$$X_{\mathrm{CG}} \sim \mathcal{T}(0, 0.5, 1.0, 1.5)\,\mathrm{ft} \tag{6.5}$$

This distribution spans the range from the main rotor hub station line to 1.5 ft aft of that point. It is hypothesized that the impact of tail rotor cant will be more favorable when using the aft-shifted CG distribution since the need to produce aerodynamic pitch down moment to counteract the inherent pitch up moment is higher.

Figures 6.10 and 6.11 plot the equivalent stress distributions and P_f predictions, respectively, for the new CG distribution. Although the stresses and resultant probabilities of failure are higher than Figures 6.8 and 6.9 overall, the impact of tail rotor cant improves, rather than degrades, the probability of failure.

All the results presented in Figure 6.11 are statistically significant at $p \approx 0$. This experiment adds an important qualification to the heuristic stated in Section 6.1.2, where it was hypothesized that tail rotor cant will always improve probability of main rotor blade fatigue failure. Instead, tail rotor cant will only improve P_f if the CG distribution is shifted aftwards.

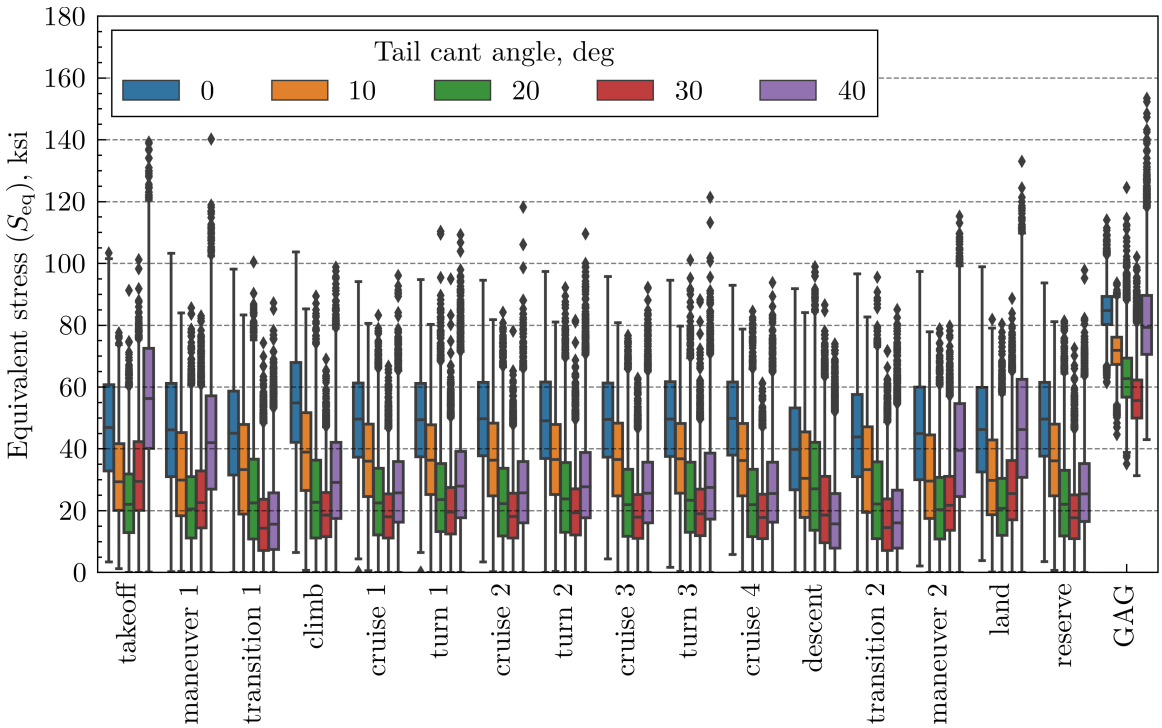


Figure 6.10: Equivalent stress distributions produced in Experiment 3.2 with the aft-shifted CG distribution.

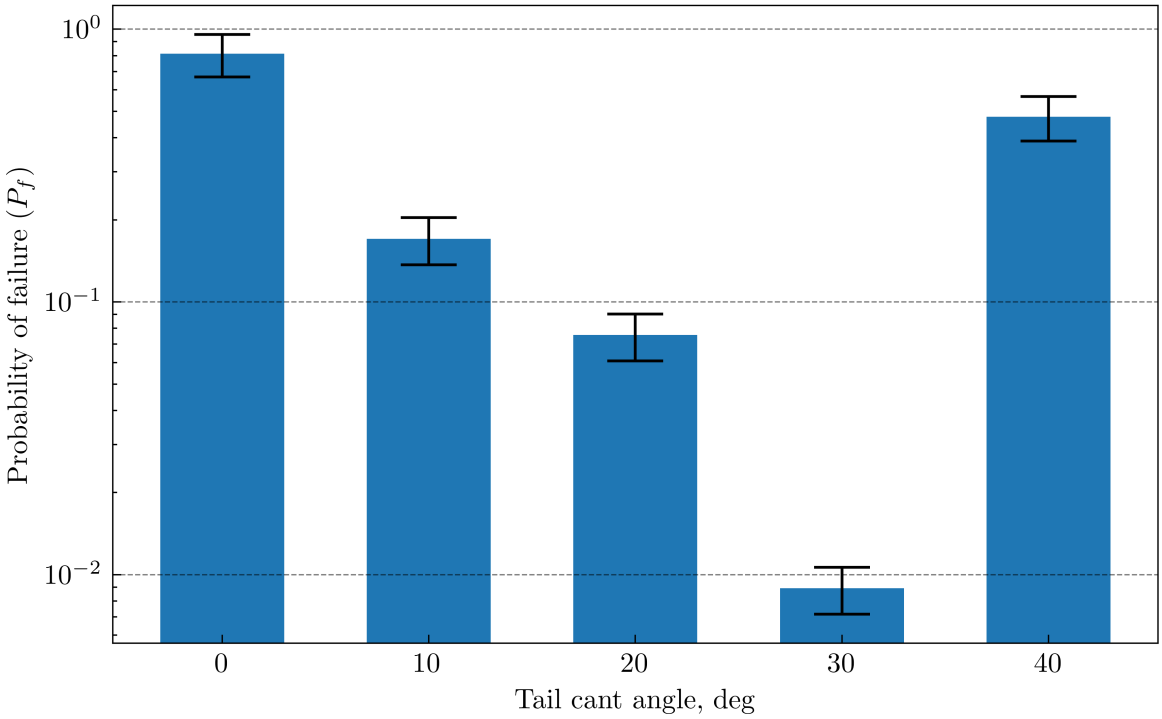


Figure 6.11: Impact of tail rotor cant angle on probability of fatigue failure at 5000 FH with the aft-shifted CG distribution.

6.5 Experiment 3.3

Figure 6.12 provides an overview of Experiment 3.3. The following sections detail the design of the experiment, modifications of the mission spectrum to reflect changes in mission variables, results, and analysis.

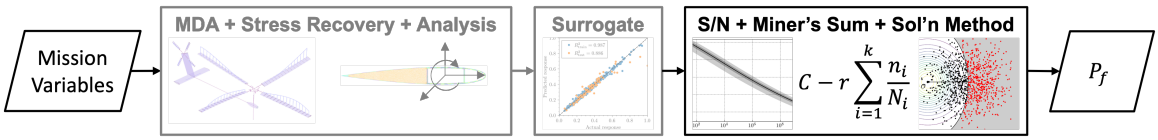


Figure 6.12: Overview of Experiment 3.3.

6.5.1 Experimental Design

Experiment 3.3 is split into two parts. First, the impact of rate of climb on the probability of main rotor blade fatigue failure by 5000 FH is examined. Next, the impact of cruise speed is evaluated. 6This experiment is intended to quantify the ability of the probabilistic fatigue design methodology to predict the impact of mission and performance requirements on fatigue life.

Unlike Experiments 3.1 and 3.2, it is not necessary to modify the generic SMR helicopter MDA model, resample the flight envelope, or retrain the ANN surrogate models, because the design of the helicopter does not change. Instead, the baseline stress surrogate model developed during Experiment 2b is used. Only the probabilistic mission spectrum, presented in full in Table 5.3, is modified in this experiment.

In the first part of Experiment 3.3, the rate of climb distribution in the climb segment is modified from its baseline value of $\Delta(600, 900, 1200)$ ft/min. In the first two cases, the mode of the triangular distribution is reduced to 600 ft/min and 700 ft/min, and the minimum and maximum are similarly adjusted to maintain a total range of 600 ft/min. In the final two cases, the mode of the triangular distribution is raised to 1000 ft/min and 1100 ft/min. The minimum is raised to 300 ft/min below the mode, but the maximum is held constant at

1200 ft/min to avoid exceeding the upper bound of the ROC input variable in the surrogate model. Table 6.3 defines each case entirely.

Table 6.3: Experimental design for the first part of Experiment 3.3.

Case	Mode of ROC (ft/min)	ROC distribution (ft/min)
Baseline	900	$\Delta(600, 900, 1200)$
1	700	$\Delta(400, 700, 1000)$
2	800	$\Delta(500, 800, 1100)$
3	1000	$\Delta(700, 1000, 1200)$
4	1100	$\Delta(800, 1100, 1200)$

For the cruise speed study, the airspeed distributions in the cruise, turning, and reserve mission segments are modified from their baseline value of $\Delta(100, 130, 160)$ kt. In the first two cases, the mode is reduced to 110 kt and 120 kt, and the minimum and maximum are reduced to maintain a range of 60 kt. In the final two cases, the mode is raised to 140 kt and 150 kt, and the minimum is raised to 30 kt below the mode. The maximum is held constant at 160 kt to respect the upper bound of the V input variable. These cases are defined in Table 6.4.

Table 6.4: Experimental design for the second part of Experiment 3.3.

Case	Mode of V (kt)	V distribution (kt)
Baseline	130	$\Delta(100, 130, 160)$
1	110	$\Delta(80, 110, 140)$
2	120	$\Delta(90, 120, 150)$
3	140	$\Delta(110, 140, 160)$
4	150	$\Delta(120, 150, 160)$

Throughout Experiment 3.3, it is assumed that the performance and mission requirements used to design the generic SMR helicopter can be adjusted as necessary. That is, there will be no conflict between the customer and the manufacturer if the target rate of climb and cruise speed for the design mission are pushed below or above their initial specifications.

6.5.2 Results and Analysis

The stress distributions produced during the rate of climb study are included in Figure 6.13. As predicted, the stress in the climb segment is lower for cases where the mode of the ROC distribution is below the baseline, and higher for cases where it is greater. These effects are replicated on a smaller scale in the GAG equivalent stress distribution. The GAG distribution is only affected by the ROC distribution for cases where the maximum stress is seen during the climb segment. Note that minor differences in the stress distributions of other segments are simply due to the random sampling of each distribution used to create the box plots; in reality, there is no change in the distributions.

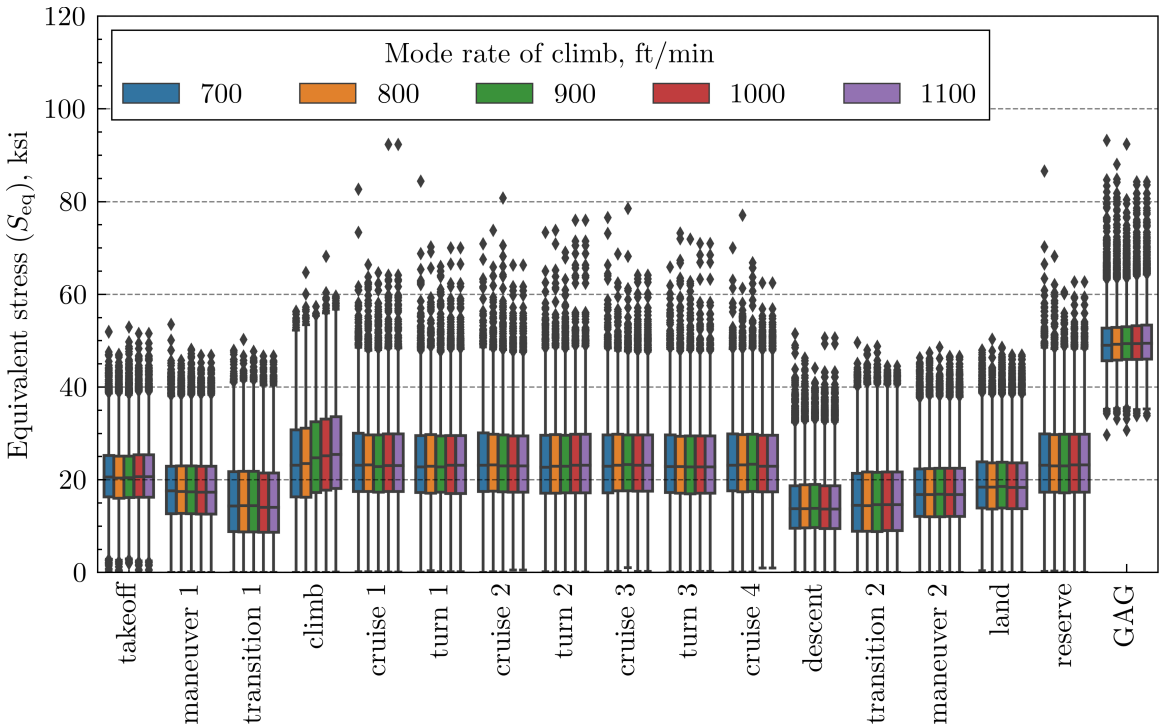


Figure 6.13: Equivalent stress distributions produced by the ROC study in Experiment 3.3.

Figure 6.14 presents the results of the Monte Carlo structural reliability calculations for P_f. The results of each case are nearly indistinguishable from the baseline case. Clearly, the effect of modifying the ROC distribution is not impactful enough to significantly influence the resultant P_f calculation, despite clearly impacting the S_{eq} distributions that drive P_f.

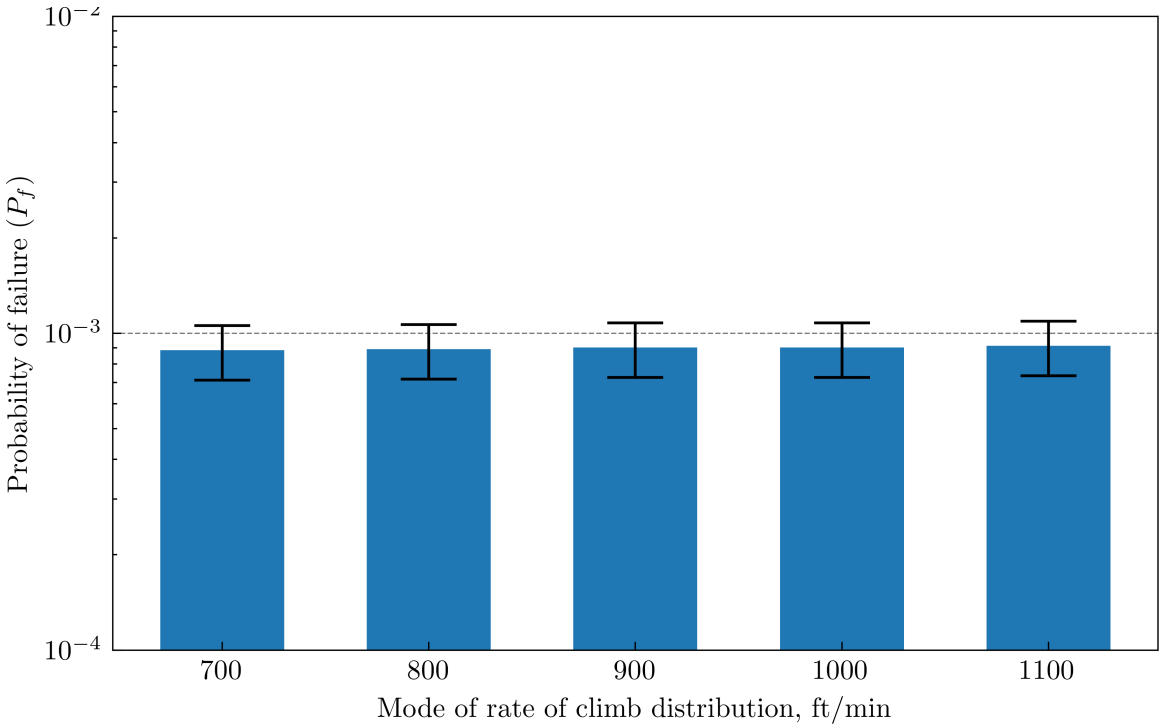


Figure 6.14: Impact of rate of climb on probability of fatigue failure at 5000 FH.

The results of cases 1 to 4 are statistically insignificant at $p = 0.45$, $p = 0.47$, $p = 0.50$, and $p = 0.47$. This indicates that the current form of the preliminary fatigue design methodology is not sensitive enough to predict extremely minor changes in P_f. This drawback is inherent to any sampling solution methods and could be improved by decreasing the maximum coefficient of variation convergence criterion at the expense of solution runtime.

The stress distributions produced during the airspeed study are presented in Figure 6.15. As predicted, the severity of the equivalent stress distributions scales proportionally to airspeed in the cruise, turning, and reserve segments. There is also a noticeable impact on the GAG S_{eq} distribution.

Also note that the tails of the stress distributions in the cruise and turning segments are highest for the cases where the mode of V is 130 kt, 140 kt, and 150 kt. In previous experiments, it was hypothesized that the tails of the stress distributions for these segments was driven by high speed flight in the 150 kt to 160 kt regime. Figure 6.15 confirms this

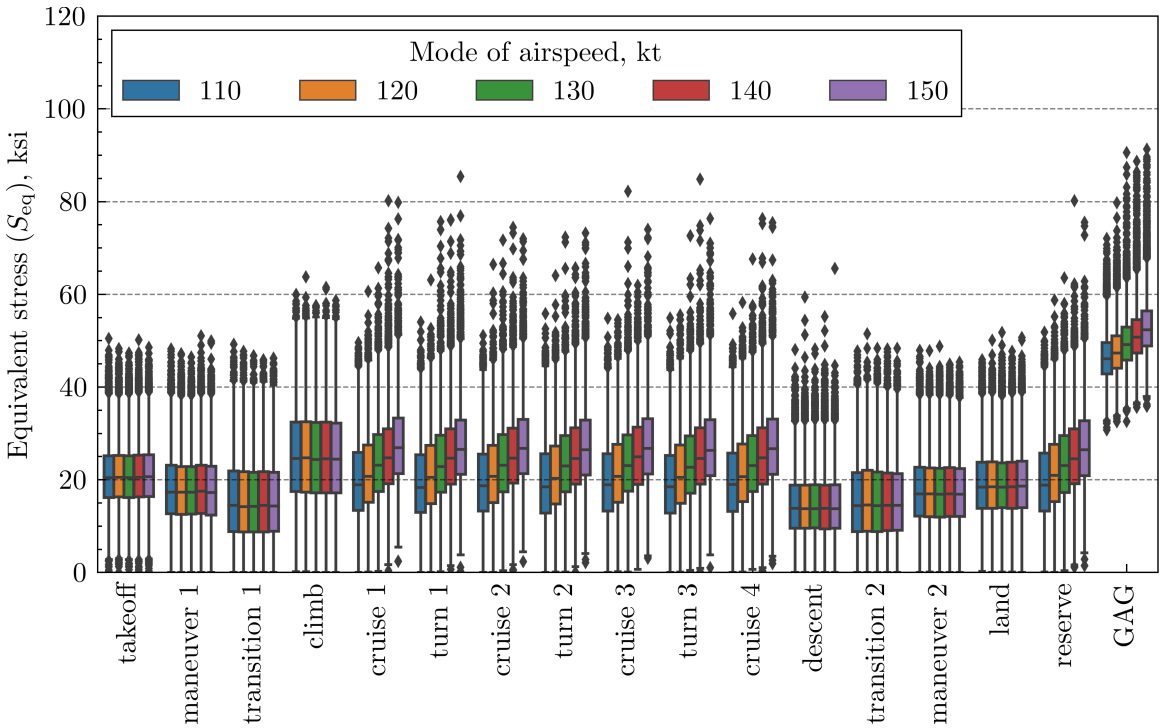


Figure 6.15: Equivalent stress distributions produced by the cruise speed study in Experiment 3.3.

hypothesis, as the stress distributions for the 110 kt and 120 kt cases, which have maxima less than 150 kt, have comparatively short tails.

The influence of these stress distributions on P_f is presented in Figure 6.16. These results also follow the expected trend: reducing cruise speed improves P_f, while cruising at faster speeds elevates P_f. The results in Figure 6.16 are statistically significant: for cases 1 to 4, $p \approx 0$, $p \approx 0$, $p = 0.001$, and $p \approx 0$.

While the first part of Experiment 3.3 suggested that the probabilistic fatigue design methodology may not be sufficiently sensitive to predict the results of changes in mission requirements, the cruise speed study demonstrated that stronger effects are relatively easy to predict with statistical significance.

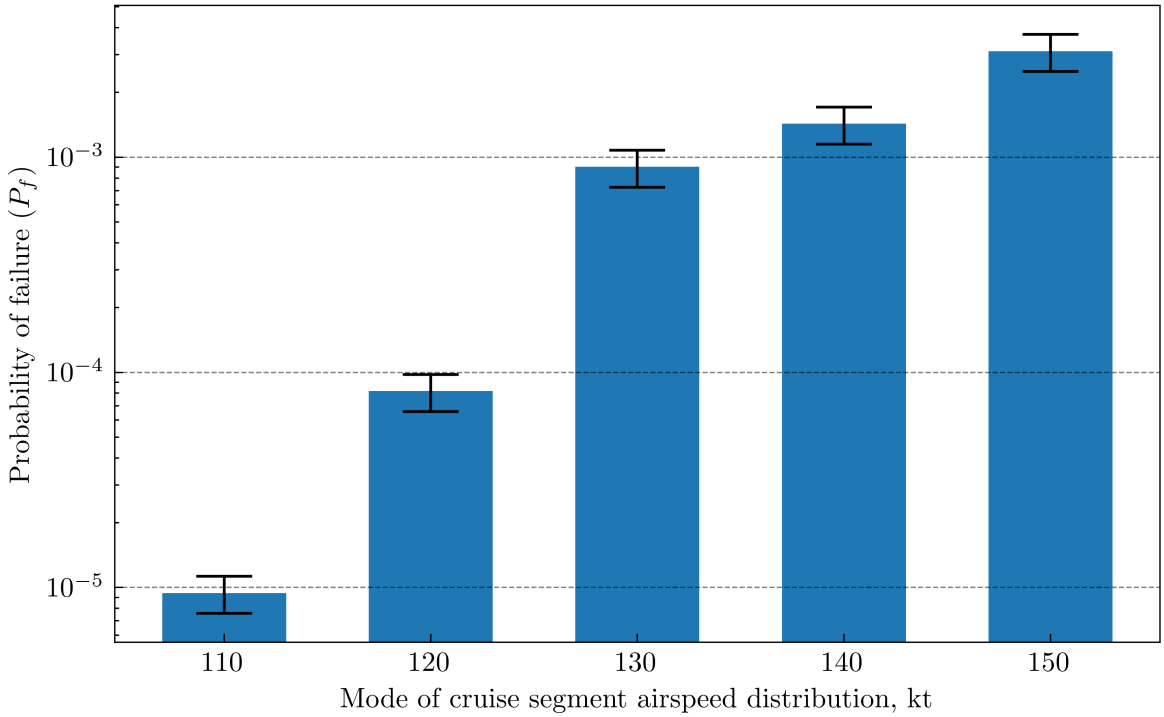


Figure 6.16: Impact of cruise speed on probability of fatigue failure at 5000 FH.

6.6 Conclusions

Experiment 3 tested the ability of the probabilistic fatigue design methodology to predict changes in the fatigue life characteristics of a conceptual rotorcraft design based on changes to that design or its mission requirements. This experiment also served as a proof-of-concept demonstrating the different manners by which this methodology can be integrated into rotorcraft design and analysis environments.

Experiment 3.1 tested the methodology's ability to detect changes to the rotor blade cross section design. Recall Hypothesis 3.1:

Hypothesis 3.1

The preliminary fatigue design methodology will enable quantification of the impact of blade spar thickness on probability of rotor blade fatigue failure at a specified service life.

The results of Experiment 3.1 *support* Hypothesis 3.1. An increase in blade spar

thickness was found to improve probability of fatigue failure up to a certain point, after which the probability of failure increased. This effect was not predictable prior to Experiment 3.1, which reinforces the usefulness of the methodology.

Experiment 3.2 tested the ability of the probabilistic fatigue design methodology to predict the impact of vehicle layout changes, namely the tail rotor cant angle. Recall Hypothesis 3.2:

Hypothesis 3.2

The preliminary fatigue design methodology will enable quantification of the impact of tail rotor cant on probability of rotor blade fatigue failure at a specified service life.

The results of Experiment 3.2 *support* Hypothesis 3.2. Increasing tail rotor cant angle was found to have negative effects on P_f when the baseline mission spectrum was used, but modifications to the baseline mission spectrum revealed that tail rotor cant can improve P_f if aft-biased center of gravity locations are more common.

Finally, Experiment 3.3 explored if the methodology could predict the influence of changes to performance or design mission requirements. Recall Hypothesis 3.3:

Hypothesis 3.3

1. The preliminary fatigue design methodology will enable quantification of the impact of average rate of climb on probability of rotor blade fatigue failure at a specified service life.
2. The preliminary fatigue design methodology will enable quantification of the impact of average cruise speed on probability of rotor blade fatigue failure at a specified service life.

The results of Experiment 3.3 *reject* Hypothesis 3.3.1 (i.e. the null hypothesis cannot be rejected) but *support* Hypothesis 3.3.2. Inherent uncertainty in the P_f predictions produced by all structural reliability sampling solution methods mean that the preliminary fatigue design methodology is not sensitive enough to calculate the relatively minor changes in P_f due to changes in the rate of climb requirement. However, more impactful effects, like those

of the airspeed requirement, were easy to predict with statistical significance. This indicates that the methodology is better-suited to predicting major effects than minor effects.

CHAPTER 7
CONCLUDING REMARKS

7.1 Research Summary

This thesis details the development of a proposed preliminary fatigue design methodology and the execution of several experiments intended to inform its development and explore its capabilities.

In Chapter 1, two of the principal disadvantages of rotorcraft, high operating cost and poor safety record, were investigated. A review of relevant literature and accident statistics found that both areas could likely be improved by reducing the amount of flight-critical component failures (Observations 1.1, 1.2, and 1.4). A significant proportion of these failures are caused by material fatigue, which is an unavoidable damage mode that accumulates constantly in rotary-wing aircraft during flight (Observation 1.5). Given the need for highly reliable and maintainable vertical lift aircraft in the near future, new rotorcraft development programs should aim to guarantee high fatigue life through fatigue-driven design decisions (Observation 1.3). These observations led to Research Question 0:

Research Question 0
How can the fatigue life of rotor system components be efficiently evaluated for use as a design driver in a rotorcraft design framework?

In order to answer Research Question 0, four research objectives were proposed:

1. Review the strengths and weaknesses of traditional and modern rotorcraft fatigue design methods.
2. Formulate a new rotorcraft fatigue design methodology to address the weaknesses of traditional methods.
3. Conduct virtual experiments to test, validate, and refine different elements of the

methodology.

4. Demonstrate the viability of the methodology using a hypothetical fatigue design exercise.

In Chapter 2, literature related to Research Objective 1 was reviewed, guided by Literature Questions 1 to 3. First, the process of aircraft design in general, and rotary-wing aircraft design in particular, was researched. It was determined that fatigue improvement efforts could begin in the preliminary design stage at the earliest (Observation 2.1).

Next, frameworks and tools used for rotary-wing vehicle design and rotor system analysis and design were reviewed. It was found that most RAM-C–focused vehicle design tools are plagued by difficulties populating the inputs with realistic data without resorting to historical projections (Observation 2.2). Additionally, most physics-based rotor system design tools only consider fundamental physical characteristics of the system and do not make efforts to predict reliability and maintainability (Observations 2.3 and 2.4).

Finally, the rotorcraft fatigue design process was discussed, including fatigue damage theories, traditional fatigue design methods, and new approaches to the fatigue design problem. Primarily, traditional fatigue design methods are limited by their reliance on historical flight loads surveys to predict component loads and safety factors to achieve high fatigue life reliability (Observations 2.5 to 2.7). Several new approaches show promise but in each case limitations in scope or capability prevent these methods from being used as a complete design methodology (Observation 2.8).

At the end of Chapter 2, three gaps in the literature were identified that must be closed in order to answer Research Question 0. Gap 1 addresses the missing link between vehicle design tools and rotor design tools:

Gap 1

There is a missing link between rotor design tools and rotary-wing vehicle design tools that prevents physics-based prediction of component service lives and resulting RAM-C characteristics in the preliminary design stage.

Gap 2 addresses the challenge of building a realistic load spectrum for arbitrary vehicle configurations rapidly and without relying on historical data:

Gap 2

No well-developed methods are available to rapidly derive the load spectrum of revolutionary vertical lift configurations in the preliminary design stage.

Finally, Gap 3 highlights the necessity of removing arbitrary safety factors from the fatigue design process:

Gap 3

There are no consistent and repeatable methods to ensure high fatigue life reliability in regular use in the rotorcraft industry.

In Chapter 3, a conjecture to Research Question 0 was formulated based on Gap 1:

Conjecture 0

A new preliminary fatigue design methodology can be created by enhancing traditional fatigue design methodologies with modern rotorcraft analysis tools and integrating new methods to improve flexibility and runtime. This enables the use of rotor component fatigue life as a design driver for future rotary-wing vehicle development programs.

Additionally, two subsequent research questions based on Gaps 2 and 3 were posed. Research Question 1 addresses the computational expense of using comprehensive analysis load predictions to build a complete load spectrum:

Research Question 1

How can a complete load spectrum be rapidly derived using physics-based comprehensive analysis tools?

Research Question 2 addresses the need to probabilistically solve the fatigue life problem rather than solving deterministically with safety factors:

Research Question 2

How can probabilistic methods be applied to efficiently remove the dependence of traditional fatigue design methodologies on reductions and safety factors?

To address Research Objective 2, an initial design of the preliminary fatigue design methodology was diagrammed, which can be found in Figure 3.1. Research Question 3 was proposed to test the capabilities of the methodology:

Research Question 3

Does the preliminary fatigue design methodology enable evaluation of the relative impact of common preliminary design variables on the probability of fatigue failure of a flight-critical component in a conceptual helicopter design?

In Chapter 4, scalar surrogate modeling methods were reviewed and proposed as a solution to Research Question 1:

Hypothesis 1

At least one of the surveyed surrogate modeling methods can be used to derive a complete load spectrum from comprehensive analysis results, enabling rapid design space exploration.

To address Research Objective 3, Experiment 1 was developed to test Hypothesis 1 and determine if surrogate models could improve upon the slow execution speed of comprehensive analyses.

In Experiment 1a, a multidisciplinary analysis environment was developed coupling three state-of-the-art helicopter analysis programs: NDARC for weight prediction and performance analysis, RCAS for comprehensive and structural analysis, and VABS for rotor blade cross section design and analysis. A generic single main rotor helicopter model was implemented within the MDA environment to serve as a test case. The generic SMR helicopter was subject to a number of extreme flight conditions to identify the critical fatigue point and study the sensitivity of equivalent stress at that point to different flight envelope variables.

In Experiment 1b, three popular surrogate modeling techniques (response surface methods, artificial neural networks, and Gaussian process models) were compared based on their ability to accurately predict mean stress and stress amplitude at the critical fatigue point.

Experiment 1 supported Hypothesis 1: ANN and GPM surrogate models were found to offer satisfactory predictive performance. Furthermore, the application of surrogate modeling greatly reduces the number of MDA samples that are required to solve a probabilistic fatigue life problem.

In Chapter 5, structural reliability methods were reviewed and proposed as solution to Research Question 2:

Hypothesis 2

At least one of the surveyed structural reliability sampling methods can be used to efficiently remove the dependence on reductions and safety factors by quantifying the reliability of fatigue life predictions using Miner's sum.

To further Research Objective 3, Experiment 2 was designed to test Hypothesis 2 and determine if structural reliability methods could provide more trustworthy solutions to the fatigue life prediction problem than traditional deterministic methods.

In Experiment 2a, a notional fatigue reliability problem was developed based on an assumed load distribution, a probabilistic S-N curve, and the Miner's sum fatigue life prediction model. A number of popular solution methods including analytical methods (first- and second-order structural reliability methods), sampling methods (Monte Carlo, quasi-Monte Carlo, Latin hypercube sampling, and directional sampling), and one hybrid method (importance sampling) were compared based on their ability to efficiently and accurately solve the notional fatigue life problem.

These results were validated in Experiment 2b, where the notional fatigue life problem was replaced by a complete helicopter fatigue analysis using a realistic mission spectrum and stress predictions from the surrogate models developed in Experiment 1b. The structural reliability solutions were also compared against hypothetical deterministic solutions based

on traditional fatigue design methods available in the literature.

Experiment 2 partially supported Hypothesis 2: solutions to the fatigue life problem were found using structural reliability methods, but the complexity of the problem and its input distributions meant that Monte Carlo simulation was the only useful method, which suffers from low efficiency at low probabilities of failure.

After Experiments 1 and 2, the probabilistic fatigue design methodology was finalized based on the results of these experiments. For the reader's convenience, a flow chart describing the final methodology is reprinted below.

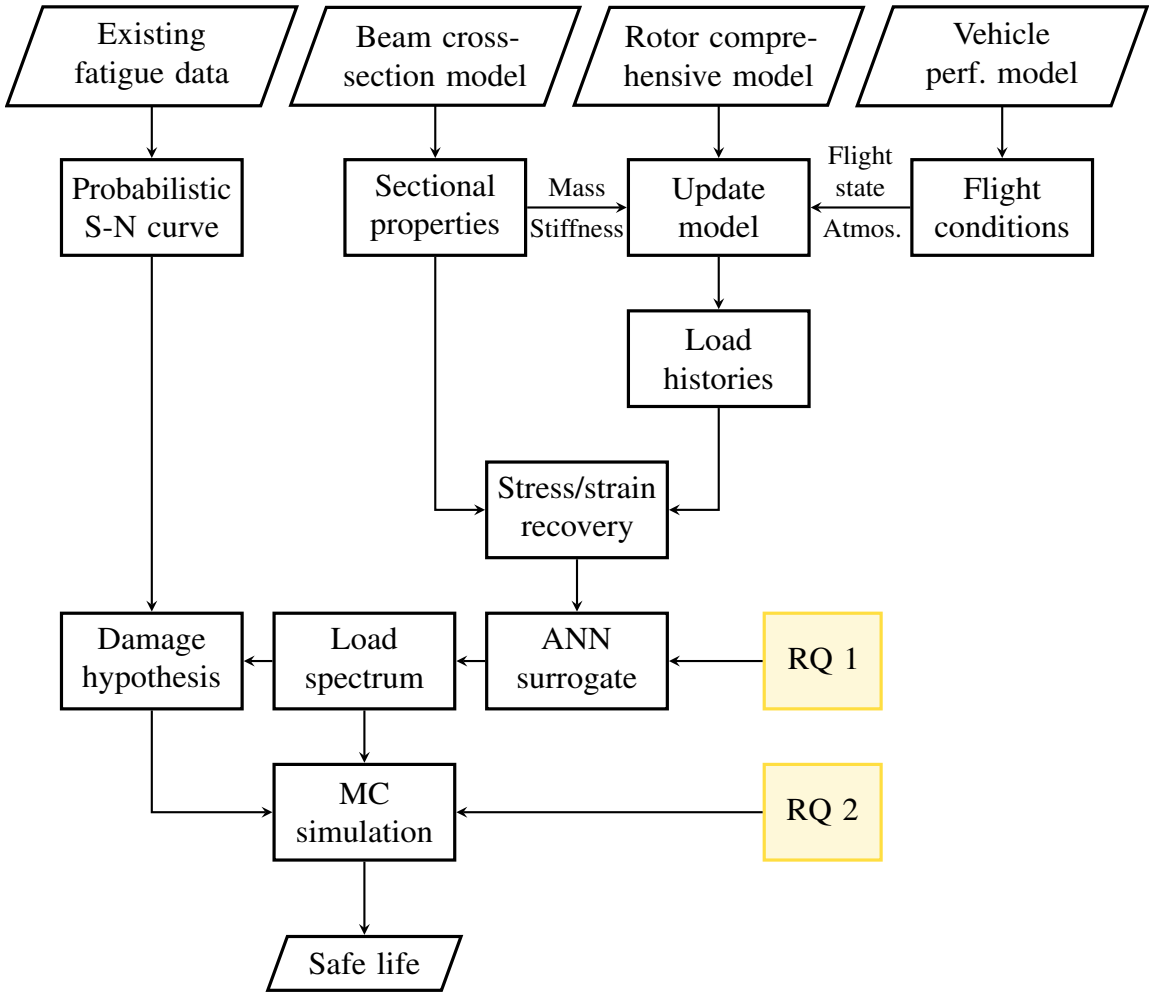


Figure 5.15: Flowchart of the preliminary fatigue design methodology after Experiments 1 and 2 (reprinted from Page 263).

Finally, to address Research Objective 4, Chapter 6 poses three hypotheses intended to

establish the capabilities of the preliminary fatigue design methodology. These hypotheses were based on literature review and observations related to the fatigue life problem in Experiments 1 and 2:

Hypothesis 3.1

The preliminary fatigue design methodology will enable quantification of the impact of blade spar thickness on probability of rotor blade fatigue failure at a specified service life.

Hypothesis 3.2

The preliminary fatigue design methodology will enable quantification of the impact of tail rotor cant on probability of rotor blade fatigue failure at a specified service life.

Hypothesis 3.3

1. The preliminary fatigue design methodology will enable quantification of the impact of average rate of climb on probability of rotor blade fatigue failure at a specified service life.
2. The preliminary fatigue design methodology will enable quantification of the impact of average cruise speed on probability of rotor blade fatigue failure at a specified service life.

Experiment 3 was designed to address Hypotheses 3.1 to 3.3 and serve as a case study to demonstrate different potential applications of the preliminary fatigue design methodology.

Experiment 3.1 studied the impact of blade spar thickness on probability of fatigue failure of the main rotor blade. This experiment supported Hypothesis 3.1 by demonstrating that the methodology could predict, with statistical significance, how the probability of fatigue failure decreases with slight increases in thickness, then increases as potentially-damaging aeroelastic effects are encountered in the high speed flight regime.

Next, Experiment 3.2 established a link between tail rotor cant angle and probability of main rotor blade fatigue failure. It was found that a positive tail rotor cant angle can

improve fatigue life if the mission spectrum used in fatigue life prediction defines a tail-heavy flight condition. This experiment supported Hypothesis 3.2 by establishing that the fatigue methodology is able to quantify these effects with statistical significance.

Finally, Experiment 3.3 attempted to predict the influence of design mission requirements on fatigue life. Although this experiment failed to support a link between rate of climb and fatigue life, leading to the rejection of Hypothesis 3.3.1, it supported Hypothesis 3.3.2. The more significant effects of airspeed on fatigue life were easier to quantify than the subtle effects of rate of climb. As a whole, the research described in Chapters 4 to 6 supported Conjecture 0.

7.2 Contributions

The preliminary fatigue design methodology provides a significant advantage over the traditional fatigue design methodologies described in Section 2.2.2 and the new approaches described in Section 2.3.

This methodology utilizes artificial neural network surrogate models to speed up the prediction of fatigue stress components derived from physics-based multidisciplinary analysis tools. Notably, this removes the need for extrapolations from historical fatigue data when designing new rotorcraft, which is a notable drawback of traditional fatigue design methods. If the physics-based models that underlie the loads and stress predictions are able to accurately predict results for revolutionary vertical lift aircraft, this methodology will be applicable to exotic configurations. Because little to no historical data exists for this type of aircraft, traditional fatigue design methodologies would be inapplicable.

Furthermore, structural reliability solution methods were proven effective at predicting fatigue life or probability of fatigue failure. This obviates the need for overly-conservative and potentially unreliable safety factors and reductions that plague traditional fatigue design methodologies. Shifting fatigue design towards probabilistic design methods provides greater confidence in fatigue life predictions which can improve flight safety, reduce weight

and over-engineering, and reduce waste due to overly-aggressive component replacement times.

Finally, the preliminary fatigue design methodology was proven to enable the use of fatigue life as a design driver for new rotorcraft programs through a series of case studies. Although the methodology as demonstrated in this thesis is *not* a complete rotorcraft design environment, it could be incorporated into a preexisting physics-based design environment, such as those described in Section 2.1.3, to provide richer results. This would require adapting this methodology to the environment's preferred performance tool, comprehensive code, and beam analysis, then incorporating the fatigue life predictions as a design objective or constraint.

This methodology is applicable to a number of specific design activities:

1. In the field of rotor blade design, engineers could make use of the methodology to run trade-off studies or optimization problems concerning rotor blade weight, dynamic characteristics, ultimate strength, and resistance to fatigue failure.
2. During the preliminary design stage, designers can conduct layout excursions trading performance, flight dynamics, handling qualities, and fatigue life of flight-critical components. Notably, the fatigue influence of minor changes in vehicle layout can be predicted with confidence.
3. During requirements definition or verification, customers such as government agencies, military procurement teams, or urban air mobility operators can assess the impact of their mission requirements on flight-critical component replacement intervals.

As described in Chapter 1, the fatigue life of certain components has a direct connection to the reliability, availability, maintainability, and cost of the fielded helicopter design.

7.3 Findings and Recommendations

This research enables more rigorous fatigue design than the heuristics and trends described in Section 1.5. Based on the fatigue analysis of the generic SMR helicopter in Experiment 3,

some general recommendations regarding rotorcraft fatigue design can be made.

First, Experiment 3.1 revealed unexpected fatigue-related effects that were not predicted prior to the experiment. Namely, certain rotor blade designs experience intense aeroelastic effects during high-speed forward flight that significantly increased the equivalent stress at the critical fatigue point. This type of high-order structural–aerodynamic effect is extremely difficult to account for using heuristics alone as it depends on the rotor blade design, including structural layout and choice of materials; the configuration of the rotor system and its controls; and the flight condition and configuration of the helicopter. Thus, when considering changes to such critical components, rotorcraft engineers should base their decisions on rigorous analysis and fatigue life predictions rather than relying on prior knowledge or experience.

Next, Experiment 3.2 highlighted the sensitivity of rotor blade fatigue life to changes in vehicle layout, relative subsystem positions and orientations, and payload configuration. This emphasizes the need to conduct fatigue design activities from a holistic viewpoint. Rotor blade design and vehicle design are on different "scales", which means they would traditionally be designed independently from one another. In reality, the two are highly intertwined, especially in the context of fatigue life prediction, which is inherently a multidisciplinary process. In fatigue design, the whole is not equal to the sum of its parts. Rotorcraft designers should ensure that their design methods are able to account for all systems on the helicopter regardless of scale or important interactions and holistic effects could be missed.

Finally, Experiment 3.3 reveals the significance that design mission and point performance requirements can have on helicopter fatigue life. Minor changes in the design cruise speed resulted in significant changes in probability of rotor blade fatigue failure. Organizations that write requirements for new rotorcraft should be aware of the impact these requirements have on the RAM-C characteristics of the final design. For example, it is desirable to achieve a high cruising speed for UAM concepts in order to offer significant time savings over terrestrial modes of transportation. However, this may come at the expense

of reduced rotor blade fatigue life, which will significantly increase operating costs and may endanger the economic feasibility of the service as a whole.

7.4 Limitations and Future Work

The preliminary fatigue design methodology developed as a part of this research is a promising proof-of-concept. However, it features a number of limitations that restrict its effectiveness which could be addressed in future research by interested parties. As limitations specific to individual experiments were already described in previous chapters, this section focuses only on limitations with the overall implementation.

First, this iteration of the methodology only considers a single critical point of failure on the component of interest. This restriction was brought about by the limitations of the scalar surrogate models used to predict mean stress and stress amplitude. Recently, techniques known as reduced-order modeling (ROM) have become popular, especially in the field of fluid-structure interaction [134, 135], which includes rotorcraft aeromechanics. ROMs allow the prediction of entire fields rather than single scalar values. The increased computational complexity associated with ROMs puts their application beyond the scope of this thesis, but they could potentially be used to predict a field of stress invariants or stress tensors on the entire rotor blade cross section, which would make the single critical fatigue point analysis obsolete.

The cyclic load analysis used throughout this thesis considered only the mean and amplitude of the periodic stress signal, effectively limiting analysis to the first harmonic of stress and neglecting any higher-frequency harmonics. If a Fourier analysis was performed on the periodic stress signal, a number of harmonics, including 1/rev, 2/rev, 3/rev, 4/rev, etc. terms, would be apparent [29]. The higher-order terms have lower magnitudes than the 1/rev component, but they account for a greater number of total fatigue cycles due to their higher frequency. This could have a significant impact on the predicted fatigue life of the rotor blade. Higher-order harmonics are also a concern for helicopter components

in the non-rotating system. In conventional designs, the main rotor passes vibratory loads at multiples of n_B/rev (where n_B is the number of blades in the rotor) through the hub to the fuselage [136]. In the case of the four-bladed generic SMR helicopter, 4/rev, 8/rev, 12/rev, etc. vibratory loads would be required to predict the fatigue life of components in the non-rotating system. These higher-order harmonics could be captured using Fourier analysis of the periodic stress waveform, or by implementing the rainflow cycle counting method described in Section 2.2.1.4.

A related limitation is that the multidisciplinary analysis environment developed in this research only considers periodic flight conditions. Transient conditions such as in-flight maneuvers, turbulence, rotor spin-up, and engine shut down, which are known to contribute significantly to fatigue damage, are not modeled [4]. To address this limitation, simulations of transient conditions could be developed in the comprehensive code. The stress histories produced during each transient condition can then be simplified using cycle counting and approximated with a separate surrogate model or reduced-order model to incorporate these conditions into the fatigue life prediction problem.

Finally, Experiment 1b revealed the potential unsuitability of the signed von Mises stress invariant when the data includes stress histories that cross zero. A signed stress invariant was necessary in this research to properly model both compression and tension loading, but the nonlinearities introduced by the signed von Mises stress reduced the accuracy of the surrogate models used to predict the stress responses. Future work should explore other invariants of the stress tensor [137] and qualify the advantages and disadvantages of each in the context of the rotorcraft fatigue design problem Alternatively, ROMs may enable prediction of the entire stress tensor, which would enable the use of multiaxial fatigue life prediction models, replacing the uniaxial S-N curve used in this research.

7.5 Future Applications

Beyond the future work required to address the limitations identified previously, this method-

ology can be expanded for applications to both existent conventional rotorcraft and conceptual revolutionary vertical lift concepts.

7.5.1 Applications to Existent Rotorcraft

The generic SMR helicopter model used in this research can serve as a starting point for modeling other helicopters of the same configuration. Because the NDARC and RCAS models are modular, it is relatively simple to swap or resize components as necessary to tune the models to match different rotorcraft. A new VABS model would need to be created to match the construction and materials of the rotor blade. In most cases, this would require a partnership with a rotorcraft manufacturer since the design details of nearly all helicopters are intellectual property.

The fidelity of the models could also be improved. For example, the NDARC rotor power consumption model can be tuned to wind tunnel test results or flight data provided by the manufacturer. Detailed analysis of rotor wake characteristics and aerodynamic interactions would benefit the accuracy of the RCAS model immensely.

Then, this model can be used for validation and verification of the preliminary fatigue design methodology, which was not possible in this thesis. Predicted rotor blade loads and stresses can be compared to results from instrumented rotor blades to validate the accuracy of the NDARC, RCAS, and VABS models. If these results show good agreement, a new probabilistic mission spectrum based on the industry partner's design missions or fleet usage data will be created. Then, the preliminary fatigue design methodology can be used to predict fatigue life at the manufacturer's desired level of reliability. These results can be compared directly to the manufacturer's own fatigue life substantiation to verify the new methodology's accuracy.

In certain cases, such as advanced composite rotor blades with infinite fatigue life, it may not be practical or desirable to apply this methodology to the rotor blade. The methodology could instead be applied to other components on the rotorcraft with a few

simple modifications. The RCAS model would need to include a finite element node at one or more locations on that component, and a new structural model would be required to predict stresses throughout the structure. If the component cannot be approximated with a one-dimensional beam, then a three-dimensional finite element model may be required. This approach could be used to substantiate the fatigue life of other flight critical components in the rotor system, like the pitch links or rotor shaft, or structural elements in the non-rotating system.

If a partnership cannot be formed with a rotorcraft manufacturer, then the methodology could be validated by analogy to wind turbines or even fixed-wing aircraft. In the wind turbine case, RCAS and VABS can still be used to provide the aeroelastic rotor model, and NDARC would not be required. Design and loads data could be obtained through partnership with NREL, who have expressed interest in using RCAS for wind turbine modeling [112]. Adapting the methodology to a fixed-wing aircraft would require more significant reworking, since the sources of fatigue damage in fixed-wing aircraft differ significantly from their rotary-wing counterparts.

7.5.2 Applications to Revolutionary Vertical Lift Concepts

After validating the preliminary fatigue design methodology by application to existent rotorcraft, it can be used for the design, development, and eventual certification of revolutionary vertical lift aircraft. This will require partnering with manufacturers to develop entirely new NDARC, RCAS, and VABS models for each vehicle, since they are unlikely to match the single main rotor configuration used in this research. Since many of the higher-fidelity aerodynamic models in RCAS require tuning to match wind tunnel or flight test data, it will be necessary to expand the comprehensive code with higher-fidelity first-principles aerodynamic models, such as vortex particle methods [16] or dual-solver hybrid methods [48].

Then, the preliminary fatigue design methodology can be integrated into the manufac-

turer's preliminary design processes to enable the use of fatigue life as a design driver or constraint. This enables the designers to make fatigue-oriented design decisions, ultimately improving the safety and RAM-C metrics of the aircraft. As the design process progresses, the models and mission spectrum can be upgraded to reflect new knowledge about the aircraft and its intended use. A sufficiently accurate model tuned to match data from wind tunnel and flight tests can also be used to support the certification process.

The procedures, models, and methodology developed in this research represent a significant down payment towards enabling physics-based fatigue life substantiation in the preliminary design stage, which is the first step in completing the "missing link" between physics-based design tools and RAM-C prediction tools. Physics-based RAM-C assessment methodologies such as that described in this thesis are essential for the design of successful, reliable, and financially-competitive rotorcraft. As transformative vertical lift aircraft take to the skies, operators and passengers alike will take comfort in the fact that these vehicles have been rigorously engineered to ensure a safe, smooth, and trouble-free flight.

Appendices

APPENDIX A
OPENMDAO PYTHON WRAPPERS

This appendix describes a series of OpenMDAO wrappers that enable control and execution of the different software elements of the multidisciplinary analysis (MDA) environment. Wrappers were created for the RCAS and VABS programs. An initial version of the RCAS wrapper was generously provided by Michael Avera of the Army Research Laboratory. The VABS wrapper was created from scratch for this research; it is designed to work only with the PreVABS pre-processor. For NDARC, the publicly available RCOTOOLS wrapper was used. Each wrapper is written in the Python programming language. The basic functionality of each wrapper is as follows:

1. Read a pre-existing input file for the program and create a representation of that information using Python data structures.
2. Implement a syntax ("access strings") that allow the user to easily map OpenMDAO variables to specific locations within the Python data structure.
3. Generate a new input file that represents the original model with changes to specific variables from OpenMDAO.
4. Execute the new input file, parse the results, and return the results to OpenMDAO.

A.1 RCAS Wrapper

The RCAS OpenMDAO wrapper is divided into two files, `RCASparse.py` and `RCASwrapper.py`. `RCASparse.py` is responsible for parsing the RCAS input and output files as well as building a new input file reflecting variable changes from OpenMDAO. This file is capable of parsing the RCAS master log and arbitrary tabular outputs generated by the user-defined output module. `RCASwrapper.py` provides an interface between OpenMDAO and `RCASparse.py`. This wrapper was tested with RCAS version 17, Python version 3.9,

and OpenMDAO version 3.8.

Listing A.1: RCASparse.py

```
"""
RCAS Input/Output File Parser for OpenMDAO v3.2

Original Author: Michael Avera - U.S. Army Research Laboratory - VTD
Current Author: Joseph Robinson - Aerospace Systems Design Laboratory
"""

from pyparsing import CaselessLiteral, Combine, ZeroOrMore, Literal, \
                    Optional, Word, alphanums, \
                    oneOf, nums, LineStart, \
                    OneOrMore, srange
from numpy import array, nan, append
from openmdao.utils.file_wrap import ToFloat, ToInteger, FileParser
from collections import OrderedDict

class RCASlist(object):
    """Utility to ease the task of constructing a formatted RCAS input file."""

    def __init__(self, comp):
        # self.rcasfile = None
        self.rcas_dict = None
        self.title = ""
        self.lines = []
        self.comp = comp

    def parse_input(self,file):
        """
        Parses an existing RCAS script file and creates a dict to hold the
        data.

        Parameters
        ----------
        file : string
            Path to the original RCAS input deck.

        Returns
        -------
        None.

        """

        self.lines = []
        currentscreen = ""
        currentpage = 0
        linenum = 0

        infile = open(file, 'r')
        data = infile.readlines()
        infile.close()

        # tokens
        script_begin_token = LineStart()+Literal("***begin-RCAS-file: scriptfile ***")
        script_end_token =   LineStart()+Literal("*****end-RCAS-file: scriptfile ***")
        screen_token =       LineStart()+CaselessLiteral("S ")
```

```
        page_token =         LineStart()+CaselessLiteral("N")
        data_token =         LineStart()+CaselessLiteral("a␣")
        data2_token =        LineStart()+CaselessLiteral("c␣")

        comment_token =      Literal("!")
        menu_token =         LineStart()+CaselessLiteral("M␣")

        # types of pieces of data
        digits = Word(nums)
        dot = "."
        sign = oneOf("+␣-")
        nan = Literal("--")
        ee = CaselessLiteral('E') | CaselessLiteral('D')
        # num_int = ToInteger(Combine( Optional(sign) + Optional(digits) + digits )) #RCAS can use 2 digit int
        num_int = ToInteger(Combine( Optional(sign) + digits )) #JNR debug
        num_float = ToFloat(Combine( Optional(sign) +
                                    ((digits + dot + Optional(digits)) |
                                    (dot + digits)) +
                                    Optional(ee + Optional(sign) + digits)))
        textvalue = Word(alphanums+"."+"_"+"&"+"("+")")
        commands = Word(srange("[a-zA-Z]")) + ZeroOrMore(Word(srange("[a-zA-Z]")))
        mixed_exp = ToFloat(Combine( Word(nums) + Optional(dot) + Optional(Word(nums) ) + ee + Optional(sign) + digits ))
        vector = Combine(digits+Literal(":")+digits)
        numval = nan | vector | mixed_exp | num_float | num_int

        # types of different lines of data
        screenline = (screen_token.setResultsName("token") + \
                Word(srange("[A-Z]")).setResultsName("name"))
        menuline = (menu_token.setResultsName("token") + \
                Word(srange("[A-Z]")).setResultsName("name"))
        dataline   = (data_token.setResultsName("token") + \
                OneOrMore(numval | textvalue).setResultsName("data"))
        dataline2   = (data2_token.setResultsName("token") + \
                OneOrMore(numval | textvalue).setResultsName("data"))

        # initialize variables for the loop
        self.header=[]
        self.footer=[]
        scriptflag = False
        record_name_flag = False
        rcas_dict = OrderedDict()
        screen_dict = self.generate_screen_dict()
        current_name = None
        current_naming_screen = None
        cmd_ctr = 0
        menu_ctr = 0

        # loop through the input file
        for line in data:
            base_line = line
            line = line.strip()

            # empty lines
            if not line:
                continue

            # comment lines
            if comment_token.searchString(line):  #removes comment but doesnt continue parsing
```

```
113                 if scriptflag:
114                     line = line.split("!",1)[0]
115                     if not line:
116                         continue
117 
118             # no further parsing if script header hasn't been found yet
119             if script_begin_token.searchString(line):
120                 scriptflag = True
121                 self.header.append(base_line)
122                 continue
123             elif not scriptflag:
124                 self.footer.append(base_line)
125                 continue
126 
127             elif screenline.searchString(line):            #Lines starting with "S"
128                 screen = screenline.parseString(line)
129                 currentscreen = screen.name
130                 currentpage = 0
131                 linenum = 0
132                 rcas_dict[currentscreen] = {
133                     "data": {currentpage : { linenum : {}}},
134                     "type": "screen",
135                     }
136 
137                 if currentscreen.upper() in screen_dict:
138                     # this screen will lend its data to its non-unique screens
139                     record_name_flag = True
140                     current_naming_screen = currentscreen
141 
142                 if current_naming_screen and (currentscreen.upper() in screen_dict[
                        ↪ current_naming_screen.upper()]):
143                     # this screen is non-unique
144                     new_current_screen_key = current_name + "-" + currentscreen
145                     rcas_dict[new_current_screen_key] = rcas_dict.pop(currentscreen)
146                     currentscreen = new_current_screen_key
147 
148             elif page_token.searchString(line):            #Lines starting with "N"
149                 currentpage = currentpage + 1
150                 linenum = 0
151                 rcas_dict[currentscreen]["data"][currentpage] = { linenum : {}}
152 
153             elif dataline.searchString(line):            #Lines starting with "a"
154                 values = dataline.parseString(line)
155                 rcas_dict[currentscreen]["data"][currentpage][linenum] = values.data.
                        ↪ asList()
156                 linenum = linenum + 1
157 
158                 if record_name_flag:
159                     # looking for the name of current_naming_screen
160                     current_name = values.data.asList()[0].upper()
161                     new_naming_screen_key = current_name + "-" +
                            ↪ current_naming_screen
162                     rcas_dict[new_naming_screen_key] = rcas_dict.pop(
                            ↪ current_naming_screen)
163                     record_name_flag = False
164 
165             elif dataline2.searchString(line):            #Lines starting with "c"
166                 values = dataline2.parseString(line)
167                 linenum = linenum - 1
168                 rcas_dict[currentscreen]["data"][currentpage][linenum] = values.data.
                        ↪ asList()
```

```
                linenum = linenum + 1

            elif script_end_token.searchString(line):
                scriptflag = False
                self.footer.append(base_line)
                self.footer.append("\n")

            elif menuline.searchString(line):              #Lines starting with "M"
                menu = menuline.parseString(line)
                currentmenu = menu.name
                currentmenu_key = str(menu_ctr) + "-" + currentmenu
                rcas_dict[currentmenu_key] = {
                    "data": {},
                    "type": "menu",
                    }
                menu_ctr = menu_ctr + 1

            # anything left over should just be commands like EXIT, COPYAEROCOMP, etc
            else:
                command = commands.parseString(line)
                currentcommand = " ".join(command)
                currentcommand_key = str(cmd_ctr) + "-" + currentcommand
                rcas_dict[currentcommand_key] = {
                    "data": {},
                    "type": "command",
                    }
                cmd_ctr = cmd_ctr + 1

        self.rcas_dict = rcas_dict

    def parse_rcasmasterlog(self,log_file):
        """
        Parse the rcasmaster001.log file.

        Parameters
        ----------
        log_file : string
            Path to RCAS log file.

        Returns
        -------
        dataout : dict
            Dictionary of output data from log file.

        """

        # f = open(self.rcasfile+str(num)+'/'+"rcasmaster001.log", 'r')
        f = open(log_file, 'r')
        log = f.readlines()
        f.close()

        PWRTSH_token = Literal("Shaft Total Power")
        PWRISH_token = Literal("Shaft Induced Power")
        PWRPSH_token = Literal("Shaft Profile Power")
        sigma_token  = Literal("Solidity")
        radius_token = Literal("Rotor Radius")
        coll_token   = Literal("Swashplate Coll")
        ct_token     = Literal("Thrust Coeff")
        cp_token     = Literal("Power Coeff")
        aoa_token    = Literal("TPP Ang. of Attack")
```

```
        conv_token   = Literal("Trim solution has been reached")
        nonconv_token= Literal("Trim did not converge")
        nonconv2_token=Literal("No trim data available")
        nonconv3_token=Literal("Floating point exception: Invalid operation")
        wind_token    =Literal("Wind X")
        density_token =Literal('Air Density')
        vtip_token    =Literal('Blade Tip Speed')

        dataout={'PWRTSH':array([]), 'PWRISH':array([]), 'PWRPSH':array([]), 'sigma':array([]), 'radius':array([]),
            'conv':False, 'ct':array([]), 'cp':array([]), 'aoa':array([]), 'wind':array([]), 'density':array([]), 'vtip':array([]) }
        skip   = False

        for line in log:
            line = line.strip()
            if (nonconv_token.searchString(line) or nonconv2_token.searchString(line)
                or nonconv3_token.searchString(line)):
                skip=True
                dataout['conv'] = False
                for k,v in dataout.items():
                    if k != "conv":
                        dataout[k]=append(dataout[k],nan)
            elif conv_token.searchString(line):
                dataout['conv']=True
                skip=False

            if skip==False:
                if PWRTSH_token.searchString(line):
                    dataout['PWRTSH']=append(dataout['PWRTSH'],float(line.split("(HP)=")[1]))
                if PWRISH_token.searchString(line):
                    dataout['PWRISH']= append(dataout['PWRISH'],float(line.split("(HP)=")[1]))
                if PWRPSH_token.searchString(line):
                    dataout['PWRPSH']=append(dataout['PWRPSH'],float(line.split("(HP)=")[1]))
                if sigma_token.searchString(line):
                    dataout['sigma']= append(dataout['sigma'],float(line.split()[2]))
                if radius_token.searchString(line):
                    dataout['radius']=append(dataout['radius'],float(line.split()[3]))
                if ct_token.searchString(line):
                    dataout['ct']=append(dataout['ct'],float(line.split("=")[1].split()[0]))
                if cp_token.searchString(line):
                    dataout['cp']= append(dataout['cp'],float(line.split("=")[1].split()[0]))
                if aoa_token.searchString(line):
                    dataout['aoa']=append(dataout['aoa'],float(line.split("=")[1].split()[0]))
                if wind_token.searchString(line):
                    dataout['wind']=append(dataout['wind'],float(line.split("=")[2].split()[0]))
                if density_token.searchString(line):
                    dataout['density']=append(dataout['density'],float(line.split("=")[1].split()[0]))
                if vtip_token.searchString(line):
                    dataout['vtip']=append(dataout['vtip'],float(line.split("=")[1].split()[0]))

```

```
        return dataout

    def parse_rcastable(self,tab_file,cols):
        """
        Parse a RCAS output table file.

        Parameters
        ----------
        tab_file : string
            Path of the table file to be parsed.
        cols : string
            Range of columns to include.
            Example: "2" will include only the second column.
                     "2:11" will include the second through eleventh columns.

        Returns
        -------
        data_out : numpy.array
            Array containing the requested data.

        """

        # parse column choice
        if ":" in cols:
            colstart = int(cols.split(":")[0])
            colend = int(cols.split(":")[1])
        else:
            colstart = int(cols.split(":")[0])
            colend = colstart

        # open file and set marker
        parser = FileParser()
        parser.set_file(tab_file)
        parser.mark_anchor("!M␣YY1")

        # find number of rows in table
        num_lines = len(open(tab_file, 'r').readlines())
        rows = num_lines - (parser._current_row+1)

        data_out = parser.transfer_2Darray(1, colstart, rows, colend)

        return data_out

    def generate(self,output_filename):
        """
        Writes RCAS input file from self.rcas_dict

        Parameters
        ----------
        output_filename : string
            Path to the file to be written.

        Returns
        -------
        None.

        """

        currentkey = ""
        currentpage = 0
```

```
        self.output=[]
        for line in self.header:
            self.output.append(line)

        for key,value in self.rcas_dict.items():
            pages = value["data"]
            if '-' in key:
                trimmed_key = key.split('-')[1]
            else:
                trimmed_key = key

            if value["type"] == "command":
                self.output.append("\n")
                self.output.append("%s\n" % trimmed_key)
                continue

            if key != currentkey:
                self.output.append("\n")
                if value["type"] == "screen":
                    self.output.append("S␣%s\n" % trimmed_key)
                elif value["type"] == "menu":
                    self.output.append("M␣%s\n" % trimmed_key)

                currentkey = key
                currentpage = 0

            for page,lines in pages.items():
                if page != currentpage:
                    while page !=currentpage:
                        self.output.append("N\n")
                        currentpage = currentpage + 1

                for line,data in lines.items():
                    if data:
                        self.output.append("a␣%s\n" % '␣'.join(map(str,data)))

        self.output.append("\n")

        for line in self.footer:
            self.output.append(line)

        outfile = open(output_filename, 'w')
        outfile.writelines(self.output)
        outfile.close()

    def generate_screen_dict(self):
        """
        A function to generate a dict of non-unique RCAS screens

        Returns
        -------
        screen_dict : dict
            A dictionary of the form:
                <NAMING_SCREEN_1>: [
                    <NAMED_SCREEN_1>,
                    <NAMED_SCREEN_2>,
                    <etc.>,
                    ],
                <NAMING_SCREEN_2>: [
                    <etc.>,
```

```
                    ],
                <etc.>

        """
        screen_dict = {
            "SELSUBSYS": [
                "SUBSYSTYP",
                "SUBSYSCOMP",
                "PSORIGIN",
                "PSORIENT",
                "CONNCONST",
                "CORNODE",
                "BLADECOMP",
                "ROTORPARAM",
                "MBC",
                "SINGLEBLADE"
                ],
            "PRIMITIVEID": [
                "ELDATASETID",
                "FENODE",
                "RIGIDBODYMASS",
                "RIGIDBAR",
                "HINGE",
                "PSMODALDAMP",
                "SLIDE",
                "SPRELE",
                "NLBEAMDEF",
                "GCBEAMDEF",
                "CONTROLCONNECT",
                ],
            "AEROSUPCOMPID": [
                "SUPCMPTYP",
                "COMPID",
                "CPORIGIN",
                "CPORIENT",
                "INFLOW",
                "DYNINFDATA",
                "AEROPTION",
                "THRUSTAVE",
                "SUPCMPTOSS",
                "TIPLOSS"
                ],
            "AEROCOMPID": [
                "COMPTYPE",
                "AERONODE",
                "AEROSEG",
                "BODYAEROTAB",
                ],
        }

        return screen_dict
```

Listing A.2: RCASwrapper.py

```
"""
RCAS Wrapper for OpenMDAO v3.2

Original Author: Michael Avera - U.S. Army Research Laboratory - VTD
Current Author: Joseph Robinson - Aerospace Systems Design Laboratory
"""
```

```python

from openmdao.api import ExternalCodeComp
import os, shutil
from modules.RCASparse import RCASlist
from modules.SupportingFunctions import add_runid

class RCASwrapper(ExternalCodeComp):
    """A file wrapper for RCAS."""

    def __init__(self,rcaspath,inputpath,rcasfile,inputs_list,outputs_list,
                 save_parsed_original=False,force_linear=False,
                 parallel=False,saveallruns=False,logger=None):
        # execute ExternalCodeComp.__init__()
        super(RCASwrapper,self).__init__()

        # assign fields
        self.rcaspath = rcaspath
        self.inputpath = inputpath
        self.rcasfile = rcasfile
        self.inputs_list = inputs_list
        self.outputs_list = outputs_list
        self.inputs_dict = {}
        self.outputs_dict ={}
        self.save_parsed_original = save_parsed_original
        self.force_linear = force_linear
        self.parallel = parallel
        self.saveallruns = saveallruns
        self.logger = logger

    def setup(self):
        """
        Setup method.

        Returns
        -------
        None.

        """

        # File I/O variables
        # These should be set by run script before executing setup()
        if not self.inputpath:
            print("[RCASwrapper]:INPUT_FILEPATH_NOT_SET!_Did_you_forget_to_set_rcas.inputpath?")
        if not self.rcaspath:
            print("[RCASwrapper]:RCAS_FILEPATH_NOT_SET!_Did_you_forget_to_set_rcas.rcaspath?")
        if not self.rcasfile:
            print("[RCASwrapper]:RCAS_INPUT_FILE_NOT_SET!_Did_you_forget_to_set_rcas.rcasfile?")

        # External Code public variables
        self.orig_input_file = self.rcasfile + ".rcas"
        self.script = RCASlist(self)
        self.rcaspath = shutil.copy(src=self.rcaspath,dst=os.getcwd())

        # Parse the input file
        self.script.parse_input(os.path.join(self.inputpath,self.orig_input_file))

        if self.save_parsed_original:
            self.script.generate("Original_Parsed_RCAS_job.rcas")
```

```python

        # Prepare inputs and outputs
        for item in self.inputs_list:
            self.add_input(item[0],**item[2])
            self.inputs_dict[item[0]] = item[1]
        for item in self.outputs_list:
            om_var = item[0] # variable name in openMDAO
            rcas_var = item[1] # variable name in RCAS wrapper
            opt_dic = item[2] # dictionary of options

            discrete = opt_dic.pop("discrete",None) #True, False, or None; remove discrete key
            if not discrete:
                self.add_output(om_var,**opt_dic)
            else:
                self.add_discrete_output(om_var,**opt_dic)
            self.outputs_dict[om_var] = rcas_var

    def compute(self, inputs, outputs, discrete_inputs, discrete_outputs):
        """
        Generates new RCAS deck, executes RCAS, and parses outputs.

        Parameters
        ----------
        inputs : dict
            Dictionary of inputs in the RCAS subsystem.
        outputs : dict
            Dictionary of outputs in the RCAS subsystem.

        Returns
        -------
        None.

        """

        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            # New job name with random ID added
            if self.parallel:
                self.new_rcasfile = add_runid(self.rcasfile,length=4)
            else:
                self.new_rcasfile = self.rcasfile

            # Set new filenames and copy original files
            self.input_file = self.new_rcasfile + ".rcas"
            # copyfile(os.path.join(self.inputpath,self.orig_input_file),self.input_file)
            self.output_file = os.path.join(self.new_rcasfile+'00',"rcasmaster001.log")

            self.options['external_input_files'] = [self.input_file,]
            self.options['external_output_files'] = [self.output_file,]

            # Prepare the command
            self.options['command'] = [self.rcaspath,self.input_file]

            # parse inputs
            for key, value in self.inputs_dict.items():
```

```
                if "." not in value: # standard input
                    input_value = inputs[key][0]

                    screen = value.split("%")[0]
                    page =   int(value.split("%")[1]) - 1 # convert to indexing from 0
                    line =   int(value.split("%")[2]) - 1 # convert to indexing from 0
                    index =  int(value.split("%")[3]) - 1 # convert to indexing from 0

                    # replace data in rcas_dict
                    oldval = self.script.rcas_dict[screen]["data"][page][line][index]
                    if oldval != input_value:
                        if input_value.is_integer():
                            input_value=int(input_value)
                        self.script.rcas_dict[screen]["data"][page][line][index] = input_value

                else: # changing a value in the blade elastic properties in the footer
                    input_value = inputs[key]

                    file = value.split("%")[0]
                    entry = value.split("%")[1]

                    # Find location of file in self.script.footer
                    file_location = [i for i in range(len(self.script.footer)) if file in self.script.footer[i]]
                    file_slice = slice(file_location[0]+1,file_location[1])

                    # Find location of entry in file
                    entry_location = [i for i in range(len(self.script.footer[file_slice])) if entry in self.script.footer[file_slice][i]]
                    if not entry_location:
                        raise Exception('Entry %s could not be found in %s' % (entry, file))
                    elif len(entry_location) > 1:
                        raise Exception('Entry %s found multiple times in %s' % (entry,file))
                    else:
                        entry_location = file_location[0] + entry_location[0] + 1

                    # Overwrite existing data with new data
                    for i in range(len(input_value)):
                        data = input_value[i,:]
                        new_line = ''
                        for j in range(len(data)):
                            new_line = new_line + '%0.8e   ' % data[j]
                        new_line = new_line + '\n'
                        self.script.footer[entry_location + i + 1] = new_line

            # generate new RCAS input deck
            self.script.generate(self.input_file)

            # run rcas and parse and load outputs
            print("RCASwrapper: executing %s" % self.input_file)
            super(RCASwrapper, self).compute(inputs, outputs)

```

```
            # parse outputs
            outs = []
            for key, value in self.outputs_dict.items():
                outs.append(value)
            log_outs = [item for item in outs if ("log" in item)]
            tab_outs = [item for item in outs if ("tab" in item)]

            # read log file outputs
            if log_outs:
                logdata = self.script.parse_rcasmasterlog(self.output_file)
                for item in log_outs:
                    key = item.split('-')[1]
                    if key == "conv":
                        discrete_outputs[self.get_key(self.outputs_dict,item)] = logdata[key]
                    else:
                        outputs[self.get_key(self.outputs_dict,item)] = logdata[key]

            # read tab file outputs
            if tab_outs:
                for item in tab_outs:
                    tab_file = item.split('-')[1].split('%')[0]
                    tab_file = os.path.join(os.path.dirname(self.output_file), tab_file + '.tab')
                    cols = item.split('-')[1].split('%')[1]
                    outputs[self.get_key(self.outputs_dict,item)] = self.script.parse_rcastable(tab_file, cols)

            # Remove run files
            if not self.saveallruns:
                os.remove(self.input_file)
                shutil.rmtree(self.new_rcasfile + '00') # folder output
                # don't remove .csh file because another process may be using it


    def get_key(self,my_dict,val):
        for key, value in my_dict.items():
             if val == value:
                 return key
```

A.2 PreVABS+VABS Wrapper

The PreVABS+VABS OpenMDAO wrapper is divided into two files, `PreVabsParse.py` and `PreVabsWrapper.py`. `PreVabsParse.py` is responsible for parsing and generating the XML-formatted input files for PreVABS into a Python dictionary using the `xmltodict` package. This module also handles parsing geometric, inertial, elastic, and stress recovery information from the VABS output files. `PreVabsWrapper.py` includes two classes that provide an interface between OpenMDAO and `PreVabsParse.py`. The `PreVabsWrapper` class is used to execute VABS in homogenization mode, and `PreVabsRecover` is used to

execute VABS in stress recovery mode. This wrapper was tested with PreVABS version 1.2, VABS version 3.8, Python version 3.9, and OpenMDAO version 3.8.

Listing A.3: PreVabsParse.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
PreVABS parser for thesis work. Will integrate with PreVABS wrapper.

Created on Thu Sep 17 15:52:04 2020

@author: josephrobinson
"""

import os
import xmltodict
from openmdao.utils.file_wrap import FileParser
from modules.SupportingFunctions import has_handle
import numpy as np
import time, random

class PreVabsParse(object):
    """ Class for parsing PreVABS documents """

    def __init__(self,input_dir,cross_section_file,output_dir):
        self.input_dir = input_dir
        self.output_dir = output_dir
        self.cross_section_file = cross_section_file

        self.baselines_file = None
        self.basepoints_file = None
        self.layups_file = None
        self.materials_file = None

        self.prevabs_dict = None # Original PreVABS input dictionary
        self.cross_section = None # reference to cross section location in self.prevabs_dict
        self.baselines = None # reference to baselines location in self.prevabs_dict
        self.layups = None # reference to layups location in self.prevabs_dict
        self.materials = None # reference to materials location in self.prevabs_dict
        self.basepoints = None # reference to basepoints location in self.prevabs_dict
        self.combined = False # True if PreVABS 1.1 combined format is being used

        self.M = np.empty((6,6)) # mass matrix
        self.S = np.empty((6,6)) # stiffness matrix
        self.num_nodes = 0
        self.num_elems = 0

    def parse_input(self):
        """
        Parses the input file specified in self.cross_section file and all
        associated files in the <include> tag.

        Returns
        -------
        None.

```

```
        """
        # Write cross section file
        cross_section_file = os.path.join(self.input_dir,self.cross_section_file+".xml")

        self.prevabs_dict = {}
        self.prevabs_dict["cross_section"] = self.read_xml(cross_section_file)["cross_section"]
        self.cross_section = self.prevabs_dict["cross_section"]

        # Check if the PreVABS 1.1 combined format is being used
        if "@format" in self.cross_section:
            if self.cross_section["@format"] == '1':
                self.combined = True
            else:
                self.combined = False
        else:
            self.combined = False

        # Read in supporting files if any exist
        if "include" in self.prevabs_dict["cross_section"]:
            include = self.prevabs_dict["cross_section"]["include"]
            if include: # make sure include isn't empty
                if "baseline" in include:
                    self.baselines_file = include["baseline"][0]
                    baselines_file = os.path.join(self.input_dir,self.baselines_file+".xml")
                    self.prevabs_dict["baselines"] = self.read_xml(baselines_file)["baselines"]
                    self.baselines = self.prevabs_dict["baselines"]
                if "layup" in include:
                    self.layups_file = include["layup"][0]
                    layups_file = os.path.join(self.input_dir,self.layups_file+".xml")
                    self.prevabs_dict["layups"] = self.read_xml(layups_file)["layups"]
                    self.layups = self.prevabs_dict["layups"]
                if "material" in include:
                    self.materials_file = include["material"][0]
                    materials_file = os.path.join(self.input_dir,self.materials_file+".xml")
                    self.prevabs_dict["materials"] = self.read_xml(materials_file)["materials"]
                    self.materials = self.prevabs_dict["materials"]

        # Assign sub-dicts of cross_section dict if combined format is being used
        if self.combined:
            if "baselines" in self.cross_section:
                self.baselines = self.cross_section["baselines"]
            if "layups" in self.cross_section:
                self.layups = self.cross_section["layups"]

        # Read basepoints if it exists
        if "include" in self.baselines["basepoints"]:
            self.basepoints_file = self.baselines["basepoints"]["include"]
            basepoints_file = os.path.join(self.input_dir,self.basepoints_file+".dat")

            self.prevabs_dict["basepoints"] = {}
            self.basepoints = self.prevabs_dict["basepoints"]
            names = []
```

```
        coords = []
        while has_handle(basepoints_file):
            time.sleep(random.random()) # check if file is already being read by another process
        with open(basepoints_file, 'r') as fd:
            for line in fd:
                item = line.rstrip().split() # strip off newline and any other trailing whitespace
                if item:
                    names.append(item[0])
                    coords.append([float(item[1]),float(item[2])])
        self.basepoints["names"] = names
        self.basepoints["coords"] = coords

    def parse_output(self,filename):
        """
        Parse M and S matrices from the .sg.K VABS output file.

        Returns
        -------
        None.

        """
        output_file = os.path.join(self.output_dir,filename+".sg.K")

        # open file and set marker
        parser = FileParser()
        parser.set_file(output_file)

        # read mass matrix
        parser.mark_anchor("The 6X6 Mass Matrix")
        self.M = parser.transfer_2Darray(3,1,8,6)

        # read stiffness matrix
        parser.mark_anchor(" Timoshenko Stiffness Matrix (1-extension; 2,3-shear, 4-twist; 5,6-bending)")
        self.S = parser.transfer_2Darray(3,1,8,6)

    def parse_recovery(self,filename,ext,cols,timeout=np.inf):
        """
        Parse arbitrary columns from any recover file (.E, .S, .EM, .SM, etc.)

        Parameters
        ----------
        ext : str
            Extension of the file to parse ("E", "S", "EM", etc.).
        cols : str
            Specification of the columns to extract.

        Returns
        -------
        np.array
            Array of data corresponding to requested file and columns.

        """
        recovery_file = os.path.join(self.output_dir,filename)+".sg."+ext

        # parse column choice
        if ":" in cols:
            colstart = int(cols.split(":")[0])
            colend = int(cols.split(":")[1])
```

```
        else:
            colstart = int(cols.split(":")[0])
            colend = colstart

        col_slice = slice(colstart-1,colend)

        # read data in file

        total_wait = 0
        wait_time = 0.5
        while not os.path.exists(recovery_file) and total_wait < timeout:
            time.sleep(wait_time)
            total_wait += wait_time

        if os.path.isfile(recovery_file):
            data = np.loadtxt(recovery_file)
        else:
            raise ValueError('%s could not be found' % recovery_file)

        return data[:,col_slice]

    def parse_vabs_file(self,filename):
        """
        Parse some mesh information from the VABS .sg input file.

        Returns
        -------
        None.

        """
        vabs_file = os.path.join(self.output_dir,filename+".sg")

        # Custom parser since there may or may not be newlines between rows
        line_idx = 0
        with open(vabs_file, 'r') as fd:
            for line in fd:
                item = line.rstrip().split() # strip off newline and any other trailing whitespace
                if item and line_idx == 3:
                    self.num_nodes = int(item[0])
                    self.num_elems = int(item[1])
                    break
                if item:
                    line_idx = line_idx+1

    def process_dict(self):
        """
        Processes the original PreVABS dictionary by converting specific entries
        to lists of floats or ints.

        Raises
        ------
        Exception
            Errors if the layup is specified as a stack sequence instead of an
            explicit list.

        Returns
        -------
        None.

```

```
        """
        # process baselines dict
        if self.baselines:
            # convert coordinates to float list
            if "point" in self.baselines["basepoints"]:
                for item in self.baselines["basepoints"]["point"]:
                    self.quick_convert(item,"#text","coords")

            # convert angles and radii to floats
            for item in self.baselines["baseline"]:
                if "angle" in item:
                    self.quick_convert(item,"angle","angle")
                if "radius" in item:
                    self.quick_convert(item,"radius","radius")

        # process materials dict
        if self.materials:
            for item in self.materials["material"]:
                # convert density to float
                if "density" in item:
                    self.quick_convert(item,"density","density")

                # convert elastic properties to float
                if "elastic" in item:
                    for prop in item["elastic"].keys():
                        self.quick_convert(item["elastic"],prop,prop)

            for item in self.materials["lamina"]:
                self.quick_convert(item,"thickness","thickness")

        # process layups dict
        if self.layups:
            for layup in self.layups["layup"]:
                if "@method" in layup and layup["@method"] != "explicit_list":
                    raise Exception("Only_explicit_list_layup_methods_are_supported_
                        ↪ by_the_parser")

                for layer in layup["layer"]:
                    if "#text" in layer:
                        text = layer["#text"]
                        layer.pop("#text")
                    else:
                        text = ""

                    if ":" in text: # angle:stack
                        text = text.split(":")
                        layer["angle"] = [float(text[0])]
                        layer["stack"] = [int(text[1])]
                    elif text: # angle:1
                        layer["angle"] = [float(text)]
                        layer["stack"] = [1]
                    else: # 0:1
                        layer["angle"] = [0.0]
                        layer["stack"] = [1]

        # process cross_section dict
        if self.cross_section:
            # convert translate, scale, and mesh_size to float
            if "general" in self.cross_section:
                if "translate" in self.cross_section["general"]:
```

```
                    self.quick_convert(self.cross_section["general"],"translate","translate")
                if "scale" in self.cross_section["general"]:
                    self.quick_convert(self.cross_section["general"],"scale","scale")
                if "mesh_size" in self.cross_section["general"]:
                    self.quick_convert(self.cross_section["general"],"mesh_size","mesh_size")

            # convert recovery data
            if "recover" in self.cross_section:
                for field in ["displacements","rotations","forces","moments"]:
                    self.quick_convert(self.cross_section["recover"],field,field)
                for field in self.cross_section["recover"]["distributed"].keys():
                    self.quick_convert(self.cross_section["recover"]["distributed"],field,field)

    def unprocess_dict(self):
        """
        "Unprocesses" the processed PreVABS dictionary by converting specific
        values back into strings.

        Returns
        -------
        None.

        """
        # unprocess baselines dict
        if self.baselines:
            # revert coordinates
            if "point" in self.baselines["basepoints"]:
                for item in self.baselines["basepoints"]["point"]:
                    self.quick_revert(item,"coords","#text")

            # revert angles and radii
            for item in self.baselines["baseline"]:
                if "angle" in item:
                    self.quick_revert(item,"angle","angle")
                if "radius" in item:
                    self.quick_revert(item,"radius","radius")

        # unprocess materials dict
        if self.materials:
            for item in self.materials["material"]:
                # revert density
                if "density" in item:
                    self.quick_revert(item,"density","density")

                # revert elastic properties
                if "elastic" in item:
                    for prop in item["elastic"].keys():
                        self.quick_revert(item["elastic"],prop,prop)

            for item in self.materials["lamina"]:
                self.quick_revert(item,"thickness","thickness")

        # unprocess layups dict
        if self.layups:
            for layup in self.layups["layup"]:
                for layer in layup["layer"]:
                    text = str(layer["angle"][0]) + ":" + str(layer["stack"][0])
                    layer["#text"] = text
```

```
                    layer.pop("angle")
                    layer.pop("stack")

        # unprocess cross_section dict
        if self.cross_section:
            # revert translate, scale, and mesh_size
            if "general" in self.cross_section:
                if "translate" in self.cross_section["general"]:
                    self.quick_revert(self.cross_section["general"],"translate","translate")
                if "scale" in self.cross_section["general"]:
                    self.quick_revert(self.cross_section["general"],"scale","scale")
                if "mesh_size" in self.cross_section["general"]:
                    self.quick_revert(self.cross_section["general"],"mesh_size","mesh_size")

            # revert recovery data
            if "recover" in self.cross_section:
                for field in ["displacements","rotations","forces","moments"]:
                    self.quick_revert(self.cross_section["recover"],field,field)
                for field in self.cross_section["recover"]["distributed"].keys():
                    self.quick_revert(self.cross_section["recover"]["distributed"],field,field)

    def generate(self,filename):
        """
        Writes the updated "unprocessed" PreVABS dict to XML files. Also writes
        all associated files in the <include> tag.

        Parameters
        ----------
        filename : str
            Name of cross section file.

        Returns
        -------
        None.

        """
        # Generate cross section file
        cross_section_file = os.path.join(self.output_dir,filename + ".xml")
        cross_section_dict = {}
        cross_section_dict["cross_section"] = self.prevabs_dict["cross_section"]
        self.write_xml(cross_section_dict,cross_section_file)

        # Generate baselines file
        if self.baselines_file:
            baselines_file = os.path.join(self.output_dir,self.baselines_file + ".xml")
            baselines_dict = {}
            baselines_dict["baselines"] = self.prevabs_dict["baselines"]
            self.write_xml(baselines_dict,baselines_file)

        # Generate layups file
        if self.layups_file:
            layups_file = os.path.join(self.output_dir,self.layups_file + ".xml")
            layups_dict = {}
            layups_dict["layups"] = self.prevabs_dict["layups"]
            self.write_xml(layups_dict,layups_file)

        # Generate materials file
```

```
        if self.materials_file:
            materials_file = os.path.join(self.output_dir,self.materials_file + ".xml")
            materials_dict = {}
            materials_dict["materials"] = self.prevabs_dict["materials"]
            self.write_xml(materials_dict,materials_file)

        # Generate basepoints file
        if self.basepoints_file:
            basepoints_file = os.path.join(self.output_dir,self.basepoints_file + ".dat")

            if not os.path.exists(os.path.dirname(basepoints_file)):
                os.makedirs(os.path.dirname(basepoints_file))

            basepoints = self.prevabs_dict["basepoints"]
            output = []
            for i in range(len(basepoints["names"])):
                line = [basepoints["names"][i], basepoints["coords"][i][0], basepoints["coords"][i][1]]
                output.append('%s\n' % '\t'.join(map(str,line)))

            with open(basepoints_file,'w') as fd:
                fd.writelines(output)

    def read_xml(self,input_file):
        """
        Uses xmltodict to read a PreVABS XML file as a dictionary.

        Parameters
        ----------
        input_file : str
            Path to input file.

        Returns
        -------
        doc : dict
            Dictionary output.

        """

        while has_handle(input_file):
            time.sleep(random.random()) # check if file is already being read by another process

        with open(input_file,'r') as fd:
            doc = xmltodict.parse(fd.read(),
                                  force_list=('layup',
                                              'layer',
                                              'component',
                                              'segment',
                                              'baseline',
                                              'material',
                                              'point'))

        return doc

    def write_xml(self,output_dict,output_file):
        """
        Uses xmltodic to write a PreVABS XML file from a dictionary.

```

```
        Parameters
        ----------
        output_dict : dict
            Dictionary to be written.
        output_file : str
            Path to output file.

        Returns
        -------
        None.

        """
        if not os.path.exists(os.path.dirname(output_file)):
            os.makedirs(os.path.dirname(output_file))

        while has_handle(output_file):
            time.sleep(random.random()) # check if file is already being read by another process

        with open(output_file,'w') as fd:
            xmltodict.unparse(output_dict,output=fd,pretty=True)

    def quick_convert(self,item,old_key,new_key):
        """
        Quickly converts an entry to a list of floats.

        Parameters
        ----------
        item : dict
            Bottom-level entry to be converted.
        old_key : str
            Name of the key in the original dictionary.
        new_key : str
            New name for the key.

        Returns
        -------
        None.

        """
        text = item[old_key].split()
        converted = [float(i) for i in text]
        if old_key != new_key:
            item.pop(old_key)
        item[new_key] = converted

    def quick_revert(self,item,old_key,new_key):
        """
        Quickly reverts an entry containing a list of floats to a string.

        Parameters
        ----------
        item : dict
            Bottom-level entry to be converted.
        old_key : str
            Name of the key in the original dictionary.
        new_key : str
            New name for the key.

        Returns
```

```
        -------
        None.

        """
        val = item[old_key]
        string = '␣'.join(map(str,val))
        if old_key != new_key:
            item.pop(old_key)
        item[new_key] = string
```

Listing A.4: PreVabsWrapper.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 23 14:56:44 2020

@author: josephrobinson
"""

from openmdao.api import ExternalCodeComp
import os
from modules.PreVabsParse import PreVabsParse
from modules.SupportingFunctions import add_runid
from collections import OrderedDict
from shutil import copyfile

class PreVabsWrapper(ExternalCodeComp):
    """ A file wrapper for PreVABS focused on calculating elastic properties """

    def __init__(self,input_path,prevabs_file,supporting_files,
                 inputs_list,outputs_list,
                 force_linear=False,parallel=False,saveallruns=False,logger=None):
        # execute ExternalCodeComp.__init__()
        super(PreVabsWrapper,self).__init__()

        # assign fields
        self.input_path = input_path
        self.prevabs_file = prevabs_file
        self.supporting_files = supporting_files
        self.inputs_list = inputs_list
        self.outputs_list = outputs_list
        self.inputs_dict = {}
        self.outputs_dict = {}
        self.force_linear = force_linear
        self.parallel = parallel
        self.saveallruns = saveallruns
        self.logger = logger

    def setup(self):
        """
        Setup method.

        Returns
        -------
        None.

        """

```

```
        # We need to create a unique filename for each job for parallel processing to work appropriately
        # Original prevabs input file
        self.script = PreVabsParse(self.input_path,self.prevabs_file,os.getcwd())

        # Parse the input file
        self.script.parse_input()

        # Prepare inputs and outputs
        for item in self.inputs_list:
            self.add_input(item[0],**item[2])
            self.inputs_dict[item[0]] = item[1]
        for item in self.outputs_list:
            self.add_output(item[0],**item[2])
            self.outputs_dict[item[0]] = item[1]

    def compute(self, inputs, outputs):
        """
        Generates new PreVABS files, executes PreVABS, and parses outputs.

        Parameters
        ----------
        inputs : dict
            Dictionary of inputs in the PreVABS subsystem.
        outputs : dict
            Dictionary of outputs in the PreVABS subsystem.

        Returns
        -------
        None.

        """

        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            # New job name with random ID added
            if self.parallel:
                self.new_prevabs_file = add_runid(self.prevabs_file)
            else:
                self.new_prevabs_file = self.prevabs_file

            # Set new filenames and copy original files
            self.input_file = self.new_prevabs_file + ".xml"
            for file in self.supporting_files: # supporting files such as basepoints.dat
                copyfile(os.path.join(self.input_path,file),file)

            self.output_file = self.new_prevabs_file + ".sg.K"
            self.options['external_input_files']=[self.input_file,]
            self.options['external_output_files']=[self.output_file,]

            # Prepare the command
            self.options['command'] = ('prevabs␣-i␣{}␣-h␣-e␣>␣/dev/null').format(self.input_file)

            # Modify input deck
            self.script.process_dict()
            for key,access_str in self.inputs_dict.items():
```

```
                input_value = []
                for i in range(len(inputs[key])):
                    if inputs[key][i].is_integer():
                        input_value.append(int(inputs[key][i]))
                    else:
                        input_value.append(inputs[key][i])
                access_list = create_access_list(self.script.prevabs_dict,access_str)
                old_val = get_from_dict(self.script.prevabs_dict,access_list)
                if old_val != input_value:
                    set_in_dict(self.script.prevabs_dict,access_list,input_value)

            # generate new PreVABS input deck
            self.script.unprocess_dict()
            self.script.generate(self.new_prevabs_file)

            # run PreVABS and parse
            print("PreVabsWrapper: Executing %s" % self.new_prevabs_file)
            super(PreVabsWrapper, self).compute(inputs, outputs)
            try:
                # Check if elasticity file exists
                self.script.parse_output(self.new_prevabs_file) # M and S matrices
            except ValueError:
                print("PreVabsWrapper: %s timed out, retrying" % self.new_prevabs_file)
                super(PreVabsWrapper, self).compute(inputs, outputs)
                self.script.parse_output(self.new_prevabs_file) # M and S matrices

            # parse and assign outputs
            self.script.parse_vabs_file(self.new_prevabs_file) # mesh information
            outs = []
            for key, value in self.outputs_dict.items():
                outs.append(value)

            for attr in outs:
                outputs[get_key(self.outputs_dict,attr)] = getattr(self.script,attr)

            # Remove run files
            if not self.saveallruns:
                os.remove(self.input_file)
                ext_list = ['.sg','.sg.ech','.sg.K','.sg.opt','.sg.v0','.sg.v1S','.sg.v22','.txt']
                for ext in ext_list:
                    file = self.new_prevabs_file+ext
                    if os.path.isfile(file):
                        os.remove(file)

class PreVabsRecover(ExternalCodeComp):
    """
    A file wrapper for PreVABS focused on stress/strain recovery.
    Should be run after PreVabsWrapper
    """

    def __init__(self,input_path,prevabs_file,
                 inputs_list,outputs_list,
                 force_linear=False,parallel=False,saveallruns=False,logger=None):
        # execute ExternalCodeComp.__init__()
        super(PreVabsRecover,self).__init__()

        # assign fields
        self.input_path = input_path
        self.prevabs_file = prevabs_file
```

```
        self.inputs_list = inputs_list
        self.outputs_list = outputs_list
        self.inputs_dict = {}
        self.outputs_dict = {}
        self.force_linear = force_linear
        self.parallel = parallel
        self.saveallruns = saveallruns
        self.logger = logger

    def setup(self):
        """
        Set up the component.

        Returns
        -------
        None.

        """
        # Prepare inputs and outputs
        for item in self.inputs_list:
            self.add_input(item[0],**item[2])
            self.inputs_dict[item[0]] = item[1]
        for item in self.outputs_list:
            self.add_output(item[0],**item[2])
            self.outputs_dict[item[0]] = item[1]

    def compute(self, inputs, outputs):
        """
        Generates new PreVABS files, executes PreVABS, and parses outputs (TBD).

        Parameters
        ----------
        inputs : dict
            Dictionary of inputs in the PreVABS subsystem.
        outputs : dict
            Dictionary of outputs in the PreVABS subsystem.

        Returns
        -------
        None.

        """

        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            # We need to create a unique filename for each job for parallel
                ↪ processing to work appropriately
            # Original prevabs input file
            self.orig_input_file = self.prevabs_file + ".xml"

            # New job name with random ID added
            if self.parallel:
                self.new_prevabs_file = add_runid(self.prevabs_file)
            else:
                self.new_prevabs_file = self.prevabs_file

            # Set new filenames and copy original files
```

```
224 |             self.input_file = self.new_prevabs_file + ".xml"
225 |             copyfile(self.orig_input_file,self.input_file)
226 |             ext_list = ['.sg','.sg.ech','.sg.K','.sg.opt','.sg.v0','.sg.v1S','.sg.v22
                    ↪ '] # all files VABS needs for recovery
227 |             for ext in ext_list:
228 |                 copyfile(self.prevabs_file+ext,self.new_prevabs_file+ext)
229 |
230 |             # Prepare the command
231 |             self.options['command'] = ('prevabs␣-i␣{}␣-d␣-e␣>␣/dev/null').format(self
                    ↪ .input_file)
232 |
233 |             # Parse input file at runtime since it may have been changed by
                    ↪ PreVabsWrapper
234 |             self.script = PreVabsParse(self.input_path,self.new_prevabs_file,os.
                    ↪ getcwd())
235 |             self.script.parse_input()
236 |             self.script.process_dict()
237 |
238 |             # Modify input deck
239 |             for key,access_str in self.inputs_dict.items():
240 |                 input_value = []
241 |                 for i in range(len(inputs[key])):
242 |                     if inputs[key][i].is_integer():
243 |                         input_value.append(int(inputs[key][i]))
244 |                     else:
245 |                         input_value.append(inputs[key][i])
246 |
247 |                 # pprint(access_list)
248 |                 access_list = create_access_list(self.script.prevabs_dict,access_str)
249 |                 old_val = get_from_dict(self.script.prevabs_dict,access_list)
250 |                 if old_val != input_value:
251 |                     set_in_dict(self.script.prevabs_dict,access_list,input_value)
252 |
253 |             # generate new PreVABS input deck
254 |             self.script.unprocess_dict()
255 |             self.script.generate(self.new_prevabs_file)
256 |
257 |             # run PreVABS and parse
258 |             print("PreVabsWrapper:␣Executing␣%s" % self.new_prevabs_file)
259 |             super(PreVabsRecover, self).compute(inputs, outputs)
260 |             try:
261 |                 # Check if recovery files exist
262 |                 self.script.parse_recovery(self.new_prevabs_file,'SM', '1', timeout
                        ↪ =10)
263 |             except ValueError:
264 |                 print("PreVabsWrapper:␣%s␣timed␣out,␣retrying" % self.
                        ↪ new_prevabs_file)
265 |                 super(PreVabsRecover, self).compute(inputs, outputs)
266 |
267 |             # read all requested outputs
268 |             outs = []
269 |             for key, value in self.outputs_dict.items():
270 |                 outs.append(value)
271 |
272 |             for item in outs:
273 |                 ext = item.split('%')[0]
274 |                 cols = item.split('%')[1]
275 |                 outputs[get_key(self.outputs_dict,item)] = self.script.parse_recovery
                        ↪ (self.new_prevabs_file, ext, cols, timeout=10)
276 |
277 |             # Remove run files
```

```
            if not self.saveallruns:
                os.remove(self.input_file)
                ext_list.extend(['.sg.E','.sg.ELE','.sg.EM','.sg.EMN','.sg.EN','.sg.S',
                                '.sg.S','.sg.SM','.sg.SMN','.sg.SN','.sg.U','.txt']) # all files VABS recovery generates
                for ext in ext_list:
                    file = self.new_prevabs_file+ext
                    if os.path.isfile(file):
                        os.remove(file)

def create_access_list(prevabs_dict,access_str):
    """
    Build an "access list" to traverse a nested dictionary.

    Parameters
    ----------
    prevabs_dict : dict
        Processed dictionary produced by PreVabsParse.process_dict.
    access_str : str
        Access string from input dictionary.

    Returns
    -------
    access_list : list
        List of keys to use for traversing the nested dictionary.

    """
    dict_chain = access_str.split("%")
    access_list = []

    # Iterate through dictionary to find the indicated value
    for level in dict_chain:
        level_type = None
        if "(" in level: # indexed level
            index = level.split("(")[1].split(")")[0]
            try:
                int(index)
            except:
                level_type = "named"
            else:
                level_type = "indexed"
        else: # unindexed level
            try:
                int(level)
            except:
                level_type = "normal"
            else:
                level_type = "index"

        if level_type == "normal": # enter next level normally
            prevabs_dict = prevabs_dict[level]
            access_list.append(level)
        elif level_type == "index": # enter by index
            index = int(level) - 1 # convert to indexed from 0
            prevabs_dict = prevabs_dict[index]
            access_list.append(index)
        elif level_type == "indexed": # enter by index of a key
            index = int(level.split("(")[1].split(")")[0]) - 1 # convert to indexed from 0
            level = level.split("(")[0]
```

```
            prevabs_dict = prevabs_dict[level][index]
            access_list.extend([level,index])
        elif level_type == "named": # enter by searching for matching name
            name = level.split("(")[1].split(")")[0]
            level = level.split("(")[0]
            to_search = prevabs_dict[level]
            results = [item for item in to_search if item["@name"] == name]
            if len(results) == 0:
                raise Exception("No name matching %s found in %s" % (name,level))
            elif len(results) > 1:
                raise Exception("Multiple names matching %s found in %s" % (name, level))
            else:
                prevabs_dict = results[0]
                access_list.extend([level,to_search.index(results[0])])

    if type(prevabs_dict) == dict or type(prevabs_dict) == OrderedDict:
        raise Exception("%s did not lead to a bottom-level value" % access_str)

    return access_list

# From: https://stackoverflow.com/questions/14692690/access-nested-dictionary-items-via-a-list-of-keys
def get_from_dict(dic, keys):
    dic_new = dic
    for k in keys:
        dic_new = dic_new[k]
    return dic_new

def set_in_dict(dic, keys, value):
    for key in keys[:-1]:
        # dic = dic.setdefault(key, {})
        dic = dic[key]

    if type(dic[keys[-1]]) == int or type(dic[keys[-1]]) == float:
        dic[keys[-1]] = value[0]
    elif type(dic[keys[-1]]) == list:
        dic[keys[-1]] = value
    else:
        raise Exception("%s is of type %s but it should be int, float, or list" %
                        dic[keys[-1]], type(dic[keys[-1]]))

def get_key(my_dict,val):
    for key, value in my_dict.items():
         if val == value:
             return key
```

APPENDIX B

GENERIC SINGLE MAIN ROTOR HELICOPTER MODELS

This appendix contains the descriptions of the NDARC, RCAS, and PreVABS+VABS models used to define the generic single main rotor (SMR) helicopter model. These models were used throughout Experiments 1 to 3. The NDARC model consists of a simple performance definition of the helicopter. The RCAS model serves as a more physically-accurate aeroelastic model of the helicopter. The PreVABS+VABS model defines the cross-section of the RCAS model's main rotor blade.

B.1 NDARC Model

The NDARC model is composed of four files. This model was derived from example files that are distributed to NDARC users via the NDARC website. The NDARC job, defined by `heli.njob`, instructs NDARC to skip the sizing task and run the performance analysis task. The aircraft definition is included in `heli.airc`. The engine performance map is defined by `gen2000.list`. A single generic performance condition is defined by `generic.cond`; these parameters are intended to be overwritten by OpenMDAO based on the defined flight condition for a given case.

Tables B.1 to B.3 describe the mapping between the variables in these files and the OpenMDAO variables. The RCOTOOLS access string syntax is described by the RCOTOOLS documentation. The OpenMDAO variables are further described in Appendix C.

Listing B.1: heli.njob

```
! Single Main Rotor Helicopter job file
! Originally created March 2009
! Forked by Joseph Robinson, 2020-07-14

&JOB open_status=1,&END
&DEFN action='ident',
    title='Heli job file',
    created='2020-07-14',
```

```
&END
!##############################################################################
&DEFN action='read file',file='gen2000.list',&END
&DEFN action='read file',file='heli.airc',&END
&DEFN action='read file',file='generic.cond',&END
! &DEFN action='read file',file='heli.cond',&END
! &DEFN action='read file',file='heli.miss',&END
!==============================================================================
&DEFN quant='Cases',&END
&VALUE
    title='Heli Cases',
  ! Tasks
    TASK_size=0,TASK_mission=0,TASK_perf=1,
  ! Outputs
    OUT_design=0,OUT_perf=0,OUT_geometry=0,
    OUT_aircraft=0,OUT_solution=1,OUT_sketch=0,
  ! Files
    FILE_design='heli.design',FILE_perf='heli.perf',
    FILE_geometry='heli.geom',FILE_sketch='heli.dxf',
    FILE_aircraft='heli.acd',FILE_solution='heli.soln',
    FILE_aero='heli.aero',FILE_engine='heli.eng',
    FILE_error='heli.err',
&END
!==============================================================================
&DEFN quant='Size',&END
&VALUE
    title='Heli Size',
    nFltCond=0,nMission=0,
    SIZE_perf='none',SET_rotor='radius+Vtip+sigma','radius+Vtip+sigma',
    FIX_DGW=1,FIX_WE=0,
    SET_tank='input',SET_SDGW='input',SET_WMTO='input',SET_limit_ds='input',
&END
!==============================================================================
&DEFN quant='Solution',&END
&VALUE
    title='Heli Solution',
  ! Trim
    niter_trim=120,
    mpid_trim=40,
    trace_trim=1,
&END
!==============================================================================
&DEFN action='endofcase',&END
&DEFN action='endofjob',&END
!##############################################################################
'endofinput'
```

Listing B.2: heli.airc

```
! Single Main Rotor Helicopter aircraft file
! Originally created March 2009
! Forked by Joseph Robinson, 2020-07-14

&DEFN action='ident',
    title='Heli aircraft file',
    created='2020-07-14',
&END
!##############################################################################
! default helicopter
&DEFN action='configuration',&END
```

```
&VALUE config='helicopter',rotate=1,&END
!==============================================================================
&DEFN quant='Cost',&END
&VALUE
    year_inf=2010,
    FuelPrice=5.00,
    Npass=10,
&END
&DEFN quant='Emissions',&END
&VALUE &END
!==============================================================================
&DEFN quant='Aircraft',&END
&VALUE
    title='Heli Aircraft',
    DGW=16000.,SDGW=17000.,WMTO=21000.,nz_ult=4.0,
    altitude=4000.,SET_atmos='temp',temp=95.,
    FIX_drag=0,FIX_DL=0,  ! FIX_drag=1 DoQ, 2 CD, 3 kDrag; FIX_DL=1 DoQV, 2 kDL
    INPUT_geom=1, ! fixed dimensional geometry (SL,BL,WL)
        KIND_scale=1,kScale=1, ! mr reference length for kx,ky,kz
    KIND_Ref=3,kRef=1,  ! fuselage reference point
    kx=0.1,ky=0.3,kz=0.3, ! radii of gyration scaled to reference length
&END
!------------------------------------------------------------------------------
&DEFN quant='Systems',&END
&VALUE
    title='Heli Systems',
    SET_Wpayload=2,Upass=200.,
    SET_Wcrew=2,Ncrew=2,Ucrew=220.,Wcrew=160.,
    nWoful=2,Woful_name='baggage','survival kit',Woful=60.,40.,
    Wtrap=75.,
    SET_Wvib=1,Wvib=0.,fWvib=0.025,
    SET_Wcont=2,Wcont=0.,
    Wfc_cc=100.,Wfc_afcs=100.,
    fRWfc_nb=1.25,fRWhyd=0.4,
    MODEL_FWfc=1,MODEL_WFWfc=2,fFWfc_nb=0.15,
    Wauxpower=200.,Winstrument=200.,Wpneumatic=0.,Welectrical=400.0,
    WMEQ=400.,
    Wfurnish=600.,Wenviron=100.,
    Ncrew_seat=2,Npass_seat=10,Ucrew_seat_inc=50.,Upass_seat_inc=40.,
    MODEL_DI=1,kDeIce_elec=0.25,0.25,kDeIce_rotor=0.25,0.08,kDeIce_air=0.006,
    SET_fold=0,
&END
!------------------------------------------------------------------------------
&DEFN quant='Fuselage',&END
&VALUE
    title='Heli Fuselage',
  ! geometry
    SET_length=4,SET_nose=1,SET_aft=2,Length_nose=12.,fLength_aft=-0.2,
       fRef_fus=0.4,Width_fus=8.,
    SET_Swet=3,fSwet=0.7,fSproj=1.0,
       Height_fus=6.,
       Circum_boom=18.,Width_boom=2.7,
  ! aerodynamics
    DoQ_cont=0.,
    AoA_zl=0.,AoA_max=35.,SS_max=35.,
    SET_lift=2,dCLda=0.1,
    SET_moment=2,CM0=0.,dCMda=0.05,
    ! SET_side=2,dCYdb=-0.15,         ! not possible in RCAS (closed-form)
    SET_side=2,dCYdb=0.,
    ! SET_yaw=2,CN0=0.,dCNdb=-0.03,   ! not possible in RCAS (closed-form)
```

```
    SET_yaw=2,CN0=0.,dCNdb=-0.,
    SET_drag=2,SET_Dfit=2,SET_Drb=2,SET_Vdrag=2,SET_Sdrag=2,
    CD=0.0065,CD_fit=0.0065,CD_rb=0.,0.,CDV=0.40,CDS=0.04,
    MODEL_drag=2,AoA_Dmin=0.,Kdrag=15.,AoA_tran=25.,
  ! weight
    fWbody_crash=0.06,
&END
!-----------------------------------------------------------------------------
&DEFN quant='LandingGear',&END
&VALUE
    title='Heli Landing Gear',
  ! drag
    DoQ=0.,
  ! weight
    nLG=3,fWLG_crash=0.15,fWLG_ret=0.,
&END
!=============================================================================
&DEFN quant='Geometry',&END
&VALUE
  ! fixed geometry (INPUT_geom=1); SL +aft, BL +right, WL +up
  ! fuselage reference
    loc_cg%SL           = 0.504, loc_cg%BL           = 0.007, loc_cg%WL           = 1.098,
    loc_fuselage%SL     =  0.00, loc_fuselage%BL     =  0.00, loc_fuselage%WL     = 0.00,
    loc_gear%SL         =  0.00, loc_gear%BL         =  0.00, loc_gear%WL         = -5.36,
    loc_rotor(1)%SL     =-0.351, loc_rotor(1)%BL     =  0.00, loc_rotor(1)%WL     = 6.691, ! 6.7 ft from fuselage center with 3 degree forward shaft tilt
    loc_pylon(1)%SL     =  0.00, loc_pylon(1)%BL     =  0.00, loc_pylon(1)%WL     = 3.752,
    loc_rotor(2)%SL     = 33.05, loc_rotor(2)%BL     = 1.072, loc_rotor(2)%WL     = 7.772,
    loc_pylon(2)%SL     =  0.00, loc_pylon(2)%BL     = 1.072, loc_pylon(2)%WL     = 7.772, ! SET_geom=tailrotor: pylon SL relative hub
    loc_tail(1)%SL      = 29.48, loc_tail(1)%BL      =  0.00, loc_tail(1)%WL      = 0.00,
    loc_tail(2)%SL      =31.265, loc_tail(2)%BL      =  0.00, loc_tail(2)%WL      = 3.886, ! halfway between boom endpoint and tail rotor
    loc_auxtank(1,1)%SL =  0.00, loc_auxtank(1,1)%BL =  0.00, loc_auxtank(1,1)%WL = 0.00,
    loc_engine(1)%SL    =  0.00, loc_engine(1)%BL    =  0.00, loc_engine(1)%WL    = 4.02,
&END
!=============================================================================
&DEFN quant='Rotor 1',&END
&VALUE
    title='Heli Main Rotor',
    Vtip_ref=723.6,
       INPUT_Vtip=2,fRPM_cruise=1.,fRPM_man=1.,fRPM_oei=1.,fRPM_xmsn=1.,
    Plimit_rs=2880.,fPlimit_rs=2.,
    radius=26.8,sigma=0.08314064191,rotate=1,nblade=4,
    SET_chord=2,SET_twist=1,twistL=-10,
       !nprop=3,rprop=0.,.9,1.,fchord=1.,1.,.6, ! for SET_chord=3
       taper=1, ! for SET_chord=2
    KIND_hub=1,flapfreq=1.035,gamma=8.26,precone=0.,
    dclda=5.7,tiploss=.97,xroot=0.15,thick=0.09,
    incid_hub=-3.,cant_hub=0.,
  ! performance
    MODEL_Ftpp=2,MODEL_Fpro=2,
  ! induced power
    Ki_hover=1.125,Ki_climb=1.125,Ki_prop=1.125,Ki_edge=2.0,Ki_min=1.1,
```

```
    CTs_Hind=.10,kh1=1.25,kh2=0.,
    mu_edge=0.35,ke1=.8,ke2=0.,ke3=1.,Xe=4.5,
  ! profile power
    MODEL_basic=2,
    CTs_Dmin=.05,d0_hel=.0080,d0_prop=.0080,d1_hel=0.,d1_prop=0.,d2_hel=.5,d2_prop=.5,
    CTs_sep=.07,dsep=4.,Xsep=3.,
    MODEL_stall=1,CTs_stall(1)=0., ! default
    fstall=1.,dstall1=2.,dstall2=40.,Xstall1=2.,Xstall2=3.,
    MODEL_comp=1,Mdd0=.73,Mddcl=0.,dm1=.005,dm2=1.,Xm=3.,
  ! drag
    SET_Spylon=2,kSwet_pylon=0.9, ! scaled pylon wetted area
    SET_Dhub=2,SET_Vhub=2,SET_Dpylon=2,SET_Vpylon=2, ! scale drag based on CD
    CD_hub=0.0025,CDV_hub=0.0025,CD_pylon=0.04,CDV_pylon=0.04, ! CD values for above
  ! weight
    TECH_blade=0, ! set custom blade weight
    dWblade=737.815, ! from VABS + RCAS blade model
  ! controls
    KIND_control=4, ! pitch and NFP
&END
!-------------------------------------------------------------------------------
&DEFN quant='Rotor 2',&END
&VALUE
    title='Heli Tail Rotor',
    INPUT_gear=1,Vtip_ref=648.,
    Plimit_rs=640.,fPlimit_rs=2.,
    radius=6.,sigma=0.16,rotate=1,nblade=4,!clearance_tr=0.25,
    SET_chord=2,taper=1.,SET_twist=1,twistL=0.,
    !KIND_hub=1,flapfreq=1.2,gamma=4.,precone=0.,delta3=45., ! articulated
    KIND_hub=2,flapfreq=1.2,gamma=1.89,precone=0., ! hingeless
    dclda=5.7,tiploss=.97,xroot=0.4,thick=.09,
    incid_hub=0.,cant_hub=0.,!20.,
  ! performance
    MODEL_Ftpp=2,MODEL_Fpro=2,
  ! induced power
    Ki_hover=1.65,Ki_climb=1.65,Ki_prop=1.65,Ki_edge=2.0,Ki_min=1.65,
    CTs_Hind=.05,kh1=0.,kh2=80.,
    mu_edge=0.35,ke1=.8,ke2=0.,ke3=1.,Xe=4.5,
  ! profile power
    MODEL_basic=2,
    CTs_Dmin=.04,d0_hel=.0090,d0_prop=.0090,d1_hel=0.,d1_prop=0.,d2_hel=.9,d2_prop=.9,
    CTs_sep=.06,dsep=20.,Xsep=3.,
    MODEL_stall=1,CTs_stall(1)=0., ! default
    fstall=1.,dstall1=5.,dstall2=40.,Xstall1=2.,Xstall2=3.,
    MODEL_comp=1,Mdd0=.68,Mddcl=0.,dm1=.005,dm2=1.,Xm=3.,
  ! no interference (vertical tail at low speed)
    MODEL_int=0,
  ! drag
    SET_Spylon=1,Swet_pylon=0.,
    SET_Dhub=2,SET_Vhub=2,SET_Dpylon=2,SET_Vpylon=2,
    CD_hub=0.030,CDV_hub=0.030,CD_pylon=0.,CDV_pylon=0.,
  ! controls
    KIND_control=4, ! pitch and NFP
&END
!===============================================================================
&DEFN quant='Tail 1',&END
&VALUE
    title='Heli Horizontal Tail',
    SET_tail='vol+aspect',
    TailVol=0.020,AspectRatio=5.,
```

```
    taper=1.,sweep=0.,dihedral=0.,thick=0.12,fchord_cont=0.,
    cant=0.,incid=0.,
  ! aerodynamics
    AoA_zl=1.,CLmax=0.8,SET_lift=3,dCLda=2.3,Tind=1.,Eind=.9,
    SET_drag=2,SET_Vdrag=2,
    CD=0.015,CDV=0.015,
    AoA_Dmin=0.,MODEL_drag=0,
  ! weight
    Vdive=250.,
&END
!-------------------------------------------------------------------------------
&DEFN quant='Tail 2',&END
&VALUE
    title='Heli Vertical Tail',
    ! SET_tail='vol+aspect',
    SET_tail='vol+span',
    TailVol=0.015,!AspectRatio=2.,
    span=8.5527, ! calculated from RCAS model (boom end to tail rotor)
    taper=1.,dihedral=0.,thick=0.20,fchord_cont=0.,
    sweep=24.67, ! calculated from RCAS model (boom end to tail rotor)
    cant=0.,incid=0.,
  ! aerodynamics
    AoA_zl=0.,CLmax=1.,SET_lift=3,dCLda=1.5,Tind=1.,Eind=.9,
    SET_drag=2,SET_Vdrag=2,
    CD=0.020,CDV=0.020,
    AoA_Dmin=0.,MODEL_drag=0,
  ! weight
    Vdive=210.,
    place_AntiQ=2, ! place=tail
&END
!===============================================================================
&DEFN quant='FuelTank',&END
&VALUE
    title='Heli Fuel Tank',
    Wfuel_cap=2500.,
  ! aux tanks
    Waux_cap=5000.,DoQ_auxtank=0.,fWauxtank=0.11,
  ! weight
    ntank_int=2,nplumb=2,Ktoler=1.8,
    K0_plumb=120.,K1_plumb=3.,
&END
!===============================================================================
&DEFN quant='Propulsion',&END
&VALUE
    title='Heli Propulsion',
  ! losses
    MODEL_Xloss=1,
    fPloss_xmsn=0.02,Ploss_windage=0.0,Pacc_0=60.,Pacc_d=0.,Pacc_n=0.,fPacc_ECU=0.,
        ↪ fPacc_IRfan=0.,
  ! geometry
    SET_length=2,fLength_ds=0.9,
  ! torque limit
    Plimit_ds=2880.,fPlimit_ds=0.9,
  ! weight
    ngearbox=5,ndriveshaft=4,fShaft=0.10,
&END
!-------------------------------------------------------------------------------
&DEFN quant='EngineGroup',&END
&VALUE
    title='Heli Engine Group',
```

```
    nEngine=2,nEngine_main=2,
    IDENT_engine='GEN2000',
    Peng=1600.,rating_to='MRP',
    SET_Swet=2,kSwet=0.8,
  ! torque limit
    Plimit_es=2880.,fPlimit_es=0.9,
  ! installation
    eta_d=0.99,
    Kffd=1.05,fPloss_inlet=.007,fPloss_ps=0.,
    fPloss_exh=.02,fMF_auxair=.01,eta_auxair=.75,
    fPloss_exh_IRon=.02,fMF_auxair_IRon=.01,eta_auxair_IRon=.75,
  ! drag
    SET_drag=2,SET_Vdrag=2,
    CD=0.0100,CDV=0.0100,
  ! weight
    fWpylon=0.,fWair=0.45,
    Kwt0_exh=0.,Kwt1_exh=0.06,
    MODEL_lub=1,
&END
!==============================================================================
&DEFN quant='TechFactors',&END
&VALUE
  ! cost
    TECH_cost_af=1.,TECH_cost_maint=1.,
  ! flight control
    TECH_RWfc_b=1.,TECH_RWfc_mb=1.,TECH_RWfc_nb=1.,TECH_RWhyd=1.,
    TECH_FWfc_nb=1.,TECH_FWfc_mb=1.,TECH_FWhyd=1.,
  ! anti-icing
    TECH_DIelect=1.,TECH_DIsys=1.,
  ! fuselage
    TECH_body=1.,TECH_mar=1.,TECH_press=1.,TECH_crash=1.,TECH_ftfold=1.,TECH_fwfold
        ↪ =1.,
    TECH_LG=1.,TECH_LGret=1.,TECH_LGcrash=1.,
  ! rotor profile power
    TECH_drag=1.,1.,
  ! rotor
    TECH_blade(2)=1., ! only automatically solve weight for tail rotor
    TECH_hub=1.,1.,TECH_spin=1.,1.,TECH_rfold=1.,1.,TECH_tr=1.,1.,TECH_aux=1.,1.,
  ! tail
    TECH_tail=1.,1.,
  ! fuel tank
    TECH_tank=1.,TECH_plumb=1.,
  ! drive system
    TECH_gb=1.,TECH_rs=1.,TECH_ds=1.,TECH_rb=1.,
  ! engine group
    TECH_eng=1.,TECH_cowl=1.,TECH_pylon=1.,TECH_supt=1.,TECH_air=1.,TECH_exh=1.,
        ↪ TECH_acc=1.,
&END
!==============================================================================
!##############################################################################
&DEFN action='endoffile',&END
```

Listing B.3: gen2000.list

```
&DEFN action='ident',created='October 2008',
     title = 'Turboshaft Engine Math Model Data (GEN2000)',
&END
&DEFN quant='EngineModel',&END
&VALUE
! Turboshaft Engine Model
```

```
     title = 'Turboshaft Engine Math Model Data (GEN2000)',
     notes = 'Generic 2000 hp Engine',
     ident = 'GEN2000',
! Engine Ratings
     nrate = 4,
     rating = 'MCP  ', 'IRP  ', 'MRP  ', 'CRP  ',
! Weight
     Kwt0_eng =       0.000,
     Kwt1_eng =       0.180,
     Kwt2_eng =       0.000,
     Xwt_eng  =      0.0000,
! Reference
     P0_ref     =     2000.,     2400.,     2540.,     2660.,
     SP0_ref    =      134.,      145.,      164.,      171.,
     Pmech_ref  =     3000.,     3000.,     3000.,     3000.,
     sfc0C_ref  =     0.44,
     SF0C_ref   =      8.9,
     Nspec_ref  =   19100.,
     Nopt0C_ref =   19100.,
! Scaling, MF_ref=15.0
     MF_limit    =     30.,
     SP0C_limit  =    143.,
     sfc0C_limit =    0.40,
     KNspec      =  73790.,
! Optimum Power Turbine Speed (linear)
     MODEL_OptN = 1,
     KNoptA =     1.,
     KNoptB =     0.,
     XNeta  =     2.,
! Single Set of Input Parameters
     INPUT_param = 1,
! Power Available
     INPUT_lin = 1,
   ! specific power
     Nspa =  1,  1,  1,  1,
     Kspa0(1,1) =       3.80,   0.00,   0.00,   0.00,   0.00,  ! MCP
     Kspa0(1,2) =       3.20,   0.00,   0.00,   0.00,   0.00,  ! IRP
     Kspa0(1,3) =       3.00,   0.00,   0.00,   0.00,   0.00,  ! MRP
     Kspa0(1,4) =       2.80,   0.00,   0.00,   0.00,   0.00,  ! CRP
     Kspa1(1,1) =      -2.80,   0.00,   0.00,   0.00,   0.00,  ! MCP
     Kspa1(1,2) =      -2.20,   0.00,   0.00,   0.00,   0.00,  ! IRP
     Kspa1(1,3) =      -2.00,   0.00,   0.00,   0.00,   0.00,  ! MRP
     Kspa1(1,4) =      -1.80,   0.00,   0.00,   0.00,   0.00,  ! CRP
     Xspa0(1,1) =      -0.22,   0.00,   0.00,   0.00,   0.00,  ! MCP
     Xspa0(1,2) =      -0.22,   0.00,   0.00,   0.00,   0.00,  ! IRP
     Xspa0(1,3) =      -0.22,   0.00,   0.00,   0.00,   0.00,  ! MRP
     Xspa0(1,4) =      -0.22,   0.00,   0.00,   0.00,   0.00,  ! CRP
     Xspa1(1,1) =       0.00,   0.00,   0.00,   0.00,   0.00,  ! MCP
     Xspa1(1,2) =       0.00,   0.00,   0.00,   0.00,   0.00,  ! IRP
     Xspa1(1,3) =       0.00,   0.00,   0.00,   0.00,   0.00,  ! MRP
     Xspa1(1,4) =       0.00,   0.00,   0.00,   0.00,   0.00,  ! CRP
   ! mass flow
     Nmfa =  2,  2,  2,  2,
     Kmfa0(1,1) =       0.30,   0.70,   0.00,   0.00,   0.00,  ! MCP
     Kmfa0(1,2) =       0.29,   0.45,   0.00,   0.00,   0.00,  ! IRP
     Kmfa0(1,3) =       0.28,   0.35,   0.00,   0.00,   0.00,  ! MRP
     Kmfa0(1,4) =       0.27,   0.30,   0.00,   0.00,   0.00,  ! CRP
     Kmfa1(1,1) =      -0.30,  -0.70,   0.00,   0.00,   0.00,  ! MCP
     Kmfa1(1,2) =      -0.29,  -0.45,   0.00,   0.00,   0.00,  ! IRP
     Kmfa1(1,3) =      -0.28,  -0.35,   0.00,   0.00,   0.00,  ! MRP
```

```
      Kmfa1(1,4) =      -0.27,  -0.30,   0.00,   0.00,   0.00,  ! CRP
      Xmfa0(1,1) =       1.05,   1.10,   0.00,   0.00,   0.00,  ! MCP
      Xmfa0(1,2) =       1.05,   1.10,   0.00,   0.00,   0.00,  ! IRP
      Xmfa0(1,3) =       1.05,   1.10,   0.00,   0.00,   0.00,  ! MRP
      Xmfa0(1,4) =       1.05,   1.10,   0.00,   0.00,   0.00,  ! CRP
      Xmfa1(1,1) =       0.00,  -0.01,   0.00,   0.00,   0.00,  ! MCP
      Xmfa1(1,2) =       0.00,  -0.01,   0.00,   0.00,   0.00,  ! IRP
      Xmfa1(1,3) =       0.00,  -0.01,   0.00,   0.00,   0.00,  ! MRP
      Xmfa1(1,4) =       0.00,  -0.01,   0.00,   0.00,   0.00,  ! CRP
 ! Performance at Power Required
    ! fuel flow         mass flow         gross jet thrust  net jet thrust
      Kffq0 =   0.20,   Kmfq0 =   0.60,   Kfgq0 =   0.20,   Kfgr0 =   0.80,
      Kffq1 =   0.80,   Kmfq1 =   0.78,   Kfgq1 =   0.80,   Kfgr1 =   0.60,
      Kffq2 =   0.00,   Kmfq2 =  -0.48,   Kfgq2 =   0.00,   Kfgr2 =   0.00,
      Kffq3 =   0.00,   Kmfq3 =   0.10,   Kfgq3 =   0.00,   Kfgr3 =   0.00,
      Xffq  =   1.30,   Xmfq  =   3.50,   Xfgq  =   2.00,
 &END
```

Listing B.4: generic.cond

```
! Generic performance file
! Created by Joseph Robinson, 2020-07-15

&DEFN action='ident',
    title='Generic performance file',
    created='2020-07-15',
&END
!##############################################################################
!==============================================================================
&DEFN quant='Performance',&END
&VALUE
    title='Heli performance analysis',
    nFltCond=1,
&END

!------------------------------------------------------------------------------
&DEFN quant='PerfCondition 1',&END
&VALUE
    title='Generic',label='generic',
    SET_GW='input',GW=16000., ! gross weight, lbs
    SET_UL='fuel',fFuel=1., ! top off fuel tanks
    SET_atmos='std',altitude=0., ! density altitude, ft
    Vkts=120., ! forward flight velocity, kn
    ROC=0., ! rate of climb, ft/min
    SET_turn=1,rate_turn=0., ! turn rate, deg/s
    STATE_trim='free',
    pitch=-1.,coll=7.5,pedal=-5.,lngcyc=5.,! trim initial conditions, deg
    relax_trim=.2,
&END
!==============================================================================
!##############################################################################
&DEFN action='endoffile',&END
```

Table B.1: Mapping from OpenMDAO variables to NDARC variables (weights).

OpenMDAO variable	RCOTOOLS access string[a]	Units
weight_main_blade	Rotor(1)%dWblade	lb
weight_fuselage	Aircraft%WEIGHT%W_FUSELAGE	lb
weight_systems	Aircraft%WEIGHT%W_EQUIP	lb
weight_fuelsystem	Aircraft%WEIGHT%W_FUELSYS	lb
weight_gear	Aircraft%WEIGHT%W_GEAR	lb
weight_main_hub	Aircraft%WEIGHT%W_ROTOR_HUB	lb
weight_main_blades	Aircraft%WEIGHT%W_ROTOR_BLADE	lb
weight_tailrotor	Aircraft%WEIGHT%W_TAILROTOR	lb
weight_horizstab	Aircraft%WEIGHT%W_HTAIL	lb
weight_vertstab	Aircraft%WEIGHT%W_VTAIL	lb
weight_drive	Aircraft%WEIGHT%W_DRIVE	lb
weight_engstruct	EngineGroup(1)%WEIGHT%W_STRUCTURE	lb
weight_engine	Aircraft%WEIGHT%W_ENGSYS	lb
weight_fuel	Performance%PerfCondition(1)%FltAircraft%WFUEL_TOTAL	lb
weight_usefulload	Performance%PerfCondition(1)%FltAircraft%WFIXUL	lb
weight_payload	Performance%PerfCondition(1)%FltAircraft%WPAYLOAD	lb

[a] The RCOTOOLS access string identifies a specific location in the NDARC data structure.

Table B.2: Mapping from OpenMDAO variables to NDARC variables (geometry).

OpenMDAO variable	RCOTOOLS access string	Units
loc_fuselage_sl	Fuselage%LOC_FUSELAGE_SLLOC	ft
loc_fuselage_bl	Fuselage%LOC_FUSELAGE_BLLOC	ft
loc_fuselage_wl	Fuselage%LOC_FUSELAGE_WLLOC	ft
loc_gear_sl	LandingGear%LOC_GEAR_SLLOC	ft
loc_gear_bl	LandingGear%LOC_GEAR_BLLOC	ft
loc_gear_wl	LandingGear%LOC_GEAR_WLLOC	ft
loc_mainrotor_sl	Rotor(1)%LOC_ROTOR_SLLOC	ft
loc_mainrotor_bl	Rotor(1)%LOC_ROTOR_BLLOC	ft
loc_mainrotor_wl	Rotor(1)%LOC_ROTOR_WLLOC	ft
loc_tailrotor_sl	Rotor(2)%LOC_ROTOR_SLLOC	ft
loc_tailrotor_bl	Rotor(2)%LOC_ROTOR_BLLOC	ft
loc_tailrotor_wl	Rotor(2)%LOC_ROTOR_WLLOC	ft
loc_horizstab_sl	Tail(1)%LOC_TAIL_SLLOC	ft
loc_horizstab_bl	Tail(1)%LOC_TAIL_BLLOC	ft
loc_horizstab_wl	Tail(1)%LOC_TAIL_WLLOC	ft
loc_vertstab_sl	Tail(2)%LOC_TAIL_SLLOC	ft
loc_vertstab_bl	Tail(2)%LOC_TAIL_BLLOC	ft
loc_vertstab_wl	Tail(2)%LOC_TAIL_WLLOC	ft
loc_fueltank_sl	FuelTank(1)%LOC_AUXTANK(1)_SLLOC	ft
loc_fueltank_bl	FuelTank(1)%LOC_AUXTANK(1)_BLLOC	ft
loc_fueltank_wl	FuelTank(1)%LOC_AUXTANK(1)_WLLOC	ft
loc_engine_sl	EngineGroup(1)%LOC_ENGINE_SLLOC	ft
loc_engine_bl	EngineGroup(1)%LOC_ENGINE_BLLOC	ft
loc_engine_wl	EngineGroup(1)%LOC_ENGINE_WLLOC	ft
tail_rotor_cant	Rotor(2)%cant_hub	deg

Table B.3: Mapping from OpenMDAO variables to NDARC variables (flight condition).

OpenMDAO variable	RCOTOOLS access string	Units
airspeed	PerfCondition(1)%Vkts	kt
altitude	PerfCondition(1)%altitude	ft
gross_weight	PerfCondition(1)%GW	lb
rate_of_climb	PerfCondition(1)%ROC	ft/min
turn_rate	PerfCondition(1)%rate_turn	deg/s
loc_cg_sl	Geometry%loc_cg%SL	ft
loc_cg_bl	Geometry%loc_cg%BL	ft
loc_cg_wl	Geometry%loc_cg%WL	ft
att_pitch	Performance%PerfCondition(1)%FltAircraft%PITCH_TRIM	deg
att_roll	Performance%PerfCondition(1)%FltAircraft%ROLL_TRIM	deg
ndarc_cont_coll	PerfPerformance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(1)	deg
ndarc_cont_lat_cyc	Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(2)	deg
ndarc_cont_lon_cyc	Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(3)	deg
ndarc_cont_pedal	Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(4)	deg
air_viscosity	Performance%PerfCondition(1)%FltAircraft%VISCOSITY	slug/(ft s)
air_density	Performance%PerfCondition(1)%FltAircraft%DENSITY	slug/ft^3
air_sound_speed	Performance%PerfCondition(1)%FltAircraft%CSOUND	ft/s

B.2 RCAS Model

The RCAS model is composed of a single file titled `heli.rcas`. Because the original RCAS file cannot be published, this section consists of a series of paragraphs and tables that describe the construction of the RCAS module.

B.2.1 Structural Model

The structural model consists of three subsystems. The `mrotor` subsystem describes the main rotor, the `fusess` subsystem describes the fuselage, and the `trotor` subsystem describes the tail rotor. The `mrotor` and `trotor` subsystems are attached to the `fusess` subsystem using a rotating-nonrotating constraint at each subsystem's respective hub node.

The model is attached to the world frame at the center point of the fuselage. All degrees of freedom are enabled. The trim springs and dampers that connect the vehicle to the world frame are described in Table B.4.

Table B.4: Coefficients of RCAS trim springs and dampers.

Object	Units	x	y	z
Translational spring	lbf/ft	7.5×10^5	7.5×10^5	7.5×10^5
Rotational spring	ft lbf/rad	4.0×10^6	8.5×10^7	6.5×10^6
Translational damper	lbf s/ft	7.5×10^3	7.5×10^3	7.5×10^3
Rotational damper	ft lbf s/rad	4.5×10^5	1.0×10^6	7.5×10^5

The geometric origins of the three subsystems are described in Table B.5 and their orientations are described in Table B.6.

Table B.5: Origins of RCAS structural model subsystems.

Subsystem	Units	x	y	z
`mrotor`	ft	0	0	0
`fusess`	ft	0	0	0
`trotor`	ft	–33.05	0	–7.772

Table B.6: Orientations of RCAS structural model subsystems.

Subsystem	Units	First rotation (y)	Second rotation (x)
`mrotor`	deg	177	0
`fusess`	deg	0	0
`trotor`	deg	180	–90

Pilot controls are mapped to the helicopter's control surface using a control mixer. The control mixer is described in Table B.7.

Table B.7: Control mixer of RCAS model.

Control	Value at 0	Coll.	Lat. cyclic	Long. cyclic	Pedal	Throttle
SP coll.	0	–0.0123	0	0	0	0
SP lat. cyc.	0	0	–0.00853	0	0	0
SP long. cyc.	0	0	0	–0.00853	0	0
TR coll.	0	0	0	0	–0.017453	0

B.2.1.1 Fuselage Subsystem

The `fusess` subsystem is composed of three primitive structures. `fuseps` models the fuselage, `vstabps` models the vertical stabilizer, and `hstabps` models the horizontal stabilizer. The `vstabps` and `hstabps` structures are constrained to the `fuseps` structure at their respective root nodes.

The geometric origins of the three primitive structures are described in Table B.8 and their orientations are described in Table B.9.

Table B.8: Origins of RCAS model fuselage subsystem primitive structures.

Subsystem	Units	x	y	z
`mrotor`	ft	0	0	0
`fusess`	ft	–29.48	0	0
`trotor`	ft	–29.48	0	0

Table B.9: Orientations of RCAS model fuselage subsystem primitive structures.

Subsystem	Units	First rotation (z)	Second rotation (y)
`mrotor`	deg	0	0
`fusess`	deg	−90	90
`trotor`	deg	−90	180

Fuselage Primitive Structure The nodes comprising the `fuseps` primitive structure are listed in Table B.10. These nodes are connected by five massless rigid bars running from node 2 to each other node.

Table B.10: Nodes of RCAS model fuselage primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Main rotor attachment point
2	ft	0	0	0	Fuselage center point
3	ft	−29.48	0	0	Stabilizer attachment point
4	ft	2	0	0	Payload CG
5	ft	0	0	5.36	Landing gear CG
6	ft	0	0	−4.02	Engine CG

The rigid body masses in this primitive structure are listed in Table B.11. Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized.

Table B.11: Rigid body masses of RCAS model fuselage primitive structure.

Elem. ID	Node	m (slug)	I_{xx} (slug ft^2)	I_{yy} (slug ft^2)	I_{zz} (slug ft^2)	Description
1	2	173.231	866.154	14,371.257	14,128.734	Fuselage
11	2	24.088	0	0	0	Fixed UL
12	5	19.582	0	0	0	Gear
13	6	84.731	169.658	789.617	789.617	Engine
14	2	77.703	0	0	0	Fuel
20	4	67.405	0	0	0	Payload

* The cross products of inertia (I_{xy}, I_{xz}, and I_{yz}) for all rigid body masses are zero.

Vertical Stabilizer Primitive Structure The nodes comprising the `vstabps` primitive structure are listed in Table B.12.

Table B.12: Nodes of RCAS model vertical stabilizer primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Root
2	ft	7.772	−3.57	0	Tip

The two nodes of this structure are connected by a single rigid bar with a center of gravity offset 4.276 ft along the x-axis from node 1, which models the mass and inertial of the vertical stabilizer. This rigid bar has the following mass properties: $m = 2.125\,\text{slug}$, $I_{xx} = 2.231\,\text{slug}\,\text{ft}^2$, $I_{yy} = 15.176\,\text{slug}\,\text{ft}^2$, $I_{zz} = 12.959\,\text{slug}\,\text{ft}^2$, and $I_{xy} = I_{yz} = I_{xz} = 0$. Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized.

Horizontal Stabilizer Primitive Structure The nodes comprising the `hstabps` primitive structure are listed in Table B.13.

Table B.13: Nodes of RCAS model horizontal stabilizer primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Root
2	ft	7.32968	0	0	Starboard tip
2	ft	−7.32968	0	0	Port tip

The two nodes of this structure are connected by a single rigid bar with its center of gravity at node 1 which models the mass and inertial of the horizontal stabilizer. This rigid bar has the following mass properties: $m = 3.241\,\text{slug}$, $I_{xx} = 2.325\,\text{slug}\,\text{ft}^2$, $I_{yy} = 58.038\,\text{slug}\,\text{ft}^2$, $I_{zz} = 60.356\,\text{slug}\,\text{ft}^2$, and $I_{xy} = I_{yz} = I_{xz} = 0$. Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized.

B.2.1.2 Main Rotor Subsystem

The `mrotor` subsystem is composed of two primitive structures. `mshaft` models the main rotor shaft and `blade1` models a single rotor blade and a corresponding portion of the swashplate. The single blade analysis option is used to duplicate `blade1` to model a four-bladed rotor. The center of rotation of the rotor is at the root node of `mshaft`. The rotor is defined to have a nominal rotation speed of 27.0 rad/s.

The geometric origins of the three primitive structures are described in Table B.14. These structures are not rotated.

Table B.14: Origins of RCAS model main rotor subsystem primitive structures.

Primitive	Units	x	y	z
`mshaft`	ft	0	0	0
`blade1`	ft	0	0	6.7

The `blade1` structure is constrained to the `mshaft` structure at the root of the blade and at the pitch link.

Main Rotor Shaft Primitive Structure The nodes comprising the `mshaft` primitive structure are listed in Table B.15.

Table B.15: Nodes of RCAS model main rotor shaft primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Connection to fixed system
2	ft	0	0	0	
3	ft	0	0	6.7	
30	ft	0	0	5.8	Swashplate center

A free z-oriented hinge is placed between nodes 1 and 2 to allow the shaft to spin. A rigid bar connects nodes 2 and 3 to model the mass and inertia of the shaft. The center of gravity of this bar is located at node 3. The bar has the following mass properties: $m = 18.605\,\text{slug}$, $I_{xx} = 14.535\,\text{slug ft}^2$, $I_{yy} = 7.268\,\text{slug ft}^2$, $I_{zz} = 7.268\,\text{slug ft}^2$, and $I_{xy} = I_{yz} = I_{xz} = 0$.

Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized. A second, massless rigid bar connects nodes 1 and 30.

Main Rotor Blade Primitive Structure The nodes comprising the `blade1` primitive structure are listed in Table B.16.

Table B.16: Nodes of RCAS model main rotor blade primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Blade root
2	ft	1.25	0	0	Lag hinge first node
3	ft	1.25	0	0	Lag hinge second node
4	ft	1.25	0	0	Flap hinge second node
5	ft	1.25	0	0	Pitch bearing second node
6	ft	3.805	0	0	Element node
7	ft	6.360	0	0	Element node
8	ft	8.915	0	0	Element node
9	ft	11.470	0	0	Element node
10	ft	14.025	0	0	Element node
11	ft	16.580	0	0	Element node
12	ft	19.135	0	0	Element node
13	ft	21.690	0	0	Element node
14	ft	24.245	0	0	Element node
20	ft	26.80	0	0	Blade tip
30	ft	0	0	–0.9	Swashplate center
31	ft	0	0	–0.9	Swashplate center
32	ft	0	0	–0.9	Swashplate center
33	ft	1.25	0	–0.9	Swashplate radial spoke node
34	ft	1.25	–0.7	–0.9	Bottom of pitch link
35	ft	1.25	–0.7	0	Pitch horn
36	ft	2.00	–0.7	0	Top of pitch link

A rigid bar connects nodes 1 and 2 to model the mass and inertia of the blade root. The center of gravity of this bar is offset 0.625 ft from node 1. The bar has the following mass properties: $m = 0.112\,\text{slug}$ and $I_{xx} = I_{yy} = I_{zz} = I_{xy} = I_{yz} = I_{xz} = 0$.

A z-oriented slide element connects nodes 30 and 31 for collective swashplate control. Nodes 34 and 35 are connected with translational linear spring with a stiffness coefficient of

8.15×10^4 lbf/ft to model control stiffness. A series of hinges allow for blade movement and cyclic swashplate control. These hinges are defined in Table B.17.

Table B.17: Hinges of RCAS model main rotor blade primitive structure.

Elem. ID	Node 1	Node 2	Type	Free?	k (lbf/ft)	c (lbf s/ft)	Description
2	2	3	Lag	Yes	0	4500	Lag hinge
3	3	4	Flap	Yes	2000	0	Flap hinge
4	4	5	Pitch	Yes	0	0	Pitch hinge
41	31	32	Pitch	No	0	0	Control bearing*

* The control bearing is rotated by 60.75° about the z-axis.

A series of 10 geometrically-exact composite beams, connecting nodes 5 to 20, model the inertia and elasticity of the rotor blade. Each of these beams has six Gauss integration points, and has two active degrees-of-freedom in axial, lag, flap, torsion, and both shear directions. The blade has a structural twist of 0.131 rad at the root and −0.044 rad at the tip. The mass and stiffness matrices that define the beam are produced by the VABS model. The baseline values of these matrices are defined in Appendix B.2.3.

B.2.1.3 Tail Rotor Subsystem

The `trotor` subsystem is composed of two primitive structures. `tshaft` models the tail rotor shaft and `tblade1` models a single tail rotor blade. The single blade analysis option is used to duplicate `tblade1` to model a four-bladed rotor. The center of rotation of the rotor is at the root node of `tshaft`. The rotor is defined to have a nominal rotation speed of 108.0 rad/s.

The geometric origins of the three primitive structures are described in Table B.18 and their orientations are defined in

Table B.18: Origins of RCAS model tail rotor subsystem primitive structures.

Primitive	Units	x	y	z
`tshaft`	ft	0	0	0
`tblade1`	ft	0	0	1.072

Table B.19: Orientations of RCAS model tail rotor subsystem primitive structures.

Primitive	Units	First rotation (z)
`tshaft`	deg	0
`tblade1`	deg	–45

The `tblade1` structure is constrained to the `tshaft` structure at the root of the blade.

Tail Rotor Shaft Primitive Structure The nodes comprising the `tshaft` primitive structure are listed in Table B.20.

Table B.20: Nodes of RCAS model tail rotor shaft primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Connection to fixed system
2	ft	0	0	0	
3	ft	0	0	1.072	Connection to blade root

A free z-oriented hinge is placed between nodes 1 and 2 to allow the shaft to spin. A rigid bar connects nodes 2 and 3 to model the mass and inertia of the shaft. The center of gravity of this bar is located at node 3. The bar has the following mass properties: $m = 1.479\,\text{slug}$, $I_{xx} = 1.065\,\text{slug}\,\text{ft}^2$, $I_{yy} = 0.532\,\text{slug}\,\text{ft}^2$, $I_{zz} = 0.532\,\text{slug}\,\text{ft}^2$, and $I_{xy} = I_{yz} = I_{xz} = 0$. Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized.

Tail Rotor Blade Primitive Structure The nodes comprising the `tblade1` primitive structure are listed in Table B.21.

Table B.21: Nodes of RCAS model tail rotor blade primitive structure.

Node ID	Units	x	y	z	Description
1	ft	0	0	0	Blade root
2	ft	1.20	0	0	Control bearing first node
3	ft	1.20	0	0	Control bearing second node
20	ft	6.00	0	0	Blade tip

A massless rigid bar connects nodes 1 and 2. A second rigid bar with mass connects nodes 3 and 20 to model the mass and inertia of the blade. The CG of this bar is offset 2.4 ft from node 3. The bar has the following mass properties: $m = 0.452\,\text{slug}$, $I_{xx} = 0.022\,\text{slug}\,\text{ft}^2$, $I_{yy} = 0.868\,\text{slug}\,\text{ft}^2$, $I_{zz} = 0.868\,\text{slug}\,\text{ft}^2$, and $I_{xy} = I_{yz} = I_{xz} = 0$. Note that these values can be overwritten by OpenMDAO based on mass and inertia values predicted by NDARC when the model is initialized.

A controlled pitch bearing is placed between nodes 2 and 3 to allow for collective control of the tail rotor.

B.2.2 Aerodynamic Model

The aerodynamic model consists of five aerodynamic supercomponents. The `amrotor` and `atrotor` supercomponents model the main and tail rotor systems, respectively. The `fusesc`, `vstabsc`, and `hstabsc` supercomponents model the fuselage, vertical stabilizer, and horizontal stabilizer, respectively.

The geometric origins of the five supercomponents are described in Table B.22 and their orientations are described in Table B.23.

Table B.22: Origins of RCAS aerodynamic model supercomponents.

Subsystem	Units	x	y	z
`amrotor`	ft	0	0	0
`atrotor`	ft	−33.05	0	−7.772
`fusesc`	ft	0	0	0
`vstabsc`	ft	−29.48	0	0
`hstabsc`	ft	−29.48	0	0

No supercomponent-to-supercomponent interference is included in this model.

B.2.2.1 Main Rotor Supercomponent

The `amrotor` aerodynamic supercomponent defines the aerodynamic model used by the main rotor. It consists of a single aerodynamic component, `amblade1`, which is positioned

Table B.23: Orientations of RCAS aerodynamic model supercomponents.

	First rotation		Second rotation	
Subsystem	Axis	Angle (deg)	Axis	Angle (deg)
amrotor	y	177	x	0
atrotor	y	180	x	–90
fusesc	z	0	y	0
vstabsc	z	–90	y	90
hstabsc	z	–90	y	180

to match initial location of the `blade1` primitive structure and constrained to the tip of `blade1`.

The supercomponent is configured to use blade element momentum inflow. Yawed flow effects, tip loss effects, linear unsteady effects, and compressibility effects are enabled. The tip loss factor is 0.97.

Main Rotor Blade Component The `amblade1` aerodynamic component contains the aerodynamic definition of the main rotor blade. The aerodynamic nodes that define this component are defined in Table B.24.

The aerodynamic nodes are connected with aerodynamic segments that describe the airfoil properties at each point. These segments are defined in Table B.25.

The main rotor blade uses a lookup table to model the aerodynamics of a NACA0012 airfoil. This will be described in Appendix B.2.3.

B.2.2.2 Tail Rotor Supercomponent

The `atrotor` aerodynamic supercomponent defines the aerodynamic model used by the tail rotor. It consists of a single aerodynamic component, `atblade1`, which is positioned to match initial location of the `tblade1` primitive structure and constrained to the tip of `tblade1`.

The supercomponent is configured to use blade element momentum inflow. Yawed flow effects, tip loss effects, linear unsteady effects, and compressibility effects are enabled. The

Table B.24: Aerodynamic nodes of RCAS model main rotor blade aerodynamic component.

Node ID	Units	x	y	z	Description
1	ft	4.02	0	0	Root cut-off
2	ft	5.538667	0	0	
3	ft	7.057333	0	0	
4	ft	8.576	0	0	
5	ft	10.094667	0	0	
6	ft	11.613333	0	0	
7	ft	13.132	0	0	
8	ft	14.650667	0	0	
9	ft	16.169333	0	0	
10	ft	17.688	0	0	
11	ft	19.206667	0	0	
12	ft	20.725333	0	0	
13	ft	22.244	0	0	
14	ft	23.762667	0	0	
15	ft	25.281333	0	0	
16	ft	26.8	0	0	Tip

Table B.25: Aerodynamic segments of RCAS model main rotor blade aerodynamic component.

Seg. ID	Node 1	Node 2	Chord (ft)	Twist (rad)	Shear (rad)
1	1	2	1.75	0.0997747	0
2	2	3	1.75	0.0898845	0
3	3	4	1.75	0.0799943	0
4	4	5	1.75	0.0701041	0
5	5	6	1.75	0.0602139	0
6	6	7	1.75	0.0503237	0
7	7	8	1.75	0.0404335	0
8	8	9	1.75	0.0305433	0
9	9	10	1.75	0.0206531	0
10	10	11	1.75	0.0107629	0
11	11	12	1.75	0.0008727	0
12	12	13	1.75	–0.0090175	0
13	13	14	1.75	–0.0189077	0
14	14	15	1.75	–0.0287979	0
15	15	17	1.75	–0.0386881	0

tip loss factor is 0.97.

Tail Rotor Blade Component The `atblade1` aerodynamic component contains the aerodynamic definition of the main rotor blade. The aerodynamic nodes that define this component are defined in Table B.26.

Table B.26: Aerodynamic nodes of RCAS model tail rotor blade aerodynamic component.

Node ID	Units	x	y	z	Description
1	ft	2.4	0	0	Root cut-off
2	ft	3.12	0	0	
3	ft	3.84	0	0	
4	ft	4.56	0	0	
5	ft	5.28	0	0	
6	ft	6	0	0	

The aerodynamic nodes are connected with aerodynamic segments that describe the airfoil properties at each point. These segments are defined in Table B.27.

Table B.27: Aerodynamic segments of RCAS model tail rotor blade aerodynamic component.

Seg. ID	Node 1	Node 2	Chord (ft)	Twist (rad)	Shear (rad)
1	1	2	0.7539822	0	0
2	2	3	0.7539822	0	0
3	3	4	0.7539822	0	0
4	4	5	0.7539822	0	0
5	5	6	0.7539822	0	0

The tail rotor blade uses a lookup table to model the aerodynamics of a NACA0012 airfoil. This will be described in Appendix B.2.3.

B.2.2.3 Fuselage Supercomponent

The `fusesc` aerodynamic supercomponent defines the aerodynamic model used by the fuselage. It consists of a single aerodynamic component, `fusecp`, which is positioned to match initial location of the `fuseps` primitive structure and constrained to the root node of `fuseps`.

Fuselage Component The `fusecp` component represents a lifting body whose aerodynamics are modeled by a series of lift, drag, and moment curves. The aerodynamic coefficients are $\frac{L_0}{q} = 0$, $\frac{L_1}{q} = 78.48$, $\frac{D_0}{q} = 24.0383$, $\frac{D_1}{q} = 0$, $\frac{D_2}{q} = 0$, $\frac{M_0}{q} = 0$, and $\frac{M_1}{q} = 1567.245$. In these equations, L, D, and M are lift, drag, and pitching moment, respectively, while q is the dynamic pressure. The subscripts correspond to the zeroth-, first-, or second-order coefficients of each curve. These parameters were adapted from the NDARC model.

B.2.2.4 Vertical Stabilizer Supercomponent

The `vstabsc` aerodynamic supercomponent defines the aerodynamic model used by the vertical stabilizer. It consists of a single aerodynamic component, `vstabcp`, which is positioned to match initial location of the `vstabps` primitive structure and constrained to the root and tip nodes of `vstabps`. This supercomponent uses the uniform momentum inflow model for half wings.

Vertical Stabilizer Component The `vstabcp` component models the aerodynamic surface of the vertical stabilizer. The aerodynamic nodes match the nodes of the `vstabps` primitive structure (see Table B.12). A single aerodynamic segment connects these nodes with a chord of 3.544117 ft. The vertical stabilizer uses a linear airfoil model with $C_{L_\alpha} = 1.5$, $C_D = 0.02$, and a zero lift angle-of-attack of 0°.

B.2.2.5 Horizontal Stabilizer Supercomponent

The `hstabsc` aerodynamic supercomponent defines the aerodynamic model used by the horizontal stabilizer. It consists of a single aerodynamic component, `hstabcp`, which is positioned to match initial location of the `hstabps` primitive structure and constrained to the port and starboard nodes of `hstabps`. This supercomponent uses the uniform momentum inflow model for full wings.

Horizontal Stabilizer Component The `hstabcp` component models the aerodynamic surface of the horizontal stabilizer. The aerodynamic nodes match the nodes of the `hstabps` primitive structure (see Table B.13). Two aerodynamic segments connect these nodes with a chord of 2.931872 ft. The vertical stabilizer uses a linear airfoil model with $C_{L_\alpha} = 2.3$, $C_D = 0.015$, and a zero lift angle-of-attack of $1°$.

B.2.3 Supporting Files

A number of supporting files are required to complete the RCAS model. The `blade_int_frc.topr` and `blade_int_mom.topr` files define the output files which describe blade forces and moments at each blade station for each time step in the periodic solution. The `BLADE_GCB_PROB.TAB` file describes the inertial and elastic properties of the main rotor blade, although this will be overwritten with results from the VABS model at runtime. Finally, the `NACA0012.C81` file provides a lookup table for the NACA0012 airfoil.

Listing B.5: blade_int_frc.topr

```
***begin-RCAS-file: blade_int_frc.topr ***
oileltdrv
ANALYS   TRIM
JOBCAS   TESTING_00.01
LODTYP   FRC
AXIS     X
AXIS     Y
AXIS     Z
ELMSEL   MROTOR_BLADE1_E5GCB
ELMSEL   MROTOR_BLADE1_E6GCB
ELMSEL   MROTOR_BLADE1_E7GCB
ELMSEL   MROTOR_BLADE1_E8GCB
ELMSEL   MROTOR_BLADE1_E9GCB
ELMSEL   MROTOR_BLADE1_E10GCB
ELMSEL   MROTOR_BLADE1_E11GCB
ELMSEL   MROTOR_BLADE1_E12GCB
ELMSEL   MROTOR_BLADE1_E13GCB
ELMSEL   MROTOR_BLADE1_E14GCB
ELMSEL   MROTOR_BLADE1_TIP
HARMFLG  N
NUMHARM  0
MCFLAG   ELMS
COMPVAR  NONE
XRANGE   0
XRANGE   0
XRANGE   0
YRANGE   0
YRANGE   0
YRANGE   0
```

```
SCALE    1
SCALE    1
LABEL    INTERNAL LOADS
DATADEST BladeIntFrc.dat
PLOTDEST NO
REPTDEST NO
REPSIZ   80
REPSIZ   60
*****end-RCAS-file: blade_int_frc.topr ***
```

Listing B.6: blade_int_mom.topr

```
***begin-RCAS-file: blade_int_mom.topr ***
oileltdrv
ANALYS   TRIM
JOBCAS   TESTING_00.01
LODTYP   MOM
AXIS     X
AXIS     Y
AXIS     Z
ELMSEL   MROTOR_BLADE1_E5GCB
ELMSEL   MROTOR_BLADE1_E6GCB
ELMSEL   MROTOR_BLADE1_E7GCB
ELMSEL   MROTOR_BLADE1_E8GCB
ELMSEL   MROTOR_BLADE1_E9GCB
ELMSEL   MROTOR_BLADE1_E10GCB
ELMSEL   MROTOR_BLADE1_E11GCB
ELMSEL   MROTOR_BLADE1_E12GCB
ELMSEL   MROTOR_BLADE1_E13GCB
ELMSEL   MROTOR_BLADE1_E14GCB
ELMSEL   MROTOR_BLADE1_TIP
HARMFLG  N
NUMHARM  0
MCFLAG   ELMS
COMPVAR  NONE
XRANGE   0
XRANGE   0
XRANGE   0
YRANGE   0
YRANGE   0
YRANGE   0
SCALE    1
SCALE    1
LABEL    INTERNAL LOADS
DATADEST BladeIntMom.dat
PLOTDEST NO
REPTDEST NO
REPSIZ   80
REPSIZ   60
*****end-RCAS-file: blade_int_mom.topr ***
```

Listing B.7: BLADE_GCB_PROP.TAB

```
***begin-RCAS-file: BLADE_GCB_PROP.TAB ***
!M REFLENGTH
 26.8

!M POS1
```

```
0.0

!M POS2
1.0

!M TWIST1
0.131

!M TWIST2
-0.044

!M MMAT1
2.20000000e-01   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00   -3.85000000e-03
0.00000000e+00   2.20000000e-01   0.00000000e+00   -0.00000000e+00   0.00000000e+00   0.00000000e+00
0.00000000e+00   0.00000000e+00   2.20000000e-01   3.85000000e-03   0.00000000e+00   0.00000000e+00
0.00000000e+00   -0.00000000e+00   3.85000000e-03   2.69982400e-02   0.00000000e+00   0.00000000e+00
0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00   6.17784133e-04   -1.37398934e-11
-3.85000000e-03   0.00000000e+00   0.00000000e+00   0.00000000e+00   -1.37398934e-11   2.63804559e-02

!M MMAT2
2.20000000e-01   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00   -3.85000000e-03
0.00000000e+00   2.20000000e-01   0.00000000e+00   -0.00000000e+00   0.00000000e+00   0.00000000e+00
0.00000000e+00   0.00000000e+00   2.20000000e-01   3.85000000e-03   0.00000000e+00   0.00000000e+00
0.00000000e+00   -0.00000000e+00   3.85000000e-03   2.69982400e-02   0.00000000e+00   0.00000000e+00
0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00   6.17784133e-04   -1.37398934e-11
-3.85000000e-03   0.00000000e+00   0.00000000e+00   0.00000000e+00   -1.37398934e-11   2.63804559e-02

!M SSTIF1
4.55369321e+07   1.87537770e-01   -3.46871241e+04   8.66582986e+03   5.33918393e-03   -2.89533006e+06
1.87537770e-01   3.76254571e+06   -1.22630524e+01   4.42762792e+00   -1.74621444e+03   -1.94971694e-02
-3.46871241e+04   -1.22630524e+01   1.94825569e+06   6.61410812e+04   -2.04712555e-02   9.33465719e+02
8.66582986e+03   4.42762792e+00   6.61410812e+04   2.17211720e+05   4.33683356e-03   8.80212979e+02
5.33918393e-03   -1.74621444e+03   -2.04712555e-02   4.33683356e-03   2.66378324e+05   -3.52369028e-02
-2.89533006e+06   -1.94971694e-02   9.33465719e+02   8.80212979e+02   -3.52369028e-02   4.83001616e+06

!M SSTIF2
4.55369321e+07   1.87537770e-01   -3.46871241e+04   8.66582986e+03   5.33918393e-03   -2.89533006e+06
1.87537770e-01   3.76254571e+06   -1.22630524e+01   4.42762792e+00   -1.74621444e+03   -1.94971694e-02
-3.46871241e+04   -1.22630524e+01   1.94825569e+06   6.61410812e+04   -2.04712555e-02   9.33465719e+02
8.66582986e+03   4.42762792e+00   6.61410812e+04   2.17211720e+05   4.33683356e-03   8.80212979e+02
5.33918393e-03   -1.74621444e+03   -2.04712555e-02   4.33683356e-03   2.66378324e+05
```

```
    ↪   -3.52369028e-02
-2.89533006e+06   -1.94971694e-02   9.33465719e+02   8.80212979e+02   -3.52369028e-02
    ↪    4.83001616e+06

!M ASTIF1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M ASTIF2
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M BSTIF1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M BSTIF2
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M CSTIF1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M CSTIF2
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M DSTIF1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

!M DSTIF2
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
*****end-RCAS-file: BLADE_GCB_PROP.TAB ***
```

Listing B.8: NACA0012.C81

```
***begin-RCAS-file: NACA0012.C81 ***
      0012                    11391165 947
       0.     .20    .30    .40    .50    .60    .7     .75    .8
       .9     1.
-180.  0.     0.     0.     0.     0.     0.     0.     0.     0.
```

```
        0.     0.
-172.5 .78    .78    .78    .78    .78    .78    .78    .78    .78
        .78    .78
-161.  .62    .62    .62    .62    .62    .62    .62    .62    .62
        .62    .62
-147.  1.     1.     1.     1.     1.     1.     1.     1.     1.
        1.     1.
-129.  1.     1.     1.     1.     1.     1.     1.     1.     1.
        1.     1.
-49.   -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18
        -1.18  -1.18
-39.   -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18  -1.18
        -1.18  -1.18
-21.   -.8    -.8    -.81   -.83   -.85   -.85   -.85   -.71   -.68
        -.64   -.64
-16.5  -1.007 -1.007 -.944  -.96   -.965  -.965  -.965  -.795  -.76
        -.7    -.7
-15.   -1.19  -1.19  -1.09  -1.055 -.99   -.98   -.98   -.83   -.79
        -.72   -.72
-14.   -1.333 -1.333 -1.22  -1.096 -1.    -.97   -.97   -.84   -.805
        -.73   -.73
-13.   -1.334 -1.334 -1.28  -1.12  -1.    -.96   -.96   -.85   -.815
        -.735  -.735
-12.   -1.255 -1.255 -1.26  -1.13  -1.    -.947  -.94   -.85   -.82
        -.74   -.74
-11.    -1.161-1.161 -1.19  -1.12  -.994  -.93   -.923  -.85   -.81
        -.74   -.74
-10.   -1.055 -1.055 -1.01  -1.082 -.985  -.91   -.90   -.845  -.805
        -.73   -.73
-8.    -.844  -.844  -.88   -.907  -.922  -.87   -.84   -.82   -.77
        -.695  -.695
-6.    -.633  -.633  -.66   -.684  -.741  -.77   -.75   -.77   -.72
        -.593  -.593
-4.    -.422  -.422  -.440  -.456  -.494  -.544  -.578  -.627  -.603
        -.396  -.396
-2.    -.211  -.211  -.22   -.228  -.247  -.272  -.313  -.350  -.395
        -.2    -.2
0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
        0.     0.
2.     .211   .211   .22    .228   .247   .272   .313   .350   .395
        .2     .2
4.     .422   .422   .44    .456   .494   .544   .578   .627   .603
        .396   .396
6.     .633   .633   .66    .684   .741   .77    .75    .77    .72
        .593   .593
8.     .844   .844   .88    .907   .922   .87    .84    .82    .77
        .695   .695
10.    1.055  1.055  1.1    1.082  .985   .91    .90    .845   .805
        .73    .73
11.    1.161  1.161  1.19   1.12   .994   .93    .923   .850   .810
        .74    .74
12.    1.255  1.255  1.26   1.13   1.     .947   .94    .85    .82
        .74    .74
13.    1.334  1.334  1.28   1.12   1.     .96    .96    .85    .815
        .735   .735
14.    1.333  1.333  1.22   1.096  1.     .97    .97    .84    .805
        .73    .73
15.    1.19   1.19   1.09   1.055  .99    .98    .98    .83    .79
        .73    .73
16.5   1.007  1.007  .944   .96    .965   .965   .965   .795   .76
```

```
|        .7     .7
|21.     .8     .8     .81    .83    .85    .85    .85    .71    .68
|        .64    .64
|39.     1.18   1.18   1.18   1.18   1.18   1.18   1.18   1.18   1.18
|        1.18   1.18
|49.     1.18   1.18   1.18   1.18   1.18   1.18   1.18   1.18   1.18
|        1.18   1.18
|129.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.
|        -1.    -1.
|147.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.
|        -1.    -1.
|161.    -.62   -.62   -.62   -.62   -.62   -.62   -.62   -.62   -.62
|        -.62   -.62
|172.5   -.78   -.78   -.78   -.78   -.78   -.78   -.78   -.78   -.78
|        -.78   -.78
|180.    0.     0.     0.     0.     0.     0.     0.     0.     0.
|        0.     0.
|        0.     .18    .28    .38    .48    .62    .72    .77    .82
|        .92    1.0
|-180.   .022   .022   .022   .022   .022   .022   .022   .022   .022
|        .022   .022
|-175.   .062   .062   .062   .062   .062   .062   .062   .062   .062
|        .062   .062
|-170.   .132   .132   .132   .132   .132   .132   .132   .132   .132
|        .132   .132
|-165.   .242   .242   .242   .242   .242   .242   .242   .242   .242
|        .242   .242
|-160.   .302   .302   .302   .302   .302   .302   .302   .302   .302
|        .302   .302
|-140.   1.042  1.042  1.042  1.042  1.042  1.042  1.042  1.042  1.042
|        1.042  1.042
|-120.   1.652  1.652  1.652  1.652  1.652  1.652  1.652  1.652  1.652
|        1.652  1.652
|-110.   1.852  1.852  1.852  1.852  1.852  1.852  1.852  1.852  1.852
|        1.852  1.852
|-100.   2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022
|        2.022  2.022
|-90.    2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022
|        2.022  2.022
|-80.    1.962  1.962  1.962  1.962  1.962  1.962  1.962  1.962  1.962
|        1.962  1.962
|-70.    1.842  1.842  1.842  1.842  1.842  1.842  1.842  1.842  1.842
|        1.842  1.842
|-60.    1.662  1.662  1.662  1.662  1.662  1.662  1.662  1.662  1.662
|        1.662  1.662
|-50.    1.392  1.392  1.392  1.392  1.392  1.399  1.392  1.392  1.392
|        1.392  1.392
|-30.    .562   .562   .562   .562   .562   .562   .562   .562   .562
|        .562   .562
|-21.    .332   .332   .332   .332   .332   .332   .332   .332   .332
|        .332   .332
|-16.    .155   .155   .181   .207   .235   .257   .274   .292   .305
|        .342   .342
|-15.    .102   .102   .148   .181   .209   .233   .252   .271   .282
|        .298   .298
|-14.    .038   .038   .099   .146   .180   .212   .233   .249   .260
|        .293   .293
|-13.    .0264  .0264  .0455  .094   .148   .191   .216   .231   .239
|        .272   .292
|-12.    .022   .022   .030   .06    .111   .164   .198   .211   .220
```

```
|        .252   .291
|-11.    .0196  .0196  .0232  .038   .078   .135   .17    .192   .202
|        .232   .275
|-10.    .0174  .0174  .0189  .0259  .053   .105   .145   .176   .186
|        .213   .254
|-9.     .0154  .0154  .0159  .0187  .0351  .077   .122   .159   .172
|        .199   .232
|-8.     .0138  .0138  .0138  .0147  .0220  .053   .101   .140   .155
|        .183   .214
|-7.     .0122  .0122  .0122  .0123  .0141  .035   .082   .111   .139
|        .169   .192
|-6.     .011   .011   .011   .011   .011   .0212  .0615  .082   .12
|        .14    .17
|-5.     .01    .01    .01    .01    .01    .0132  .038   .054   .084
|        .111   .14
|-4.     .0093  .0093  .0093  .0093  .0093  .01    .0167  .03    .0575
|        .095   .112
|-3.     .0088  .0088  .0088  .0088  .0088  .009   .0102  .0175  .0355
|        .086   .102
|-2.     .0085  .0085  .0085  .0085  .0085  .0085  .0086  .0117  .0240
|        .081   .098
|-1.     .0083  .0083  .0083  .0083  .0083  .0083  .0083  .0091  .0175
|        .078   .096
|0.      .008   .008   .008   .008   .008   .008   .008   .008   .0137
|        .078   .095
|1.      .0083  .0083  .0083  .0083  .0083  .0083  .0083  .0091  .0175
|        .078   .096
|2.      .0085  .0085  .0085  .0085  .0085  .0085  .0086  .0117  .024
|        .081   .098
|3.      .0088  .0088  .0088  .0088  .0088  .0090  .0102  .0175  .0355
|        .086   .102
|4.      .0093  .0093  .0093  .0093  .0093  .01    .0167  .03    .0575
|        .095   .112
|5.      .01    .01    .01    .01    .01    .0132  .038   .054   .084
|        .111   .14
|6.      .011   .011   .011   .011   .011   .0212  .0615  .082   .12
|        .14    .17
|7.      .0122  .0122  .0122  .0123  .0141  .035   .082   .111   .139
|        .169   .192
|8.      .0138  .0138  .0138  .0147  .022   .053   .101   .14    .155
|        .183   .214
|9.      .0154  .0154  .0159  .0187  .0351  .077   .122   .159   .172
|        .199   .232
|10.     .0174  .0174  .0189  .0259  .053   .105   .145   .176   .186
|        .213   .254
|11.     .0196  .0196  .0232  .038   .078   .135   .17    .192   .202
|        .232   .275
|12.     .022   .022   .03    .06    .111   .164   .198   .211   .22
|        .252   .291
|13.     .0264  .0264  .0455  .094   .148   .191   .216   .231   .239
|        .272   .292
|14.     .038   .038   .099   .146   .18    .212   .233   .249   .26
|        .293   .293
|15.     .102   .102   .148   .181   .209   .233   .252   .271   .282
|        .298   .298
|16.     .155   .155   .181   .207   .235   .257   .274   .292    .305
|        .342   .342
|21.     .332   .332   .332   .332   .332   .332   .332   .332   .332
|        .332   .332
|30.     .562   .562   .562   .562   .562   .562   .562   .562   .562
```

```
|        .562   .562
|50.     1.392  1.392  1.392  1.392  1.392  1.392  1.392  1.392  1.392
|        1.392  1.392
|60.     1.662  1.662  1.662  1.662  1.662  1.662  1.662  1.662  1.662
|        1.662  1.662
|70.     1.842  1.842  1.842  1.842  1.842  1.842  1.842  1.842  1.842
|        1.842  1.842
|80.     1.962  1.962  1.962  1.962  1.962  1.962  1.962  1.962  1.962
|        1.962  1.962
|90.     2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022
|        2.022  2.022
|100.    2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022  2.022
|        2.022  2.022
|110.    1.852  1.852  1.852  1.852  1.852  1.852  1.852  1.852  1.852
|        1.852  1.852
|120.    1.652  1.652  1.652  1.652  1.652  1.652  1.652  1.652  1.652
|        1.652  1.652
|140.    1.042  1.042  1.042  1.042  1.042  1.042  1.042  1.042  1.042
|        1.042  1.042
|160.    .302   .302   .302   .302   .302   .302   .302   .302   .302
|        .302   .302
|165.    .242   .242   .242   .242   .242   .242   .242   .242   .242
|        .242   .242
|170.    .132   .132   .132   .132   .132   .132   .132   .132   .132
|        .132   .132
|175.    .062   .062   .062   .062   .062   .062   .062   .062   .062
|        .062   .062
|180.    .022   .022   .022   .022   .022   .022   .022   .022   .022
|        .022   .022
|        .20    .30    .40    .50    .6     .7     .75    .8     .9
|-180.   0.     0.     0.     0.     0.     0.     0.     0.     0.
|-170.   .4     .4     .4     .4     .4     .4     .4     .4     .4
|-165.   .3     .3     .3     .3     .3     .3     .3     .3     .3
|-160.   .3     .3     .3     .3     .3     .3     .3     .3     .3
|-135.   .5     .5     .5     .5     .5     .5     .5     .5     .5
|-90.    .5     .5     .5     .5     .5     .5     .5     .5     .5
|-30.    .174   .184   .196   .214   .235   .25    .264   .277   .298
|-23.    .112   .118   .128   .144   .157   .171   .183   .206   .232
|-16.    .073   .078   .086   .097   .108   .117   .137   .176   .200
|-15.    .054   .065   .073   .084   .097   .111   .133   .173   .195
|-14.    0.     .027   .054   .068   .086   .103   .127   .167   .189
|-13.    0.     .0015  .025   .05    .074   .093   .122   .163   .184
|-12.    0.     0.     .002   .03    .06    .083   .116   .157   .176
|-11.    0.     0.     -.003  .014   .046   .074   .108   .149   .17
|-10.    0.     0.     -.0015 .002   .032   .065   .10    .142   .163
|-9.     0.     0.     0.     -.003  .016   .054   .089   .132   .154
|-8.     0.     0.     0.     -.004  .005   .041   .082   .123   .145
|-7.     0.     0.     0.     0.     -.004  .0275  .072   .1125  .136
|-6.     0.     0.     0.     0.     -.003  .016   .0625  .10    .125
|-4.     0.     0.     0.     0.     0.     .005   .04    .076   .102
|-3.     0.     0.     0.     0.     0.     -.0025 .026   .0665  .087
|-2.     0.     0.     0.     0.     0.     0.     .013   .053   .07
|-1.     0.     0.     0.     0.     0.     0.     .0035  .033   .045
|0.      0.     0.     0.     0.     0.     0.     0.     0.     0.
|1.      0.     0.     0.     0.     0.     0.     -.0035 -.033  -.045
|2.      0.     0.     0.     0.     0.     0.     -.013  -.053  -.07
|3.      0.     0.     0.     0.     0.     .0025  -.026  -.0665 -.087
|4.      0.     0.     0.     0.     0.     -.005  -.04   -.076  -.102
|6.      0.     0.     0.     0.     .003   -.016  -.0625 -.1    -.125
|7.      0.     0.     0.     0.     .004   -.0275 -.072  -.1125 -.136
```

```
8.     0.     0.     0.     .004   -.005  -.041  -.082  -.123   -.145
9.     0.     0.     0.     .003   -.016  -.054  -.089  -.132   -.154
10.    0.     0.     .0015  -.002  -.032  -.065  -.1    -.142   -.163
11.    0.     0.     .003   -.014  -.046  -.074  -.108  -.149   -.17
12.    0.     0.     -.002  -.03   -.06   -.083  -.116  -.157   -.176
13.    0.     -.0015 -.025  -.05   -.074  -.093  -.122  -.163   -.184
14.    0.     -.027  -.054  -.068  -.086  -.103  -.127  -.167   -.189
15.    -.054  -.065  -.073  -.084  -.097  -.111  -.133  -.173  -.195
16.    -.073  -.078  -.086  -.097  -.108  -.117  -.137  -.176  -.20
23.    -.112  -.118  -.128  -.144  -.157  -.171  -.183  -.206  -.232
30.    -.174  -.184  -.196  -.214  -.235  -.250  -.264  -.277  -.298
90.    -.5    -.5    -.5    -.5    -.5    -.5    -.5    -.5    -.5
135.   -.5    -.5    -.5    -.5    -.5    -.5    -.5    -.5    -.5
160.   -.3    -.3    -.3    -.3    -.3    -.3    -.3    -.3    -.3
165.   -.3    -.3    -.3    -.3    -.3    -.3    -.3    -.3    -.3
170.   -.4    -.4    -.4    -.4    -.4    -.4    -.4    -.4    -.4
180.   0.     0.     0.     0.     0.     0.     0.     0.     0.

NACA0012 for Puma
from Bousman, US Army, October 1988

corrected cd typo at alpha=-10; October 1988

*****end-RCAS-file: NACA0012.C81 ***
```

B.2.4 OpenMDAO–RCAS Variable Mapping

Tables B.28 to B.31 describe the mapping between the variables in this file and the OpenMDAO variables. The RCAS access string syntax is described in the notes to Table B.28. The OpenMDAO variables are further described in Appendix C.

Table B.28: Mapping from OpenMDAO variables to RCAS variables (flight condition).

OpenMDAO variable	RCAS access string[a]	Units
airspeed	INITCOND%3%1%1	ft/s
neg_rate_of_climb[b]	INITCOND%3%1%3	ft/s
att_pitch	INITCOND%2%1%5	rad
att_roll	INITCOND%2%1%4	rad
rcas_cont_coll	INITCOND%1%1%1	deg
rcas_cont_lat_cyc	INITCOND%1%1%2	rad
rcas_cont_lon_cyc	INITCOND%1%1%3	rad
rcas_cont_pedal	INITCOND%1%1%4	rad
turn_radius	INITCOND%5%1%1	ft
turn_rate	INITCOND%5%1%2	rad/s
air_density	AEROSTATCONST%1%1%4	slug/ft^3
air_sound_speed	AEROSTATCONST%1%1%5	ft/s
air_viscosity	AEROSTATCONST%1%1%6	slug/(ft s)
neg_loc_payload_sl[c]	FUSEPS-FENODE%1%4%2	ft
azimuths[d]	tab-BladeIntFrc001%1	deg
blade_int_force_x	tab-BladeIntFrc001%2:12	lbf
blade_int_force_y	tab-BladeIntFrc002%2:12	lbf
blade_int_force_z	tab-BladeIntFrc003%2:12	lbf
blade_int_moment_x	tab-BladeIntMom001%2:12	ft lbf
blade_int_moment_y	tab-BladeIntMom002%2:12	ft lbf
blade_int_moment_z	tab-BladeIntMom003%2:12	ft lbf

[a] The RCAS access string identifies a specific point in the RCAS input file. For most inputs, this takes the form <SCREEN>%<page>%<line>%<column. If the screen is non-unique, then its "parent" screen is appended to the beginning of the screen name and separated with a hyphen. For supplementary files, such as the mass and stiffness matrices, the access string is of the form <filename>%<element>. For tabular output files, the access string is of the form tab-<filename>%<first_col>:<last_col>.

[b] Because RCAS uses a z-down reference frame, a negative V_z implies a positive rate of climb.

[c] Because RCAS uses a x-forward reference frame, a negative CG shift implies an aftward movement of the CG.

[d] Provides a list of all azimuth angles in the periodic solution.

Table B.29: Mapping from OpenMDAO variables to RCAS variables (fuselage mass and inertia).

OpenMDAO variable	RCAS access string	Units
mass_payload	FUSEPS-RIGIDBODYMASS%1%6%3	slug
inert_payload_Ixx	FUSEPS-RIGIDBODYMASS%1%6%4	slug ft^2
inert_payload_Ixy	FUSEPS-RIGIDBODYMASS%2%6%2	slug ft^2
inert_payload_Ixz	FUSEPS-RIGIDBODYMASS%2%6%3	slug ft^2
inert_payload_Iyy	FUSEPS-RIGIDBODYMASS%1%6%5	slug ft^2
inert_payload_Iyz	FUSEPS-RIGIDBODYMASS%2%6%4	slug ft^2
inert_payload_Izz	FUSEPS-RIGIDBODYMASS%1%6%6	slug ft^2
mass_fuel	FUSEPS-RIGIDBODYMASS%1%5%3	slug
inert_fuel_Ixx	FUSEPS-RIGIDBODYMASS%1%5%4	slug ft^2
inert_fuel_Ixy	FUSEPS-RIGIDBODYMASS%2%5%2	slug ft^2
inert_fuel_Ixz	FUSEPS-RIGIDBODYMASS%2%5%3	slug ft^2
inert_fuel_Iyy	FUSEPS-RIGIDBODYMASS%1%5%5	slug ft^2
inert_fuel_Iyz	FUSEPS-RIGIDBODYMASS%2%5%4	slug ft^2
inert_fuel_Izz	FUSEPS-RIGIDBODYMASS%1%5%6	slug ft^2
mass_fuselage	FUSEPS-RIGIDBODYMASS%1%1%3	slug
inert_fuselage_Ixx	FUSEPS-RIGIDBODYMASS%1%1%4	slug ft^2
inert_fuselage_Ixy	FUSEPS-RIGIDBODYMASS%2%1%2	slug ft^2
inert_fuselage_Ixz	FUSEPS-RIGIDBODYMASS%2%1%3	slug ft^2
inert_fuselage_Iyy	FUSEPS-RIGIDBODYMASS%1%1%5	slug ft^2
inert_fuselage_Iyz	FUSEPS-RIGIDBODYMASS%2%1%4	slug ft^2
inert_fuselage_Izz	FUSEPS-RIGIDBODYMASS%1%1%6	slug ft^2
mass_usefulload	FUSEPS-RIGIDBODYMASS%1%2%3	slug
inert_usefulload_Ixx	FUSEPS-RIGIDBODYMASS%1%2%4	slug ft^2
inert_usefulload_Ixy	FUSEPS-RIGIDBODYMASS%2%2%2	slug ft^2
inert_usefulload_Ixz	FUSEPS-RIGIDBODYMASS%2%2%3	slug ft^2
inert_usefulload_Iyy	FUSEPS-RIGIDBODYMASS%1%2%5	slug ft^2
inert_usefulload_Iyz	FUSEPS-RIGIDBODYMASS%2%2%4	slug ft^2
inert_usefulload_Izz	FUSEPS-RIGIDBODYMASS%1%2%6	slug ft^2
mass_gear	FUSEPS-RIGIDBODYMASS%1%3%3	slug
inert_gear_Ixx	FUSEPS-RIGIDBODYMASS%1%3%4	slug ft^2
inert_gear_Ixy	FUSEPS-RIGIDBODYMASS%2%3%2	slug ft^2
inert_gear_Ixz	FUSEPS-RIGIDBODYMASS%2%3%3	slug ft^2
inert_gear_Iyy	FUSEPS-RIGIDBODYMASS%1%3%5	slug ft^2
inert_gear_Iyz	FUSEPS-RIGIDBODYMASS%2%3%4	slug ft^2
inert_gear_Izz	FUSEPS-RIGIDBODYMASS%1%3%6	slug ft^2

Table B.30: Mapping from OpenMDAO variables to RCAS variables (rotor mass and inertia).

OpenMDAO variable	RCAS access string	Units
mass_matrix_1[a]	BLADE_GCB_PROP.TAB%MMAT1	—
mass_matrix_2[a]	BLADE_GCB_PROP.TAB%MMAT2	—
stiff_matrix_1[a]	BLADE_GCB_PROP.TAB%SSTIF1	—
stiff_matrix_2[a]	BLADE_GCB_PROP.TAB%SSTIF2	—
mass_main_hub	MSHAFT-RIGIDBAR%2%1%2	slug
inert_main_hub_Ixx	MSHAFT-RIGIDBAR%2%1%3	slug ft^2
inert_main_hub_Ixy	MSHAFT-RIGIDBAR%2%1%4	slug ft^2
inert_main_hub_Ixz	MSHAFT-RIGIDBAR%2%1%5	slug ft^2
inert_main_hub_Iyy	MSHAFT-RIGIDBAR%2%1%6	slug ft^2
inert_main_hub_Iyz	MSHAFT-RIGIDBAR%2%1%7	slug ft^2
inert_main_hub_Izz	MSHAFT-RIGIDBAR%2%1%8	slug ft^2
mass_tail_blade	TBLADE1-RIGIDBAR%2%1%2	slug
inert_tail_blade_Ixx	TBLADE1-RIGIDBAR%2%1%3	slug ft^2
inert_tail_blade_Ixy	TBLADE1-RIGIDBAR%2%1%4	slug ft^2
inert_tail_blade_Ixz	TBLADE1-RIGIDBAR%2%1%5	slug ft^2
inert_tail_blade_Iyy	TBLADE1-RIGIDBAR%2%1%6	slug ft^2
inert_tail_blade_Iyz	TBLADE1-RIGIDBAR%2%1%7	slug ft^2
inert_tail_blade_Izz	TBLADE1-RIGIDBAR%2%1%8	slug ft^2
mass_tail_hub	TSHAFT-RIGIDBAR%2%1%2	slug
inert_tail_hub_Ixx	TSHAFT-RIGIDBAR%2%1%3	slug ft^2
inert_tail_hub_Ixy	TSHAFT-RIGIDBAR%2%1%4	slug ft^2
inert_tail_hub_Ixz	TSHAFT-RIGIDBAR%2%1%5	slug ft^2
inert_tail_hub_Iyy	TSHAFT-RIGIDBAR%2%1%6	slug ft^2
inert_tail_hub_Iyz	TSHAFT-RIGIDBAR%2%1%7	slug ft^2
inert_tail_hub_Izz	TSHAFT-RIGIDBAR%2%1%8	slug ft^2
tail_rotor_cant_struct	SSORIENT%1%3%5	deg
tail_rotor_cant_aero	SCORIENT%1%2%5	deg

[a] The main rotor blade uses the GECB formulation of the NLB element. Its mass and stiffness are defined by the generalized Timoshenko mass and stiffness matrices from VABS.

Table B.31: Mapping from OpenMDAO variables to RCAS variables (other mass and inertia).

OpenMDAO variable	RCAS access string	Units
mass_horizstab	HSTABPS-RIGIDBAR%2%1%2	slug
inert_horizstab_Ixx	HSTABPS-RIGIDBAR%2%1%3	slug ft^2
inert_horizstab_Ixy	HSTABPS-RIGIDBAR%2%1%4	slug ft^2
inert_horizstab_Ixz	HSTABPS-RIGIDBAR%2%1%5	slug ft^2
inert_horizstab_Iyy	HSTABPS-RIGIDBAR%2%1%6	slug ft^2
inert_horizstab_Iyz	HSTABPS-RIGIDBAR%2%1%7	slug ft^2
inert_horizstab_Izz	HSTABPS-RIGIDBAR%2%1%8	slug ft^2
mass_vertstab	VSTABPS-RIGIDBAR%2%1%2	slug
inert_vertstab_Ixx	VSTABPS-RIGIDBAR%2%1%3	slug ft^2
inert_vertstab_Ixy	VSTABPS-RIGIDBAR%2%1%4	slug ft^2
inert_vertstab_Ixz	VSTABPS-RIGIDBAR%2%1%5	slug ft^2
inert_vertstab_Iyy	VSTABPS-RIGIDBAR%2%1%6	slug ft^2
inert_vertstab_Iyz	VSTABPS-RIGIDBAR%2%1%7	slug ft^2
inert_vertstab_Izz	VSTABPS-RIGIDBAR%2%1%8	slug ft^2
mass_engine	FUSEPS-RIGIDBODYMASS%1%4%3	slug
inert_engine_Ixx	FUSEPS-RIGIDBODYMASS%1%4%4	slug ft^2
inert_engine_Ixy	FUSEPS-RIGIDBODYMASS%2%4%2	slug ft^2
inert_engine_Ixz	FUSEPS-RIGIDBODYMASS%2%4%3	slug ft^2
inert_engine_Iyy	FUSEPS-RIGIDBODYMASS%1%4%5	slug ft^2
inert_engine_Iyz	FUSEPS-RIGIDBODYMASS%2%4%4	slug ft^2
inert_engine_Izz	FUSEPS-RIGIDBODYMASS%1%4%6	slug ft^2

B.3 PreVABS+VABS Model

The PreVABS+VABS model consists of three different files. The primary file defines the cross section geometry, layups, and structural components. This file is titled `naca0012_comb.xml`. The second file is a list of material and lamina definitions which must be inserted into the `MaterialDB.xml` file in the PreVABS root directory. The third is a list of basepoints, which define the NACA0012 outer mold line. This file is titled `basepoints_rev.dat`.

Table B.32 describes the mapping between the variables in the PreVABS+VABS model and the OpenMDAO variables. The PreVABS access string syntax is described in the notes to Table B.32. The OpenMDAO variables are further described in Appendix C.

Listing B.9: naca0012_comb.xml

```
<cross_section name="naca0012" format="1">
  <include>
    <!-- <baseline>baselines_rev</baseline> -->
      <!-- <material>materials</material> -->
    <!-- <layup>layups_fixed</layup> -->
  </include>
  <analysis>
    <model>1</model>
  </analysis>
  <general>
    <translate>0.25  0</translate>
    <scale>1.75</scale>
    <mesh_size>0.010</mesh_size>
    <element_type>linear</element_type>
  </general>
  <recover>
    <displacements>0 0 0</displacements>
    <rotations>1 0 0 0 1 0 0 0 1</rotations>
    <forces>0 0 0</forces>
    <moments>0 0 0</moments>
    <distributed>
      <forces>0 0 0</forces>
      <forces_d1>0 0 0</forces_d1>
      <forces_d2>0 0 0</forces_d2>
      <forces_d3>0 0 0</forces_d3>
      <moments>0 0 0</moments>
      <moments_d1>0 0 0</moments_d1>
      <moments_d2>0 0 0</moments_d2>
      <moments_d3>0 0 0</moments_d3>
    </distributed>
  </recover>
  <!--   baselines file -->
  <baselines>
    <basepoints>
      <include>basepoints_rev</include>
    </basepoints>
```

```
    <baseline name="bl_lps_for" type="straight">
      <points>le:law</points>
    </baseline>
    <baseline name="bl_lps_spar" type="straight">
      <points>law:lmw</points>
    </baseline>
    <baseline name="bl_lps_aft" type="straight">
      <points>lmw:lte</points>
    </baseline>
    <baseline name="bl_te" type="straight">
      <points>lte,te,hte</points>
    </baseline>
    <baseline name="bl_hps_aft" type="straight">
      <points>hte:hmw</points>
    </baseline>
    <baseline name="bl_hps_spar" type="straight">
      <points>hmw:haw</points>
    </baseline>
    <baseline name="bl_hps_for" type="straight">
      <points>haw:122,le</points>
    </baseline>
    <baseline name="bl_lps_ero" type="straight">
      <points>le:ler</points>
    </baseline>
    <baseline name="bl_hps_ero" type="straight">
      <points>her:122,le</points>
    </baseline>
    <baseline name="bl_aux_web" type="straight">
      <point>aw</point>
    <angle>90</angle>
    </baseline>
    <baseline name="bl_main_web" type="straight">
      <point>mw</point>
    <angle>90</angle>
    </baseline>
    <baseline name="bl_tef" type="straight">
      <point>tef</point>
    <angle>90</angle>
    </baseline>
  </baselines>
  <!--   layups file -->
  <layups>
    <layup name="layup_spar">
      <layer lamina="la_im7_410">0</layer>
      <layer lamina="la_im7_372">45</layer>
      <layer lamina="la_im7_372">-45</layer>
      <layer lamina="la_im7_100">90</layer>
    </layup>
    <layup name="layup_ovw">
      <layer lamina="la_eglass_050">0</layer>
      <layer lamina="la_eglass_050">45</layer>
    <layer lamina="la_eglass_050">-45</layer>
      <layer lamina="la_eglass_050">0</layer>
    </layup>
    <layup name="layup_spar_with_ovw">
      <layer lamina="la_eglass_050">0</layer>
      <layer lamina="la_eglass_050">45</layer>
      <layer lamina="la_eglass_050">-45</layer>
      <layer lamina="la_eglass_050">0</layer>
      <layer lamina="la_im7_410">0</layer>
```

```
      <layer lamina="la_im7_372">45</layer>
    <layer lamina="la_im7_372">-45</layer>
      <layer lamina="la_im7_100">90</layer>
    </layup>
    <layup name="layup_erosion_strip">
      <layer lamina="la_steel_164">0</layer>
    </layup>
    <layup name="layup_le_fill">
      <layer lamina="la_im7_782">0</layer>
    </layup>
    <layup name="layup_le_fill_with_ovw">
      <layer lamina="la_eglass_050">0</layer>
      <layer lamina="la_eglass_050">45</layer>
    <layer lamina="la_eglass_050">-45</layer>
      <layer lamina="la_eglass_050">0</layer>
      <layer lamina="la_im7_782">0</layer>
    </layup>
  </layups>
  <!--   components -->
  <!--   airfoil surface except erosion strip -->
  <component name="surface">
    <segment name="sg_lps_for">
      <baseline>bl_lps_for</baseline>
      <layup direction="left">layup_le_fill_with_ovw</layup>
    </segment>
    <segment name="sg_hps_for">
      <baseline>bl_hps_for</baseline>
      <layup direction="left">layup_le_fill_with_ovw</layup>
    </segment>
    <segment name="sg_lps_spar">
      <baseline>bl_lps_spar</baseline>
      <layup direction="left">layup_spar_with_ovw</layup>
    </segment>
    <segment name="sg_hps_spar">
      <baseline>bl_hps_spar</baseline>
      <layup direction="left">layup_spar_with_ovw</layup>
    </segment>
    <segment name="sg_lps_aft">
      <baseline>bl_lps_aft</baseline>
      <layup direction="left">layup_ovw</layup>
    </segment>
    <segment name="sg_te">
      <baseline>bl_te</baseline>
      <layup direction="left">layup_ovw</layup>
    </segment>
    <segment name="sg_hps_aft">
      <baseline>bl_hps_aft</baseline>
      <layup direction="left">layup_ovw</layup>
    </segment>
  </component>
  <!--   erosion strip -->
  <component name="erosion_strip">
    <segment name="sg_lps_erosion">
      <baseline>bl_lps_ero</baseline>
      <layup direction="right">layup_erosion_strip</layup>
    </segment>
    <segment name="sg_hps_erosion">
      <baseline>bl_hps_ero</baseline>
      <layup direction="right">layup_erosion_strip</layup>
    </segment>
```

```
  </component>
  <!--   auxiliary (leading) web -->
  <component name="aux_web" depend="surface">
    <segment name="sg_aux_web">
      <baseline>bl_aux_web</baseline>
      <layup direction="left">layup_spar</layup>
    </segment>
  </component>
  <!--   main (trailing) web -->
  <component name="main_web" depend="surface">
    <segment name="sg_main_web">
      <baseline>bl_main_web</baseline>
      <layup direction="right">layup_spar</layup>
    </segment>
  </component>
  <!--   sglass fill -->
  <component name="sglass_fill" type="fill" depend="surface">
    <baseline fillside="left">bl_tef</baseline>
    <material>sglass</material>
  </component>
  <!--   plascore fill -->
  <component name="plascore_fill" type="fill" depend="surface,main_web,sglass_fill">
    <location>pf</location>
    <material>plascore</material>
  </component>
</cross_section>
```

Listing B.10: Excerpt from MaterialDB.xml

```
<materials>
  <material name="eglass" type="orthotropic">
    <!-- E-Glass (slug-ft-lbf) -->
    <density>3.34</density>
    <elastic>
      <e1>4.3229e+08</e1>
      <e2>4.3229e+08</e2>
      <e3>4.3229e+08</e3>
      <g12>8.5536e+07</g12>
      <g13>8.5536e+07</g13>
      <g23>8.5536e+07</g23>
      <nu12>0.15</nu12>
      <nu13>0.15</nu13>
      <nu23>0.30</nu23>
    </elastic>
  </material>
  <material name="im7" type="orthotropic">
    <!-- IM7 (slug-ft-lbf) -->
    <density>3.01</density>
    <elastic>
      <e1>3.4464e+09</e1>
      <e2>1.8374e+08</e2>
      <e3>1.8374e+08</e3>
      <g12>1.0224e+08</g12>
      <g13>1.0224e+08</g13>
      <g23>1.0224e+08</g23>
      <nu12>0.34</nu12>
      <nu13>0.34</nu13>
      <nu23>0.30</nu23>
    </elastic>
  </material>
```

```
  <material name="steel" type="orthotropic">
    <!-- Steel (slug-ft-lbf) -->
    <density>15.13</density>
    <elastic>
      <e1>4.2820e+09</e1>
      <e2>4.2820e+09</e2>
      <e3>4.2820e+09</e3>
      <g12>1.6083e+09</g12>
      <g13>1.6083e+09</g13>
      <g23>1.6083e+09</g23>
      <nu12>0.30</nu12>
      <nu13>0.30</nu13>
      <nu23>0.30</nu23>
    </elastic>
  </material>
  <material name="sglass" type="orthotropic">
    <!-- S-Glass (slug-ft-lbf) -->
    <density>3.61</density>
    <elastic>
      <e1>9.0648e+08</e1>
      <e2>2.5056e+08</e2>
      <e3>2.5056e+08</e3>
      <g12>7.5168e+07</g12>
      <g13>7.5168e+07</g13>
      <g23>7.5168e+07</g23>
      <nu12>0.28</nu12>
      <nu13>0.28</nu13>
      <nu23>0.30</nu23>
    </elastic>
  </material>
  <material name="plascore" type="orthotropic">
    <!-- Plascore (slug-ft-lbf) -->
    <density>0.09</density>
    <elastic>
      <e1>1.4400e+05</e1>
      <e2>2.8829e+06</e2>
      <e3>1.4400e+05</e3>
      <g12>5.0112e+05</g12>
      <g13>1.4400e+05</g13>
      <g23>8.3520e+05</g23>
      <nu12>0.01</nu12>
      <nu13>0.30</nu13>
      <nu23>0.01</nu23>
    </elastic>
  </material>
  <lamina name="la_im7_410">
    <material>im7</material>
    <thickness>0.00410</thickness>
  </lamina>
  <lamina name="la_im7_372">
    <material>im7</material>
    <thickness>0.00372</thickness>
  </lamina>
  <lamina name="la_im7_100">
    <material>im7</material>
    <thickness>0.00100</thickness>
  </lamina>
  <lamina name="la_eglass_050">
    <material>eglass</material>
    <thickness>0.00050</thickness>
```

```
  </lamina>
  <lamina name="la_steel_164">
    <material>steel</material>
    <thickness>0.00164</thickness>
  </lamina>
  <lamina name="la_im7_782">
    <material>im7</material>
    <thickness>0.00782</thickness>
  </lamina>
</materials>
```

Listing B.11: basepoints_rev.dat

```
le   0.0000000000   0.0000000000
2   -0.0006629550   0.0045356860
3   -0.0026500620   0.0089652630
4   -0.0059560520   0.0132828070
5   -0.0105721570   0.0174805780
6   -0.0164861380   0.0215491810
7   -0.0236823100   0.0254777740
8   -0.0321415900   0.0292543320
9   -0.0418415480   0.0328659420
10  -0.0527564590   0.0362991350
11  -0.0648573790   0.0395402370
12  -0.0781122200   0.0425757210
13  -0.0924858310   0.0453925700
14  -0.1079400970   0.0479786130
15  -0.1244340350   0.0503228470
law -0.1419239060   0.0524157230
ler -0.1603633310   0.0542493930
18  -0.1797034110   0.0558179110
19  -0.1998928600   0.0571173780
20  -0.2208781390   0.0581460450
21  -0.2426035990   0.0589043460
22  -0.2650116280   0.0593948930
23  -0.2880428050   0.0596224090
24  -0.3116360530   0.0595936220
25  -0.3357288090   0.0593171150
26  -0.3602571830   0.0588031480
27  -0.3851561290   0.0580634490
lmw -0.4103596210   0.0571109880
29  -0.4358008220   0.0559597540
30  -0.4614122690   0.0546245110
31  -0.4871260430   0.0531205840
32  -0.5128739570   0.0514636420
33  -0.5385877310   0.0496695110
34  -0.5641991780   0.0477540100
35  -0.5896403790   0.0457328110
36  -0.6148438710   0.0436213350
37  -0.6397428170   0.0414346730
38  -0.6642711910   0.0391875450
39  -0.6883639470   0.0368942710
40  -0.7119571950   0.0345687830
41  -0.7349883720   0.0322246420
42  -0.7573964010   0.0298750750
43  -0.7791218610   0.0275330210
44  -0.8001071400   0.0252111770
45  -0.8202965890   0.0229220430
46  -0.8396366690   0.0206779550
47  -0.8580760940   0.0184911130
```

```
48 |48  -0.8755659650   0.0163735830
49 |49  -0.8920599030   0.0143372880
50 |50  -0.9075141690   0.0123939730
51 |51  -0.9218877800   0.0105551550
52 |52  -0.9351426210   0.0088320460
53 |lte -0.9472435410   0.0072354620
54 |54  -0.9581584520   0.0057757170
55 |55  -0.9678584100   0.0044625050
56 |56  -0.9763176900   0.0033047740
57 |57  -0.9835138620   0.0023105990
58 |58  -0.9894278430   0.0014870610
59 |59  -0.9940439480   0.0008401290
60 |60  -0.9973499380   0.0003745600
61 |61  -0.9993370450   0.0000938160
62 |te  -1.0000000000   0.0000000000
63 |63  -0.9993370450   -0.0000938160
64 |64  -0.9973499380   -0.0003745600
65 |65  -0.9940439480   -0.0008401290
66 |66  -0.9894278430   -0.0014870610
67 |67  -0.9835138620   -0.0023105990
68 |68  -0.9763176900   -0.0033047740
69 |69  -0.9678584100   -0.0044625050
70 |70  -0.9581584520   -0.0057757170
71 |hte -0.9472435410   -0.0072354620
72 |72  -0.9351426210   -0.0088320460
73 |73  -0.9218877800   -0.0105551550
74 |74  -0.9075141690   -0.0123939730
75 |75  -0.8920599030   -0.0143372870
76 |76  -0.8755659650   -0.0163735830
77 |77  -0.8580760940   -0.0184911130
78 |78  -0.8396366690   -0.0206779550
79 |79  -0.8202965890   -0.0229220430
80 |80  -0.8001071400   -0.0252111770
81 |81  -0.7791218610   -0.0275330210
82 |82  -0.7573964010   -0.0298750740
83 |83  -0.7349883720   -0.0322246420
84 |84  -0.7119571950   -0.0345687830
85 |85  -0.6883639470   -0.0368942710
86 |86  -0.6642711910   -0.0391875440
87 |87  -0.6397428170   -0.0414346730
88 |88  -0.6148438710   -0.0436213340
89 |89  -0.5896403790   -0.0457328110
90 |90  -0.5641991780   -0.0477540100
91 |91  -0.5385877310   -0.0496695110
92 |92  -0.5128739570   -0.0514636420
93 |93  -0.4871260430   -0.0531205840
94 |94  -0.4614122690   -0.0546245110
95 |95  -0.4358008220   -0.0559597530
96 |hmw -0.4103596210   -0.0571109880
97 |97  -0.3851561290   -0.0580634480
98 |98  -0.3602571830   -0.0588031480
99 |99  -0.3357288090   -0.0593171150
100 |100 -0.3116360530   -0.0595936220
101 |101 -0.2880428050   -0.0596224090
102 |102 -0.2650116280   -0.0593948930
103 |103 -0.2426035990   -0.0589043460
104 |104 -0.2208781390   -0.0581460450
105 |105 -0.1998928600   -0.0571173780
106 |106 -0.1797034110   -0.0558179100
107 |her -0.1603633310   -0.0542493930
```

```
haw -0.1419239060   -0.0524157230
109 -0.1244340350   -0.0503228470
110 -0.1079400970   -0.0479786130
111 -0.0924858310   -0.0453925700
112 -0.0781122200   -0.0425757210
113 -0.0648573790   -0.0395402360
114 -0.0527564590   -0.0362991350
115 -0.0418415480   -0.0328659420
116 -0.0321415900   -0.0292543320
117 -0.0236823100   -0.0254777740
118 -0.0164861380   -0.0215491810
119 -0.0105721570   -0.0174805780
120 -0.0059560520   -0.0132828070
121 -0.0026500620   -0.0089652630
122 -0.0006629550   -0.0045356860
aw  -0.145 0
mw  -0.408 0
pf  -0.750 0
tef -0.950 0
sf  -0.970 0
```

Table B.32: Mapping from OpenMDAO variables to PreVABS+VABS variables.

OpenMDAO variable	PreVABS+VABS access string[a]	Units
spar_layer2_stacks	cross_section%layups%layup(layup_spar)%layer(2)%stack	—
spar_layer3_stacks	cross_section%layups%layup(layup_spar)%layer(3)%stack	—
mass_matrix	M	—
stiff_matrix	S	—
mesh_elems	num_elems	—
force_x	cross_section%recover%forces%1	lbf
force_y	cross_section%recover%forces%2	lbf
force_z	cross_section%recover%forces%3	lbf
moment_x	cross_section%recover%moments%1	ft lbf
moment_y	cross_section%recover%moments%2	ft lbf
moment_z	cross_section%recover%moments%3	ft lbf
gauss_points	SM%1:2	ft
stress	SM%3:8	lbf/ft^2

[a] The PreVABS+VABS access string typically points to a specific position in the cross section XML file. Two special strings, M and S, point to the locations of the mass and stiffness matrices, respectively, in the VABS homogenization output file. Another special string, num_elems, references the number of mesh elements in the VABS input file. In recovery mode, stress or strain output files can be specified with the form <extension>%<first_col>:<last_col>.

APPENDIX C

OPENMDAO MODULES AND SUPPORTING TOOLS

The OpenMDAO modules and supporting tools are used to provide additional capabilities which are necessary to execute the MDA and process its results. The MDA group controls the execution order and data handling for the MDA. The blade ballast calculator and mass calculator ensure consistency between the NDARC, RCAS, and VABS models. The von Mises stress calculator and stress analyzer are used for data reduction and analysis after the MDA execution completes.

C.1 MDA Group

The MDA group, which is responsible for executing each of the individual components of the MDA and handling the data that passes between components, is implemented as a subclass of the OpenMDAO `Group` class. This class can be imported into any other Python script so the MDA environment and the generic SMR helicopter model can easily be reused in different applications.

Listing C.1: HeliGroup

```
import os,sys,inspect
import openmdao.api as om
import numpy as np
import rcotools.ndarc.ndarc_mdao as nm
import modules.RCASwrapper as rm
import modules.PreVabsWrapper as pv
from modules.OmClasses import (HeliMassCalculator, TurnRadiusCalculator,
    BladeBallastCalculator, ControlDamper, HeliCG)

class HeliGroup(om.Group):

    def __init__(self,ndarc_job_path,
                 rcas_run_file,rcas_job_dir,rcas_job_file,
                 prevabs_job_dir,prevabs_job_file,prevabs_supporting_files,
                 run_parallel=False,force_linear=False,saveallruns=False,logger=None)
                     ↪ :
        """
        OpenMDAO Group subclass containing the PreVABS, NDARC, and RCAS models for
            ↪ the
```

```
        generic helicopter along with all necessary connecting and supporting functions.
        Reworked to set vehicle CG instead of payload CG.

        Parameters
        ----------
        ndarc_job_path : string
            Path to the original NDARC njob file.
        rcas_run_file : string
            Path to the rcas_master_run.csh file.
        rcas_job_dir : string
            Path to the RCAS job directory.
        rcas_job_file : string
            Name of the RCAS job file, without extension
        prevabs_job_dir : string
            Path to the PreVABS job directory.
        prevabs_job_file : string
            Name of the PreVABS job file, without extension.

        Returns
        -------
        None.

        """

        super(HeliGroup,self).__init__()

        self.ndarc_job_path = ndarc_job_path

        self.rcas_run_file = rcas_run_file
        self.rcas_job_dir = rcas_job_dir
        self.rcas_job_file = rcas_job_file

        self.prevabs_job_dir = prevabs_job_dir
        self.prevabs_job_file = prevabs_job_file
        self.prevabs_supporting_files = prevabs_supporting_files

        self.run_parallel = run_parallel
        self.force_linear = force_linear
        self.saveallruns = saveallruns
        self.logger = logger

    def setup(self):
        ### Independent variables component
        # For variables that will probably change
        indep_vars = om.IndepVarComp()
        indep_vars.add_discrete_output('casenum', 1) #dummy variable for keeping cases ordered
        indep_vars.add_output('airspeed', 100.0, units='kn')
        indep_vars.add_output('altitude', 0.0, units='ft')
        indep_vars.add_output('gross_weight', 16000.0, units='lb')
        indep_vars.add_output('rate_of_climb', 0.0, units='ft / min') #climb +, descent -
        indep_vars.add_output('turn_rate',  0.0, units='deg / s') #right +, left -
        indep_vars.add_output('cg', 0.503, units='ft') #forward -, backwards +

        ### Static variables component
        # For variables you probably won't change but might want easy access to
        static_vars = om.IndepVarComp()
        static_vars.add_output('blade_mpl', units='slug / ft', val=0.22)
```

```
        static_vars.add_output('blade_cg_y', units='ft', val=0.0175) # + towards leading edge, - towards trailing edge, ref quarter chord
        static_vars.add_output('blade_cg_z', units='ft', val=0.0) # + towards low pressure surface, - towards high pressure surface, ref chordline
        static_vars.add_output('tail_rotor_cant', units='deg', val=0.0) # tail rotor cant for exp 3b. + cants tail rotor upwards


        ### NDARC weight initialization component
        ndarc_weight_init_inputs = [
            ('gross_weight', 'PerfCondition(1)%GW', {'val': 16000.0, 'units': 'lb'}),
            ('weight_main_blade', 'Rotor(1)%dWblade', {'val': 750.0, 'units': 'lbm'}),
            ('tail_rotor_cant', 'Rotor(2)%cant_hub', {'val': 0.0, 'units': 'deg'}), # added for exp 3b
        ]

        ndarc_weight_init_outputs = []
        loc_list = [
            ('loc_fuselage', 'Fuselage%LOC_FUSELAGE'),
            ('loc_gear', 'LandingGear%LOC_GEAR'),
            ('loc_mainrotor', 'Rotor(1)%LOC_ROTOR'),
            ('loc_tailrotor', 'Rotor(2)%LOC_ROTOR'),
            ('loc_horizstab', 'Tail(1)%LOC_TAIL'),
            ('loc_vertstab', 'Tail(2)%LOC_TAIL'),
            ('loc_fueltank', 'FuelTank(1)%LOC_AUXTANK(1)'),
            ('loc_engine', 'EngineGroup(1)%LOC_ENGINE'),
            ]
        for item in loc_list:
            ndarc_weight_init_outputs.extend([
                (item[0]+"_sl", item[1]+"%SLLOC", {'units': 'ft', 'val': 0.0}),
                (item[0]+"_bl", item[1]+"%BLLOC", {'units': 'ft', 'val': 0.0}),
                (item[0]+"_wl", item[1]+"%WLLOC", {'units': 'ft', 'val': 0.0}),
                ])
        ndarc_weight_init_outputs.extend([
            # Fixed weights
            ('weight_fuselage', 'Aircraft%WEIGHT%W_FUSELAGE', {'units': 'lbm', 'val': 0.0}),
            ('weight_systems', 'Aircraft%WEIGHT%W_EQUIP', {'units': 'lbm', 'val': 0.0}),
            ('weight_vibration', 'Aircraft%WEIGHT%W_VIB', {'units': 'lbm', 'val': 0.0}),
            ('weight_fuelsystem', 'Aircraft%WEIGHT%W_FUELSYS', {'units': 'lbm', 'val': 0.0}),
            ('weight_gear', 'Aircraft%WEIGHT%W_GEAR', {'units': 'lbm', 'val': 0.0}),
            ('weight_main_hub', 'Aircraft%WEIGHT%W_ROTOR_HUB', {'units': 'lbm', 'val': 0.0}),
            ('weight_main_blades', 'Aircraft%WEIGHT%W_ROTOR_BLADE', {'units': 'lbm', 'val': 0.0}),
            ('weight_tailrotor', 'Aircraft%WEIGHT%W_TAILROTOR', {'units': 'lbm', 'val': 0.0}),
            ('weight_horizstab', 'Aircraft%WEIGHT%W_HTAIL', {'units': 'lbm', 'val': 0.0}),
            ('weight_vertstab', 'Aircraft%WEIGHT%W_VTAIL', {'units': 'lbm', 'val': 0.0}),
            ('weight_drive', 'Aircraft%WEIGHT%W_DRIVE', {'units': 'lbm', 'val': 0.0}),
            ('weight_engstruct', 'EngineGroup(1)%WEIGHT%W_STRUCTURE', {'units': 'lbm', 'val': 0.0}),
            ('weight_engine', 'Aircraft%WEIGHT%W_ENGSYS', {'units': 'lbm', 'val': 0.0}),
            # Flexible weights
```

```
            ('weight_fuel', 'Performance%PerfCondition(1)%FltAircraft%WFUEL_TOTAL', {'units': 'lbm', 'val': 0.0}),
            ('weight_usefulload', 'Performance%PerfCondition(1)%FltAircraft%WFIXUL', {'units': 'lbm', 'val': 0.0}),
            ('weight_payload', 'Performance%PerfCondition(1)%FltAircraft%WPAYLOAD', {'units': 'lbm', 'val': 0.0}),
            ])

        ndarc_weight_init = nm.NdarcWrapper(self.ndarc_job_path,
                                            ndarc_weight_init_inputs, ndarc_weight_init_outputs,
                                            parallel=self.run_parallel,
                                            saveallruns=self.saveallruns, saveinputfile=self.saveallruns)

        blade_weight_calculator = om.ExecComp('weight_main_blade = 4*(blade_mpl *(26.8-1.25) + 0.112)', # calculate weight of four main blades from mass/length
                                              blade_mpl = {'value': 0.22, 'units': 'slug / ft'},
                                              weight_main_blade = {'value': 0.0, 'units': 'slug'})

        ### NDARC flight condition component
        ndarc_flight_cond_inputs = [
            ('airspeed', 'PerfCondition(1)%Vkts', {'val': 100.0, 'units': 'kn'}),
            ('altitude', 'PerfCondition(1)%altitude', {'val': 0.0, 'units': 'ft'}),
            ('gross_weight', 'PerfCondition(1)%GW', {'val': 16000.0, 'units': 'lb'}),
            ('rate_of_climb', 'PerfCondition(1)%ROC', {'val': 0.0, 'units': 'ft / min'}),
            ('turn_rate', 'PerfCondition(1)%rate_turn', {'val': 0.0, 'units': 'deg / s'}),
            ('loc_cg_sl', 'Geometry%loc_cg%SL', {'val': 0.0, 'units': 'ft'}),
            ('loc_cg_bl', 'Geometry%loc_cg%BL', {'val': 0.0, 'units': 'ft'}),
            ('loc_cg_wl', 'Geometry%loc_cg%WL', {'val': 0.0, 'units': 'ft'}),
            ('weight_main_blade', 'Rotor(1)%dWblade', {'val': 750.0, 'units': 'lbm'}),
            ('tail_rotor_cant', 'Rotor(2)%cant_hub', {'val': 0.0, 'units': 'deg'}), # added for exp 3b
            ]

        ndarc_flight_cond_outputs = [
            # Trimmed fuselage attitude
            ('att_pitch', 'Performance%PerfCondition(1)%FltAircraft%PITCH_TRIM', {'units': 'deg', 'val': 0.0}),
            ('att_roll', 'Performance%PerfCondition(1)%FltAircraft%ROLL_TRIM', {'units': 'deg', 'val': 0.0}),
            # Trimmed pilot control positions
            ('ndarc_cont_coll', 'Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(1)', {'units': 'deg', 'val': 0.0}),
            ('ndarc_cont_lat_cyc','Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(2)', {'units': 'deg', 'val': 0.0}),
            ('ndarc_cont_lon_cyc', 'Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(3)', {'units': 'deg', 'val': 0.0}),
            ('ndarc_cont_pedal', 'Performance%PerfCondition(1)%FltAircraft%CONTROL_TRIM(4)', {'units': 'deg', 'val': 0.0}),
            # Atmospheric conditions
            ('air_viscosity', 'Performance%PerfCondition(1)%FltAircraft%VISCOSITY', {'units': 'slug / (ft * s)', 'val': 0.0}),
            ('air_density', 'Performance%PerfCondition(1)%FltAircraft%DENSITY', {'units': 'slug / ft ** 3', 'val': 0.0}),
            ('air_sound_speed', 'Performance%PerfCondition(1)%FltAircraft%CSOUND', {'units': 'ft / s', 'val': 0.0}),
```

```
            ('ndarc_success', '%success', {'val': False, 'discrete': True}),
            ('ndarc_converged', '%converged', {'val': False, 'discrete': True}),
            ]

        ndarc_flight_cond = nm.NdarcWrapper(self.ndarc_job_path,
                                            ndarc_flight_cond_inputs,
                                                ↪ ndarc_flight_cond_outputs,
                                            parallel=self.run_parallel,
                                            saveallruns=self.saveallruns,
                                                ↪ saveinputfile=self.saveallruns)

        ### RCAS component
        rcas_inputs = [
            # Blade elastic properties
            ('mass_matrix_1','BLADE_GCB_PROP.TAB%MMAT1',{'val': np.zeros([6,6]), '
                ↪ units': None}),
            ('mass_matrix_2','BLADE_GCB_PROP.TAB%MMAT2',{'val': np.zeros([6,6]), '
                ↪ units': None}),
            ('stiff_matrix_1','BLADE_GCB_PROP.TAB%SSTIF1',{'val': np.zeros([6,6]), '
                ↪ units': None}),
            ('stiff_matrix_2','BLADE_GCB_PROP.TAB%SSTIF2',{'val': np.zeros([6,6]), '
                ↪ units': None}),
            # Speeds
            ('airspeed','INITCOND%3%1%1', {'val': 168.0, 'units': 'ft␣/␣s'}),
            ('neg_rate_of_climb', 'INITCOND%3%1%3', {'val': 0.0, 'units': 'ft␣/␣s'}),
            # Fuselage attitude initial conditions
            ('att_pitch', 'INITCOND%2%1%5', {'units': 'rad', 'val': 0.0}),
            ('att_roll', 'INITCOND%2%1%4', {'units': 'rad', 'val': 0.0}),
            # Pilot control initial conditions
            ('rcas_cont_coll', 'INITCOND%1%1%1', {'units': 'deg', 'val': 0.0}),
            ('rcas_cont_lat_cyc','INITCOND%1%1%2', {'units': 'deg', 'val': 0.0}),
            ('rcas_cont_lon_cyc', 'INITCOND%1%1%3', {'units': 'deg', 'val': 0.0}),
            ('rcas_cont_pedal', 'INITCOND%1%1%4', {'units': 'deg', 'val': 0.0}),
            # Turn rate and radius
            ('turn_radius', 'INITCOND%5%1%1', {'units': 'ft', 'val': 0.0}),
            ('turn_rate', 'INITCOND%5%1%2', {'units': 'rad␣/␣s', 'val': 0.0}),
            # Atmospheric
            ('air_density', 'AEROSTATCONST%1%1%4', {'val': 2.37e-3, 'units': 'slug␣/␣
                ↪ ft␣**␣3'}),
            ('air_sound_speed', 'AEROSTATCONST%1%1%5', {'val': 1116, 'units': 'ft␣/␣s
                ↪ '}),
            ('air_viscosity', 'AEROSTATCONST%1%1%6', {'val': 3.62e-7, 'units': 'slug␣
                ↪ /␣(ft␣*␣s)'}),
            # Payload stationline
            ('neg_loc_payload_sl', 'FUSEPS-FENODE%1%4%2', {'val': 0., 'units': 'ft'})
                ↪ ,
            # Tail rotor cant (exp 3b)
            ('tail_rotor_cant_struct', 'SSORIENT%1%3%5', {'val': -90., 'units': 'deg'
                ↪ }),
            ('tail_rotor_cant_aero', 'SCORIENT%1%2%5', {'val': -90., 'units': 'deg'})
                ↪ ,
            # Mass properties
            ('mass_payload', 'FUSEPS-RIGIDBODYMASS%1%6%3', {'val': 67.61, 'units': '
                ↪ slug'}),
            ('inert_payload_Ixx', 'FUSEPS-RIGIDBODYMASS%1%6%4', {'val': 0., 'units':
                ↪ 'slug*ft**2'}),
            ('inert_payload_Ixy', 'FUSEPS-RIGIDBODYMASS%2%6%2', {'val': 0., 'units':
                ↪ 'slug*ft**2'}),
            ('inert_payload_Ixz', 'FUSEPS-RIGIDBODYMASS%2%6%3', {'val': 0., 'units':
                ↪ 'slug*ft**2'}),
            ('inert_payload_Iyy', 'FUSEPS-RIGIDBODYMASS%1%6%5', {'val': 0., 'units':
                ↪ 'slug*ft**2'}),
```

```
            ('inert_payload_Iyz', 'FUSEPS-RIGIDBODYMASS%2%6%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_payload_Izz', 'FUSEPS-RIGIDBODYMASS%1%6%6', {'val': 0., 'units': 'slug*ft**2'}),
            ('mass_fuel', 'FUSEPS-RIGIDBODYMASS%1%5%3', {'val': 77.70, 'units': 'slug'}),
            ('inert_fuel_Ixx', 'FUSEPS-RIGIDBODYMASS%1%5%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuel_Ixy', 'FUSEPS-RIGIDBODYMASS%2%5%2', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuel_Ixz', 'FUSEPS-RIGIDBODYMASS%2%5%3', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuel_Iyy', 'FUSEPS-RIGIDBODYMASS%1%5%5', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuel_Iyz', 'FUSEPS-RIGIDBODYMASS%2%5%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuel_Izz', 'FUSEPS-RIGIDBODYMASS%1%5%6', {'val': 0., 'units': 'slug*ft**2'}),
            ('mass_fuselage', 'FUSEPS-RIGIDBODYMASS%1%1%3', {'val': 173.19, 'units': 'slug'}),
            ('inert_fuselage_Ixx', 'FUSEPS-RIGIDBODYMASS%1%1%4', {'val': 865.99, 'units': 'slug*ft**2'}),
            ('inert_fuselage_Ixy', 'FUSEPS-RIGIDBODYMASS%2%1%2', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuselage_Ixz', 'FUSEPS-RIGIDBODYMASS%2%1%3', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuselage_Iyy', 'FUSEPS-RIGIDBODYMASS%1%1%5', {'val': 14126.19, 'units': 'slug*ft**2'}),
            ('inert_fuselage_Iyz', 'FUSEPS-RIGIDBODYMASS%2%1%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_fuselage_Izz', 'FUSEPS-RIGIDBODYMASS%1%1%6', {'val': 14368.67, 'units': 'slug*ft**2'}),
            ('mass_usefulload', 'FUSEPS-RIGIDBODYMASS%1%2%3', {'val': 24.08, 'units': 'slug'}),
            ('inert_usefulload_Ixx', 'FUSEPS-RIGIDBODYMASS%1%2%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_usefulload_Ixy', 'FUSEPS-RIGIDBODYMASS%2%2%2', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_usefulload_Ixz', 'FUSEPS-RIGIDBODYMASS%2%2%3', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_usefulload_Iyy', 'FUSEPS-RIGIDBODYMASS%1%2%5', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_usefulload_Iyz', 'FUSEPS-RIGIDBODYMASS%2%2%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_usefulload_Izz', 'FUSEPS-RIGIDBODYMASS%1%2%6', {'val': 0., 'units': 'slug*ft**2'}),
            ('mass_gear', 'FUSEPS-RIGIDBODYMASS%1%3%3', {'val': 19.58, 'units': 'slug'}),
            ('inert_gear_Ixx', 'FUSEPS-RIGIDBODYMASS%1%3%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_gear_Ixy', 'FUSEPS-RIGIDBODYMASS%2%3%2', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_gear_Ixz', 'FUSEPS-RIGIDBODYMASS%2%3%3', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_gear_Iyy', 'FUSEPS-RIGIDBODYMASS%1%3%5', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_gear_Iyz', 'FUSEPS-RIGIDBODYMASS%2%3%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_gear_Izz', 'FUSEPS-RIGIDBODYMASS%1%3%6', {'val': 0., 'units': 'slug*ft**2'}),
            ('mass_main_hub', 'MSHAFT-RIGIDBAR%2%1%2', {'val': 18.60, 'units': 'slug'}),
            ('inert_main_hub_Ixx', 'MSHAFT-RIGIDBAR%2%1%3', {'val': 14.53, 'units': 'slug*ft**2'}),
```

```
            ('inert_main_hub_Ixy', 'MSHAFT-RIGIDBAR%2%1%4', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_main_hub_Ixz', 'MSHAFT-RIGIDBAR%2%1%5', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_main_hub_Iyy', 'MSHAFT-RIGIDBAR%2%1%6', {'val': 7.26, 'units': '
                ↪ slug*ft**2'}),
            ('inert_main_hub_Iyz', 'MSHAFT-RIGIDBAR%2%1%7', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_main_hub_Izz', 'MSHAFT-RIGIDBAR%2%1%8', {'val': 7.26, 'units': '
                ↪ slug*ft**2'}),
            ('mass_tail_blade', 'TBLADE1-RIGIDBAR%2%1%2', {'val': 0.45, 'units': '
                ↪ slug'}),
            ('inert_tail_blade_Ixx', 'TBLADE1-RIGIDBAR%2%1%3', {'val': 0.02, 'units':
                ↪  'slug*ft**2'}),
            ('inert_tail_blade_Ixy', 'TBLADE1-RIGIDBAR%2%1%4', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_blade_Ixz', 'TBLADE1-RIGIDBAR%2%1%5', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_blade_Iyy', 'TBLADE1-RIGIDBAR%2%1%6', {'val': 0.86, 'units':
                ↪  'slug*ft**2'}),
            ('inert_tail_blade_Iyz', 'TBLADE1-RIGIDBAR%2%1%7', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_blade_Izz', 'TBLADE1-RIGIDBAR%2%1%8', {'val': 0.88, 'units':
                ↪  'slug*ft**2'}),
            ('mass_tail_hub', 'TSHAFT-RIGIDBAR%2%1%2', {'val': 1.47, 'units': 'slug'
                ↪ }),
            ('inert_tail_hub_Ixx', 'TSHAFT-RIGIDBAR%2%1%3', {'val': 1.06, 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_hub_Ixy', 'TSHAFT-RIGIDBAR%2%1%4', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_hub_Ixz', 'TSHAFT-RIGIDBAR%2%1%5', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_hub_Iyy', 'TSHAFT-RIGIDBAR%2%1%6', {'val': 0.53, 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_hub_Iyz', 'TSHAFT-RIGIDBAR%2%1%7', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_tail_hub_Izz', 'TSHAFT-RIGIDBAR%2%1%8', {'val': 0.53, 'units': '
                ↪ slug*ft**2'}),
            ('mass_horizstab', 'HSTABPS-RIGIDBAR%2%1%2', {'val': 3.09, 'units': 'slug
                ↪ '}),
            ('inert_horizstab_Ixx', 'HSTABPS-RIGIDBAR%2%1%3', {'val': 2.25, 'units':
                ↪ 'slug*ft**2'}),
            ('inert_horizstab_Ixy', 'HSTABPS-RIGIDBAR%2%1%4', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_horizstab_Ixz', 'HSTABPS-RIGIDBAR%2%1%5', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_horizstab_Iyy', 'HSTABPS-RIGIDBAR%2%1%6', {'val': 55.48, 'units':
                ↪  'slug*ft**2'}),
            ('inert_horizstab_Iyz', 'HSTABPS-RIGIDBAR%2%1%7', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_horizstab_Izz', 'HSTABPS-RIGIDBAR%2%1%8', {'val': 57.67, 'units':
                ↪  'slug*ft**2'}),
            ('mass_vertstab', 'VSTABPS-RIGIDBAR%2%1%2', {'val': 2.09, 'units': 'slug'
                ↪ }),
            ('inert_vertstab_Ixx', 'VSTABPS-RIGIDBAR%2%1%3', {'val': 2.27, 'units': '
                ↪ slug*ft**2'}),
            ('inert_vertstab_Ixy', 'VSTABPS-RIGIDBAR%2%1%4', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_vertstab_Ixz', 'VSTABPS-RIGIDBAR%2%1%5', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
            ('inert_vertstab_Iyy', 'VSTABPS-RIGIDBAR%2%1%6', {'val': 14.95, 'units':
                ↪ 'slug*ft**2'}),
            ('inert_vertstab_Iyz', 'VSTABPS-RIGIDBAR%2%1%7', {'val': 0., 'units': '
                ↪ slug*ft**2'}),
```

```
            ('inert_vertstab_Izz', 'VSTABPS-RIGIDBAR%2%1%8', {'val': 12.84, 'units': 'slug*ft**2'}),
            ('mass_engine', 'FUSEPS-RIGIDBODYMASS%1%4%3', {'val': 84.73, 'units': 'slug'}),
            ('inert_engine_Ixx', 'FUSEPS-RIGIDBODYMASS%1%4%4', {'val': 169.46, 'units': 'slug*ft**2'}),
            ('inert_engine_Ixy', 'FUSEPS-RIGIDBODYMASS%2%4%2', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_engine_Ixz', 'FUSEPS-RIGIDBODYMASS%2%4%3', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_engine_Iyy', 'FUSEPS-RIGIDBODYMASS%1%4%5', {'val': 788.70, 'units': 'slug*ft**2'}),
            ('inert_engine_Iyz', 'FUSEPS-RIGIDBODYMASS%2%4%4', {'val': 0., 'units': 'slug*ft**2'}),
            ('inert_engine_Izz', 'FUSEPS-RIGIDBODYMASS%1%4%6', {'val': 788.70, 'units': 'slug*ft**2'}),
            ]

        rcas_outputs = [
            ('rcas_converged','log-conv',{'val': True, 'discrete': True}),
            ('azimuths','tab-BladeIntFrc001%1',{'val': np.zeros([72,1]), 'units': 'deg'}),
            ('blade_int_force_x','tab-BladeIntFrc001%2:12',{'val': np.zeros([72,11]), 'units': 'lbf'}),
            ('blade_int_force_y','tab-BladeIntFrc002%2:12',{'val': np.zeros([72,11]), 'units': 'lbf'}),
            ('blade_int_force_z','tab-BladeIntFrc003%2:12',{'val': np.zeros([72,11]), 'units': 'lbf'}),
            ('blade_int_moment_x','tab-BladeIntMom001%2:12',{'val': np.zeros([72,11]), 'units': 'ft*lbf'}),
            ('blade_int_moment_y','tab-BladeIntMom002%2:12',{'val': np.zeros([72,11]), 'units': 'ft*lbf'}),
            ('blade_int_moment_z','tab-BladeIntMom003%2:12',{'val': np.zeros([72,11]), 'units': 'ft*lbf'}),
            ]

        rcas_wrapper = rm.RCASwrapper(self.rcas_run_file,self.rcas_job_dir,self.rcas_job_file,
                                      rcas_inputs,rcas_outputs,
                                      parallel=self.run_parallel,
                                      force_linear=self.force_linear,
                                      saveallruns=self.saveallruns,
                                      logger=self.logger)

        roc_negatizer = om.ExecComp('neg_rate_of_climb = -rate_of_climb', # needed to align with RCAS inertial frame
                                    rate_of_climb = {'value': 0.0, 'units': 'ft / min'},
                                    neg_rate_of_climb = {'value': 0.0, 'units': 'ft / min'})

        payload_negatizer = om.ExecComp('neg_loc_payload_sl = -loc_payload_sl', # needed since RCAS uses x-forward
                                    loc_payload_sl = {'value': 0.0, 'units': 'ft'},
                                    neg_loc_payload_sl = {'value': 0.0, 'units': 'ft'})

        rcas_tail_rotor_cant = om.ExecComp(['tail_rotor_cant_struct = -90 + tail_rotor_cant',
                                            'tail_rotor_cant_aero = -90 + tail_rotor_cant'], # need to convert tail rotor cant to RCAS tail rotor reference system
```

```
                                        tail_rotor_cant = {'value': 0.0, 'units': 'deg'},
                                        tail_rotor_cant_struct = {'value': -90.0, 'units': 'deg'},
                                        tail_rotor_cant_aero = {'value': -90.0, 'units': 'deg'},
                                        )


        control_damper = ControlDamper(damp_fact=0.75,force_linear=True)

        ### PreVABS component
        prevabs_inputs = [
            ('spar_layer2_stacks','cross_section%layups%layup(layup_spar)%layer(2)%stack',{'val': 1, 'units': None}),
            ('spar_layer3_stacks','cross_section%layups%layup(layup_spar)%layer(3)%stack',{'val': 1, 'units': None}),
            ]
        prevabs_outputs = [
            ('mass_matrix','M',{'val': np.zeros([6,6]), 'units': None}),
            ('stiff_matrix','S',{'val': np.zeros([6,6]), 'units': None}),
            ('mesh_elems','num_elems',{'val': 0, 'units': None}),
        ]

        prevabs_wrapper = pv.PreVabsWrapper(self.prevabs_job_dir,self.prevabs_job_file,self.prevabs_supporting_files,
                                            prevabs_inputs,prevabs_outputs,
                                            parallel=self.run_parallel,
                                            force_linear=self.force_linear,
                                            saveallruns=self.saveallruns,
                                            logger=self.logger)

        ### Add subsystems
        self.add_subsystem('indepVars', indep_vars)
        self.add_subsystem('staticVars', static_vars)
        self.add_subsystem('preVabs', prevabs_wrapper,
                           promotes_outputs=['*'])
        self.add_subsystem('bladeBallastCalculator', BladeBallastCalculator(force_linear=self.force_linear),
                           promotes_inputs=['*'], promotes_outputs=['*'])
        self.add_subsystem('bladeWeightCalculator', blade_weight_calculator,
                           promotes=['*'])
        self.add_subsystem('rcasTailRotorCant', rcas_tail_rotor_cant,
                           promotes=['*'])
        self.add_subsystem('ndarcWeightInit', ndarc_weight_init,
                           promotes_inputs=['*'], promotes_outputs=['*'])
        self.add_subsystem('heliCG', HeliCG(single_blade_analysis=True,
                           force_linear=self.force_linear),
                           promotes_inputs=['*'], promotes_outputs=['*'])
        self.add_subsystem('ndarcFlightCond', ndarc_flight_cond,
                           promotes_inputs=['*'], promotes_outputs=['*'])
        self.add_subsystem('rocNegatizer', roc_negatizer,
                           promotes=['*'])
        self.add_subsystem('payloadNegatizer', payload_negatizer,
                           promotes=['*'])
        self.add_subsystem('turnRadiusCalculator', TurnRadiusCalculator(force_linear=self.force_linear),
                           promotes=['*'])
        self.add_subsystem('controlDamper', control_damper,
                           promotes=['*'])
        self.add_subsystem('rcas',rcas_wrapper,
```

```
                            promotes_inputs=['*'], promotes_outputs=['rcas_converged'
                                ↪ ])

        ### Set the subsystem execution order just in case
        self.set_order(['indepVars',
                        'staticVars',
                        'preVabs',
                        'bladeBallastCalculator',
                        'bladeWeightCalculator',
                        'rcasTailRotorCant',
                        'ndarcWeightInit',
                        'heliCG',
                        'ndarcFlightCond',
                        'rocNegatizer',
                        'payloadNegatizer',
                        'turnRadiusCalculator',
                        'controlDamper',
                        'rcas',
                        ])

    def configure(self):
        ### Define remaining connections
        # Independent variables
        self.connect('indepVars.airspeed','airspeed')
        self.connect('indepVars.altitude','altitude')
        self.connect('indepVars.gross_weight','gross_weight')
        self.connect('indepVars.rate_of_climb','rate_of_climb')
        self.connect('indepVars.turn_rate','turn_rate')
        self.connect('indepVars.cg','set_cg_sl')
        # Static variables
        self.connect('staticVars.blade_mpl','blade_mpl')
        self.connect('staticVars.blade_cg_y','blade_cg_y')
        self.connect('staticVars.blade_cg_z','blade_cg_z')
        self.connect('staticVars.tail_rotor_cant','tail_rotor_cant')
        # Blade elastic properties
        self.connect('mass_matrix_new',['mass_matrix_1','mass_matrix_2'])
        self.connect('stiff_matrix',['stiff_matrix_1','stiff_matrix_2'])
```

C.2 Blade Ballast Calculator

The blade ballast calculator, which is responsible for balancing the rotor blade to ensure the CG position does not move as the cross section design changes, is implemented as a subclass of the OpenMDAO `ExplicitComponent` class. This class makes use of a function called `add_ballast` to perform the calculations. Table C.1 lists the inputs and outputs of this module.

Listing C.2: BladeBallastCalculator

```
class BladeBallastCalculator(ExplicitComponent):
    """
    Explicit component for calculating the appropriate blade cross-section ballast
```

```
    for the desired mass per unit length and CG location
    """
    def __init__(self,force_linear=False):
        super(BladeBallastCalculator, self).__init__()
        self.force_linear = force_linear

    def setup(self):
        self.add_input("mass_matrix", units=None, val=np.empty((6,6)))
        self.add_input("blade_mpl", units="slug / ft", val=0.22)
        self.add_input("blade_cg_y", units="ft", val=0.0175)
        self.add_input("blade_cg_z", units="ft", val=0.0)
        self.add_output("mass_matrix_new", units=None, val=np.empty((6,6)))

    def compute(self, inputs, outputs):
        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            outputs["mass_matrix_new"] = add_ballast(inputs["mass_matrix"],
                                                     inputs["blade_mpl"][0],
                                                     inputs["blade_cg_y"][0],
                                                     inputs["blade_cg_z"][0])
```

Listing C.3: add_ballast

```
def add_ballast(M_0,mu_f,Xm2_f,Xm3_f):
    """
    Calculates ballast mass and location for mass/CG matching of airfoil.

    Parameters
    ----------
    M_0 : numpy.array
        Original mass matrix from VABS.
    mu_f : float
        Desired value of mass per unit length.
    Xm2_f : float
        Desired i2 location of center of mass.
    Xm3_f : float
        Desired i3 location of center of mass.

    Returns
    -------
    M_f : numpy.array
        New mass matrix in VABS format.

    """

    # Original properties
    mu_0 = M_0[0][0]
    Xm2_0 = M_0[0][5] / -mu_0
    Xm3_0 = M_0[0][4] / mu_0
    i22_0 = M_0[4][4]
    i33_0 = M_0[5][5]
    i23_0 = M_0[4][5]

    # Calculations
    mu_b = mu_f - mu_0 # ballast mass
    if mu_b <= 0:
        raise ValueError("Inputs produce a non-positive value for ballast mass")
```

```
    Xm2_b = (Xm2_f*mu_f - Xm2_0*mu_0) / mu_b # ballast location
    Xm3_b = (Xm3_f*mu_f - Xm3_0*mu_0) / mu_b # ballast location

    i22_f = i22_0 + mu_b*Xm3_b**2 # ballast is basically with i22 axis so this shouldn't change much
    i33_f = i33_0 + mu_b*Xm2_b**2 # this should increase
    i23_f = i23_0 + mu_b*Xm2_b*Xm3_b # this should change slightly

    # New mass matrix
    M_f = np.array([[ mu_f,       0,          0,          0,              mu_f*Xm3_f, -mu_f*Xm2_f],
                    [ 0,          mu_f,       0,         -mu_f*Xm3_f,     0,          0 ],
                    [ 0,          0,          mu_f,       mu_f*Xm2_f,     0,          0 ],
                    [ 0,         -mu_f*Xm3_f, mu_f*Xm2_f, i22_f + i33_f, 0,          0 ],
                    [ mu_f*Xm3_f, 0,          0,          0,              i22_f,      i23_f ],
                    [-mu_f*Xm2_f, 0,          0,          0,              i23_f,      i33_f ]])

    return M_f
```

Table C.1: Inputs and outputs for the blade ballast calculator.

OpenMDAO variable	Units	Notes
Inputs		
`mass_matrix`	—	From VABS outputs
`blade_mpl`	slug/ft	μ of baseline cross section (default is 0.22 slug/ft)
`blade_cg_y`	ft	x_{m_2} of baseline cross section (default is 0.0175 ft)
`blade_cg_z`	ft	x_{m_3} of baseline cross section (default is 0.0 ft)
Outputs		
`mass_matrix_new`	—	To RCAS GECB definition

C.3 Mass Calculator

The mass calculator is responsible for calculating the inertial properties of various subsystems on the helicopter model to maintain consistency between the NDARC and RCAS models. It is implemented as a subclass of the OpenMDAO `ExplicitComponent` class. This class makes use of a function called `calculate_mass_props` to perform the calculations. Table C.1 lists the inputs and outputs of this module.

Listing C.4: HeliCG

```
class HeliCG(ExplicitComponent):
    """
    Explicit component for calculating the mass and inertia of the helicopter
    model. Provides a payload SL location for a given payload weight and desired
    CG location.
    """
    def __init__(self,single_blade_analysis=False,force_linear=False):
        super(HeliCG,self).__init__()
        self.single_blade_analysis = single_blade_analysis
        # RCAS has a bug where the single blade analysis also multiplies the mass of the hub/shaft by the number of blades.
        # Setting this option to true will divide the output masses/inertias by four to counteract this.
        self.force_linear = force_linear

    def setup(self):
        # Location inputs
        loc_list = [
            'loc_fuselage',
            'loc_gear',
            'loc_mainrotor',
            'loc_tailrotor',
            'loc_horizstab',
            'loc_vertstab',
            'loc_fueltank',
            'loc_engine',
            ]

        for item in loc_list:
            for dim in ["sl","bl","wl"]:
                self.add_input(item+"_"+dim, units="ft", val=0.0)

        # self.add_input("loc_payload_sl", units="ft", val=0.0)
        self.add_input("set_cg_sl",units='ft',val=0.0)

        # Weight inputs
        self.add_input("weight_fuselage", units="slug", val=0.0)
        self.add_input("weight_systems", units="slug", val=0.0)
        self.add_input("weight_vibration", units="slug", val=0.0)
        self.add_input("weight_fuelsystem", units="slug", val=0.0)
        self.add_input("weight_gear", units="slug", val=0.0)
        self.add_input("weight_main_hub", units="slug", val=0.0)
        self.add_input("weight_main_blades", units="slug", val=0.0)
        self.add_input("weight_tailrotor", units="slug", val=0.0)
        self.add_input("weight_horizstab", units="slug", val=0.0)
        self.add_input("weight_vertstab", units="slug", val=0.0)
        self.add_input("weight_drive", units="slug", val=0.0)
        self.add_input("weight_engstruct", units="slug", val=0.0)
        self.add_input("weight_engine", units="slug", val=0.0)
        self.add_input("weight_fuel", units="slug", val=0.0)
        self.add_input("weight_usefulload", units="slug", val=0.0)
        self.add_input("weight_payload", units="slug", val=0.0)

        # CG output
        self.add_output("loc_cg_sl",units='ft',val=0.0)
        self.add_output("loc_cg_bl",units='ft',val=0.0)
        self.add_output("loc_cg_wl",units='ft',val=0.0)
        self.add_output("loc_payload_sl", units="ft", val=0.0)

```

```
        # Mass and inertia outputs
        sys_list = [
            "payload",
            "fuel",
            "fuselage",
            "usefulload",
            "gear",
            "main_hub",
            "tail_blade",
            "tail_hub",
            "horizstab",
            "vertstab",
            "engine",
            ]

        for item in sys_list:
            self.add_output("mass_"+item,units='slug')
            for elem in ["xx","xy","xz","yy","yz","zz"]:
                self.add_output("inert_"+item+"_I"+elem,units='slug*ft**2',val=0.0)

    def compute(self,inputs,outputs):
        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            # Set up the vehicle dictionary without payload
            fuel = {
                "location": np.array([inputs["loc_fueltank_sl"][0],
                                      inputs["loc_fueltank_bl"][0],
                                      inputs["loc_fueltank_wl"][0]]), # SL, BL, WL
                "mass": {
                    "fuel": inputs["weight_fuel"][0], # slug
                    },
                "inertia": {
                    "model": None,
                    }
                }

            fuselage = {
                "location": np.array([inputs["loc_fuselage_sl"][0],
                                      inputs["loc_fuselage_bl"][0],
                                      inputs["loc_fuselage_wl"][0]]), # SL, BL, WL
                "mass": {
                    "fuselage_group": inputs["weight_fuselage"][0], # slug
                    "systems_and_equipment": inputs["weight_systems"][0],
                    "vibration": inputs["weight_vibration"][0],
                    "fuel_system": inputs["weight_fuelsystem"][0],
                    },
                "inertia": {
                    "model": "ellipsoid",
                    "x": 39.94, # ft
                    "y": 8, # ft
                    "z": 6 # ft
                    }
                }

            useful_load = {
                "location": np.array([inputs["loc_fuselage_sl"][0],
                                      inputs["loc_fuselage_bl"][0],
```

```
                                    inputs["loc_fuselage_wl"][0]]), # SL, BL, WL
                "mass": {
                    "fixed_useful_load": inputs["weight_usefulload"][0], # slug
                    },
                "inertia": {
                    "model": None,
                    }
                }

            gear = {
                "location": np.array([inputs["loc_gear_sl"][0],
                                    inputs["loc_gear_bl"][0],
                                    inputs["loc_gear_wl"][0]]), # SL, BL, WL
                "mass": {
                    "alighting_gear": inputs["weight_gear"][0], # slug
                    },
                "inertia": {
                    "model": None,
                    }
                }

            main_blade_1 = {
                "location": np.array([inputs["loc_mainrotor_sl"][0],
                                    inputs["loc_mainrotor_bl"][0],
                                    inputs["loc_mainrotor_wl"][0]]), # SL, BL, WL
                "mass": {
                    "blade": inputs["weight_main_blades"][0] / 4, # slug
                    },
                "inertia": {
                    "model": None,
                    }
                }

            main_blade_2 = main_blade_1.copy()
            main_blade_3 = main_blade_1.copy()
            main_blade_4 = main_blade_1.copy()

            main_hub = {
                "location": np.array([inputs["loc_mainrotor_sl"][0],
                                    inputs["loc_mainrotor_bl"][0],
                                    inputs["loc_mainrotor_wl"][0]]), # SL, BL, WL
                "mass": {
                    "hub_and_hinge": inputs["weight_main_hub"][0], # slug
                    },
                "inertia": {
                    "model": "flat_disc",
                    "orientation": "x",
                    "radius": 1.25,
                    }
                }

            tail_blade_1 = {
                "location": np.array([inputs["loc_tailrotor_sl"][0],
                                    inputs["loc_tailrotor_bl"][0],
                                    inputs["loc_tailrotor_wl"][0]]), # SL, BL, WL
                "mass": {
                    "blade": (0.55*inputs["weight_tailrotor"][0]) / 4, # slug
                    },
                "inertia": {
                    "model": "rect_prism",
```

```
                "x": 4.8, # ft
                "y": 0.7539822,
                "z": 0.090478,
                }
            }

        tail_blade_2 = tail_blade_1.copy()
        tail_blade_3 = tail_blade_1.copy()
        tail_blade_4 = tail_blade_1.copy()

        tail_hub = {
            "location": np.array([inputs["loc_tailrotor_sl"][0],
                                  inputs["loc_tailrotor_bl"][0],
                                  inputs["loc_tailrotor_wl"][0]]), # SL, BL, WL
            "mass": {
                "hub_and_hinge": 0.45*inputs["weight_tailrotor"][0], # slug
                },
            "inertia": {
                "model": "flat_disc",
                "orientation": "x",
                "radius": 1.2,
                }
            }

        horiz_stab = {
            "location": np.array([inputs["loc_horizstab_sl"][0],
                                  inputs["loc_horizstab_bl"][0],
                                  inputs["loc_horizstab_wl"][0]]), # SL, BL, WL
            "mass": {
                "horiz_stab": inputs["weight_horizstab"][0], # slug
                },
            "inertia": {
                "model": "rect_prism",
                "x": 14.936,
                "y": 2.931872,
                "z": 0.35182464,
                }
            }

        vert_stab = {
            "location": np.array([inputs["loc_vertstab_sl"][0],
                                  inputs["loc_vertstab_bl"][0],
                                  inputs["loc_vertstab_wl"][0]]), # SL, BL, WL
            "mass": {
                "vert_stab": inputs["weight_vertstab"][0], # slug
                },
            "inertia": {
                "model": "rect_prism",
                "x": 8.5527,
                "y": 0.7088234,
                "z": 3.544117, # switched chord and thickness to align with RCAS coordinate system
                }
            }

        engine_group = {
            "location": np.array([inputs["loc_engine_sl"][0],
                                  inputs["loc_engine_bl"][0],
                                  inputs["loc_engine_wl"][0]]), # SL, BL, WL
            "mass": {
```

```
                    "drive_system": inputs["weight_drive"][0], # slug
                    "engine_structure": inputs["weight_engstruct"][0],
                    "engine_system": inputs["weight_engine"][0],
                    },
                "inertia": {
                    "model": "cylinder",
                    "orientation": "x",
                    "length": 9.985, # ft
                    "radius": 2, # ft
                    }
                }

            no_payload = {
                # "payload": payload,
                "fuel": fuel,
                "fuselage": fuselage,
                "useful_load": useful_load,
                "gear": gear,
                "main_blade_1": main_blade_1,
                "main_blade_2": main_blade_2,
                "main_blade_3": main_blade_3,
                "main_blade_4": main_blade_4,
                "main_hub": main_hub,
                "tail_blade_1": tail_blade_1,
                "tail_blade_2": tail_blade_2,
                "tail_blade_3": tail_blade_3,
                "tail_blade_4": tail_blade_4,
                "tail_hub": tail_hub,
                "horiz_stab": horiz_stab,
                "vert_stab": vert_stab,
                "engine_group": engine_group,
                }
            vehicle = deepcopy(no_payload) # copy before it gets processed

            # Run calculate_mass_props with no payload
            no_pl_out, no_pl_mass, no_pl_cg = calculate_mass_props(no_payload)
            no_pl_cg_sl = no_pl_cg[0]
            # caclulate payload SL for desired CG position
            loc_payload_sl = (inputs["set_cg_sl"][0] * (no_pl_mass + inputs["weight_payload"][0]) -
                              no_pl_mass * no_pl_cg_sl ) / inputs["weight_payload"][0]

            # Build payload dict
            payload = {
                "location": np.array([loc_payload_sl,0,0]), # SL, BL, WL
                "mass": {
                    "payload": inputs["weight_payload"][0], # slug
                    },
                "inertia": {
                    "model": None,
                    }
                }
            # Add payload to vehicle dict
            vehicle["payload"] = payload
            # Rerun calculatemass_props with full vehicle
            vehicle_out, total_mass, cg_position = calculate_mass_props(vehicle)

            # Set outputs
            outputs["loc_cg_sl"] = cg_position[0]
            outputs["loc_cg_bl"] = cg_position[1]
```

```
            outputs["loc_cg_wl"] = cg_position[2]
            outputs["loc_payload_sl"] = loc_payload_sl

            sys_list = [
                ("payload","payload"),
                ("fuel","fuel"),
                ("fuselage","fuselage"),
                ("usefulload","useful_load"),
                ("gear","gear"),
                ("main_hub","main_hub"),
                ("tail_blade","tail_blade_1"),
                ("tail_hub","tail_hub"),
                ("horizstab","horiz_stab"),
                ("vertstab","vert_stab"),
                ("engine","engine_group"),
                ]

            for item in sys_list:
                if not self.single_blade_analysis or "hub" not in item[1]:
                    outputs["mass_"+item[0]] = vehicle_out[item[1]]["system_mass"]
                    for elem in ["xx","xy","xz","yy","yz","zz"]:
                        outputs["inert_"+item[0]+"_I"+elem] = vehicle_out[item[1]]["inertia"]["I"+elem]
                elif self.single_blade_analysis and "hub" in item[1]: # divide hub masses and inertias by four
                    outputs["mass_"+item[0]] = vehicle_out[item[1]]["system_mass"] / 4
                    for elem in ["xx","xy","xz","yy","yz","zz"]:
                        outputs["inert_"+item[0]+"_I"+elem] = vehicle_out[item[1]]["inertia"]["I"+elem] / 4
```

Listing C.5: calculate_mass_props

```
def calculate_mass_props(system_dict):
    """
    Calculates the total mass, center of gravity position, and inertia matrices
    for the vehicle.

    Parameters
    ----------
    system_list : dict of dicts
        A dict of dictionaries. Each dictionary describes a system on the
        vehicle. Dictionaries should be formatted as follows:

            example_system = {
                "location": np.array([<SL>,<BL>,<WL>]),
                "mass": {
                    "example_mass_1": <example_mass_value>,
                    "example_mass_2": <example_mass_value>,
                    "example_mass_3": <example_mass_value>,
                    },
                "inertia": {
                    "model": "<inertia_model>",
                    "orientation": "<axis_of_orientation>",
                    <supporting_entries>,
                    }
                }

        Valid inertia models include "cylinder", "flat_disc", "rect_prism",
        "ellipsoid", or None.
```

```
        "cylinder" should have a specified orientation and additional "length"
        and "radius" entries.

        "flat_disc" should have a specified orientation and additional "radius"
        entry.

        "rect_prism" should have additional "x" "y" and "z" entries.

        "ellipsoid" should have additional "x" "y" and "z" entries.

        Orientation need not be specified for "rect_prism" and "ellipsoid".

        All units should be in the US customary (slug-foot-second) system.

    Returns
    -------
    system_dict : dict of dicts
        Updated version of the input dict containing additional fields for
        calculated values.
    total_mass : float
        Total mass of the vehicle described by systems_list.
    cg_position : numpy.array
        Center of gravity position, in the order SL, BL, WL.

    """

    # Initialize variables
    total_mass = 0
    moment_arms = np.array([0,0,0])

    # Iterate through systems
    for key in system_dict:
        system = system_dict[key]

        # Mass and moment arms
        system["system_mass"] = sum(system["mass"].values())
        total_mass = total_mass + system["system_mass"]
        moment_arms = moment_arms + system["system_mass"] * system["location"]

        # Inertia properties
        M = system["system_mass"] # shorthand for equations
        # All equations from Prof. Saleh's notes for AE 6210 Advanced Dynamics
        inertia = system["inertia"]

        if inertia["model"] is None:
            inertia["Ixx"] = 0
            inertia["Iyy"] = 0
            inertia["Izz"] = 0

            inertia["Ixy"] = 0
            inertia["Ixz"] = 0
            inertia["Iyz"] = 0
        else:
            if inertia["model"] == "ellipsoid":
                a = inertia["z"]/2
                b = inertia["y"]/2
                c = inertia["x"]/2

                inertia["Ixx"] = (1/5)*M*(a**2 + b**2)
```

```
                inertia["Iyy"] = (1/5)*M*(a**2 + c**2)
                inertia["Izz"] = (1/5)*M*(b**2 + c**2)

                inertia["Ixy"] = 0
                inertia["Ixz"] = 0
                inertia["Iyz"] = 0

            elif inertia["model"] == "cylinder":
                L = inertia["length"]
                R = inertia["radius"]

                if inertia["orientation"] == "x":
                    inertia["Ixx"] = M*R**2 / 2
                    inertia["Iyy"] = (M/12) * (L**2 + 3*R**2)
                    inertia["Izz"] = (M/12) * (L**2 + 3*R**2)
                elif inertia["orientation"] == "y":
                    inertia["Ixx"] = (M/12) * (L**2 + 3*R**2)
                    inertia["Iyy"] = M*R**2 / 2
                    inertia["Izz"] = (M/12) * (L**2 + 3*R**2)
                elif inertia["orientation"] == "z":
                    inertia["Ixx"] = (M/12) * (L**2 + 3*R**2)
                    inertia["Iyy"] = (M/12) * (L**2 + 3*R**2)
                    inertia["Izz"] = M*R**2 / 2
                else:
                    raise ValueError(inertia["orientation"] + "␣is␣not␣a␣valid␣orientation")

                inertia["Ixy"] = 0
                inertia["Ixz"] = 0
                inertia["Iyz"] = 0

            elif inertia["model"] == "rect_prism":
                L = inertia["y"]
                W = inertia["x"]
                H = inertia["z"]

                inertia["Ixx"] = (1/12)*M*(L**2 + H**2)
                inertia["Iyy"] = (1/12)*M*(W**2 + H**2)
                inertia["Izz"] = (1/12)*M*(L**2 + W**2)

                inertia["Ixy"] = 0
                inertia["Ixz"] = 0
                inertia["Iyz"] = 0

            elif inertia["model"] == "flat_disc":
                R = inertia["radius"]

                if inertia["orientation"] == "x":
                    inertia["Ixx"] = M*R**2 / 2
                    inertia["Iyy"] = M*R**2 / 4
                    inertia["Izz"] = M*R**2 / 4
                elif inertia["orientation"] == "y":
                    inertia["Ixx"] = M*R**2 / 4
                    inertia["Iyy"] = M*R**2 / 2
                    inertia["Izz"] = M*R**2 / 4
                elif inertia["orientation"] == "z":
                    inertia["Ixx"] = M*R**2 / 4
                    inertia["Iyy"] = M*R**2 / 4
                    inertia["Izz"] = M*R**2 / 2
                else:
```

```
                    raise ValueError(inertia["orientation"] + "␣is␣not␣a␣valid␣orientation")

                inertia["Ixy"] = 0
                inertia["Ixz"] = 0
                inertia["Iyz"] = 0

            else:
                raise ValueError(inertia["model"] + "␣is␣not␣a␣valid␣inertia␣model")

        inertia["matrix"] = np.array([[inertia["Ixx"], inertia["Ixy"], inertia["Ixz"]],
                                      [inertia["Ixy"], inertia["Iyy"], inertia["Iyz"]],
                                      [inertia["Ixz"], inertia["Iyz"], inertia["Izz"]]])

    # Calculate center of gravity position
    cg_position = moment_arms / total_mass

    return system_dict, total_mass, cg_position
```

Table C.2: Inputs and outputs for the mass calculator.

OpenMDAO variable	Units	Notes
Inputs		
loc_<system>_sl[a]	ft	From NDARC outputs
loc_<system>_bl[a]	ft	From NDARC outputs
loc_<system>_wl[a]	ft	From NDARC outputs
set_cg_sl	ft	Defined by flight condition
weight_<system>[b]	slug	From NDARC outputs
Outputs		
loc_cg_sl	ft	Included to verify match with set_cg_sl
loc_cg_bl	ft	
loc_cg_wl	ft	
loc_payload_sl	ft	To RCAS model
mass_<system>[c]	slug ft^2	To RCAS model
inert_<system>_Ixx[c]	slug ft^2	To RCAS model
inert_<system>_Iyy[c]	slug ft^2	To RCAS model
inert_<system>_Izz[c]	slug ft^2	To RCAS model
inert_<system>_Ixy[c]	slug ft^2	To RCAS model
inert_<system>_Ixz[c]	slug ft^2	To RCAS model
inert_<system>_Iyz[c]	slug ft^2	To RCAS model

[a] <system> is replaced with fuselage, gear, mainrotor, tailrotor, horizstab, vertstab, fueltank, and engine.

[b] <system> is replaced with fuselage, systems, vibration, fuelsystem, gear, main_hub, main_blades, tailrotor, horizstab, vertstab, drive, engstruct, engine, fuel, usefulload, and payload.

[c] <system> is replaced with payload, fuel, fuselage, usefulload, gear, main_hub, tail_blade, tail_hub, horizstab, vertstab, and engine.

C.4 Von Mises Stress Calculator

The von Mises stress calculator, which reduces a stress tensor field to a signed von Mises stress field, is implemented as a subclass of the OpenMDAO `ExplicitComponent` class. Table C.3 lists the inputs and outputs of this module.

Listing C.6: VonMisesStress

```
class VonMisesStress(ExplicitComponent):
    """
    Explicit component for calculating von Mises stress and strain from VABS recovery
        ↪ .
```

```
    """

    def __init__(self,num_gauss_points,force_linear=False,use_critical_point=False):
        super(VonMisesStress, self).__init__()
        self.num_gauss_points = num_gauss_points
        self.force_linear = force_linear
        self.use_critical_point = use_critical_point

    def setup(self):
        self.add_input("gauss_points", units="ft", val=np.zeros([self.num_gauss_points,2]))
        self.add_input("stress", units="lbf / ft**2", val=np.zeros([self.num_gauss_points,6]))
        self.add_discrete_input("use_critical_point", val=False)
        self.add_input("critical_point", units="ft", val=np.zeros(2))

        if self.use_critical_point:
            self.add_output("stress_vm", units="lbf / ft**2", val=np.zeros(1))
            self.add_output("stress_vm_signed", units="lbf / ft**2", val=np.zeros(1))
        else:
            self.add_output("stress_vm", units="lbf / ft**2", val=np.zeros(self.num_gauss_points))
            self.add_output("stress_vm_signed", units="lbf / ft**2", val=np.zeros(self.num_gauss_points))

    def compute(self, inputs, outputs, discrete_inputs, discrete_outputs):
        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            # Assign values to arrays:
            s11 = inputs["stress"][:,0]
            s12 = inputs["stress"][:,1]
            s13 = inputs["stress"][:,2]
            s22 = inputs["stress"][:,3]
            s23 = inputs["stress"][:,4]
            s33 = inputs["stress"][:,5]

            # Stress calculations
            shs = (s11 + s22 + s33) / 3 # hydrostatic stress
            svm = np.sqrt(0.5*((s11-s22)**2 + (s22-s33)**2 + (s33-s11)**2) + 3*(s12**2 + s23**2 + s13**2)) # von mises stress
            # signed von mises stress:
            svms = svm
            svms[shs < 0] = -svms[shs < 0] # make von mises stress negative if hydrostatic stress is less than 0

            # Select critical point if desired
            if discrete_inputs["use_critical_point"]:
                critical = inputs["critical_point"]
                gauss_points = inputs["gauss_points"]
                dist_to_key = np.sqrt((gauss_points[:,0] - critical[0])**2 + (gauss_points[:,1] - critical[1])**2).flatten()
                nearest = dist_to_key.argmin()
                # print(critical,nearest)
                outputs["stress_vm"] = svm.flatten()[nearest]
                outputs["stress_vm_signed"] = svms.flatten()[nearest]
            else:
                outputs["stress_vm"] = svm.flatten()
                outputs["stress_vm_signed"] = svms.flatten()
```

Table C.3: Inputs and outputs for the von Mises stress calculator.

OpenMDAO variable	Units	Notes
Inputs		
`gauss_points`	ft	Array of Gauss point locations
`stress`	lbf/ft^2	Array of stress tensors corresponding to Gauss point locations
`use_critical_point`	`True` or `False`	Boolean flag to limit analysis to a single Gauss point rather than the entire stress field
`critical_point`	ft	Critical point location if `use_critical_point` is `True`
Outputs		
`stress_vm`	lbf/ft^2	Von Mises stress at each Gauss point location
`stress_vm_signed`	lbf/ft^2	Signed von Mises stress at each Gauss point location

C.5 Stress Analyzer

The stress analyzer, which calculates the first-harmonic components of the signed von Mises stress cycle and applies the Goodman relation to derive equivalent stress, is implemented as a subclass of the OpenMDAO `ExplicitComponent` class. Table C.4 lists the inputs and outputs of this module.

Listing C.7: VonMisesStress

```
class StressAnalyzer(ExplicitComponent):
    """
    Explicit component for calculating certain stress values of interest from von
        ↪ Mises stress.
    """
    def __init__(self,num_gauss_points,num_time_steps,force_linear=False):
        super(StressAnalyzer, self).__init__()
        self.num_gauss_points = num_gauss_points
        self.num_time_steps = num_time_steps
        self.force_linear = force_linear

    def setup(self):
        self.add_input("stress_ultimate", units="lbf␣/␣ft**2", val=2.304e7) # default
            ↪  is 160 ksi
        self.add_input("stress_vm_signed", units="lbf␣/␣ft**2", val=np.zeros((self.
            ↪ num_gauss_points,self.num_time_steps)))
        self.add_input("gauss_points", units="ft", val=np.zeros((self.
            ↪ num_gauss_points,2)))
        self.add_input("critical_point",units="ft", val=np.zeros(2))

        self.add_output("stress_amplitude", units="lbf␣/␣ft**2", val=np.zeros((self.
            ↪ num_gauss_points)))
        self.add_output("stress_mean", units="lbf␣/␣ft**2", val=np.zeros((self.
            ↪ num_gauss_points)))
        self.add_output("stress_equivalent", units="lbf␣/␣ft**2", val=np.zeros((self.
            ↪ num_gauss_points)))

        self.add_output("max_stress_amplitude", units="lbf␣/␣ft**2", val=0.0)
        self.add_output("max_stress_mean", units="lbf␣/␣ft**2", val=0.0)
        self.add_output("max_stress_equivalent", units="lbf␣/␣ft**2", val=0.0)

        self.add_output("loc_max_stress_amplitude", units="ft", val=np.zeros(2))
        self.add_output("loc_max_stress_mean", units="ft", val=np.zeros(2))
        self.add_output("loc_max_stress_equivalent", units="ft", val=np.zeros(2))

        self.add_output("critical_stress_mean", units="lbf␣/␣ft**2", val=0.0)
        self.add_output("critical_stress_amplitude", units="lbf␣/␣ft**2", val=0.0)
        self.add_output("critical_stress_equivalent", units="lbf␣/␣ft**2", val=0.0)

    def compute(self, inputs, outputs):
        do_compute = True
        if self.force_linear and self.comm.rank != 0:
            do_compute = False

        if do_compute:
            stress = inputs["stress_vm_signed"]
```

```
        s_u = inputs["stress_ultimate"]
        gauss_points = inputs["gauss_points"] # shouldn't change throughout the cycle

        # Stress mean/amplitude/equivalent at each point
        stress_amp = np.zeros(self.num_gauss_points)
        stress_mean = np.zeros(self.num_gauss_points)
        stress_eq = np.zeros(self.num_gauss_points)
        for i in range(self.num_gauss_points):
            stress_amp[i] = stress[i,:].max() - stress[i,:].min()
            stress_mean[i] = stress[i,:].min() + stress_amp[i] / 2
            stress_eq[i] = (stress_amp[i] * s_u) / (s_u - np.abs(stress_mean[i]))

        outputs["stress_amplitude"] = stress_amp
        outputs["stress_mean"] = stress_mean
        outputs["stress_equivalent"] = stress_eq

        # Max stress amplitude point
        stress_amp_max_ind = stress_amp.argmax()
        stress_amp_max = stress_amp[stress_amp_max_ind]
        stress_amp_max_loc = gauss_points[stress_amp_max_ind,:]
        outputs["max_stress_amplitude"] = stress_amp_max
        outputs["loc_max_stress_amplitude"] = stress_amp_max_loc

        # Max mean stress point
        stress_mean_max_ind = stress_mean.argmax()
        stress_mean_max = stress_mean[stress_mean_max_ind]
        stress_mean_max_loc = gauss_points[stress_mean_max_ind,:]
        outputs["max_stress_mean"] = stress_mean_max
        outputs["loc_max_stress_mean"] = stress_mean_max_loc

        # Max equivalent stress piont
        stress_eq_max_ind = stress_eq.argmax()
        stress_eq_max = stress_eq[stress_eq_max_ind]
        stress_eq_max_loc = gauss_points[stress_eq_max_ind,:]
        outputs["max_stress_equivalent"] = stress_eq_max
        outputs["loc_max_stress_equivalent"] = stress_eq_max_loc

        # Key point stress values
        critical = inputs["critical_point"]
        dist_to_key = np.sqrt((gauss_points[:,0] - critical[0])**2 + (gauss_points[:,1] - critical[1])**2).flatten()
        nearest = dist_to_key.argmin()
        outputs["critical_stress_mean"] = stress_mean[nearest]
        outputs["critical_stress_amplitude"] = stress_amp[nearest]
        outputs["critical_stress_equivalent"] = stress_eq[nearest]
```

Table C.4: Inputs and outputs for the stress analyzer.

OpenMDAO variable	Units	Notes
Inputs		
`stress_ultimate`	lbf/ft^2	Assumed value of S_u
`stress_vm_signed`	lbf/ft^2	$S_{\mathrm{vm},s}$ array from the von Mises stress calculator
`gauss_points`	ft	Array of Gauss point locations
`critical_point`	ft	Critical point location
Outputs		
`stress_amplitude`	lbf/ft^2	S_{amp} field
`stress_mean`	lbf/ft^2	S_{mean} field
`stress_equivalent`	lbf/ft^2	S_{eq} field
`max_stress_amplitude`	lbf/ft^2	Maximum value of S_{amp}
`max_stress_mean`	lbf/ft^2	Maximum value of S_{mean}
`max_stress_equivalent`	lbf/ft^2	Maximum value of S_{eq}
`loc_max_stress_amplitude`	ft	Location of maximum value of S_{amp}
`loc_max_stress_mean`	ft	Location of maximum value of S_{mean}
`loc_max_stress_equivalent`	ft	Location of maximum value of S_{eq}
`critical_stress_amplitude`	lbf/ft^2	S_{amp} at the critical stress point
`critical_stress_mean`	lbf/ft^2	S_{mean} at the critical stress point
`critical_stress_equivalent`	lbf/ft^2	S_{eq} at the critical stress point

REFERENCES

[1] F. D. Harris, *Introduction to Autogyros, Helicopters, and Other V/STOL Aircraft*. National Aeronautics and Space Administration, May 2012, vol. 2.

[2] H. K. Reddick Jr., "Army helicopter cost drivers," U. S. Army Air Mobility Research and Development Laboratory, Tech. Rep., Aug. 1975.

[3] F. D. Harris, "Operating costs," in *Introduction to Autogyros, Helicopters, and Other V/STOL Aircraft*, vol. 2, National Aeronautics and Space Administration, May 2012, ch. 2.9, pp. 579–662.

[4] D. C. Lombardo, "Helicopter structures—a review of loads, fatigue design techniques and usage monitoring," Aeronautical Research Laboratory, Tech. Rep. 15, May 1993.

[5] S. Kim, "Are helicopters less safe than planes?" *The Telegraph*, Oct. 2018.

[6] J. Drake et al., "The compendium report: The U.S. JHSAT baseline of helicopter accident analysis, volume I," International Helicopter Safety Team, Tech. Rep., Aug. 2011.

[7] ——, "The compendium report: The U.S. JHSAT baseline of helicopter accident analysis, volume II," International Helicopter Safety Team, Tech. Rep., Jul. 2011.

[8] L. Roskop, "Comparative report, volume 1—U.S. JHIMDAT data to U.S. JHSAT data," United States Helicopter Safety Team, Tech. Rep., Mar. 2014.

[9] ——, "Comparative report, volume 2—U.S. JHIMDAT data to U.S. JHSAT data," United States Helicopter Safety Team, Tech. Rep., Aug. 2014.

[10] F. D. Harris, "Accident record," in *Introduction to Autogyros, Helicopters, and Other V/STOL Aircraft*, vol. 2, National Aeronautics and Space Administration, May 2012, ch. 2.10, pp. 663–698.

[11] P. Tobias, "Assessing product reliability," in *NIST/SEMATECH e-Handbook of Statistical Methods*, C. Croarkin and P. Tobias, Eds., NIST, Oct. 2013, ch. 8, p. 8.1.2.4.

[12] Uber, *Fast-forwarding to a future of on-demand urban air transportation*, Web, Uber Elevate, Oct. 2016.

[13] Booz Allen Hamilton, *Executive briefing: Urban air mobility (UAM) market study*, Web, Presentation, Oct. 2018.

[14] National Aeronautics and Space Administration, *Urban air mobility (UAM) market study*, Web, Presentation, Nov. 2018.

[15] R. McDonald and B. German, "eVTOL stored energy overview," in *Uber Elevate Summit 2017*, Dallas, Texas, Apr. 2017.

[16] M. Avera, "Vortex particle analysis of side-by-side overlapping rotors in forward flight," in *AHS 73rd Annual Forum*, Fort Worth, Texas, May 2017.

[17] Y. Ito and N. Furue, "Design-oriented study on the public acceptance of cargo eVTOL aircraft as a revolutionary vehicle concept," in *Vertical Flight Society's 75th Annual Forum & Technology Display*, Vertical Flight Society, May 2019.

[18] N. Zart, *Opener officially launches single-person EVTOL personal aerial vehicle, BlackFly*, Web, Jul. 2018.

[19] SureFly, *SureFly personal eVTOL air vehicle*, Web, 2018.

[20] Grug Group, *Grug group personal eVTOL jet*, Web, 2018.

[21] A. T. Bellocchio, "A framework to enable rotorcraft maintenance free operating periods," Ph.D. dissertation, Georgia Institute of Technology, May 2018.

[22] D. P. Davies, S. L. Jenkins, and F. R. Belben, "Survey of fatigue failures in helicopter components and some lessons learnt," *Engineering Failure Analysis*, vol. 32, pp. 134–151, Sep. 2013.

[23] G. S. Campbell and R. T. C. Lahey, "A survey of serious aircraft accidents involving fatigue fracture, vol. 2: Rotary-wing aircraft," National Aeronautical Establishment, Tech. Rep., Apr. 1983.

[24] ——, "A survey of serious aircraft accidents involving fatigue fracture," *International Journal of Fatigue*, vol. 6, no. 1, pp. 25–30, Jan. 1984.

[25] M. Florian and J. Sørensen, "Wind turbine blade life-time assessment model for preventive planning of operation and maintenance," *Journal of Marine Science and Engineering*, vol. 3, no. 3, pp. 1027–1040, Sep. 2015.

[26] K. O. Ronold, J. Wedel-Heinen, and C. J. Christensen, "Reliability-based fatigue design of wind-turbine rotor blades," *Engineering Structures*, vol. 21, no. 12, pp. 1101–1114, Dec. 1999.

[27] H. S. Toft and J. D. Sørensen, "Reliability-based design of wind turbine blades," *Structural Safety*, vol. 33, no. 6, pp. 333–342, Sep. 2011.

[28] J. D. Anderson Jr., "The philosophy of airplane design," in *Aircraft Performance & Design*, McGraw-Hill, 2010, ch. 7, pp. 379–396.

[29] W. Johnson, "Design," in *Helicopter Theory*, Dover Publications Inc., 1994, ch. 7, pp. 313–343.

[30] G. J. Leishman, "Conceptual design of helicopters," in *Principles of Helicopter Aerodynamics*, Cambridge University Press, 2006, ch. 6, pp. 277–346.

[31] A. Bagai, "Aerodynamic design of the X2 Technology Demonstrator main rotor blade," in *64th Annual Forum of the American Helicopter Society, International*, May 2008.

[32] R. Blackwell and T. Millott, "Dynamics design characteristics of the Sikorsky X2 Technology Demonstrator aircraft," in *American Helicopter Society 64th Annual Forum*, American Helicopter Society, Apr. 2008.

[33] R. W. Arden, D. P. Chappell, and H. K. Reddick, "Development and qualification procedures," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 5, pp. 207–238.

[34] D. N. Mavris, *Fixed wing aircraft design I: Classical design methods*, Web, Class notes, Aug. 2017.

[35] W. Johnson, "NDARC: NASA design and analysis of rotorcraft," NASA Ames Research Center, Moffett Field, California, Tech. Rep. TP-2009-215402, Dec. 2009.

[36] ——, "NDARC: NASA design and analysis of rotorcraft—theory," NASA Ames Research Center, Moffett Field, California, Tech. Rep. TP–20220000355/Vol 1, Jan. 2022.

[37] F. D. Harris and M. P. Scully, "Rotorcraft cost too much," *Journal of the American Helicopter Society*, vol. 43, no. 1, pp. 3–13, Jan. 1998.

[38] R. Scott, "A new rotorcraft design framework based on reliability and cost," Ph.D. dissertation, Georgia Institute of Technology, Aug. 2016.

[39] J. K. Price, S. Ashok, R. Armstrong, K. B. Collins, D. Mavris, and D. Schrage, "Integrated discrete-event simulation environment for analysis of rotorcraft reliability, availability, and maintainability," in *AHS International 73rd Annual Forum & Technology Display*, Vertical Flight Society, May 2017.

[40] R. Scott, D. Schrage, and A. Sirirojvisuth, "Development of a rotorcraft lifecycle cost model incorporating reliability and maintainability with application to rotorcraft preliminary design," in *AHS 69th Annual Forum*, May 2013.

[41] R. Scott, "Conceptual design & affordability assessment of a highly reliable helicopter," in *AHS Specialists' Meeting, Capability and Affordability in the Future of the Vertical Lift Industry*, Sep. 2015.

[42] ——, "Reliability-focused design of advanced rotorcraft configurations," in *AHS 72nd Annual Forum*, May 2016.

[43] R. Biggs and J. Key, *PC-based development and recurring cost model users' guide*, 2001.

[44] S. Bhattacharya, V. Nagaraju, L. Fiondella, E. Spero, and A. Ghoshal, "Rotorcraft tradespace exploration incorporating reliability engineering," in *AHS 71st Annual Forum*, 2015.

[45] ——, "Process improvement for rotorcraft tradespace exploration incorporating reliability and availability," in *AHS 72nd Annual Forum*, American Helicopter Society International, Inc., May 2016.

[46] S. Bhattacharya, V. Nagaraju, E. Spero, A. Ghoshal, and L. Fiondella, "Incorporating quantitative reliability engineering measures into tradespace exploration," *Research in Engineering Design*, vol. 29, no. 4, pp. 589–603, Jul. 2018.

[47] W. Johnson, "A history of rotorcraft comprehensive analyses," NASA Ames Research Center, Moffett Field, CA, Tech. Rep. TP-2012-216012, Apr. 2012.

[48] M. J. Smith and A. Moushegian, "Dual-solver hybrid computational approaches for design and analysis of vertical lift vehicles," *The Aeronautical Journal*, vol. 126, no. 1295, pp. 187–208, Dec. 2021.

[49] I. C. Wilbur, A. Moushegian, M. J. Smith, and G. R. Whitehouse, "UH-60A rotor analysis with an accurate dual-formulation hybrid aeroelastic methodology," *Journal of Aircraft*, vol. 57, no. 1, pp. 113–127, Jan. 2020.

[50] F. Tarzanin and D. Young, "Boeing rotorcraft experience with rotor design and optimization," in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, Sep. 1998.

[51] J. Sinsay and G. Nuñez, "Toward right-fidelity rotorcraft conceptual design," in *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, Apr. 2010.

[52] J. D. Sinsay and J. J. Alonso, "Optimization of a lift-offset compound helicopter in a multidisciplinary analysis environment," in *AHS 71st Annual Forum*, May 2015.

[53] K. Collins et al., "Toward a high-fidelity helicopter rotor redesign framework," in *AHS International 64th Annual Forum & Technology Display*, American Helicopter Society International, Inc., May 2008.

[54] K. B. Collins, "A multi-fidelity framework for physics based rotor blade simulation and optimization," Ph.D. dissertation, Georgia Institute of Technology, 2008.

[55] K. Collins and L. Sankar, "Application of low and high fidelity simulation tools to helicopter rotor blade optimization," in *AHS International 65th Annual Forum & Technology Display*, American Helicopter Society International, Inc., May 2009.

[56] F. Liard, "Helicopter fatigue design guide," NATO Advisory Group for Aerospace Research and Development, Neuilly sur Seine, France, Tech. Rep. AGARD-AG-292, Nov. 1983.

[57] S. Suresh, "Introduction and overview," in *Fatigue of Materials*, Cambridge University Press, 1998, ch. 1, pp. 1–32.

[58] J. Degrieck and W. van Paepegem, "Fatigue damage modeling of fibre-reinforced composite materials: Review," *Applied Mechanics Reviews*, vol. 54, no. 4, p. 279, 2001.

[59] M. A. Miner, "Cumulative damage in fatigue," *Journal of Applied Mechanics*, vol. 12, 1945.

[60] R. Cansdale, "Service load monitoring and structural integrity evaluation," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 6, pp. 239–248.

[61] F. Och, "Fatigue strength," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 4.1, pp. 109–132.

[62] R. P. L. Nijssen, "Fatigue life prediction and strength degradation of wind turbine rotor blade composites," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2006-7810P, Nov. 2006.

[63] Z. Hashin and A. Rotem, "A cumulative damage theory of fatigue failure," *Materials Science and Engineering*, vol. 34, no. 2, pp. 147–160, Jul. 1978.

[64] J. R. Schaff and B. D. Davidson, "Life prediction methodology for composite structures. part I—constant amplitude and two-stress level fatigue," *Journal of Composite Materials*, vol. 31, no. 2, pp. 128–157, Jan. 1997.

[65] ——, "Life prediction methodology for composite structures. part II—spectrum fatigue," *Journal of Composite Materials*, vol. 31, no. 2, pp. 158–181, Jan. 1997.

[66] L. Li, V. V. Volovoi, and D. H. Hodges, "Structural design against fatigue failure for composite rotor blades," in *64th Annual Forum and Technology Display of the American Helicopter Society International*, Apr. 2008.

[67] A. Salvetti, G. Cavallini, and A. Fediani, "Fracture mechanics," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 4.2, pp. 133–178.

[68] A. Shah, M. Ruzzene, and J. J. Rimoli, "Influence of cycle counting methods on fatigue life estimation of critical rotorcraft components," in *Vertical Flight Society 75th Annual Forum & Technology Display*, AHS - The Vertical Flight Society, May 2019.

[69] J. B. de Jonge, "The analysis of load-time histories by means of counting methods," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 3.4, pp. 89–106.

[70] F. D. Harris, "Vibration," in *Introduction to Autogyros, Helicopters, and Other V/STOL Aircraft*, vol. 2, National Aeronautics and Space Administration, May 2012, ch. 2.6, pp. 315–436.

[71] W. N. Twelvetrees, "The evolution of the rotor blade," *Aircraft Engineering and Aerospace Technology*, vol. 41, no. 7, pp. 19–23, Jul. 1969.

[72] J. F. Ward and L. H. Ludi, "A review of current helicopter loads research as applied to the problem of rotor-blade fatigue substantiation," in *AHS 17th Annual Forum*, May 1961.

[73] R. A. Everett Jr., F. D. Bartlett Jr., and W. Elber, "Probabilistic fatigue methodology for six nines reliability," NASA Langley Research Center, Tech. Rep. NASA-TM-102757, Dec. 1990.

[74] A. Facchin and M. Raggi, "Statistical basis of data processing," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 4.4, pp. 193–206.

[75] G. Stievenard, "Mission spectra," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 3.1, pp. 19–28.

[76] L. Zion, "Predicting fatigue loads using regression diagnostics," in *The American Helicopter Society 50th Annual Forum*, May 1994.

[77] A. Jorio, "Load spectra: Measurement techniques," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. 3.3.1, pp. 63–80.

[78] H. L. Zion, "Safe life reliability: Evaluation of new statistical methods," in *American Helicopter Society 47th Annual Forum*, May 1991.

[79] A. E. Thompson and D. O. Adams, "A computational method for the determination of structural reliability of helicopter dynamic components," in *American Helicopter Society 46th Annual Forum*, May 1990.

[80] K. B. Amer, "A 'new' philosophy of structural reliability, fail safe versus safe life — the 1988 Alexander A. Nikolsky lecture," in *44th Annual National Forum of the American Helicopter Society*, Jun. 1988.

[81] H. K. Reddick Jr., "Safe-life and damage-tolerant design approaches for helicopter structures," US Army Research and Technology Laboratories; Applied Technology Laboratory, Tech. Rep., Aug. 1983.

[82] Army Materiel Command, "Engineering design handbook—helicopter engineering—part one: Preliminary design," U.S. Army Materiel Command, Alexandria, VA, Tech. Rep. AMCP 706-201, Aug. 1974.

[83] A. A. ten Have, "HELIX and FELIX: Loading standards for use in the fatigue evaluation of helicopter rotor components," in *Helicopter Fatigue Design Guide*, AGARD-AG-292, F. Liard, Ed., Neuilly sur Seine, France: NATO Advisory Group for Aerospace Research and Development, Nov. 1983, ch. A, pp. 249–270.

[84] P. R. Edwards and J. Darts, "Standardized fatigue loading sequences for helicopter rotors (HELIX and FELIX)—Part 1: Background and fatigue evaluation," Royal Aircraft Establishment, Tech. Rep. 84084, Aug. 1984.

[85] ——, "Standardized fatigue loading sequences for helicopter rotors (HELIX and FELIX)—Part 2: Final definition of HELIX and FELIX," Royal Aircraft Establishment, Tech. Rep. 84085, Aug. 1984.

[86] R. W. Arden, "Hypothetical fatigue life problem," in *Specialists Meeting on Helicopter Fatigue Methodology*, American Helicopter Society, Mar. 1980.

[87] L. Li, "Structural design of composite rotor blades with consideration of manufacturability, durability, and manufacturing uncertainties," Ph.D. dissertation, Georgia Institute of Technology, Aug. 2008.

[88] W. Yu, D. H. Hodges, and J. C. Ho, "Variational asymptotic beam sectional analysis – an updated version," *International Journal of Engineering Science*, vol. 59, pp. 40–64, Oct. 2012.

[89] O. Bauchau, *Dymore user's manual*, 2010.

[90] J. Arruda, L. Hamel, and K. Collins, "A method for quantitative technology analysis of active rotor technologies," in *American Helicopter Society 67th Annual Forum*, American Helicopter Society International, Inc., May 2011.

[91] B. German, *Surrogate modeling*, Class Notes, 2018.

[92] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response surface methodology: process and product optimization using designed experiments*, 3rd ed. Hoboken, N.J: John Wiley & Sons, Inc., 2009.

[93] T. C. Schank, "Optimal aeroelastic trim for rotorcraft with constrained, non-unique trim solutions," Ph.D. dissertation, Georgia Institute of Technology, Apr. 2008.

[94] N. Lappos, *Nikolsky lecture: Design advantages of an integrated cyber-physical aircraft*, Web, Presentation, May 2019.

[95] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, "Introduction," in *Response surface methodology: process and product optimization using designed experiments*, 3rd ed., Hoboken, N.J: John Wiley & Sons, Inc., 2009, ch. 1, pp. 1–12.

[96] D. N. Mavris, *Design-of-experiments for practical applications in modeling, simulation, and analysis—introduction to response surface methods*, Web, Class notes, Aug. 2017.

[97] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, "Building empirical models," in *Response surface methodology: process and product optimization using designed experiments*, 3rd ed., Hoboken, N.J: John Wiley & Sons, Inc., 2009, ch. 2, pp. 13–79.

[98] C. C. Aggarwal, "An introduction to neural networks," in *Neural Networks and Deep Learning*, Springer-Verlag GmbH, 2018, ch. 1, pp. 1–52.

[99] B. Bagdatli, *Artificial neural networks as a non-linear regression method example*, Web, Class notes, Aug. 2017.

[100] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer-Verlag GmbH, 2018.

[101] B. German and C. Heller, *Gaussian process models*, Web, Class notes, Aug. 2017.

[102] C. E. Rasmussen and C. K. I. Williams, "Introduction," in *Gaussian Processes for Machine Learning*, Cambridge, Mass: MIT Press, 2006, ch. 1, pp. 1–6.

[103] ——, "Regression," in *Gaussian Processes for Machine Learning*, Cambridge, Mass: MIT Press, 2006, ch. 2, pp. 7–32.

[104] ——, "Model selection and adaptation of hyperparameters," in *Gaussian Processes for Machine Learning*, Cambridge, Mass: MIT Press, 2006, ch. 5, pp. 105–128.

[105] ——, *Gaussian Processes for Machine Learning*. Cambridge, Mass: MIT Press, 2006.

[106] Department of the Army, "Operator's manual for UH-60A helicopter, UH-60L helicopter, EH-60A helicopter," Headquarters, Department of the Army, Washington, D.C., Tech. Rep. TM 1-1520-237-10, Oct. 1996.

[107] L. A. Meyn, "Rotorcraft optimization tools: Incorporating rotorcraft design codes into multi-disciplinary design, analysis, and optimization," in *AHS Technical Conference on Aeromechanics Design for Transformative Vertical Flight*, Jan. 2018.

[108] W. Johnson, "NDARC — NASA design and analysis of rotorcraft: Validation and demonstration," in *American Helicopter Society Aeromechanics Specialists' Conference*, San Francisco, CA: Vertical Flight Society, Jan. 2010.

[109] H. Saberi, M. Khoshlahjeh, R. A. Ormiston, and M. J. Rutkowski, "Overview of RCAS and application to advanced rotorcraft problems," in *AHS 4th Decennial Specialist's Conference on Aeromechanics*, San Francisco, California, Jan. 2004.

[110] D. H. Hodges and E. H. Dowell, "Nonlinear equations of motion for the elastic bending and torsion of twisted nonuniform rotor blades," NASA Ames Research Center, Moffett Field, California, Tech. Rep. NASA-TN-D-7818, Dec. 1974.

[111] D. H. Hodges, *Nonlinear Composite Beam Theory*, F. K. Lu, Ed. 1801 Alexander Bell Drive, Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006, 304 pp.

[112] G. S. Bir, "Structural dynamics verification of rotorcraft comprehensive analysis system (RCAS)," National Renewable Energy Laboratory, 1617 Cole Boulevard, Golden, Colorado 80401-3393, Tech. Rep. NREL/TP-500-35328, Feb. 2005.

[113] W. Yu, *VABS manual for users*, Web.

[114] S. Tian, X. Liu, and W. Yu, *PreVABS*, Web, Nov. 2017.

[115] W. Yu, V. Volovoi, D. Hodges, and X. Hong, "Validation of the variational asymptotic beam sectional analysis (VABS)," in *19th AIAA Applied Aerodynamics Conference*, American Institute of Aeronautics and Astronautics, Jun. 2001.

[116] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor, "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, vol. 59, no. 4, pp. 1075–1104, Mar. 2019.

[117] P. J. Rohl, C. E. S. Cesnik, P. Dorman, and K. Kumar, "IXGEN — a modeling tool for the preliminary design of composite rotor blades," in *American Helicopter Society Future Vertical Lift Aircraft Design Conference*, San Francisco, CA: American Helicopter Society International, Inc., Jan. 2012.

[118] D. Kumar, "Design and analysis of composite rotor blades for active/passive vibration reduction," Ph.D. dissertation, The University of Michigan, 2013.

[119] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[120] M. Lemaire, "Simulation methods," in *Structural Reliability*, J. Mazars, Ed., London Hoboken, NJ: ISTE Wiley, 2009, ch. 7, pp. 233–263.

[121] O. Ditlevsen and H. O. Madsen, *Structural Reliability Methods*. John Wiley & Sons Ltd., 1996.

[122] M. Lemaire, *Structural Reliability*, J. Mazars, Ed. London Hoboken, NJ: ISTE Wiley, 2009.

[123] A. E. Mansour, "An introduction to structural reliability theory," Mansour Engineering, Inc, Berkely, CA 94708, Tech. Rep. SSC-351, Jan. 1990.

[124] M. Lemaire, "Elementary $R - S$ case," in *Structural Reliability*, J. Mazars, Ed., London Hoboken, NJ: ISTE Wiley, 2009, ch. 3, pp. 39–76.

[125] ——, "Isoprobablistic transformation," in *Structural Reliability*, J. Mazars, Ed., London Hoboken, NJ: ISTE Wiley, 2009, ch. 4, pp. 77–114.

[126] ——, "Probability of failure," in *Structural Reliability*, J. Mazars, Ed., London Hoboken, NJ: ISTE Wiley, 2009, ch. 7, pp. 165–232.

[127] P. Bjerager, “Probability integration by directional simulation,” *Journal of Engineering Mechanics*, vol. 114, no. 8, pp. 1285–1302, Aug. 1988.

[128] M. Baudin, A. Dutfoy, B. Iooss, and A.-L. Popelin, *OpenTURNS: An industrial software for uncertainty quantification in simulation*, 2015.

[129] M. Lemaire, “Mechanical–reliability coupling,” in *Structural Reliability*, J. Mazars, Ed., London Hoboken, NJ: ISTE Wiley, 2009, ch. 11, pp. 341–391.

[130] P. V. Shevchenko, “Calculation of aggregate loss distributions,” *The Journal of Operational Risk*, vol. 5, no. 2, pp. 3–40, May 2010.

[131] R. C. S. Freire Júnior and A. S. Belísio, “Probabilistic S–N curves using exponential and power laws equations,” *Composites Part B: Engineering*, vol. 56, pp. 582–590, 2014.

[132] B. Harris, N. Gathercole, J. A. Lee, H. Reiter, and T. Adam, “Life–prediction for constant–stress fatigue in carbon–fibre composites,” *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 355, no. 1727, pp. 1259–1294, Jun. 1997.

[133] C. Silva, W. R. Johnson, E. Solis, M. D. Patterson, and K. R. Antcliff, “VTOL urban air mobility concept vehicles for technology development,” in *2018 Aviation Technology, Integration, and Operations Conference*, American Institute of Aeronautics and Astronautics, Jun. 2018.

[134] T. Lieu, C. Farhat, and M. Lesoinne, “Reduced-order fluid/structure modeling of a complete aircraft configuration,” *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 41-43, pp. 5730–5742, Aug. 2006.

[135] D. Xiao, P. Yang, F. Fang, J. Xiang, C. Pain, and I. Navon, “Non-intrusive reduced order modelling of fluid–structure interactions,” *Computer Methods in Applied Mechanics and Engineering*, vol. 303, pp. 35–54, May 2016.

[136] K. C. Kim, “Analytical investication into the helicopter vibration resulting from main rotor blade (MRB) ballistic damage,” Army Research Laboratory, Aberdeen Proving Ground, MD 21005-5068, Tech. Rep. ARL-TR-1985, Jun. 1999.

[137] J. Slavič, M. Mršnik, M. Česnik, J. Javh, and M. Boltežar, “Multiaxial vibration fatigue,” in *Vibration Fatigue by Spectral Methods*, Elsevier, 2021, pp. 115–126.

VITA

Joseph Nathaniel Robinson was born in Los Gatos, California in April 1995. He attended Loma Prieta Elementary School, C.T. English Middle School, and Los Gatos High School, where he was inspired to persue a career in the aerospace industry. In 2013, he moved to Atlanta, Georgia to attend the Georgia Institute of Technology, majoring in aerospace engineering. He completed his bachelor's coursework in 2017, earned a master's degree in 2018, and finished his doctorate degree in 2022. Joseph currently lives in Herndon, Virginia with his girlfriend, Rachel, and their dog, Tucker.