Low-Tip-Speed High-Torque Proprotor Noise Approximation for Design Cycle Analysis

Xavier G. Santacruz

Follow this and additional works at: https://commons.erau.edu/edt

Scholarly Commons Citation
https://commons.erau.edu/edt/451

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu, wolfs309@erau.edu.
LOW-TIP-SPEED HIGH-TORQUE PROPRORO

NOISE APPROXIMATION FOR DESIGN CYCLE ANALYSIS

A Thesis

Submitted to the Faculty

of

Embry-Riddle Aeronautical University

by

Xavier G. Santacruz

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Aerospace Engineering

May 2019

Embry-Riddle Aeronautical University

Daytona Beach, Florida
LOW-TIP-SPEED HIGH-TORQUE PROPROTOR NOISE APPROXIMATION FOR DESIGN CYCLE ANALYSIS

by

Xavier G. Santacruz

A Thesis prepared under the direction of the candidate's committee chairman, Dr. Richard P. Anderson, Department of Aerospace Engineering, and has been approved by the members of the thesis committee. It was submitted to the School of Graduate Studies and Research and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

Chairman, Dr. Richard P. Anderson

Member, Dr. Anastasios Lyrintzis

Member, Dr. J. Gordon Leishman

Graduate Program Coordinator, Dr. Magdy Attia

Dean of College of Engineering, Dr. Maj Mirmirani

Senior Vice President for Academic Affairs and Provost, Dr. Lon Moeller

4.25.2019

4/25/2019

4/25/19
ACKNOWLEDGMENTS

I would like to take this opportunity to write words of appreciation to all of those who made this endeavour possible. First and foremost is to my loving wife, Karen, who endured this research as if it had been her own. Not only by tolerating long nights and reduced attention, but also by providing selfless support and encouragement throughout these years.

To my advisor, Dr. Richard Anderson, for enabling me to pursue research activities at the Eagle Flight Research Center, granting me the graduate research assistantship, and providing valuable counsel. To the members of my committee, Dr. J. Gordon Leishman and Dr. Anastasios Lyrintzis, for providing invaluable guidance and support through the learning process of subject areas which I was not familiar with at the beginning. To Dr. Kenneth Brentner, from Pennsylvania State University, for providing an executable version of his famous noise approximation algorithm PSU-WOPWOPv3, as well as aid and insight when available.

To all of my new friends who endured this process with me and without whom my time spent at the university would not have been enjoyable. To old friends who provided not only encouragement but word of advice. Lastly but most importantly, to my parents, as this degree and subsequent work would not have been possible without their constant support and stimulation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>x</td>
</tr>
<tr>
<td>ABBREVIATIONS</td>
<td>xii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xiii</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Literature Review</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Rotating Blade Noise</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Propeller Theory</td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 Rotating Blade Aerodynamics</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Problem Statement</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Objectives</td>
<td>9</td>
</tr>
<tr>
<td>1.5 Presented Work Outline</td>
<td>10</td>
</tr>
<tr>
<td>Methodology</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Theory</td>
<td>12</td>
</tr>
<tr>
<td>2.1.1 Rotor Aeroacoustics</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2 Blade Element Momentum Theory</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Implementation Overview</td>
<td>24</td>
</tr>
<tr>
<td>2.2.1 PSU-WOPWOP</td>
<td>25</td>
</tr>
<tr>
<td>2.2.2 XFOIL</td>
<td>25</td>
</tr>
<tr>
<td>2.2.3 Interface</td>
<td>26</td>
</tr>
<tr>
<td>2.2.4 Flow Diagrams</td>
<td>27</td>
</tr>
<tr>
<td>2.3 Parameter Sensitivity</td>
<td>30</td>
</tr>
<tr>
<td>2.3.1 Grid Resolution</td>
<td>30</td>
</tr>
<tr>
<td>2.3.2 Data Points Sensitivity</td>
<td>36</td>
</tr>
<tr>
<td>2.3.3 Time step Sensitivity</td>
<td>38</td>
</tr>
<tr>
<td>2.4 Validation</td>
<td>39</td>
</tr>
<tr>
<td>2.4.1 Blade Aerodynamics</td>
<td>39</td>
</tr>
<tr>
<td>2.4.2 Noise</td>
<td>44</td>
</tr>
<tr>
<td>Results and Discussion</td>
<td>48</td>
</tr>
<tr>
<td>3.1 Low RPM</td>
<td>51</td>
</tr>
<tr>
<td>3.2 Medium RPM</td>
<td>53</td>
</tr>
<tr>
<td>3.3 High RPM</td>
<td>55</td>
</tr>
<tr>
<td>Conclusion</td>
<td>60</td>
</tr>
</tbody>
</table>
4.1 Limitations .............................................. 61
4.2 Future Work ............................................ 63
REFERENCES ..................................................... 64
A Appendix - Sensitivity Analysis Tables .......................... 68
B Appendix - Algorithm Structure and Details ..................... 74
  B.1 Structure and Details .................................... 74
  B.2 Running Script ........................................... 75
  B.3 Geometry Input Scripts ................................... 77
    B.3.1 Environmental and Flight Properties ..................... 78
    B.3.2 Rotor Properties ...................................... 78
    B.3.3 Blade Properties ...................................... 79
    B.3.4 Custom Aeroacoustic Options ............................ 80
    B.3.5 Pre-defined Blade Geometries ............................ 81
  B.4 Main Script ............................................. 85
  B.5 Blade Element Theory Function .............................. 87
  B.6 XFOIL Function ........................................... 90
    B.6.1 Execute XFOIL .......................................... 91
    B.6.2 Interpolation of Aerodynamic Coefficients ............... 93
  B.7 Blade Plotting and Data Processing Function .................. 95
    B.7.1 Interpolation of $C_P$ .................................. 97
  B.8 WOPWOP Function .......................................... 98
    B.8.1 Case Files ........................................... 99
    B.8.2 Geometry Input File .................................... 99
    B.8.3 Loading Input File .................................... 104
    B.8.4 Running WOPWOP ....................................... 104
    B.8.5 Reading WOPWOP results ................................. 105
C Appendix - Source Code Listings ................................ 106
  C.1 runProp.m script ......................................... 106
  C.2 geometry.m script ........................................ 107
  C.3 blade_calculations.m script ................................ 108
  C.4 BEMT_v0b.m script ....................................... 111
  C.5 xfoil.m script ........................................... 115
  C.6 aerocalcs.m script ........................................ 122
  C.7 wopwop.m script ......................................... 126
  C.8 readBinPatch.m script ..................................... 142
  C.9 blade_plotting.m script ................................... 146
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Grid Sensitivity Test Matrix</td>
<td>31</td>
</tr>
<tr>
<td>2.2 Time Step and Data Points Sensitivity Test Matrix</td>
<td>37</td>
</tr>
<tr>
<td>3.1 Result Analysis Matrix</td>
<td>51</td>
</tr>
<tr>
<td>3.2 Perceived Noise Level - Location 1, Low RPM</td>
<td>52</td>
</tr>
<tr>
<td>3.3 Perceived Noise Level - Location 2, Low RPM</td>
<td>52</td>
</tr>
<tr>
<td>3.4 Perceived Noise Level - Location 1, Medium RPM</td>
<td>53</td>
</tr>
<tr>
<td>3.5 Perceived Noise Level - Location 2, Medium RPM</td>
<td>54</td>
</tr>
<tr>
<td>3.6 Perceived Noise Level - Location 1, High RPM</td>
<td>55</td>
</tr>
<tr>
<td>3.7 Perceived Noise Level - Location 2, High RPM</td>
<td>57</td>
</tr>
<tr>
<td>3.8 Perceived Noise Level - Location 3, High RPM</td>
<td>58</td>
</tr>
<tr>
<td>A.1 Grid Sensitivity Total Run Time, No Weighting</td>
<td>68</td>
</tr>
<tr>
<td>A.2 Grid Sensitivity Total Run Time, Tip Weighted</td>
<td>68</td>
</tr>
<tr>
<td>A.3 Grid Sensitivity Total Run Time, No Bias vs. Tip Bias</td>
<td>69</td>
</tr>
<tr>
<td>A.4 $\Delta \tau$ and Data Point Sensitivity Total Run Time</td>
<td>69</td>
</tr>
<tr>
<td>A.5 Observer time and $\Delta \tau$ Sensitivity Total Run Time</td>
<td>69</td>
</tr>
<tr>
<td>A.6 Grid Sensitivity Peak Loading Pressure, No Weighting</td>
<td>70</td>
</tr>
<tr>
<td>A.7 Grid Sensitivity Peak Loading Pressure, No Weighting</td>
<td>70</td>
</tr>
<tr>
<td>A.8 Grid Sensitivity Peak Loading Pressure, No Bias vs. Tip Bias</td>
<td>70</td>
</tr>
<tr>
<td>A.9 $\Delta \tau$ and Data Point Sensitivity Peak Loading Pressure</td>
<td>71</td>
</tr>
<tr>
<td>A.10 Grid Sensitivity Peak Total Pressure, No Weighting</td>
<td>71</td>
</tr>
<tr>
<td>A.11 Grid Sensitivity Peak Total Pressure, No Weighting</td>
<td>71</td>
</tr>
<tr>
<td>A.12 Grid Sensitivity Peak Total Pressure, No Bias vs. Tip Bias</td>
<td>72</td>
</tr>
<tr>
<td>A.13 $\Delta \tau$ and Data Point Sensitivity Peak Total Pressure</td>
<td>72</td>
</tr>
<tr>
<td>A.14 $\Delta \tau$ and Data Point Sensitivity - PNL</td>
<td>72</td>
</tr>
<tr>
<td>A.15 Observer time and $\Delta \tau$ Sensitivity - PNL</td>
<td>73</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Modified aircraft for quiet flight based mainly on propeller changes (Vogeley, 1949)</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Velocities and aerodynamic forces acting on the blade element (Leishman, 2006)</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Annulus of rotor disk as used for the local momentum analysis of a hovering rotor / static propeller (Leishman, 2006)</td>
<td>21</td>
</tr>
<tr>
<td>2.3 Top Level Script Flow Diagram</td>
<td>28</td>
</tr>
<tr>
<td>2.4 Blade Element Analysis Script Flow Diagram</td>
<td>29</td>
</tr>
<tr>
<td>2.5 Aeroacoustic Analysis Script Flow Diagram</td>
<td>30</td>
</tr>
<tr>
<td>2.6 Grid resolution sensitivity analysis for the same blade, same operating conditions, with different spanwise (S) and fixed chordwise (P) elements, with constant element width (no bias)</td>
<td>33</td>
</tr>
<tr>
<td>2.7 Grid resolution sensitivity analysis for the same blade, same operating conditions, same spanwise elements (100), with varying chordwise (P) elements</td>
<td>34</td>
</tr>
<tr>
<td>2.8 Linear span-wise element distribution versus tip-biased element spacing. Sample image with 10 elements (spanwise) and 20 chordwise panels.</td>
<td>34</td>
</tr>
<tr>
<td>2.9 Grid resolution sensitivity analysis to element distribution weighting for the same blade, same operating conditions, same spanwise (100) and chordwise (200) elements</td>
<td>35</td>
</tr>
<tr>
<td>2.10 Grid resolution sensitivity analysis for the same blade, same operating conditions, same spanwise elements (100), with varying chordwise (P) elements and tip-biased element width distribution</td>
<td>35</td>
</tr>
<tr>
<td>2.11 SPL frequency distribution sensitivity test to variable $\Delta \tau$ and $nT$</td>
<td>37</td>
</tr>
<tr>
<td>2.12 Effect on varying time step sensitivity to SPL frequency distribution</td>
<td>38</td>
</tr>
<tr>
<td>2.13 Calculated pressure coefficient compared against experimentally measured pressure on a NACA0012 blade (Caradonna &amp; Tung, 1981) (a)-(e). Operating conditions: $C_T = 0.00213$, $\Omega = 1250$, $M_{Tip} = 0.434$</td>
<td>40</td>
</tr>
<tr>
<td>2.14 Calculated pressure coefficient compared against experimentally measured pressure on a NACA0012 blade (Caradonna &amp; Tung, 1981) (a)-(e). Operating conditions: $C_T = 0.00455$, $\Omega = 1750$, $M_{Tip} = 0.612$</td>
<td>42</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>2.15 Comparison of calculated $c_p$ distribution vs. extracted from Suresh et al. (2018)</td>
<td>43</td>
</tr>
<tr>
<td>2.16 Calculated compact pressure against extracted from WOPWOP Sample Case 1 (Hennes et al., 2017)</td>
<td>43</td>
</tr>
<tr>
<td>2.17 Comparison of calculated noise pressures vs. extracted from Suresh et al. (Suresh et al., 2018) (a)(b) and Suresh et al and Boxwel et al. (1978) (c)</td>
<td>45</td>
</tr>
<tr>
<td>2.18 Calculated Sound Pressures and SPL compared against sample case results from PSU-WOPWOPv3 distribution (Hennes et al., 2017)</td>
<td>46</td>
</tr>
<tr>
<td>3.1 Rendered views of the SAB Goblin TB 700 rotor blade. Chordwise view (a) is looking inboard from the tip. Planform view (b) has the leading edge towards the bottom.</td>
<td>49</td>
</tr>
<tr>
<td>3.2 Rendered views of the EFRC GOE550 &quot;Hantz&quot; (Hantz, 2018) proprotor blade. Chorwise view (a) is looking inboard from the tip. Planform view (b) has the leading edge towards the bottom.</td>
<td>50</td>
</tr>
<tr>
<td>3.3 Observer position depiction with respect to rotor disk.</td>
<td>50</td>
</tr>
<tr>
<td>3.4 Low RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>52</td>
</tr>
<tr>
<td>3.5 Low RPM test cases. Microphone positioned at 1.5R, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>53</td>
</tr>
<tr>
<td>3.6 Medium RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>54</td>
</tr>
<tr>
<td>3.7 Medium RPM test cases. Microphone positioned at 1.5R, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>55</td>
</tr>
<tr>
<td>3.8 High RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>56</td>
</tr>
<tr>
<td>3.9 High RPM test cases. Microphone positioned at 1.5R, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b).</td>
<td>57</td>
</tr>
<tr>
<td>3.10 Third microphone location: 10ft horizontally from axis of rotation and 30° down from the TPP.</td>
<td>58</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.11 High RPM test cases. Microphone positioned at 10ft horizontally from the axis of rotation, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)</td>
<td>59</td>
</tr>
<tr>
<td>B.1 Directory structure with brief explanation</td>
<td>75</td>
</tr>
<tr>
<td>B.2 EFRC proprotor blade shape render</td>
<td>82</td>
</tr>
<tr>
<td>B.3 Goblin Model Helicopter SAB700 blade shape render</td>
<td>82</td>
</tr>
<tr>
<td>B.4 NACA Propeller No.5868 blade shape render</td>
<td>83</td>
</tr>
<tr>
<td>B.5 WopWop Sample Case 1 Gyrodyne blade shape render</td>
<td>84</td>
</tr>
<tr>
<td>B.6 UH1H 1/7th Scale blade shape render</td>
<td>85</td>
</tr>
<tr>
<td>B.7 Linear span-wise element distribution versus tip-biased element spacing</td>
<td>86</td>
</tr>
<tr>
<td>B.8 Sample output of the aerodynamic coefficient interpolation results</td>
<td>95</td>
</tr>
<tr>
<td>B.9 Sample output of the Pressure Coefficient distribution along the blade surface, shown with rotor performance parameters</td>
<td>96</td>
</tr>
<tr>
<td>B.10 Node-centered area-vector calculation scheme and demonstration</td>
<td>102</td>
</tr>
<tr>
<td>B.11 Upper surface node-centered area-vector comparison between calculated and provided results</td>
<td>103</td>
</tr>
<tr>
<td>B.12 Decoded output plot shown to the user as means to ensure the correct inputs were fed to WOPWOP for execution. Units are in SI, thus the distance is in meters and gauge pressure in Pascals</td>
<td>104</td>
</tr>
<tr>
<td>B.13 Sample result extraction from WOPWOP’s execution</td>
<td>105</td>
</tr>
</tbody>
</table>
SYMBOLS

\( \Box \) wave operator, \( \Box^2 = \frac{1}{c^2} \frac{\partial^2}{\partial t^2} - \nabla^2 \)
\( c \) speed of sound in quiescent medium
\( c_d \) section drag coefficient, \( D/\frac{1}{2}\rho U^2c \)
\( c_l \) section lift coefficient, \( L/\frac{1}{2}\rho U^2c \)
\( c_{\alpha} \) lift curve slope
\( c_p \) section pressure coefficient, \( (\Delta p)/\frac{1}{2}\rho V^2_c \)
\( C_P \) rotor power coefficient, \( P/\rho A(\Omega R)^3 \)
\( C_Q \) rotor torque coefficient, \( Q/\rho A(\Omega R)^2R \)
\( C_T \) rotor thrust coefficient, \( T/\rho A(\Omega R)^2 \)
\( c_y \) element local chord distance
\( D \) drag force
\( F_x \) aerodynamic force parallel to disk plane
\( F_z \) aerodynamic force perpendicular to disk plane
\( H(f) \) Heavyside function, \( H(f < 0) = 0 \) and \( H(f > 0) = 1 \)
\( J \) propeller advance ratio
\( l_i \) components of local force intensity that acts on the fluid, \( l_i = P_{ij}n_j \)
\( L \) lift force
\( m \) mass
\( M \) Mach number or local free-stream Mach number, \( U/a \)
\( \hat{n}_i \) component of the unit outward normal vector to surface
\( N_b \) number of rotor blades
\( nT \) points in time to measure
\( p' \) acoustic pressure
\( P_{ij} \) compressive stress tensor
\( r \) non-dimensional radial position
\( R \) blade radius
\( Re \) Reynolds number based on chord, \( \rho U_c/\mu \)
\( T_{ij} \) Lighthill stress tensor, \( \rho u_iu_j + P_{ij} - c^2p'\delta_{ij} \)
\( U \) resultant velocity at blade element, \( \sqrt{U_P^2 + U_T^2} \)
\( u_i \) component of local fluid velocity
\( u_n \) normal component of local fluid velocity
\( U_P \) out-of-plane velocity normal to rotor disk plane
\( U_R \) radial velocity along blade at disk plane
\( U_T \) in-plane velocity parallel to rotor disk plane
\( v_i \) rotor induced velocity
\( v_n \) local normal velocity of source surface
\( V \) velocity
\( V_c \) axial or climb velocity
\( x/c \) chord fraction
\( \alpha \) angle of attack
\( \alpha_i \) induced angle of attack
\( \delta(f) \) Dirac delta function
\( \Delta p \) pressure change with respect to atmospheric pressure, \( p - p_\infty \)
\( \Delta \tau \) time step
\( \lambda \) rotor inflow ratio, positive downward through disk, \( U_P/\Omega R \)
\( \lambda_c \) climb inflow ratio, \( V_c/\Omega R \)
\( \lambda_i \) induced inflow ratio, \( v_i/\Omega R \)
\( \phi \) induced inflow angle, \( \arctan(U_P/U_T) \)
\( \sigma \) rotor solidity, \( N_b c/\pi R \)
\( \theta \) blade element pitch angle
\( \theta_0 \) blade collective pitch
\( \Omega \) angular velocity
# ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEMT</td>
<td>Blade Element Momentum Theory</td>
</tr>
<tr>
<td>BET</td>
<td>Blade Element Theory</td>
</tr>
<tr>
<td>CFD</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>CFR</td>
<td>Code of Federal Regulations</td>
</tr>
<tr>
<td>EASA</td>
<td>European Aviation Safety Agency</td>
</tr>
<tr>
<td>EFRC</td>
<td>Eagle Flight Research Center</td>
</tr>
<tr>
<td>EPNL</td>
<td>Equivalent Perceived Noise Level</td>
</tr>
<tr>
<td>ERAU</td>
<td>Embry-Riddle Aeronautical University</td>
</tr>
<tr>
<td>eVTOL</td>
<td>electric Vertical Take-off and Landing</td>
</tr>
<tr>
<td>F1A</td>
<td>Farrasat’s 1A Formulation of the FW-H Equation</td>
</tr>
<tr>
<td>FAA</td>
<td>Federal Aviation Administration</td>
</tr>
<tr>
<td>FAR</td>
<td>Federal Aviation Regulations</td>
</tr>
<tr>
<td>FP</td>
<td>Fixed Point Iteration Method</td>
</tr>
<tr>
<td>FW-H</td>
<td>Ffowcs Williams - Hawkins</td>
</tr>
<tr>
<td>GA</td>
<td>General Aviation</td>
</tr>
<tr>
<td>GAMA</td>
<td>General Aviation Manufacturers Association</td>
</tr>
<tr>
<td>ICE</td>
<td>Internal Combustion Engine</td>
</tr>
<tr>
<td>NACA</td>
<td>National Advisory Committee for Aeronautics</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>RC</td>
<td>Remote Controlled</td>
</tr>
<tr>
<td>RPM</td>
<td>Revolutions Per Minute</td>
</tr>
<tr>
<td>PNL</td>
<td>Perceived Noise Level</td>
</tr>
<tr>
<td>PSU</td>
<td>Pennsylvania State University</td>
</tr>
<tr>
<td>SPL</td>
<td>Sound Pressure Level</td>
</tr>
<tr>
<td>TTP</td>
<td>Tip-Path Plane</td>
</tr>
<tr>
<td>UAM</td>
<td>Urban Air Mobility</td>
</tr>
</tbody>
</table>
ABSTRACT

Santacruz, Xavier G. MSc, Embry-Riddle Aeronautical University, May 2019. Low-Tip-Speed High-Torque Proprotor Noise Approximation for Design Cycle Analysis.

Noise reduction in aviation would enable urban missions that cannot be flown with current generation helicopters because of their noisiness. This goal can be achieved by using electric motors as they are quieter and can produce higher torque at lower RPMs. Therefore, a proprotor system can be designed to exploit this characteristic potentially abating noise levels. This research performed noise approximations included with rotor aerodynamics for a single, electric-driven, hovering proprotor by creating a code meant to be used in design cycle analysis. The approximation was based on geometry by using the blade element momentum theory, and calculating the pressure distribution along the blade surface using Drela’s XFOIL (2001). The noise approximation, performed using Brentner’s PSU-WOPWOPv3 (2017), was validated with known data obtained from previous published experimental results. These were within acceptable range of error, demonstrating the feasibility of the tool to be used in a design environment. Two rotors were analyzed, concluding that a custom designed proprotor for eVTOL applications is quieter than conventional rotor.
1. Introduction

1.1 Motivation

In a world where an hour-long commute is being regarded as the expected standard, breakthrough technologies could provide a solution to reduce time spent on the road. In particular, electric vertical take-off and landing (eVTOL) type aircraft could help achieve a reduction in road vehicle utilization at a reasonable economic cost, while reducing the user’s travel time. These vehicles would enable mobility within the urban environment by adding a third dimension to the present transit grid. Current generation helicopters cannot perform this task because their operation is restricted in certain areas due to noise limits (Holden & Goel, 2016). Therefore, new vehicles must be created to exploit this new mission requirement.

Urban Air Mobility (UAM) aircraft will employ electric propulsion, which will likely be quiet enough to operate in cities and suburbs alike (Silva et al., 2018). This assumption comes from the empirical fact that electric motors produce less noise than internal combustion engines. Further, since electric motors can decouple the rotational speed from the torque generation, electric aircraft propulsion systems can be designed to operate at low RPMs and higher torque settings (Gartenberg, 2017). Therefore, propeller or proprotor noise must be considered within the design-space of the vehicle.

Current noise approximations methods abound, ranging from high-complexity computational fluid dynamics (CFD) to simpler numerical approximations. Higher
fidelity noise approximations require extensive computational time. The lower order methods do not. However, these lower order methods could provide results qualitatively similar to those of the higher order ones. Therefore, the simpler methods can be used within a design framework seeking to optimize the noise generated by the rotor without the use of computationally intensive processes.

1.2 Literature Review

1.2.1 Rotating Blade Noise

Propeller noise has been of interest in the past. Brentner & Farassat (1994) mention that propeller noise has been well understood by the 1950’s and 60’s. However, the capabilities to predict and model the complex broadband noise components of rotor noise was achieved only in recent decades (Brentner et al., 2002). These components are caused by unsteady interactions of the flow and the rotating blade.

Historically, Gutin showed the relationship between blade loading and noise generation in the 1930’s (Gutin, 1948). In the 1940’s, Demming provided a connection between blade thickness and its associated noise (Deming, 1940). By the 1950’s, relationships between tip speed, solidity, and disk loading were already known, inasmuch as aircraft were designed and tested to produce 20dB less than an unmodified counterpart, mainly with the use of altered propeller geometry (Vogeley, 1949).

Vogeley’s (1949) noise-reduced aircraft can be seen in Figure 1.1, where the increase in the number of blades, change of blade shape, and ensuing reduction in RPM, led to the significant noise abatement mentioned above. By the 1970s, propeller noise was considered predictable through empirical curve-fitting models, which are
still used industry-wide to this day (Barry & Magliozi, 1971; Worobel & Mayo, 1971).

![Figure 1.1](image1.png)

(a) Unmodified Stinson L-5E Sentinel Aircraft
(b) Modified Quiet-Flight L-5E
(c) Modified quiet blade

*Figure 1.1 Modified aircraft for quiet flight based mainly on propeller changes (Vogeley, 1949)*

However, all these approximations and noise calculations only pertain to axial flow through an actuator disk. Helicopter rotor noise is a different problem of its own due to the unsteady flow occurring throughout the rotor disk in flight (Brentner & Farassat, 1994). Rotor noise components are separated into two types: discrete and broadband (Farassat & Brentner, 1998). Loading and thickness noise are part of the discrete type. In addition to those previously mentioned, also included are high impulsive noise generated by high tip-speeds, and blade-vortex interaction (Brentner, 1986). Broadband noise is generated by unsteady flow components due to the various
interactions of the rotating blade with cyclic and collective pitch controls in forward flight (Brentner & Farassat, 2003).

Rotor noise started to gain attention in the 1960’s; and, the first predicting methods specifically for helicopters were developed by Lowson (1965) and Wright (1969). Then came Ffowcs Williams and Hawkins (1969), who introduced their now famous acoustic noise analogy equation to solve for sounds being generated by objects in motion within a fluid. Farrasat (2007) derived a formulation of the FW-H equations, known as Farrasat’s 1A formulation (F1A). The F1A formulation manipulated the FW-H in the retarded time frame, and took the time derivatives inside the first integral for a fixed observer position, (Farassat, 2007), making the numerical evaluation of the resulting integrals easier.

As more computing power became available, complex algorithms were developed to perform aeroacoustic calculations (Brentner & Farassat, 2003). Caradonna and Tung were the first ones to take advantage of such computing power (Brentner, 1986). In the 1980’s, they showed the importance of the quadrupole term of the FW-H equation for transonic flow, which was usually neglected before due to lack of resolved flow field information around the rotor (Caradonna & Tung, 1981). Currently, prediction capabilities of rotor noise are fairly accurate (Ingraham et al., 2019). Lately, due to the interest garnered by the UAM community seeking lower noise signatures on their vehicles, there has been a renewed interest in noise prediction and reduction methods for rotating blades (Zawodny et al., 2016; Brentner et al., 2018).
Brentner et al. (2002) created PSU-WOPWOP as a rotor aeroacoustic prediction tool, built upon Farrasat’s 1A formulation of the FW-H equations. Originally developed in the 1980’s as WOPWOP for NASA, it has evolved into an entirely different tool at Pennsylvania State University (PSU). Its capabilities have been expanded to include better predictions of broadband noise and unsteady sources (Goldman, 2012). Hennes et al. (2017) have thoroughly documented its proper use and Shirey et al. (2007) have provided extensive validation to the tool’s predictive capabilities. Therefore, it has been chosen as a component to this research for noise prediction even though most of these capabilities are not used in this work.

1.2.2 Propeller Theory

Propellers, having been initially the only method of aircraft propulsion, have been thoroughly investigated throughout aeronautics history (Leishman, 2006). In fact, these pre-date aircraft entirely, as the screw propeller had been already in use and studied for marine propulsion applications (McCormick Jr., 1995). A fundamental theory to study and design propellers is the blade element theory (BET) (Glauert, 1930). The blade element theory breaks a single propeller blade into multiple blade sections, where each section is treated as a 2D airfoil (Ingraham et al., 2019). These elements are analyzed independently and later integrated as a whole blade. Prandtl, Betz, and Glauert (Gutin, 1948) have all studied and designed methods applied to the propeller and rotor aerodynamics which improved upon the Blade Element Theories.

The Blade Element Momentum Theory (BEMT), discussed in detail in Section 2.1.2, has been established and proven to be very accurate predicting propeller aero-
dynamics and flow effects, compared to current computational fluid dynamics (CFD) (Gur & Rosen, 2008; Chirico et al., 2018). Correction factors for tip-effects by Prandtl and Glauert are embedded into the BEMT to provide results representative of reality, as long as the local Mach number is subsonic ($M \leq 0.7$) (Gur & Rosen, 2008). Current propeller design and noise calculations rely on BEMT as their source for loading parameters to provide representative data to use within design processes (Hambrey et al., 2017; Kotwicz Herniczek et al., 2017).

The proven reliability of the method, without requiring high computational effort, is the reason why it was chosen in this research to estimate the surface pressure distribution on a blade. A limitation of this method is the assumption that span-wise flow is neglected. It has been shown, however, that generally this radial flow on the surface of the blade is very small, providing consequently insignificant difference between the 2D and 3D flow characteristic (Juhasz et al., 2014; J. Morgado et al., 2016), except at the blade tip. This only holds true for untapered, high-aspect-ratio, low-twist blades, and away from the top (Leishman, 2006).

### 1.2.3 Rotating Blade Aerodynamics

A key element to make the BEMT accurate is the availability of airfoil coefficients of lift ($c_l$) and drag ($c_d$) for several angles of attack ($\alpha$) and Reynolds numbers ($Re$). A common method is to use tabulated data from known airfoil sections (Gur & Rosen, 2008). Another frequent method is, if small angles are assumed along the blade span, to use the linearized coefficient of lift and the drag polar model to obtain the forces on the element (Leishman, 2006; Drela, 2007; McCormick Jr., 1995). The limitation of
these mentioned methods is, however, that the applicability of the inflow calculation is bounded by the range of the data.

Several methods have been developed to improve upon these. Viterna and Janetzke (1982) developed a method to extrapolate the lift and drag coefficients of airfoils past to ±180°. Their extrapolation method is based on potential theory, and is generally used for propeller and wind turbine blade design as it is relatively simple to perform (Mahmuddin et al., 2017). The other prevailing extrapolation approach, lately favored due to its higher accuracy, is the Montgomerie method (Mahmuddin et al., 2017). Montgomerie’s (2004) approach expands on Viterna’s (1982); however, it is more complex and harder to implement. Mahmuddin et al. (2017) prove that both methods are reliable for use in blade element analyses.

Drela uses Montgomerie’s method within QPROP (Drela, 2007) and XROTOR (Drela & Youngren, 2015) as do Morgado et al. in JPROP (J. P. Morgado et al., 2013), all popular open source tools for propeller and rotor design and performance estimation. Even though airfoil lift and drag polar extrapolation is a necessity for wind-turbine blades, there is no real need to use it for the design analysis of propellers or rotors as these should not operate within the stall region (J. Morgado et al., 2016). The use of 2D airfoil analysis tools, such as Drela’s XFOIL (Drela, 1989), can, therefore, be used instead of lookup tables or linearized approximations (J. Morgado et al., 2016).

Morgado et al. (2016) have shown that a full CFD rotor solution provides little improvement upon aerodynamic results obtained through the use of XFOIL for at-
tached flows. Consequently, the problem at hand does not require the use of CFD; and, XFOIL can be used to obtain the aerodynamic parameters desired for the BEMT code. The limitation of this implementation is that the blade cannot be operating near its stall region. Furthermore, the local element velocities must be below the transonic region.

1.3 Problem Statement

Noise reduction in aviation would enable urban missions that cannot be flown with current generation helicopters. This goal can be achieved by using electric motors, as they provide two distinct qualities which aid noise reduction. First, as they are quieter than their internal combustion counterparts, the power plant noise is greatly reduced as a source. Second, electric motors reduce the rotor or propeller noise by using the capability of producing higher torque at lower revolutions per minute (RPM). Therefore, a proprotor system can be designed to exploit this characteristic, abating noise levels further.

The present research focuses on a noise approximation method to be included along with a preliminary design tool for a single electric-driven hovering proprotor. The approximation is based on the pressure distribution along the blade surface and blade geometry. This estimation mechanism is validated with known propeller noise and performance data. Finally, it is used to compare the sound generated by a custom-designed proprotor for electric propulsion applications against a standard RC helicopter rotor.
1.4 Objectives

The main objective of this research is to evaluate the sound generated by a proprotor efficiently without the need of time-consuming Computational Fluid Dynamic (CFD) solutions. The intent is to include this methodology into the design cycle analysis for eVTOL vehicles. Since the UAM market is growing and the interest in this aircraft genre focuses mainly on being able to reduce the noise when compared to current generation helicopters, being able to quickly and accurately estimate the sound level is key to bring this technology into full commercial operation. This research is the initial step, i.e. a single rotor at hover conditions.

The noise approximation method is explained in detail, as well as the aerodynamic analysis on the single proprotor system. The limitations and assumptions are stated. The results are validated and compared against published results showing that this method provides an adequate sound level generation estimation. Lastly, two rotor geometries producing the same thrust will be analyzed with the tool. One is a commercially available RC helicopter rotor. The other is a proprotor custom designed specifically for a transitioning eVTOL type vehicle.

The main objective stated above was thus discretized into specific tasks to be accomplished by this research. These are:

- Create a modular code based on the methodologies mentioned above, i.e. BEMT, XFOIL, PSU-WOPWOP.
• Study the result sensitivity to geometrical grid size, total observer time length, time step size, and total number of analysis points.

• Ensure blade aerodynamic approximations are accurate.

• Verify that sound level approximations are within a reasonable range.

• Compare a regular helicopter rotor’s noise generation against that of an eVTOL proprotor, at the same thrust and power conditions.

1.5 Presented Work Outline

This document is organized in four chapters. The current section states the motivation for this study, summarizes past work and applicable theories to solve the problem. Specific objectives are declared to aid in the development and validation of the solving tool. Section 2 describes the methodology selected to construct the algorithm, as well as a brief overview on the code structure. Validation and input sensitivity studies are shown in this section. Section 3 presents and discusses the results obtained using the tool, comparing a regular RC helicopter rotor against an proprotor designed specifically for electric propulsion applications. The fourth and last section states the concluding remarks, limitations, and future work of this study.

Supporting documentation can be found in the appendices. Appendix A contains all the results obtained by the sensitivity studies, showing the difference between each of the investigated parameters. No discussion is included in this section. Appendix B describes in detail the algorithm implementation, acting as a pseudo-manual for future implementation or expansion of this code. All calculation schemes are explained, and
a description of inputs, files, and structures is included. Lastly, all the main scripts are listed in Appendix C.
2. Methodology

The ensuing section discusses the theory used to solve the stated problem, followed by how it was implemented into the predicting tool. The two main theories applied are the Blade Element Momentum Theory (BEMT), discussed in Section 2.1.2, and Farassat’s 1A formulation (F1A) of the Ffowcs Williams-Hawkins (FW-H) aeroacoustic equation, discussed in Section 2.1.1.

2.1 Theory

2.1.1 Rotor Aeroacoustics

The Ffowcs Williams - Hawking (FW-H) equation has been used extensively due to its robustness and practicality (Brentner et al., 2002). Specifically, Farassat’s Formulation 1A of the FW-H equation is used by PSU-WOPWOPv3 (Hennes et al., 2017). This section will overview the theory applied within the tool and show the implications of the different contributors to noise generation.

Ffowcs Williams and Hawkins (1969) published their groundbreaking equation in 1969. It is a Navier-Stokes reformulation, and is shown in Equation 2.1 written in differential form:

\[ \Box^2 p'(x, t) = \frac{\delta}{\delta t} \left\{ \rho_0 v_n \delta(f) \right\} - \frac{\delta}{\delta x_i} \left\{ l_i \delta(f) \right\} + \frac{\delta^2}{\delta x_i \delta x_j} \left[ T_{ij} H(f) \right] \]  (2.1)
where: $\Box^2 = \text{wave operator}$, $\square^2 = \frac{1}{c^2} \frac{\partial^2}{\partial t^2} - \nabla^2$

$c = \text{speed of sound in quiscent medium}$

$p' = \text{acoustic pressure}$

$H(f) = \text{Heaviside function, } H(f) = 0 \text{ for } f < 0 \text{ and } H(f) = 1 \text{ for } f > 0$

$l_i = \text{components of local force intensity that acts on the fluid, } l_i = P_{ij}n_j$

$\hat{n}_i = \text{component of the unit outward normal vector to surface}$

$P_{ij} = \text{compressive stress tensor}$

$T_{ij} = \text{Lighthill stress tensor, } \rho u_iu_j + P_{ij} - c^2 \rho' \delta_{ij}$

$u_n = \text{normal component of local fluid velocity}$

$u_i = \text{component of local fluid velocity}$

$v_n = \text{local normal velocity of source surface, where } v_n = 0 \text{ for a solid surface}$

$\delta(f) = \text{Dirac delta function}$

$\rho = \text{local density of medium}$

$\rho_0 = \text{density of quiscent medium}$

To understand this equation, it is best to look at the three terms independently, as they correspond to distinct components of noise generation due to a moving surface. The first two account for monopole and dipole sources on the body surface. The last one, the quadrupole term, pertains to the volume surrounding the body.

If this surface is then considered a solid and placed exactly around a rotor blade profile, then the FW-H equation can be used to calculate the noise generated by the rotating blade. Hence, the monopole term accounts for the thickness noise, or the noise being generated by the displacement of the air mass by the physical airfoil.
The dipole meanwhile describes the loading noise, or the one being generated by the unsteady forces exerted from the surface of the blade. This encompasses blade vortex interactions (BVI), accelerating (e.g. rotating) force distributions, and unsteady loadings on the blade. Lastly, the quadrupole term accounts for the nonlinear terms due to high-speed interactions and non-uniform velocities of the flow field around the rotor, such as shock waves and turbulence. However, the quadrupole sources are usually ignored for subcritical flows.

Farassat formulated a solution in integral form by neglecting the quadrupole term and taking the time derivatives inside the resulting integrals, while considering only subsonic flow (Farassat, 2007). The integrands are evaluated at the retarded time, i.e. the time when the sound was emitted, and the integration is performed over the actual blade surface, i.e. $f = 0$, for a fixed observer position (Brentner & Farassat, 2003). This is known as Farassat’s 1A formulation of the FW-H equation, shown in Equation 2.2, with its sub components defined in equations 2.3.

$$p'(x,t) = p_T'(x,t) + p_L'(x,t)$$  \hspace{1cm} (2.2)

where: $p_T' = \text{thickness pressure fluctuation}$

and $p_L' = \text{loading pressure fluctuation}$

and

$$4\pi p_T'(x,t) = \int_{f=0} \left[ \frac{\rho_0 \dot{v}_n}{r(1 - M_r)^2} \right]_{ret} dS$$

$$+ \int_{f=0} \left[ \frac{\rho_0 v_n (r \dot{M}_r + c(M_r - M^2))}{r^2(1 - M_r)^3} \right]_{ret} dS$$  \hspace{1cm} (2.3a)
\[ 4\pi p_L'(x, t) = \int_{f=0}^{\pi} \left[ \frac{l_r}{r(1 - M_r)^2} \right]_{\text{ret}} \, dS + \int_{f=0}^{\pi} \left[ \frac{l_r - l_M}{r^2(1 - M_r)^2} \right]_{\text{ret}} \, dS \]

\[ + \frac{1}{c} \int_{f=0}^{\pi} \left[ l_r \frac{rM_r + c(M_r - M^2)}{r^2(1 - M_r)^3} \right]_{\text{ret}} \, dS \]

(2.3b)

where:
- \( c \) = speed of sound
- \( l \) = local force intensity
- \( M \) = Mach number (scalar)
- \( n \) = surface normal vector
- \( r \) = observer position relative to source
- \( p' \) = acoustic pressure, i.e. \( p - p_0 \)
- \( v \) = local velocity of source surface
- \( dS \) = blade element surface area
- \([\ldots]_{\text{ret}} \) = calculated in retarded time

It is worth noting that the dot indicates time derivatives, while the subscripts show which variable the dot product is performed against (Brentner & Farassat, 2003).

PSU-WOPWOPv3 uses this formulation to solve part of the noise level estimations. The implementation is numerical, where the integrals are treated as discrete sums and the derivatives using finite difference methods. The tool includes more complex calculations to estimate noise from other sources besides the loading and thickness, i.e. broadband noise. However, since this research is focusing only on these two noise sources, the others will not be discussed. PSU-WOPWOP has been
validated and shown to predict accurately noise generation by a rotor in different flight conditions (Shirey et al., 2007), providing a robust platform for noise prediction.

2.1.2 Blade Element Momentum Theory

Leishman (2006) and McCormick (1995) both derive Blade Element Analyses for Hover (static) or Axial Flight (climb for rotors, forward for propellers) based on classical works cited in Section 1.2. The key to the blade element analysis is to calculate the inflow. Thus, obtaining the correct resultant velocity relative to the blade and its angle of attack is paramount. Hence, the blade element momentum theory, as discussed by Leishman (2006), is mainly applied. Some corrections were applied using McCormick’s (1995) derivation to account for high blade twist pitch.

To begin, we must look at a single blade and divide it into different sections. Each section is a single blade element. This blade element will have a velocity component relative to the path of rotation, the Tip-Path Plane (TPP). The component parallel to it is denominated $U_T$, as depicted in Figure 2.1(b). This component comes from the physical rotation of the blade. The other velocity component that affects the blade element is the one perpendicular to the TPP, known as the propeller or rotor induced velocity. In this case, denominated $U_P$. There is a third component of the resultant velocity $U$, which is the span-wise flow across the blade $U_R$. For this application, the latter is assumed to be negligible, however. Knowing these two values, the resultant velocity $U$ and the relative inflow angle $\phi$ can be calculated, as shown in Equations 2.4-2.5.

$$U = \sqrt{U_T^2 + U_P^2}$$  \hspace{1cm} (2.4)
Having $\phi$, and using the known element pitch angle ($\theta$), the element angle of attack ($\alpha$) can be calculated.

Using $U$ and $\alpha$, the element’s coefficients of lift ($c_l$) and drag ($c_d$) can be obtained from which the differential lift ($dL$) and drag ($dD$) forces are computed. These forces can be resolved parallel ($dF_x$) and perpendicular ($dF_z$) to the disk, as expressed in Equations 2.7 and 2.8.

$$\alpha = \theta - \phi$$  \hspace{1cm} (2.6)
\[ dL = \frac{1}{2} \rho U^2 c_y c_t dy \quad (2.7a) \]
\[ dD = \frac{1}{2} \rho U^2 c_y c_d dy \quad (2.7b) \]

where: \( c_y \) = element’s local chord distance
\( dy \) = element spanwise thickness

Hence,

\[ dF_x = dL \sin \phi + dD \cos \phi \quad (2.8a) \]
\[ dF_z = dL \cos \phi - dD \sin \phi \quad (2.8b) \]

Therefore, these two force contributions can be resolved in terms of thrust, torque and power. These contributions can be integrated along the blade, resulting in the contribution by each blade. Finally, by multiplying this by the number of blades per rotor \( N_b \), the total thrust, torque, and power can be calculated, as seen in Equations 2.9.

\[ dT = N_b(dL \cos \phi - dD \sin \phi) \quad (2.9a) \]
\[ dQ = N_b(dL \sin \phi + dD \cos \phi) y \quad (2.9b) \]
\[ dP = N_b(dL \sin \phi + dD \cos \phi) \Omega y \quad (2.9c) \]

Having Equations 2.9 above, we can now non-dimensionalize distances in terms of the blade radius \( R \) and the velocities by the rotational tip-speed \( \Omega R \). This yields the following set of properties and identities:

\[ r = \frac{y}{R} = \frac{\Omega y}{\Omega R} = \frac{U_T}{\Omega R} \]
\[
\lambda = \frac{U_P}{\Omega y} = \frac{U_P}{\Omega R} = \frac{U_P}{U_T} \left( \frac{y}{R} \right) = \frac{U_P}{U_T} r
\]

\[
dC_T = \frac{dT}{\rho A(\Omega R)^2}
\]

\[
dC_Q = \frac{dT}{\rho A(\Omega R)^2 R}
\]

\[
dC_P = \frac{dT}{\rho A(\Omega R)^3}
\]

where: 
- \( r \) = non-dimensional radial position 
- \( \lambda \) = non-dimensional inflow 
- \( dC_T \) = differential coefficient of thrust 
- \( dC_Q \) = differential coefficient of torque 
- \( dC_P \) = differential coefficient of power 

However, these previous equations imply that the inflow is known. To calculate the thrust, power, and torque, \( \phi \) must be determined. It in turn depends on the the induced axial velocity of air. There are several methods to tackle this problem (Leishman, 2006; McCormick Jr., 1995). One of them is to calculate the inflow using numerical methods, equating the lifting surface circulation theory and the momentum theory of lift (Leishman, 2006; McCormick Jr., 1995). To do this, we must look at the blade element now as part of the disk annulus, as seen in Figure 2.2. From the non-dimensional coefficient of thrust \((dC_T)\) and knowing the contributions of \(dL\) and \(dD\) to the forces as shown in Equations 2.7-2.8, we can manipulate the thrust coefficient term in the following way:

\[
dC_T = \frac{N_b (dL \cos \phi - dD \sin \phi)}{\rho A(\Omega R)^2} \quad (2.10)
\]

\[
= \frac{1}{2} \frac{N_b \rho U^2 c_p}{\rho (\pi R^2) (\Omega R)^2} (c_l \cos \phi - c_d \sin \phi) \quad (2.11)
\]

\[
= \frac{1}{2} \left( \frac{N_b c_p}{\pi R} \right) \left( \frac{U^2}{\Omega^2 R^2} \right) (c_l \cos \phi - c_d \sin \phi) \left( \frac{y}{R} \right) \quad (2.12)
\]
\[ dC_T = \frac{1}{2} \sigma (c_i \cos \phi - c_d \sin \phi) \left( \lambda^2 + r^2 \right) dr \] (2.15)

Now, from the annulus of the disk, seen in Figure 2.2, we can calculate the radial mass flow through the disk. Here \( U_P \) gets separated into two terms: the induced velocity \((v_i)\) and the axial or climb velocity \((V_c)\). With the mass flow through the discretized annulus we obtain the incremental thrust for the element:

\[ d\dot{m} = \rho (V_c + v_i) dA \] (2.16)
\[ = \rho (V_c + v_i) (2\pi y dy) \] (2.17)
\[ dT = 2v_i d\dot{m} \] (2.18)
\[ \therefore = 4\pi \rho (V_c + v_i) v_i y dy \] (2.19)

Non-dimensionalizing we get:

\[ dC_T = \frac{4\pi \rho (V_c + v_i) v_i y dy}{\rho A(\Omega R)^2} \] (2.20)
\[ = 4\frac{V_c + v_i}{\Omega R} \left( \frac{v_i}{\Omega R} \right) \left( \frac{y}{R} \right) d \left( \frac{y}{R} \right) \] (2.21)
\[ = 4\lambda \lambda_i r dr \] (2.22)

As mentioned before, the total inflow is discretized into the induced flow by the rotor \((v_i)\) and the axial flow caused by axial translation of the rotor in space \((V_c)\). Non-dimensionalizing these terms result in the inflow ratios \(\lambda_i\) and \(\lambda_c\) respectively.
Knowing $U_p = V_c + v_i$, we can manipulate $\lambda_i$ to be in terms of the total inflow and the axial flow such that $\lambda_i = \lambda - \lambda_c$. This allows us to solve for the unknown induced inflow velocity later.

One factor to consider in this analysis is the losses towards the tip of the rotor. The loss of lift due to the effects of a finite blade can be calculated through the a method devised by Prandtl (Leishman, 2006; McCormick Jr., 1995). A correction factor, $F$, is introduced, where:

$$F = \left(\frac{2}{\pi}\right) \cos^{-1} \left( e^{-\frac{N_b}{2} \left( \frac{1-r}{r_o} \right)} \right)$$ \hspace{1cm} (2.23)

The Prandtl tip-loss correction factor ($F$) and the manipulated $\lambda_i$ are then introduced in equation 2.22 to get:

$$dC_T = 4F\lambda (\lambda - \lambda_c) r dr$$ \hspace{1cm} (2.24)
Finally, equation 2.24 can be related to Equation 2.15, from which the inflow can be formulated as:

\[ 4F\lambda (\lambda - \lambda_c) r dr = \frac{1}{2} (c_l \cos \phi - c_d \sin \phi) \sigma (\lambda^2 + r^2) dr \]  

(2.25)

\[ (\lambda^2 - \lambda \lambda_c) r = \frac{\sigma (c_l \cos \phi - c_d \sin \phi)}{8F} (\lambda^2 + r^2) \]  

(2.26)

\[ \lambda^2 - \lambda \lambda_c = \frac{\sigma (c_l \cos \phi - c_d \sin \phi)}{8F} \frac{\lambda^2}{r} + \frac{\sigma (c_l \cos \phi - c_d \sin \phi)}{8F} \frac{r}{\lambda \lambda_c} \]  

(2.27)

\[ 0 = \lambda^2 \left( 1 - \frac{\sigma (c_l \cos \phi - c_d \sin \phi)}{8Fr} \right) - \lambda \lambda_c \ldots \]  

(2.28)

Solving for the inflow, we get:

\[ \lambda(r, \lambda_c) = \frac{r 4F \lambda_c}{8Fr - \sigma (c_l \cos \phi - c_d \sin \phi) \ldots} \]  

\[ \ldots + \frac{r \sqrt{(4F \lambda_c)^2 + \sigma (c_l \cos \phi - c_d \sin \phi) (8Fr - \sigma (c_l \cos \phi - c_d \sin \phi))}}{8Fr - \sigma (c_l \cos \phi - c_d \sin \phi)} \]  

(2.29)

It is worth noting that in Equation 2.29 the term \((c_l \cos \phi - c_d \sin \phi)\) has been left intact. This was left as-is for two reasons. First, for the proprotor case we cannot assume small angles of deflection; thus, \(\phi \neq \lambda/r\). Second, in this research application, \(c_l\) and \(c_d\), which are both functions of local angle of attack \(\alpha\), Mach number \(M\), and Reynolds number \(Re\), are calculated through the implementation of XFOIL. Consequently, it was decided to leave these values without any manipulation.

A second point of discussion is the inclusion of the vertical or axial speed term \((\lambda_c)\) in equation 2.29. Even though this research focuses solely on hovering rotors, the capability to analyze them with axial speeds was desired to enable performance evaluations in the future. As the intent is to later employ these calculations within
a design scheme, this term was left in the equation as-is. For the hover calculations, nonetheless, \( \lambda_c \) was set to zero.

A repeated complication is observed, however. To calculate the inflow, we need the inflow angle. To get the inflow angle, we need the induced inflow. This problem can be solved by setting-up an iterative process. For \( n = 1, 2, 3, \ldots, N \) number of elements, the inflow is iterated for \( j = 0, 1, 2, \ldots, J \) steps, where \( J \) is determined when convergence has been achieved. For the first iteration, i.e. \( j = 0 \), though, an assumption has to be made. For this initial case, the inflow is determined by using linearized aerodynamic coefficients along with small angle approximations. Thus, the following assumptions are made for the initial condition:

\[
c_l = c_{l(\alpha)}(\theta - \alpha_0 - \phi)
\]

\[
\phi << 1 \implies \sin \phi \approx 0 & \cos \phi \approx 1
\]

\[
\therefore (c_l \cos \phi - c_d \sin \phi) \approx c_{l(\alpha)}(\theta - \frac{\lambda}{r})
\]

With which equation 2.29 simplifies for hover to:

\[
\lambda_{(n)}^{(j=0)} = \sqrt{\left(\frac{\sigma_{(n)} c_{l(\alpha)}}{16 F(n)}\right)^2 + \frac{\sigma_{(n)} c_{l(\alpha)}}{8 F(n)}\theta_{(n)} r_{(n)} - \left(\frac{\sigma_{(n)} c_{l(\alpha)}}{16 F(n)}\right)}
\] (2.30)

For the next iterative steps, equation 2.29 is reformulated for hover by setting \( \lambda_c = 0 \), and accounting the iteration steps \( j \) for each element \( n \) as:

\[
\lambda_{(n)}^{(j+1)} = \frac{r_{(n)} \sqrt{\sigma_{(n)} \left(c_l^{(j)} \cos \phi^{(j)} - c_d^{(j)} \sin \phi^{(j)}\right)}}{8 F(n) r_{(n)} - \sigma_{(n)} \left(c_l^{(j)} \cos \phi^{(j)} - c_d^{(j)} \sin \phi^{(j)}\right)} \ldots
\]

\[
\ldots \sqrt{8 F(n) r_{(n)} - \sigma_{(n)} \left(c_l^{(j)} \cos \phi^{(j)} - c_d^{(j)} \sin \phi^{(j)}\right)}
\]

\[
\frac{8 F(n) r_{(n)} - \sigma_{(n)} \left(c_l^{(j)} \cos \phi^{(j)} - c_d^{(j)} \sin \phi^{(j)}\right)}{8 F(n) r_{(n)} - \sigma_{(n)} \left(c_l^{(j)} \cos \phi^{(j)} - c_d^{(j)} \sin \phi^{(j)}\right)}
\] (2.31)
Consequently, the calculation scheme proceeds as follows:

1. Initial $\lambda^{(j=0)}$ is calculated using equation 2.30.

2. Using $\lambda^{(j)}$, calculate $\alpha$, $M$, and $Re$.

3. With $\alpha$, $M$, and $Re$, get $c_l$ and $c_d$ through XFOIL

4. Calculate new $\lambda^{(j+1)}$ with equation 2.31 using $c_l^{(j)}$, $c_t^{(j)}$, and $\phi^{(j)}$ obtained from step 3.

5. Iterate steps 2 through 4 until the difference between steps of $\lambda$ is less than 0.01% between steps by using the fixed-point (FP) method, i.e. $|\lambda^{(j+1)} - \lambda^{(j)}| \leq \mathcal{E}$.

Note that equations 2.30 and 2.31 need to be iterated to account the tip loss factor, as inflow is implied in $F$, by using equation 2.23. Once the initial inflow is approximated in step 1, these values can be used to estimate $U_P$ from the known $U_T$. With the estimated $U_P$, values for local element $M$, $Re$ and $\alpha$ are obtained in step 2. These values are then used to obtain the 2D airfoil aerodynamic coefficients through the use of XFOIL in step 3. With the obtained coefficients, equation 2.31 is used to get the inflow estimation in step 4. This last value is iterated through a FP iterative numerical method until the error in local $\lambda$ is less than 0.01%. After this last step, the process is iterated back to step 2 until the inflow converges to a stable result.

2.2 Implementation Overview

The following section goes over the details of what each implemented tool does, how they work, and describes the different algorithms that were created to enable the interaction among them. The methods and assumptions made are also discussed, as well as the limitations in the operation. Possible modifications to increase future capabilities are briefly discussed as well.
2.2.1 PSU-WOPWOP

Brentner’s PSU-WOPWOPv3 (Hennes et al., 2017) is an aeroacoustic tool based on the Farassat 1A formulation of the FW-H equations, as introduced in Section 2. The user inputs the geometry of a single blade and the pre-calculated pressure loading on it. Through the input files the rotor geometry, the atmospheric conditions, and the acoustic measurement parameters are assembled and prepared for execution. Once execution ends, several files are created each containing different sets of results, depending on which outputs were selected in the configuration.

The inputs enable the user to calculate the noise through several phases of flight, different time lengths, and multiple configurations. Multiple rotors can be analyzed, as well as rotor-fuselage, rotor-wall, wall-wall, and rotor-rotor sound interactions. Flight paths can be setup where the rotor or source assembly is be displaced in space with respect to the fixed observer location. The Federal Aviation Administration (FAA) certification files can be pre-configured, defaulting the analysis to the requirements of Federal Aviation Regulations (FAR) Part 36 Appendix A (2018). As it is observed, the complexity of the software exceeds the requirement of this research. Only the static acoustic analysis capabilities are therefore used and implemented.

2.2.2 Xfoil

XFOIL, written by Drela (2001), is a widely used tool for airfoil sections in subsonic flow. It provides accurate pressure coefficient distributions on airfoil surfaces using vortex sheet methods in the pre-stall region (J. Morgado et al., 2016). Other outputs that are useful for this application are the coefficients of lift, drag, and moment. The user inputs either xy-coordinates of the airfoil to be studied or a known airfoil file input, the angles of attack, and the Reynolds and Mach numbers at which the airfoil will be analyzed. The tool can analyze multiple angles of attack to gener-
ate the lift and drags polars, as well as include viscous losses approximations (Drela, 1989).

**XFOIL** was chosen as the source of aerodynamic coefficients for the BEMT due to the flexibility it provides. As mentioned in Section 1.2, there are two usual methods of acquiring these coefficients: look-up tables or linearized aerodynamics. By using **XFOIL** instead of these, the assumption is that the results would be better and more representative of the real conditions. Morgado et al. (2016) have shown that **XFOIL** does provide results very similar to those obtained by using CFD for propeller design. Moreover, by implementing this software within the script, the need to have look-up tables for each blade at different Reynolds numbers ($Re$) and Mach numbers ($M$) is dismissed. The interaction between the codes is simple. The BEMT parses the elements $\alpha$, $M$, and $Re$ for each blade element to **XFOIL**. It consecutively returns the aerodynamic coefficients for the blade section.

### 2.2.3 Interface

The underlying interface is written in **MATLAB**, enabling the interaction between different systems and executables, as well as being a practical language widely used in engineering and science. The script is organized in a way that enables different levels of user knowledge. The base running script just handles the inputs and includes several switches and choices the user selects. These control both the numerical and visual outcomes. They also account for different calculation types, as can be seen in Appendix B.2.

With the use of the different interfaces, the running script performs a blade element momentum analysis to calculate the local angle of attack and free-stream velocity at each blade element. With the calculated angles of attack and velocities, the blade’s surface pressure distribution is calculated with the use of **XFOIL**. This pressure distribution is then processed and used by WOPWOP to perform the aeroacoustic
analysis. Finally, all the results are post-processed according the user selection and presented. This process and the details of how these were implemented are detailed in appendix B.1.

2.2.4 Flow Diagrams

A set of flow diagrams can be seen in the following pages depicting the schemes used in the developed code. Figure 2.3 demonstrates the high level structure. There are two distinct cores or subroutines to the code. The first is the blade analysis. In this subroutine the geometry and rotor inputs get pre-processed into blade elements in order to calculate the inflow and, with it, the pressure distribution on the blade surface. The second subroutine is the noise analysis. Here the time-dependent pressure and the sound pressure level (SPL) frequency distribution data are obtained by running the noise approximating tool using the results from the previous subroutine. After each process is completed, the data is presented to the user in different plots, as well as a data file containing all the calculated parameters.

The Blade Element analysis subroutine reads the user inputs and performs the calculations as shown in Figure 2.4. The initial calculations translate the inputs into the geometrical, flight, and rotor parameters that are required to perform the modified Blade Element Theory. Each element is analyzed independently, where the inflow is calculated for each based on the theories shown by Leishman (2006) and McCormick (1995) as discussed previously in Section 2.1.2. The inflow is iteratively calculated with a fixed point (FP) iteration numerical method, accounting for changes in aerodynamics due to the blade moving through air.

Once the inflow has been estimated, the angle of attack and resultant velocity are then calculated. With these values, XFOIL is then called to calculate the $c_l$, $c_d$, and $c_p$ for each blade element. These coefficients are then assembled and combined to get the total $c_l$ and $c_d$ distribution along the blade length. The $c_p$ distribution along
the airfoil is interpolated and assigned to each geometrical node of the blade. Lastly, the data is delivered into user-friendly variable structures, and plots so the user can visualize the results.

The noise predicting subroutine, pictorially explained in Figure 2.5, uses the outputs from the blade core and performs the aeroacoustic analysis. First, the geometric data extracted before needs to be formatted and saved into the correct input file.
Second, the pressure distribution is saved into a separate loading file. The pressure data points need to correspond to the number of nodes in the input geometry. Lastly, a setup file that commands WOPWOP is generated based on the user inputs. Once all three input files are completed, WOPWOP is called. After it has finished its calculations, the output files are imported and decoded into useful parameters to the user, as well as plots.
2.3 Parameter Sensitivity

Sensitivity to the user’s inputs is of great interest. As previously mentioned in section 1.4, there are four areas in which result sensibility is a concern. These will be discussed in the following sections.

2.3.1 Grid Resolution

An area of interest is the rotor blade’s geometric grid resolution impact on acoustic calculations, as Suresh et al. (2018) performed a similar test which showed a difference in the sound pressure levels. To determine the results’ sensitivity to the blade geometry grid, calculations were performed for the same blade geometry but with varying number of spanwise and chordwise elements, onward referred to as elements
and panels respectively. The number of elements is used by the BEMT, while the number of panels affects the XFOIL analysis of that section.

Table 2.1

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Description</th>
<th>Elements</th>
<th>Panels</th>
<th>Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>10 elements, 40 panels, No bias</td>
<td>10</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>1.1.2</td>
<td>50 elements, 40 panels, No bias</td>
<td>50</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>1.1.3</td>
<td>100 elements, 40 panels, No bias</td>
<td>100</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>1.2.1</td>
<td>10 elements, 60 panels, No bias</td>
<td>10</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>1.2.2</td>
<td>50 elements, 60 panels, No bias</td>
<td>50</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>1.2.3</td>
<td>100 elements, 60 panels, No bias</td>
<td>100</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>1.3.1</td>
<td>10 elements, 100 panels, No bias</td>
<td>10</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>1.3.2</td>
<td>50 elements, 100 panels, No bias</td>
<td>50</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>1.3.3</td>
<td>100 elements, 100 panels, Tip bias</td>
<td>100</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>2.1.1</td>
<td>10 elements, 40 panels, Tip bias</td>
<td>10</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>2.1.2</td>
<td>50 elements, 40 panels, Tip bias</td>
<td>50</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>2.1.3</td>
<td>100 elements, 40 panels, Tip bias</td>
<td>100</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>2.2.1</td>
<td>10 elements, 60 panels, Tip bias</td>
<td>10</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>2.2.2</td>
<td>50 elements, 60 panels, Tip bias</td>
<td>50</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>2.2.3</td>
<td>100 elements, 60 panels, Tip bias</td>
<td>100</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>2.3.1</td>
<td>10 elements, 100 panels, Tip bias</td>
<td>10</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2.3.2</td>
<td>50 elements, 100 panels, Tip bias</td>
<td>50</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2.3.3</td>
<td>100 elements, 100 panels, Tip bias</td>
<td>100</td>
<td>60</td>
<td>1</td>
</tr>
</tbody>
</table>

The test matrix is shown in Table 2.1. Each varying parameter effect was studied and discussed independently. Full tables summarizing the total difference between each of the cases can be found in Appendix A.

The effect of varying only the spanwise elements while keeping the chordwise panels fixed is seen in Figure 2.6. It can be observed that there is a small effect of the number of spanwise elements on the acoustic loading pressure (Figure 2.6(c)), with a peak difference of 4.94% between 50 and 100 elements, and 2.59% between using 100 and 200 elements. The difference between the first and last is of 7.41%. The same behavior is observed for the thickness acoustic pressure (Figure 2.6(b)) and
total acoustic pressure (Figure 2.6(b)), with a peak percent differences having similar values as described for the loading pressure. It is therefore concluded that total number of spanwise elements has an effect on the total perceived acoustic pressure. Thus, a minimum of 100 spanwise elements must be used in the calculations.

Freezing the spanwise elements to 100, the effect of varying the total number of chordwise panels is shown in Figure 2.7. Thickness pressure variation was negligible (0.69% between end cases), and is therefore not shown. Loading pressure (Figure 2.7(a)) is affected considerably, showing a difference of -28.04% between using 50 and 100 chordwise panels, and -36.06% between 50 and 300 panels. The difference between 100 and 200 chordwise elements is of -6.26%. However, when analyzing the total acoustic pressure (Figure 2.7(b)), this severity is reduced. The total peak pressure difference is of -7.22% between the first and last case, and of -2.14% between using 100 or 200 chordwise elements. Clearly, the number of chordwise panels affects the perceived acoustic pressure. As the greatest effect is seen on the loading pressure, it is recommended that at least 100 chordwise panels are used.

As aerodynamic effects towards the tip of the blade have greater significance than those towards the root, an option available to the user is to have a biased distribution of elements towards the tip. This is, the elements are not equally distributed along the span of the blade. Figure 2.8 below shows the difference between the linearly distributed elements (Figure 2.8(a)) and the tip-biased distribution (Figure 2.8(b)) Thus, element weighting distribution was added as a secondary parameter to study. In these cases, the number of spanwise and chordwise elements were the same as with the constant width cases.

Figure 2.3.1 shows the loading acoustic pressure observed for the same blade, analyzed with 100 spanwise and 200 chordwise elements, with varying element width distribution along the blade. By having higher resolution at the blade tip, the differ-
Figure 2.6 Grid resolution sensitivity analysis for the same blade, same operating conditions, with different spanwise (S) and fixed chordwise (P) elements, with constant element width (no bias). The difference in peak pressure is of 6.07%. It was therefore concluded that tip weighting must be used to achieve better results.
Figure 2.7 Grid resolution sensitivity analysis for the same blade, same operating conditions, same spanwise elements (100), with varying chordwise (P) elements.

Figure 2.8 Linear span-wise element distribution versus tip-biased element spacing. Sample image with 10 elements (spanwise) and 20 chordwise panels.

The peak loading effect seen above propagates to the SPL frequency distribution, as depicted in Figure 2.10. The higher the amount of chordwise panels, the lower the peaks and the troughs. Distribution in the higher frequencies reveal some effect, mainly lowering the trailing frequency band. These are, however, well above the 24th
Figure 2.9 Grid resolution sensitivity analysis to element distribution weighting for the same blade, same operating conditions, same spanwise (100) and chordwise (200) elements.

1/3 octave used to calculate the perceived noise level (PNL), and can be ignored. When observing the total SPL, these effects are imperceptible. As the difference is minute, no calculations were performed to determine the real difference between the calculated values.

Figure 2.10 Grid resolution sensitivity analysis for the same blade, same operating conditions, same spanwise elements (100), with varying chordwise (P) elements and tip-biased element width distribution.

The loading acoustic pressure is thus sensitive to the blade geometric grid definition. It depends on the number of chordwise panels, given a greater resolution at
the blade tip. If a constant element width has to be used, then this parameter will affect the result. Consequently, it is recommended to perform calculations with 100 spanwise elements, 200 chordwise panels, and the tip weighted element distribution enabled.

Another effect studied was the change in total computation runtime for the same set of cases. Overall, as expected, the behavior shows that increasing the number of spanwise and chordwise elements increases the total runtime. When contrasting tip weighted versus unweighted grid, the maximum runtime change is of 1.9%. If only the biased geometry is studied, however, executing time increases conversely with both number of spanwise and chordwise elements.

In conclusion, caution should be taken when choosing the number of spanwise and chordwise elements to use. Using tip-weighting increases the accuracy of the calculation, as does increasing the number of chordwise elements. Given the conditions above, the number of spanwise elements provides no positive contribution to the computation, only delaying the execution. It is recommended to use at least 200 chordwise panels, 100 spanwise elements, and tip-bias enabled to perform any calculations with this algorithm.

This parameter therefore remains constant for each increment of observer time.

2.3.2 Data Points Sensitivity

The next variable analyzed is the total number of data points per revolution to be considered for the aeroacoustic calculation. A testing matrix was prepared, where both $\Delta \tau$ and $nT$ were alternated. This matrix can be seen in Table 2.2. Figure 2.11 shows the SPL comparison for the defined cases.

Similar to the effects seen in Figure ??, data sets agglomerate based on the number of data points, and remain insensitive to the time step change. The peaks have similar resolution, with only a variation in the troughs. When calculating the
Table 2.2
Time Step and Data Points Sensitivity Test Matrix

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Description</th>
<th>nT</th>
<th>( \Delta \tau ) [deg]</th>
<th>Observer Time [revs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>dTau 0.5 deg, 2096 nT</td>
<td>2096</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>3.1.2</td>
<td>dTau 1 deg, 2096 nT</td>
<td>2096</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3.1.3</td>
<td>dTau 2 deg, 2096 nT</td>
<td>2096</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3.2.1</td>
<td>dTau 0.5 deg, 4192 nT</td>
<td>4192</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>3.2.2</td>
<td>dTau 1 deg, 4192 nT</td>
<td>4192</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3.2.3</td>
<td>dTau 2 deg, 4192 nT</td>
<td>4192</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3.3.1</td>
<td>dTau 0.5 deg, 8384 nT</td>
<td>8384</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>3.3.2</td>
<td>dTau 1 deg, 8384 nT</td>
<td>8384</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3.3.3</td>
<td>dTau 2 deg, 8384 nT</td>
<td>8384</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

PNL, the data shows good agreement, with only a variation of 3%. Therefore, the number of data points has an effect on the overall shape of the SPL distribution, albeit not impacting the PNL.

**Figure 2.11** SPL frequency distribution sensitivity test to variable \( \Delta \tau \) and nT
2.3.3 Time step Sensitivity

The last sensitivity study pertains to the time step. This parameter has been included in the test matrices shown in tables ?? and 2.2. The effect this varying parameter has had is very small. In both cases, the results clustered around the other variable. It has also been shown that the total variation in pressure, SPL, and PNL between the varying $\Delta \tau$ are all less than 2%.

Table A shows a match in PNL, whereas Table A exhibits total pressure peaks having a mere difference of 0.1%. Figure 2.12 displays no variation in the total sound pressure wave by either of the varying parameters. It is therefore deemed that the variation of this parameter holds very small effect on the measurements as long as it is within a range of 2000 points.

![Total SPL Frequency Distribution time step (\(\Delta \tau\)) Sensitivity](image.png)

*Figure 2.12 Effect on varying time step sensitivity to SPL frequency distribution*
2.4 Validation

Validation was performed against published experimental or simulated data. For each of the cases presented below, the grid size consisted of 100 spanwise elements and 200 chordwise panels. The aeroacoustic calculations were performed with a time step of 0.5°, 4192 data points per rotor revolution, and at 50 observed revolutions. As shown before in section 2.3, this grid provides appropriate results without compromising computing time and with reduced impact on the outcome. Acoustic approximations were measured at the same observer location as in the cited references.

To validate the BEMT and blade aerodynamic solutions, three sources were used. The first is Caradonna and Tung (1981), the second is Suresh et al. (2018), and the third is WOPWOP’s sample case (Hennes et al., 2017). To validate the noise prediction, two sources were used: Suresh et al. (2018) and WOPWOP’s sample case (Hennes et al., 2017).

2.4.1 Blade Aerodynamics

Caradonna and Tung (1981) performed experimental tests on a rectangular blade with a NACA0012 airfoil. The rotor blade was tapped with 60 pressure transducers and the measured pressure coefficients were recorded and tabulated. Figures 2.13 and 2.14 show a comparison of the calculated \( c_p \) distribution versus the tabulated data provided by the researchers. The sample is contrasted at two different RPMs (1250 and 1750), obtaining the same thrust coefficient as the experiment. The \( C_T = 0.00213 \) at a \( M_{Tip} = 0.434 \) for the first RPM, and \( C_T = 0.00455 \) at a \( M_{Tip} = 0.612 \) for the later.

Figure 2.13 juxtaposes the measured and tabulated \( c_p \) for the rotor operating at 1250 RPM and a \( C_T \) of 0.00213. The values are shown for five different span-wise locations. Figures 2.13(a) through 2.13(c) show very good agreement of the estimated coefficient with respect to the measurements for span-wise locations or 50%, 68%, and
80%. Figures 2.13(d) and 2.13(e) show variation, especially at $r = 96\%$. This can be attributed to 3D effects of the blade, which were unaccounted for.

![Figure 2.13](image)

Figure 2.13 Calculated pressure coefficient compared against experimentally measured pressure on a NACA0012 blade (Caradonna & Tung, 1981) (a)-(e). Operating conditions: $C_T = 0.00213$, $\Omega = 1250$, $M_{Tip} = 0.434$
Figure 2.14 examines the $c_p$ for the rotor operating at 1750 RPM and a $C_T$ of 0.00455. As before, data was recorded at five distinct span-wise locations. Figures 2.14(a) through 2.14(c) again show congruity between the coefficients for the three inner span-wise locations. Similar to the data operating at 1250 RPM and $C_T$ of 0.00213, Figures 2.14(d) and 2.14(e) show that the 1750 RPM $C_T=0.00455$ tip pressure distribution differ slightly. As with the previous presented case, this difference can be attributed to wing tip vortices reducing the $c_p$ towards the trailing edge of the airfoil.

Suresh et al. (2018) performed a CFD analysis of a scaled UH1H rotor, based on experiments done by Boxwell et al. (1978). Their computational model was conducted using cell-centered, block-structured, parallel code SPARC with a Spalart-Allmaras turbulence model and a mesh of 3 million cells. As part of their validation, they compare the sound pressure time variation with the ones measured by Boxwel et al. Also, they include a sample coefficient of pressure distribution along an element for the 95$^{th}$ percentile radial position. This data was extracted and the analysis was performed with the `run_Prop.m` tool using the same blade geometry and operating conditions described. This correlation is shown in Figure 2.15. The $c_p$ is over-predicted by our code. However, due to the radial position being so close to the tip, similar to the above discussed Caradonna and Tung cases, the difference can be explained by the assumptions and limitations of the BEMT approach by not accounting for 3D tip-effects.

The third validation source is the sample case provided with the distribution of PSU-WOPWOP (Hennes et al., 2017) to validate our code. The sample case simulates the noise generated by a three-bladed gyrodyne rotor operating at $M_{Tip} = 0.5529$. The surface pressure is provided by a compact patch vector array, which differs from the surface pressure distribution approach used in our code.
Figure 2.14 Calculated pressure coefficient compared against experimentally measured pressure on a NACA0012 blade (Caradonna & Tung, 1981) (a)-(e). Operating conditions: $C_T = 0.00455$, $\Omega = 1750$, $M_{Tip} = 0.612$
Since the blade geometry had been coded to test proper tool implementation, this file was processed in the implemented code and the results tested against those extracted from the sample solution. As it can be seen in Figure 2.16, the calculated compact pressure distribution on the blade shows minimal difference with the one provided. It is noticeable however that in this case the calculations underpredicted the results by 4%.

**Figure 2.15** Comparison of calculated $c_p$ distribution vs. extracted from Suresh et al. (2018)

**Figure 2.16** Calculated compact pressure against extracted from WOPWOP Sample Case 1 (Hennes et al., 2017)
2.4.2 Noise

Suresh et al. (2018) modelled a UH1H rotor blade to a $1/7$th scale operating at $M_{Tip} = 0.8$, similar to what Boxwell et al. (1978) used in their experimental hover tower measurements. The model was used for CFD estimations of the flow field around the rotating blades, and from those flow field results, they applied the F1A formulation of the FW-H equations (Suresh et al., 2018). These noise pressure approximations were then compared against the experimentally measured data to prove that the computational method is valid. Similarly, the same blade, as described by Boxwell et al., was modeled (See Section B.6) and run through run_PROP.m to achieve the same thrust coefficient. Figure 2.17(a) contrasts Suresh et al.’s thickness noise pressure variation and the one obtained through this investigation’s calculations. The thickness noise is approximately equal with a negligible difference between them.

Similarly, loading noise was calculated and compared. This is shown in Figure 2.17(b), where the difference is noticeable. However, since the measurements are in-plane, loading noise would not be computed as a significant contributing factor to the total noise. What is important is that the wave behavior matches previous research, as this shows agreement between both calculations. Thus, the blade and its surface pressure show a similar behavior in space, showing the same periodicity and frequency. Therefore, the overall effect of this magnitude disparity is negligible.

Lastly, the total noise pressure variation is seen in Figure 2.17(c). Here, agreement is seen between the two approximation methods, but they differ vastly with what Boxwell et al. (1978) measured. As discussed in Sections 1.2 and 2.1.1, Farrasat’s 1A formulation of the FW-H equations does not account for the quadruple term. Hence, the broadband components are not considered in the computation. The missing noise between this author’s results and Suresh et al.’s with respect to Boxwell et al’s (1978) is due to these unaccounted sources of noise. Consequently, the calculations are in
agreement with other approximations performed by researchers. The same cannot be said when compared to actual measurements as the broadband terms are currently not captured with the current approach.
Figure 2.18 Calculated Sound Pressures and SPL compared against sample case results from PSU-WOPWOPv3 distribution (Hennes et al., 2017)

Similar to the pressure validation case above, comparing against Hennes et al’s (2017) results ensures the proper outputs are given. As seen in Figure 2.18, the thickness noise pressure output matches that of the output. Loading noise, however, is vertically offset and with a slight magnitude difference. This translates into a minute difference in total sound pressure. This is probably caused by the difference
in which the pressure input is being fed to the code. Hennes et al. used the computed compact pressure, where the pressure is resolved into a single resultant force vector for each element. The calculated case through our code uses the total surface pressure distribution, hence one pressure value per each node. Therefore, the difference in magnitude and vertical offset could be explained by this difference.
3. Results and Discussion

Having shown the noise approximations to be valid, two different rotors were analyzed. As mentioned in the introduction, the UAM vehicle proponents state that the use of electrically driven proprotors will reduce the noise signature of the vehicles when compared against current generation helicopters. Therefore, a regular helicopter type rotor was compared to a proprotor specifically designed for an eVTOL type vehicle.

The standard RC helicopter rotor system chosen uses the Goblin SAB TB 700 Blade (2018), referenced onwards as the Goblin blade. The blade is rectangular shaped, untapered, untwisted, using a NACA0012 airfoil with a chord length of 2.5in. The two-bladed rotor’s radius is of 30.886in, and the root cut-out is 6in. Its operation is stated by the manufacturer between 1400 and 2200 RPM. Figure 3.1 shows the different rendered views of the blade and rotor.

The proprotor uses the EFRC GOE550 blade developed by Christian Hantz (Hantz, 2018), hereinafter mentioned in the document as the Hantz blade. It has a straight leading edge, tapered trailing edge, cubic twist distribution, and uses a GOE550 airfoil. At zero collective, the tip angle is also zero while the root cut-out section has 21°. Rotor radius is 30.6in, and the chord length varies from 4.656in at the root cut-out to 1.5in at the tip. It’s operating range is between 950 and 1650 RPM. Figure 3.2 depicts the blade geometry.
Figure 3.1 Rendered views of the SAB Goblin TB 700 rotor blade. Chordwise view (a) is looking inboard from the tip. Planform view (b) has the leading edge towards the bottom.

The measurements were taken at three different RPM settings. For each setting the thrust, torque, and power were held constant. This approach allowed a fair comparison as both rotors were designed to operate at different rotation velocities. Two observer locations were tested at each RPM. One at 1.5 blade radii from axis of rotation and in plane with the disk. The second at the same radial distance but 30° down with respect to the TPP. These observer positions are shown in Figure 3.3 below. The results gathering matrix can be seen in Table 3.1.
Figure 3.2 Rendered views of the EFRC GOE550 "Hantz" (Hantz, 2018) proprotor blade. Chordwise view (a) is looking inboard from the tip. Planform view (b) has the leading edge towards the bottom.

Figure 3.3 Observer position depiction with respect to rotor disk
Table 3.1

Table Analysis Matrix

<table>
<thead>
<tr>
<th>Blade</th>
<th>RPM</th>
<th>θ</th>
<th>$C_T$</th>
<th>$C_P$</th>
<th>Thrust [lbs]</th>
<th>Power [hp]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hantz</td>
<td>700</td>
<td>15</td>
<td>1.46E-02</td>
<td>1.51E-03</td>
<td>11.9</td>
<td>0.4</td>
</tr>
<tr>
<td>Goblin</td>
<td>1000</td>
<td>10.04</td>
<td>6.78E-03</td>
<td>4.41E-04</td>
<td>11.9</td>
<td>0.4</td>
</tr>
<tr>
<td>Hantz</td>
<td>1000</td>
<td>14.85</td>
<td>1.47E-02</td>
<td>1.48E-03</td>
<td>24.5</td>
<td>1.2</td>
</tr>
<tr>
<td>Goblin</td>
<td>1400</td>
<td>10.4</td>
<td>7.14E-03</td>
<td>4.45E-04</td>
<td>24.5</td>
<td>1.1</td>
</tr>
<tr>
<td>Hantz</td>
<td>1300</td>
<td>13.9</td>
<td>1.49E-02</td>
<td>1.38E-03</td>
<td>41.9</td>
<td>2.5</td>
</tr>
<tr>
<td>Goblin</td>
<td>1900</td>
<td>9.2</td>
<td>6.62E-03</td>
<td>4.21E-04</td>
<td>41.9</td>
<td>2.5</td>
</tr>
</tbody>
</table>

The results discussion has been divided into the three RPM settings. The Perceived Noise Level (PNL) was compared for each, comparing only the first 24 1/3 octave bands (44.7Hz-11,220Hz). It was calculated as per the FAR Part 36, Appendix A, Section 4 (2018). SPL plots are shown both in actual frequency and normalized per blade passage frequency. This analysis focused on the qualitative difference between both.

3.1 Low RPM

The SPL frequency distribution for the first test case can be seen in Figure 3.4. In plane, the proprotor observed higher sound level peaks. Given the geometry discussed before, this is an expected behavior as in this observer location the thickness noise is predominant. Nevertheless, the distribution seen in Figure 3.4(b) exhibited fewer number of peaks than that of the rotor and lower signatures in the high frequency range, shown in Figure 3.4(a). Thus, the total perceived noisiness is less for the Hantz rotor than the Goblin one. Table 3.2 summarizes the calculated difference to be 7.9dB lower.
Figure 3.4 Low RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)

Table 3.2
Perceived Noise Level - Location 1, Low RPM

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>$\theta_c$</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.2</td>
<td>Goblin</td>
<td>1000</td>
<td>10.04</td>
<td>109.9</td>
<td>–</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Hantz</td>
<td>700</td>
<td>15.00</td>
<td>102.0</td>
<td>-7.9</td>
</tr>
</tbody>
</table>

Changing the observer location to 30° below the plane had a great impact on the SPL distribution. Displayed in Figure 3.5, the reader can see similar peak levels between both rotors. However, the helicopter rotor presented again more peaks and a higher spectrum towards the higher frequencies, as shown in Figures 3.5(a) and 3.5(b) respectively. The PNL difference, as stated in Table 3.3, again was lower for the Hantz rotor.

Table 3.3
Perceived Noise Level - Location 2, Low RPM

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>$\theta_c$</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2</td>
<td>Goblin</td>
<td>1000</td>
<td>10.04</td>
<td>66.0</td>
<td>–</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Hantz</td>
<td>700</td>
<td>15.00</td>
<td>58.6</td>
<td>-7.4</td>
</tr>
</tbody>
</table>
3.2 Medium RPM

For the medium RPM range test cases, the behavior was similar. In-plane, presented in Figure 3.6(b), higher peaks were displayed for the proprotor. These, however, tapered faster than those of the Goblin blade. Frequency-wise, more signature peaks were detected for the helicopter type rotor. Consequently, the observed PNL was lower for the Hantz rotor as quantified in Table 3.4.

Table 3.4
Perceived Noise Level - Location 1, Medium RPM

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>θ_c</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.2</td>
<td>Goblin</td>
<td>1400</td>
<td>10.40</td>
<td>116.5</td>
<td>–</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Hantz</td>
<td>1000</td>
<td>14.85</td>
<td>109.3</td>
<td>-7.2</td>
</tr>
</tbody>
</table>
Figure 3.6 Medium RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)

The first peaks were similar for both cases when moving the microphone location 30° below the plane of rotation. However, once again, the proprotor tapered quicker and exhibited lower high-frequency signatures, as seen in Figure 3.7(b). As with the previous cases at lower RPMs, the rotor displays a large quantity of peaks as observed in Figure 3.7(a). Thus, as stated in Table 3.5, the PNL was lower for the proprotor.

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>$\theta_c$</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2</td>
<td>Goblin</td>
<td>1400</td>
<td>10.40</td>
<td>81.7</td>
<td>–</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Hantz</td>
<td>1000</td>
<td>14.85</td>
<td>70.8</td>
<td>-10.9</td>
</tr>
</tbody>
</table>
Figure 3.7 Medium RPM test cases. Microphone positioned at 1.5R, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)

3.3 High RPM

At the higher RPM range, a similar trend is observed. Figure 3.8(b) shows that the proprotor initial peak is greater in magnitude than the rotor, with the subsequent peaks tapering off at a higher rate than that of the rotor. However, Figure 3.8 shows the baseline magnitude remains higher for the proprotor in this case. Even though the Goblin rotor has noticeably more peaks in these runs than before, its recorded PNL, shown in Table 3.6, is lower than the Hantz rotor. This behavior is different than what was observed before.

Table 3.6
Perceived Noise Level - Location 1, High RPM

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>θc</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.2</td>
<td>Goblin</td>
<td>1900</td>
<td>9.20</td>
<td>67.5</td>
<td>–</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Hantz</td>
<td>1300</td>
<td>13.90</td>
<td>70.9</td>
<td>+3.4</td>
</tr>
</tbody>
</table>

This difference can be traced to the change in frequency where the first peak is occurring. In cases 5.1.1-6.2.2 shown before, the first SPL peak of the proprotor
Figure 3.8 High RPM test cases. Microphone positioned 1.5R in-plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)

consistently appeared before the lower-bound cut-off region of the first 1/3 octave (44.7 Hz). It was therefore unaccounted in the PNL calculation. At this RPM, however, the first blade pass is within this range, and thus augments the perceived noisiness.

Changing the observer location by 30° produced the same tendency as the low and medium RPM test cases. The Goblin blade gained notoriety, having similar initial peak magnitudes and then narrowing at a slower rate than the proprotor, as seen in Figure 3.9(b). Contradictory to the in-plane observation and in agreement with the lower RPM runs, Figure 3.7(a) shows the baseline total SPL of the Hantz bladed rotor is lower than the rectangular bladed one. Comparing the PNLs for both presented cases, the proprotor presented -3.8dB lower noise levels than the rotor.
Figure 3.9 High RPM test cases. Microphone positioned at 1.5R, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)

Table 3.7
Perceived Noise Level - Location 2, High RPM

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>θc</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2</td>
<td>Goblin</td>
<td>1900</td>
<td>9.20</td>
<td>75.7</td>
<td>–</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Hantz</td>
<td>1300</td>
<td>13.90</td>
<td>71.9</td>
<td>-3.8</td>
</tr>
</tbody>
</table>

Given the consistency seen throughout the test cases, a third observer position was analyzed for the higher RPM setting. The initial observer position stated of 1.5 radii is an highly unlikely position from an operational standpoint. Hence, the new observer location was chosen to be at 10 feet horizontally from the axis of rotation and 30° down. This microphone location is depicted in Figure 3.10. The arbitrary chosen position suggests an observer is at a reasonable distance from a hovering vehicle.

The behavior seen for this location continues the observed conduct from the previous cases. The pressure peaks emitted by the rotor recedes at a decreased rate when compared to those of the tapered blade, as observed in Figure 3.11(b). Also, due
to the higher RPM, the former shows more peaks than the latter. Overall, in Figure 3.11(a), the magnitudes and baselines, however, are closer than in the previously discussed cases. Analyzing the PNL signature of both blades, the proprotor registered a lower noise level than the regular helicopter rotor. The total difference is of -1.6dB, as seen in Table 3.8.

Table 3.8

<table>
<thead>
<tr>
<th>Case</th>
<th>Blade</th>
<th>RPM</th>
<th>$\theta_c$</th>
<th>PNL [dB]</th>
<th>Diff [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.2</td>
<td>Goblin</td>
<td>1900</td>
<td>9.20</td>
<td>66.3</td>
<td>–</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Hantz</td>
<td>1300</td>
<td>13.90</td>
<td>64.7</td>
<td>-1.6</td>
</tr>
</tbody>
</table>
Figure 3.11 High RPM test cases. Microphone positioned at 10ft horizontally from the axis of rotation, 30° below the plane. SPL presented in actual frequency (a) and normalized per blade passes [2/rev] (b)
4. Conclusion

A script was created that performs proprotor noise approximations. The tool, written in MATLAB, uses an iterative process within the Blade Element Momentum Theory to calculate inflow. The inflow is calculated using the aerodynamic coefficients obtained with XFOIL, where the surface pressure distribution is also determined. This surface pressure is then encoded and parsed to the rotor aeroacoustic solver PSU-WOPWOPv3 to establish the time varying noise pressure levels and sound pressure level spectrum. These values were validated against established literature to ensure proper the interfacing and applied theory worked accordingly.

Through the study performed within the framework of this research, it was determined that indeed a proprotor designed specifically for eVTOL propulsion can be quieter than a conventional helicopter blade. For the three explored cases, when comparing both blades at the same thrust and power levels, the prop-rotor demonstrated a lower noise signatures at given observer positions for two, while the third showed different results. Thus, the statement and reliance of the UAM market exponents on quieter vehicles possesses credibility and enables the development of this new technology. The use of electric motors to lower the RPM while increasing the torque proved to be the key factor in abating the sound signature when no observers remain in the plane of rotation.

The algorithm is usable, readable, and explained thoroughly in this document and the scripts themselves. The theory has been clearly stated and referenced, il-
lustrating the reason why the methods and tools implemented were chosen. All assumptions are summarized in Section 4.1. A study was conducted on the outcome sensibility to different inputs. It was determined that the amount of chordwise elements, observer time, and total data points have a great impact on the results. It was, therefore, recommended to use a minimum of 100 chordwise and 100 spanwise elements, 10 revolutions total observer time, and at least 2001 data points per revolution. The aerodynamic performance and estimation capacity has been shown to obtain reasonable results when compared to both purely experimental and computational outcomes found across the literature. The sound pressure levels have been contrasted and ratified against both experimental and computational results as well, justifying a correct implementation.

4.1 Limitations

As stated previously throughout the document, several limitations and assumptions were made. For the aerodynamic analysis of the blade, no transonic effects were accounted for. Thus, tip speeds must remain subcritical. As the BEMT solution and fixed point (FP) iteration method do not consider these effects in the flow, the user must ensure the operating points are below the subcritical speeds for the airfoil.

Another limiting factor is that the airfoil cannot be operating at stall regions for extended parts of the blades. XFOIL does not provide results past stall. In some cases, being in the stall region causes the whole execution to crash. If the blade stall regions were small, the discussed inscribed interpolation methods could overcome the lack of convergence. However, this might not be the case for all runs. If many elements had
to undergo interpolation, the total result will be an extensive interpolation rather than results obtained through XFOIL, and would thus be unrealistic. This may not provide an accurate result.

On the rotor side, the model only accounts for axial flow. There is no span-wise flow on the blade, no wake interactions, and most importantly only an assumed steady state pressure distribution on the surface. This is a limitation because, even though it encompasses the vertical ascent and the forward motion for the proprotor, it does not account for any maneuvering or perturbations. These, however, are the sources for broadband noise that can increase the total perceived noise level significantly.

Unsteady loading on the blades due to vehicle angle of attack or angle of sideslip will affect the pressure distribution on the rotor disk. Blade vortex interactions also have great effect on the perceived noise. Neither of these sound sources are modeled in the current implementation. The only considered sources are blade loading and thickness. It should be noted that there is a lack of spanwise flow, and no 3D tip effects caused by vortices were considered. It has been shown, nonetheless, that the BEMT, with the tip-loss corrections, can provide accurate results.

Lastly, even though the model includes axial flow as part of the BEMT, this behavior has not been validated. Further validation would be required to demonstrate that this capability is completely functional. This section could also expand on the noise analysis capabilities, as right now only a static observer is considered.
4.2 Future Work

Future work to achieve the desired goal of a design cycle analysis tool should first focus on reducing some of the limitations stated above. Unsteady loading on the blade can be calculated iteratively by positioning the blade at the different angles through the rotating plane for advancing flow. The other unsteady loads need to be studied further in order to propose a solution to the limits they impose.

Expansion of the observer location could encompass several flight phases or a flight path. This could be done either internally with the capabilities PSU-WOPWOP has or implement an iterative method to have the capability within MATLAB. Either option is feasible. However, the internal iterative MATLAB method would enable post processing to be done within the environment, whereas by using PSU-WOPWOP’s internal capabilities will require a third party software.

Further study of the capabilities PSU-WOPWOP has could expand other areas of interest. The software can handle multiple scenarios, multiple rotor systems, and rotor-fuselage interactions. All these areas are of interest, and could provide added areas of study for future research.

Finally, adding multiple rotor capabilities to our tool to exploit PSU-WOPWOP’s competence will increase its usefulness. To simulate noise generated by a full UAM vehicle would be of great interest. This expansion should encompass not only inter-rotor interactions, but also multi-wake-interactions and rotor wake and body interactions.
REFERENCES


### Table A.1

*Grid Sensitivity Total Run Time, No Weighting*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Run-Time [sec]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.1</td>
<td>6.99</td>
<td>-</td>
</tr>
<tr>
<td>1.3.1</td>
<td>7.33</td>
<td>4.9%</td>
</tr>
<tr>
<td>1.1.1</td>
<td>9.37</td>
<td>34.1%</td>
</tr>
<tr>
<td>1.1.2</td>
<td>14.94</td>
<td>113.7%</td>
</tr>
<tr>
<td>1.2.2</td>
<td>15.66</td>
<td>124.1%</td>
</tr>
<tr>
<td>1.3.2</td>
<td>17.42</td>
<td>149.2%</td>
</tr>
<tr>
<td>1.1.3</td>
<td>25.29</td>
<td>261.8%</td>
</tr>
<tr>
<td>1.2.3</td>
<td>26.82</td>
<td>283.7%</td>
</tr>
<tr>
<td>1.3.3</td>
<td>31.19</td>
<td>346.2%</td>
</tr>
</tbody>
</table>

### Table A.2

*Grid Sensitivity Total Run Time, Tip Weighted*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Run-Time [sec]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1</td>
<td>6.41</td>
<td>-</td>
</tr>
<tr>
<td>2.2.1</td>
<td>6.53</td>
<td>1.8%</td>
</tr>
<tr>
<td>2.3.1</td>
<td>6.92</td>
<td>7.8%</td>
</tr>
<tr>
<td>2.1.2</td>
<td>14.71</td>
<td>129.2%</td>
</tr>
<tr>
<td>2.2.2</td>
<td>15.44</td>
<td>140.6%</td>
</tr>
<tr>
<td>2.3.2</td>
<td>17.73</td>
<td>176.3%</td>
</tr>
<tr>
<td>2.1.3</td>
<td>25.33</td>
<td>294.8%</td>
</tr>
<tr>
<td>2.2.3</td>
<td>27.33</td>
<td>326.0%</td>
</tr>
<tr>
<td>2.3.3</td>
<td>30.83</td>
<td>380.6%</td>
</tr>
</tbody>
</table>
Table A.3
*Grid Sensitivity Total Run Time, No Bias vs. Tip Bias*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Run-Time [sec]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.3</td>
<td>25.29</td>
<td>-</td>
</tr>
<tr>
<td>2.1.3</td>
<td>25.33</td>
<td>0.1%</td>
</tr>
<tr>
<td>1.2.3</td>
<td>26.82</td>
<td>-</td>
</tr>
<tr>
<td>2.2.3</td>
<td>27.33</td>
<td>1.9%</td>
</tr>
<tr>
<td>1.3.3</td>
<td>31.19</td>
<td>-</td>
</tr>
<tr>
<td>2.3.3</td>
<td>30.83</td>
<td>-1.2%</td>
</tr>
</tbody>
</table>

Table A.4
*Δτ and Data Point Sensitivity Total Run Time*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Run-Time [sec]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.3</td>
<td>4.36</td>
<td>-</td>
</tr>
<tr>
<td>3.2.3</td>
<td>4.83</td>
<td>1.8%</td>
</tr>
<tr>
<td>3.3.3</td>
<td>5.92</td>
<td>7.8%</td>
</tr>
<tr>
<td>3.1.2</td>
<td>6.71</td>
<td>129.2%</td>
</tr>
<tr>
<td>3.2.2</td>
<td>7.39</td>
<td>140.6%</td>
</tr>
<tr>
<td>3.3.2</td>
<td>8.43</td>
<td>176.3%</td>
</tr>
<tr>
<td>3.1.1</td>
<td>11.42</td>
<td>294.8%</td>
</tr>
<tr>
<td>3.2.1</td>
<td>11.79</td>
<td>326.0%</td>
</tr>
<tr>
<td>3.3.1</td>
<td>12.99</td>
<td>380.6%</td>
</tr>
</tbody>
</table>

Table A.5
*Observer time and Δτ Sensitivity Total Run Time*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Run-Time [sec]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1</td>
<td>4.22</td>
<td>-</td>
</tr>
<tr>
<td>4.1.2</td>
<td>3.27</td>
<td>-22.4%</td>
</tr>
<tr>
<td>4.1.3</td>
<td>2.93</td>
<td>-30.6%</td>
</tr>
<tr>
<td>4.2.1</td>
<td>26.54</td>
<td>529.3%</td>
</tr>
<tr>
<td>4.2.2</td>
<td>20.34</td>
<td>382.3%</td>
</tr>
<tr>
<td>4.2.3</td>
<td>17.34</td>
<td>311.2%</td>
</tr>
<tr>
<td>4.3.1</td>
<td>130.62</td>
<td>2997.0%</td>
</tr>
<tr>
<td>4.3.2</td>
<td>100.98</td>
<td>2294.2%</td>
</tr>
<tr>
<td>4.3.3</td>
<td>86.22</td>
<td>1944.1%</td>
</tr>
</tbody>
</table>
Table A.6
*Grid Sensitivity Peak Loading Pressure, No Weighting*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Loading Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.3</td>
<td>-1.546</td>
<td>-</td>
</tr>
<tr>
<td>1.1.2</td>
<td>-1.473</td>
<td>-4.8%</td>
</tr>
<tr>
<td>1.2.3</td>
<td>-1.195</td>
<td>-22.7%</td>
</tr>
<tr>
<td>1.2.2</td>
<td>-1.133</td>
<td>-26.7%</td>
</tr>
<tr>
<td>1.1.1</td>
<td>-1.069</td>
<td>-30.8%</td>
</tr>
<tr>
<td>1.3.3</td>
<td>-1.027</td>
<td>-33.6%</td>
</tr>
<tr>
<td>1.3.2</td>
<td>-0.971</td>
<td>-37.2%</td>
</tr>
<tr>
<td>1.2.1</td>
<td>-0.804</td>
<td>-48.0%</td>
</tr>
</tbody>
</table>

Table A.7
*Grid Sensitivity Peak Loading Pressure, No Weighting*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Loading Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.3</td>
<td>-1.627</td>
<td>-</td>
</tr>
<tr>
<td>2.1.2</td>
<td>-1.625</td>
<td>-0.1%</td>
</tr>
<tr>
<td>2.1.1</td>
<td>-1.585</td>
<td>-2.6%</td>
</tr>
<tr>
<td>2.2.3</td>
<td>-1.262</td>
<td>-22.4%</td>
</tr>
<tr>
<td>2.2.2</td>
<td>-1.261</td>
<td>-22.5%</td>
</tr>
<tr>
<td>2.2.1</td>
<td>-1.228</td>
<td>-24.5%</td>
</tr>
<tr>
<td>2.3.3</td>
<td>-1.088</td>
<td>-33.1%</td>
</tr>
<tr>
<td>2.3.2</td>
<td>-1.085</td>
<td>-33.3%</td>
</tr>
</tbody>
</table>

Table A.8
*Grid Sensitivity Peak Loading Pressure, No Bias vs. Tip Bias*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Loading Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.3</td>
<td>-35.265</td>
<td>-</td>
</tr>
<tr>
<td>2.1.3</td>
<td>-36.849</td>
<td>4.5%</td>
</tr>
<tr>
<td>1.2.3</td>
<td>-35.406</td>
<td>-</td>
</tr>
<tr>
<td>2.2.3</td>
<td>-37.000</td>
<td>4.5%</td>
</tr>
<tr>
<td>1.3.3</td>
<td>-35.562</td>
<td>-</td>
</tr>
<tr>
<td>2.3.3</td>
<td>-37.164</td>
<td>4.5%</td>
</tr>
</tbody>
</table>
### Table A.9

**Δτ** and Data Point Sensitivity Peak Loading Pressure

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Loading Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3</td>
<td>-1.261</td>
<td>-</td>
</tr>
<tr>
<td>3.3.3</td>
<td>-1.261</td>
<td>0.0%</td>
</tr>
<tr>
<td>3.1.3</td>
<td>-1.251</td>
<td>-0.8%</td>
</tr>
<tr>
<td>3.2.2</td>
<td>-1.250</td>
<td>-0.9%</td>
</tr>
<tr>
<td>3.3.2</td>
<td>-1.250</td>
<td>-0.9%</td>
</tr>
<tr>
<td>3.3.1</td>
<td>-1.243</td>
<td>-1.4%</td>
</tr>
<tr>
<td>3.2.1</td>
<td>-1.243</td>
<td>-1.4%</td>
</tr>
<tr>
<td>3.1.2</td>
<td>-1.241</td>
<td>-1.6%</td>
</tr>
<tr>
<td>3.1.1</td>
<td>-1.236</td>
<td>-2.0%</td>
</tr>
</tbody>
</table>

### Table A.10

**Grid Sensitivity Peak Total Pressure, No Weighting**

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Total Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.3</td>
<td>-35.562</td>
<td>-</td>
</tr>
<tr>
<td>1.2.3</td>
<td>-35.406</td>
<td>-0.4%</td>
</tr>
<tr>
<td>1.1.3</td>
<td>-35.265</td>
<td>-0.8%</td>
</tr>
<tr>
<td>1.3.2</td>
<td>-34.083</td>
<td>-4.2%</td>
</tr>
<tr>
<td>1.2.2</td>
<td>-33.934</td>
<td>-4.6%</td>
</tr>
<tr>
<td>1.1.2</td>
<td>-33.802</td>
<td>-4.9%</td>
</tr>
<tr>
<td>1.3.1</td>
<td>-25.669</td>
<td>-27.8%</td>
</tr>
<tr>
<td>1.2.1</td>
<td>-25.560</td>
<td>-28.1%</td>
</tr>
</tbody>
</table>

### Table A.11

**Grid Sensitivity Peak Total Pressure, No Weighting**

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Total Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3</td>
<td>-37.164</td>
<td>-</td>
</tr>
<tr>
<td>2.3.2</td>
<td>-37.136</td>
<td>-0.1%</td>
</tr>
<tr>
<td>2.2.3</td>
<td>-37.000</td>
<td>-0.4%</td>
</tr>
<tr>
<td>2.2.2</td>
<td>-36.970</td>
<td>-0.5%</td>
</tr>
<tr>
<td>2.1.3</td>
<td>-36.849</td>
<td>-0.8%</td>
</tr>
<tr>
<td>2.1.2</td>
<td>-36.821</td>
<td>-0.9%</td>
</tr>
<tr>
<td>2.3.1</td>
<td>-36.292</td>
<td>-2.3%</td>
</tr>
<tr>
<td>2.2.1</td>
<td>-36.132</td>
<td>-2.8%</td>
</tr>
</tbody>
</table>
Table A.12
*Grid Sensitivity Peak Total Pressure, No Bias vs. Tip Bias*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Total Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.3</td>
<td>-35.265</td>
<td>-</td>
</tr>
<tr>
<td>2.1.3</td>
<td>-36.849</td>
<td>4.5%</td>
</tr>
<tr>
<td>1.2.3</td>
<td>-35.406</td>
<td>-</td>
</tr>
<tr>
<td>2.2.3</td>
<td>-37.000</td>
<td>4.5%</td>
</tr>
<tr>
<td>1.3.3</td>
<td>-35.562</td>
<td>-</td>
</tr>
<tr>
<td>2.3.3</td>
<td>-37.164</td>
<td>4.5%</td>
</tr>
</tbody>
</table>

Table A.13
*Δτ and Data Point Sensitivity Peak Total Pressure*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>Total Peak Pressure [Pa]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>-36.978</td>
<td>-</td>
</tr>
<tr>
<td>3.3.2</td>
<td>-36.971</td>
<td>0.0%</td>
</tr>
<tr>
<td>3.3.3</td>
<td>-36.918</td>
<td>-0.2%</td>
</tr>
<tr>
<td>3.2.1</td>
<td>-36.914</td>
<td>-0.2%</td>
</tr>
<tr>
<td>3.2.2</td>
<td>-36.907</td>
<td>-0.2%</td>
</tr>
<tr>
<td>3.2.3</td>
<td>-36.854</td>
<td>-0.3%</td>
</tr>
<tr>
<td>3.1.1</td>
<td>-36.570</td>
<td>-1.1%</td>
</tr>
<tr>
<td>3.1.2</td>
<td>-36.565</td>
<td>-1.1%</td>
</tr>
<tr>
<td>3.1.3</td>
<td>-36.519</td>
<td>-1.2%</td>
</tr>
</tbody>
</table>

Table A.14
*Δτ and Data Point Sensitivity - PNL*

<table>
<thead>
<tr>
<th>Case Number</th>
<th>PNL [dB]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>145.1</td>
<td>-</td>
</tr>
<tr>
<td>3.1.2</td>
<td>145.1</td>
<td>0.0%</td>
</tr>
<tr>
<td>3.1.3</td>
<td>145.1</td>
<td>0.0%</td>
</tr>
<tr>
<td>3.2.1</td>
<td>143.0</td>
<td>-1.5%</td>
</tr>
<tr>
<td>3.2.2</td>
<td>143.0</td>
<td>-1.5%</td>
</tr>
<tr>
<td>3.2.3</td>
<td>142.9</td>
<td>-1.5%</td>
</tr>
<tr>
<td>3.3.1</td>
<td>140.8</td>
<td>-2.9%</td>
</tr>
<tr>
<td>3.3.2</td>
<td>140.8</td>
<td>-2.9%</td>
</tr>
<tr>
<td>3.3.3</td>
<td>140.8</td>
<td>-3.0%</td>
</tr>
</tbody>
</table>
Table A.15
Observer time and $\Delta \tau$ Sensitivity - PNL

<table>
<thead>
<tr>
<th>Case Number</th>
<th>PNL [dB]</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1</td>
<td>144.9</td>
<td>-</td>
</tr>
<tr>
<td>4.1.2</td>
<td>144.8</td>
<td>0.0%</td>
</tr>
<tr>
<td>4.1.3</td>
<td>144.8</td>
<td>0.0%</td>
</tr>
<tr>
<td>4.2.1</td>
<td>135.2</td>
<td>-6.7%</td>
</tr>
<tr>
<td>4.2.2</td>
<td>135.2</td>
<td>-6.7%</td>
</tr>
<tr>
<td>4.2.3</td>
<td>135.1</td>
<td>-6.7%</td>
</tr>
<tr>
<td>4.3.1</td>
<td>125.5</td>
<td>-13.4%</td>
</tr>
<tr>
<td>4.3.2</td>
<td>125.5</td>
<td>-13.4%</td>
</tr>
<tr>
<td>4.3.3</td>
<td>125.5</td>
<td>-13.4%</td>
</tr>
</tbody>
</table>
B. Appendix - Algorithm Structure and Details

B.1 Structure and Details

The script has been written in a multiple file, stacked directory structure. This enables breaking key components into different files, arrange them in intuitively named directories, and general common-practices. There are only two visible files on the root directory, which are the calling script and a file to load the corresponding directories as working folder paths. Neither of these upper-level files perform any calculation, avoiding the final user to interfere with or damage the code.

As seen in Figure B.1, the directory structure chosen breaks everything into three key components: input files, output files, and running scripts. Each directory contains what its name implies. The scripts directory is broken down into two sub-directories: one for core calculations and one for the tools used. The former contains all running scripts to perform the requested tasks. The latter is where all the external executable programs are housed, each in their independent sub-directory. The functions employed to handle the executable are separated from their lodging directories to reduce the chance of affecting the outcome of the tools by avoiding interference with any file packaged with it. Custom functions and scripts that are not key or core components to the running script are also located within the tools folder.

By running the main run.Prop.m file, the initialization and core scripts are called accordingly. Sections B.2 and B.4 discuss how these work in detail, highlighting key code components. Sections B.6 and B.8 describe how these external executable files
were implemented within the MATLAB environment, and go into detail on how their respective outputs are imported to the work-space.

Figure B.1 Directory structure with brief explanation

B.2 Running Script

As it was briefly introduced before, the code is separated into several sub-scripts. These do different jobs, and were separated to allow a better workflow for the author and any future user. The main calling script is where the user selects what the code will do, through logic switches. Here the user selects which of the different options available to him he will use to command the different sub-routines. Seen in Listing B.1 are the main options the user can select.

Listing B.1. User option switches in run_Prop.m

```plaintext
21  % Setup Switches [0=Off, 1=On; unless specified]
22  sw.geom = 0; % Use m-file inputs (0) or open geometry editor (1)
23  sw.aero = 1; % Use Xfoil in all elements (1) or just for general Cla (0)
24  sw.perf = 0; % Calculate hover performance curves (1), advancing flight
```
As seen above, there are five switches:

- **sw.geom** controls whether the user would like to modify a custom geometry on to be analyzed. If the switch is set to ON, when the script is executed, an editor window will open with a sample geometry file which the user can edit. Once completed, the user must save the file and click ”Ok” to continue. If set to OFF, one of the pre-loaded geometry files must be uncommented in the Blade Input section. More on this later.

- **sw.aero** commands whether XFOIL is going to be used for the aerodynamic calculations on the blade, or the user specified $C_l\alpha$ and $C_d\alpha$ will be used. If set to ON, XFOIL will be used to calculate the aerodynamic coefficients for each element.

- **sw.perf** determines what the desired output of the code will be. If set to ON, the selected geometry will be tested at different collective angles, rpms, and advancing ratios. This option will output a file containing the $C_T$, $C_P$, and $J$ maps, and plot them. NOTE: If this option is selected, **sw.noise** will be overridden and turned off.

- **sw.noise** decides if noise approximation calculations will be performed or not. If set to ON, after the blade is analyzed for the static performance, the aeroacoustic approximation will be calculated. If the previous switch (**sw.perf**) is set to ON, this selection will be overridden and turned off.
• *sw.bemt* allows the user to select which BEMT code to use. There are different version coded, mainly to provide comparative results between distinct theoretical applications.

• *sw.bias* if set to ON the blade element distribution will have a tip-weighted bias, using unequal spacing with decreasing width towards the tip. If set to OFF, the blade elements will be equally spaced across the total span.

Once the switches have been set as per the user’s selection, there are two other choices to perform before the script can be executed. The first one is which pre-loaded geometry file to use, if *sw.geom* has been set to OFF. The user has to uncomment the desired file name to load the pertaining data. Shown in Listing B.2 is the section of *run.Prop.m* where these commands are, showcasing *HantzBlade.m* as the geometry to be loaded if run. There are a total of five pre-configured geometries, each shown and discussed in Section B.3. The second one is a tool to locate the noise observer location to one of the five microphones used for the experimental measurement performed at the EFRC. Note that this second option needs to be commented out if it is to be used, and also that *sw.noise* must be turned on.

Listing B.2. Pre-loaded blade geometries, as seen in *run.Prop.m*

```matlab
33 34 35 36 37 38 39
33 \texttt{\begin{tabular}{l} 34 \texttt{HantzBlade; } \%EFRC GOE550 blade (Hantz 2018) 35 \texttt{GoblinBlade; } \%EFRC Goblin Blade 36 \texttt{Prop5868; } \%NACA TR650 Propeller \#5868–9 37 \texttt{wopCase1blade; } \%WopWop Sample Case 1 blade 38 \texttt{UH1Hscaled; } \%UHIH Rotor., 1/7 scale (Boxwell 1978) 39 \texttt{CardonnaBlade; } \%Blade Used for testing in Cardonna and Tung (1981) 33 \texttt{\end{tabular}}}
```

B.3 Geometry Input Scripts

The geometry input file has three sections: one for the flight or environmental data, one for the rotor properties, and one the blade geometry. A fourth optional section features observer location and time limits to be used with the aeroacoustic
tool desired. If \textit{sw.geom} is set to ON, a MATLAB editor window will open with a sample geometry file. This file can be edited and saved to use it with the analyzing code. Alternatively, if this geometry is going to be used repeatedly, the user can save this file as a different name and load it in the \textit{run.Prop.m} input section.

\textbf{B.3.1 Environmental and Flight Properties}

The code was originally intended to perform flight parameter calculations along with the noise approximation generated by these phases of flight. However, to limit the scope and breadth for this particular application, only static cases are being analyzed. Having said that, the user can still select a climb velocity, a forward velocity, and a flight altitude. These inputs are seen in Listing B.3. It is worth noting that in this release, forward velocity remains unused and is not considered for any calculation.

Listing B.3. Environmental and Flight inputs, as seen in \texttt{\input\geometry.m}

```matlab
7 \% flight data
8 flight.Vclimb = 2; \%[ft/s] – Climb/Axial Velocity
9 flight.Vfwd = 0; \%[ft/s] – Forward flight speed
10 flight.alt = 0;
```

\textbf{B.3.2 Rotor Properties}

The rotor properties, as shown in Listing B.4, encompass telling the code how many blades comprise a rotor, how many rotors the vehicle has, and what is the Mach number at the blade tips. Alternatively, if \textit{rotor.macht} is left as 0, then \textit{rotor.rpm} must be defined. If a tip Mach number is defined, the RPM will be calculated based on this value and \textit{rotor.rpm} will be ignored.

Listing B.4. Rotor inputs, as seen in \texttt{\input\geometry.m}

```matlab
12 \% rotor data
13 rotor.Nr = 2; %Number of rotors
14 rotorNb = 3; %Number of blades
15 rotor.macht = 0; %Tip Mach, overrides rpm. 0 uses rpm
```
B.3.3 Blade Properties

The blade properties, shown in Listing B.5, are subdivided into four categories. The first is the number of elements into which the blade will be divided. It is recommended to use at least 50. The second pertains to the total number of airfoil panels to be used in the aerodynamic calculations. If this value is omitted, the default is set to 50 total elements. A discussion on how this affects the total outcome can be found in Section 2.3.1. The next section covers the geometric properties. Here the user defines the blade radius (in feet, measured from the axis of rotation), the blade root cut-out (feet, from the axis of rotation), and the blade leading and trailing edge functions. These functions are based on $r$, the non-dimensional radial location defined as $r = y/R$. Therefore, $r = 1 \implies y = R$, and $r = 0$ is the axis of rotation. The leading edge and trailing edge must be defined as a function, or else there will be a run time error. If the desired inputs are constants, these functions can be defined as such, just ensure that the @($r$) remain ahead of the constant value. i.e.: blade.LE=@($r$)1;

The next section goes over the blade angles. Here the blade twist is defined as a function of $r$ as well. The rotation center location of the airfoil is determined here, in terms of percent chord $x/c$. If this value is set to 0, the airfoil will be rotated using in the amount determined by the twist function and span-wise location about the leading edge line. This term, therefore, construes where the axis of rotation is located with respect to the chord position. The collective angle is added to the blade twist at each element, effectively equating the angle commanded through the swash-plate or governor collective. Lastly, the airfoil name must be defined, as well as the $c_{l_{\alpha}}$ and $c_{d_{\alpha}}$ to be used if $sw.aero$ is set to OFF. A limitation of the current code implementation is that only a single airfoil can be used throughout the blade geometry. Later implementations will enable blended airfoil capabilities.
Listing B.5. Blade data inputs, as seen in `\input\geometry.m`

```matlab
18 \%
19 \% blade data
20 blade.el = 100; \% number of blade elements
21 blade.pn = 50; \% number of chordwise surface panels (TOTAL i.e.: Upper + Lower)
22 \%
23 \% blade geometry
24 blade.R = 4; \% [ft] – Blade length / rotor radius
25 blade.root = 0.5; \% [ft] – Root cut-out / hub radius
26 blade.LE = @(r) .5 – .375*r; \% [ft] – Leading Edge function wrt non-dim r
27 blade.TE = @(r) -.5+.375*r; \% [ft] – Trailing Edge function wrt non-dim r
28 \%
29 \% blade angles
30 blade.twist = @(r) -25*r+25; \% [deg] – Linear twist slope. 0 at tip.
31 \% Func wrt non-dim r
32 \% center of twist rotation
33 \%
34 \% blade airfoil
35 \%
36 \% custom airfoil name for Xfoil
37 blade.Cla = @(alpha) 3.33;
38 \%
39 \% Cd0 = @(alpha) 0.011;
```

B.3.4 Custom Aeroacoustic Options

If `sw.noise` is set to ON, then aeroacoustic calculations will be performed. The user can choose the observer position by modifying lines 37-39 in `geometry.m`, as shown in listings B.6. If this structure is omitted, the values will default to a position 60.72m away in the x direction and 5.31m below the rotor disk. Lines 41-43 pertain to the total observer time the pressure will be integrated for, and the total points to be calculated along that given time-frame. Finally, line 45 simply allows the user to describe their own custom directory where the aeroacoustic results will be saved. This is an useful function if multiple runs will be performed without overwriting the raw results.

Listing B.6. Custom aeroacoustic inputs, as seen in `\input\geometry.m`

```matlab
35 \%
36 \% WOPWOP custom commands
37 \% Microphone position
38 wop.obs.x = 3*blade.R; \% [ft]
39 wop.obs.y = 0; \% [ft]
40 wop.obs.z = 0; \% [ft]
```
B.3.5 Pre-defined Blade Geometries

As mentioned before, there are five geometries pre-loaded with the code release. These are the two used in experiments at the EFRC, and the three used for validation purposes. The two used for experiments were used to assess the prediction capabilities of this code with respect to real-life noise measurements. The other three were used to compare against published data for either performance analysis, noise prediction, or both. The following paragraphs will briefly describe each rotor and blade geometries, based on their names as displayed in Listing B.2. The user can also modify these files as he pleases, or use them as templates for other geometries.

_HantzBlade_ loads the geometry for the proprotor blade designed in-house at the EFRC by Christian Hantz (Hantz, 2018). It is a high twist, straight leading-edge, tapered trailing-edge blade that uses the _GOE-550_ airfoil profile. It was designed to carry 25 lb at hover conditions, and provide a propulsive thrust of 2.5 lb during a cruise flight of 55 mph (Hantz, 2018). The blade planform and front views can be seen in Figure B.2.

_GoblinBlade_ calls for the blade geometry for the Goblin Model Helicopter SAB700 rotor system. This rotor consists of two NACA0012 blades with no twist or taper. The rotor is used in model helicopters, can hover 25 lb, and has seen extended testing at the EFRC. This is the blade against which the Hantz’s blade is being compared. Performance-wise, the proprotor was designed to replace this blade in the rotor system for an unmanned air vehicle that transitioned from hover to forward flight.
Figure B.2 EFRC proprotor blade shape render

Figure B.3 Goblin Model Helicopter SAB700 blade shape render
Prop5868 inputs NACA’s Propeller No.5868, modeled in Figure B.4. Since NACA carried out several performance tests on different propellers in the 1940’s, this geometry was chosen because it is both referenced in several sources (McCormick Jr., 1999, 1995) and is a great specimen to test complex geometric input to the code. In the Technical Report No.684, Hartman and Biermann analyzed six full scale propellers, detailing the blade geometry and the propeller performance (Hartman & Biermann, 1940). Due to the varying thickness, high twist, and elliptical planform shape, it provided a great way to test the geometry assembly and encoding. Besides serving as a test case for the geometric input, the calculated performance was used to validate the results of the BEMT. These results are later discussed in Section 2.4.

\[ 
\begin{align*}
\text{(a) Planform view} \\
\text{(b) Front View}
\end{align*}
\]

Figure B.4 NACA Propeller No.5868 blade shape render

wopCase1blade imports WOPWOP’s Sample Case 1 Gyrodyne blade shape. With the copy of PSU-WOPWOP provided graciously by Dr. Kenneth Brentner were included five sample cases to verify the software installed correctly. One of these cases’ geometries was decoded and converted to an input to this code as means of
testing / verifying the surface pressure calculated and the noise levels generated by the same blade. The blade has a symmetric airfoil, straight leading edge, slightly tapered trailing edge, and no twist. A render of it can be seen in Figure B.5, and the results used for validation found in Section 2.4.

![Planform view](image)

(a) Planform view

![Front View](image)

(b) Front View

*Figure B.5 WopWop Sample Case 1 Gyrodyne blade shape render*

*UH1Hscaled* loads a 1/7th scale of the UH1H blade. Suresh et al. modelled this blade for aeroacoustic calculations using CFD (Suresh et al., 2018), comparing against those measured by Boxwell et al. four decades earlier (Boxwell et al., 1978). This provided another great data set to compare and validate the values calculated with this code, as Suresh et al. provided $C_p$ results as well as the Acoustic $\Delta P$ for the rotor system.
B.4 Main Script

The main script is called `blade_calculations.m` resides in the `/scripts/core` directory, and will be discussed at length in this section. As mentioned before the, running script just contains only inputs that the user can modify to perform the requested calculations. These initial calculations were separated into this main file for two reasons. First, it provides a safeguard to the underlying code by reducing the chance of disarraying any line of code. Second, it provides a clear and simple User Interface, even allowing for a future GUI implementation. This m-file grabs the data loaded through the geometry input files and performs the logic sequences according to the switches as well as the initial calculations.

The file assembles the geometry of each blade element by applying the equations defined in the geometry section. As seen in Listing B.7, the elements are not equally spaced but rather with a sine weighted distribution (line 30). A comparison between the linear element interval with respect to the sinusoidal one can be seen in Figure B.7. This distribution allowed for more definition towards the tip region, where the aerodynamic reactions are greatest. Line 31 applies the weighting to the blade geometrical constrains, while line 32 calculates the element width. Line 34 calculates
the center location of the element, while lines 35-36 the inboard and outboard element bounds.

Listing B.7. Element spanwise distribution calculations, as seen in \scripts\core\blade_calculations.m

```matlab
29 %element length (tip weighted distribution)
30 if sw.bias==1
31    rWeight = sin(0:pi/(2*blade.el):pi/2);
32 else
33    rWeight = linspace(0,1,blade.el+1);
34 end
35 rWalls = (rWeight.*(blade.R-blade.root))+blade.root;
36 blade.dr = rWalls(2:end)-rWalls(1:end-1);
```

(a) Linear Distribution  (b) Tip Biased Distribution

\textit{Figure B.7} Linear span-wise element distribution versus tip-biased element spacing

After the element distribution has been set, the values shown above are used to calculate other fixed geometrical parameters. Observed in Listings B.8 Line 38 non-dimensionalizes the elements location, while lines 40-41 calculate the element’s twist angle ($\theta$) and the pitch angle $\beta$. The chord of the elements are calculated in line 42, with a logic check to ensure there are not any negative chord values seen in line 43. Finally, lines 44-47 calculate the element’s planform area, the blades total planform area, the rotor solidity, and the differential solidity of each element.

Listing B.8. Element characteristics calculations, as seen in \scripts\core\blade_calculations.m

```matlab
38 blade.x = rWalls(1:end-1)+(blade.dr./2); %center x-coord of element
39 blade.xi = rWalls(1:end-1); %inner x-coord of element
```
Once the geometrical calculations are completed, the pertaining logic based on the switches get activated, and the script calls the BEMT function. This function is discussed at length in Section B.5. Once the BEMT calculations have been completed the main script then takes the data structure and processes it for visualization, which is discussed in Section B.7. If `sw.noise` is on, the process will then continue and call the calling function to WOPWOP, which is explained in Section B.8 Lastly, after all calculations have been completed and post-processing has been performed, the result structures are saved into the output folder as two separate files: one for the blade parameters and one for the noise results. These are separated to enable performing several noise approximations with the same rotor aerodynamics. i.e.: analyzing the same rotor conditions at different observer locations or observer times.

### B.5 Blade Element Theory Function

The BEMT function receives the data structures pre-calculated by the main script in four structures: `flight`, `rotor`, `blade`, and `sw`. The flight input structure provides the atmospheric parameters in which the rotor is operating. The rotor contains the top level information pertaining to the rotor, such as number of blades, rpm, and disk area. The blade structure contains all the geometrical parameters discussed in Section B.4. The last structure is the switch holder, which provides computational logic so the function performs what the user intends to do.
Listing B.9.Blade Element Momentum Theory MATLAB function header
and initial calculations

```matlab
function [coeffs, airfoil, blade, rotor] = BEMT_v0(flight, rotor, blade, sw)

% BEMT V3

a = flight.a;%[ft/s]
rho = flight.rho;%[lb/ft^3]
u = flight.nu;%[ft^2/s]
rotor.Vaxial = flight.Vclimb;%[ft/s]

%number of airfoil panels

omega = rotor.rpm*(2*pi/60); %[rad/s]

omega_r = omega.*blade.x;%[ft/s]
V=rotor.Vaxial.*ones(size(blade.x)); %[ft/s]
VT = blade.R*omega;%[ft/s] Tip Speed (SCALAR)
x = blade.x;%[ft]
r = x/blade.R;%[nd] Non-dimensionalized element location
dr = blade.dr/blade.R;
lambda_c = rotor.Vaxial./omega_r;
lambda = V./VT; %[nd] See bottom of pg302 (V + VT)./omegar
beta = (blade.collective + blade.twist(blade.xi./blade.R)).*(pi/180);%

c = blade.c;%[ft]
Nb = rotor.Nb;%[nd]
D = rotor.D;%[ft]
A = rotor.A;%[ft^2]
u = rotor.rpm/60;%[rev/s]
J = rotor.Vaxial/(n*D);
```

The first section of the function, depicted in Listing B.9, grabs the data of interest
from the input structures and renames them to variables that are easier to keep track
of and understand. The variable names try to depict as closely as possible the actual
variable it is representing from the formulation. For example: Ω is represented as
omeg,
λ is represented as lambda. Before Proceeding to the actual BEMT calculation
however, XFOIL is called to pre-calculate the linearized $c_{l\alpha}$, $c_{d\alpha}$ and the drag polar
coefficients $c_{d0}$, $c_{d1}$, and $c_{d2}$ for the blade airfoil. This process is shown in Listing
B.10, where the airfoil is analyzed at the $M$ and $Re$ located at the $r = 0.75$ for
$0 \leq \alpha \leq 10$. The obtained $c_l$ and $c_d$ are then fitted accordingly to get the above
mentioned coefficients. Further detail on how the called function works is discussed
in Section B.6.
Having now the values to calculate the inflow based on the BEMT as discussed in Section 2.1.2, the function proceeds to do an element by element calculation of the inflow, using Equations 2.30 and 2.29. As mentioned above, the same formulations shown in Equations 2.30 and 2.29 are used, and the variable naming convention has been set to be as descriptive and similar to the equation formulation as possible. Showed below in Listing B.11 is the initial inflow calculation using Equation 2.30. This calculation is repeated iteratively for each blade element.

Listing B.11 Blade Element Momentum Theory MATLAB function initial inflow calculation

```matlab
% Predefine 1st F as 1
F = 1;

% Iterate for 5 times to get modified inflow
while m <= 10
    lambda(i) = sqrt(((sigma_r*Cl/(16*F))-(lambda_c(i)/2))^2 + ... (sigma_r*Cl+dtetha_r(i)*r(i)/(8*F)))-...
```

Listing B.10 Blade Element Momentum Theory MATLAB function initial linearized aerodynamic calculations

```matlab
%%run Xfoil once from 0 to 10 deg at tip mach and re to get average Cla ,
% Cd0 , and t_max/c of airfoil to use in equations
temp.coeff.aoa = [0:2:10];
[temp.coeff,temp.foil]=aerocalcs(blade.airfoil,temp.coeff.aoa,...
    blade.Re(fix(blade.el*.75)).*ones(size(temp.coeff.aoa)),...
    blade.Mach(fix(blade.el*.75)).*ones(size(temp.coeff.aoa)),false);
%convert aoa to rads for calculations
temp.coeff.aoa=temp.coeff.aoa.*((pi/180));

%%linear fit the Cl curve to get Cla and C10
f=fit(temp.coeff.aoa',temp.coeff.Cl',poly1');
%fit in the form 'f(x) = p1*x + p2'
Cla=f.p1;%[1/rad]
C10=f.p2;%[nd]
dC10=C10;

%%quadratic fit the Cd curve to get Cd0, Cd1 and Cd2
f=fit(temp.coeff.aoa',temp.coeff.Cd',poly2');
%fit in the form 'f(x) = p1*x^2 + p2*x + p3'
Cd0=f.p3;%[nd]
Cd1=f.p2;%[1/rad]
Cd2=f.p1;%[1/rad^2]
```
\[
((\sigma_r \cdot C_{la}/(16 \cdot F)) - (\lambda_{c(i)}/2)); \quad \% \text{Eq. 3.126 modified with 3.58}
\]

\[
f = (N/2) \cdot ((1 - \tau(i))/((\lambda(i) - (\lambda_{c(i)}/Nr))); \quad \% \text{Eq. 3.121}
\]

\[
F = (2/\pi()) \cdot \arccos(\exp(-f)); \quad \% \text{Eq. 3.120}
\]

\[
m = m + 1;
\]

\[
lambda(i) = \sqrt{((\sigma_r \cdot C_{la}/(16 \cdot F)) - (\lambda_{c(i)}/2))^2 +}
\]

\[
((\sigma_r \cdot C_{la} \cdot \theta_r(i) \cdot \tau(i)/(8 \cdot F))^2 -
\]

\[
((\sigma_r \cdot C_{la}/(16 \cdot F)) - (\lambda_{c(i)}/2)); \quad \% \text{Eq. 3.126 modified with 3.58}
\]

A note on the comments. The equation numbers commented at the end of each line relate to the ones as found in the literature reference used. This reference is stated in the function header, and unless otherwise specified, the equation number is the same as the one in that reference.

Once the inflow has been calculated, then a last run is performed to get the pressure coefficient distribution along the blade surface, as well as the lift and drag coefficients. These values are used to calculate the thrust, power and torque coefficients using Equations 2.8 2.9. These calculated values are appended to the blade and rotor structures and sent as an output. A new structure is also outputted which carries the coefficients and airfoil data extracted using XFOIL. The function can be seen in its entirety in Appendix C.4.

B.6 Xfoil Function

The XFOIL calling functions consists of two parts. The first is a function dedicated exclusively to run XFOIL with the given parameters, read the output files, and return the values in variable form. This is discussed in subsection B.6.1. The second prepares the variables to send the former, and post-processes it’s outputs in case there were errors during the execution. This approach was selected as means to avoid any issues if the execution stopped, or if no results were converged by the tool. This function is described in subsection B.6.2.
B.6.1 Execute Xfoil

The execution code, located in `\scripts\tools\Xfoil\xfoil.m` takes a total of five inputs. These are:

- Airfoil Coordinates, as an XY array, or known airfoil name, as a string. Within the execution directory lies a folder where airfoil data files are stored. The airfoil name must match any of those.

- Angle of Attack ($\alpha$) of the airfoil. Can be an array of several angles of attack. However, if this is the case, the next two inputs must also be arrays of the same size.

- Reynolds Number ($Re$) relative to the airfoil. As noted above, this input can be an array with the caveat that the number of array elements must match between the previous and next input variable.

- Mach Number ($M$) relative to the airfoil. Again, this can be an array of values as long as the length of the array is equal to that of the previous two inputs.

- Extra commands for XFOIL. Since there are plenty of options available, including the possibility to change airfoil shape, number of panels, and iterations until the code decides it does not converge to name a few, this variable input was setup to accept text commands to be parsed to the command window. To use this option, the user must input a string or an cell array of strings each containing the desired input. Each space character will be translated to a return command, and a plus (+) sign equates a single space. The user must ensure that there is an equal number of spaces at the end of the command to those used before in order to avoid improper execution.

The outputs are two structures. One contains the aerodynamic polar coefficients for the range of $\alpha$’s selected. The other contains the airfoil parameters that are specific
to each angle of attack, such as $c_p$ and the airfoil coordinates read from the data file. If the inputs are arrays, the polar will simply contain the derived $c_{l\alpha}$, $c_m$ and $c_{d\alpha}$ for the range of angles of attack and speeds. The foil structure will, however, contain a value for each of the array elements.

The script seen in Appendix C.5 has around 318 lines of code and is therefore not suitable to present in its totality in this section. However, it can be easily explained as it consists of five concise and distinctive steps. Lines 71-114 deal with the inputs, arranging them and filling any void should there be one. If no inputs are made, there are defaults set to avoid the code to crash.

The first step is to write the input command file. The way the XFOIL is interfaced within the MATLAB environment is by using calling the Window’s command terminal. To take advantage of this very useful MATLAB embedded function, the text inputs can be written to a file, which will be fed to the command terminal as is. The input commands to XFOIL are written into a temporary file in lines 118-181. For more information on the commands used or to understand what each command means, refer to Drela et al.’s XFOIL 6.94 User Guide (Drela & Youngren, 2001).

The second step consists on calling the windows executable file through windows command line. This can be seen in Listing B.12 below. Basically the code tells windows command to find the directory where the exe file lies, run it with the inputs found on the file called xfoil.inp, and save any of the written outputs to a file called xfoil.out. Lines 186-189 check whether there was a run time error, and will stop the execution of the MATLAB script if this happens.

Listing B.12. XFOIL executable calling MATLAB script

```
181   % execute xfoil
182   cmd = sprintf(['cd %s && xfoil.exe < xfoil.inp > xfoil.out',wd]);
183   [status,result] = system(cmd);
184   if (status~=0)
185       disp(result);
```
The third and fourth steps read the respective files containing the results. The file mentioned in step 2 above (xfoil.out) does not contain any actual result. Rather, it just shows any printed output XFOIL would have sent to the command screen for a user to see. Through the input command file, XFOIL was told to save the polar results into one file and the airfoil dump files into another. Lines 193-264 read the polar DAT file, and translates the results into MATLAB variables. Lines 265-299 do the same but for the dump file. Once both these files are read, they are deleted to avoid using hard drive space.

The fifth and last step assembles the data read from the files into data structures that are then outputted. If for some reason one of the files does not contain any data points due a failure during the execution, an error flag is added to the result structure and all values are set to zero. This comes in handy to detect any possible errors during the execution and allows to perform corrective actions if necessary.

**B.6.2 Interpolation of Aerodynamic Coefficients**

Since the execution of XFOIL is prone to not converge for certain values due to the changing flow parameters used in the BEMT solution, the obtained values must be post-processed to counter for any missing data points. To do so, a second function was created which can be found in \scripts\tools\aerocalcs.m. The first four inputs to aerocalcs.m are the same as xfoil.m. The last one is user a switch to plot the aerodynamic coefficients with respect to angle of attack. The first section of the function simply runs XFOIL through the user of xfoil.m one time per $\alpha$, $M$ and $Re$. This can be seen in Listing B.13.

```matlab
Listing B.13. XFOIL outputs post-processing MATLAB function
```
The next section of the function looks for any errors in the extracted results. This is now an easy endeavor thanks to the error flag added in the main calling function. If any errors were detected during the operation, then the missing data points will be generated through interpolation using the converged results. However, due to limitations in MATLAB’s interpolation methods, the data set needs to be divided into sections, separated by the maximum \( \alpha \). MATLAB cannot interpolate when there are two different dependent value for a single independent one. Since the inflow has an elliptical distribution, there is a local \( \alpha \) maximum, enabling to split the data set into two curves. The error seeking logic looks for the error flag, finds the index number of this case, and compares it against the \textit{alpha} max. Once identified which curve the missing point can be interpolated against, the missing value is calculated. A sample output of this execution plot is seen in Figure B.8. This plot will be generated if the fifth output to \textit{aerocalcs.m} is set to \texttt{true}. In it, blue points are valid output values from the XFOIL execution. The red marks are interpolated data points that had to be performed due to an error during the analysis. Having this implemented avoids
having "dead zones" of data on the blade distribution, as well as limiting the cases where the scripts crash and all data is lost.

![Graphs showing aerodynamic coefficients](image)

Figure B.8 Sample output of the aerodynamic coefficient interpolation results

B.7 Blade Plotting and Data Processing Function

Once the aerodynamic coefficients on the blade surface have been established, the obtained data needs to be post-processed for two purposes. First, to visualize the information. Second, to parse the information to WOPWOP in order to perform the acoustic approximation. To do so, a function was created. Function `blade_plotting.m` catches the airfoil XY coordinates, the coefficient of pressure, and the blade geometry data. It processes the data and assembles a 3D coordinates for the blade, where it assigns each node a pressure coefficient value. This values are in turn converted to gauge pressure values. After this post-processing has been completed,
the new 3d coordinates are exported and saved into the blade result structure. Figure B.9 below shows a sample representation of the output plot created by this function.

![Blade Pressure Distribution]

*Figure B.9* Sample output of the Pressure Coefficient distribution along the blade surface, shown with rotor performance parameters

The function also outputs a render of the full rotor, mainly for verification purposes. A key component to achieve these graphics is using a direction cosine matrix rotation of the blade airfoil based on the local pitch angle, airfoil chord length, and element location. These calculations are shown in Listing B.14, where the dimensions are applied to the non-dimensional 2D airfoil coordinates and then rotated according to the element geometry. This process is repeated for each element until the 3D coordinate grid is assembled.

Listing B.14.Blade Plotting function detail: 3D coordinate generation

```matlab
49   for i=n:-1:1
50       %get Angle of blade Element
51       theta=blade.beta(i);
52       %scale center to local element chord
53       center = repmat([blade.tw_center*blade.c(i); 0], 1, length(airfoil.x(airfoil.x<=1)));
54       %reset unscaled airfoil coordinates
55       v=[airfoil.x(airfoil.x<=1)'; airfoil.y(airfoil.x<=1)'];
```
%scale them to current element chord
v(1,:) = blade.c(i).*v(1,:);%scale to element chord
v(2,:) = blade.c(i).*v(2,:);%scale to element chord

%get thickness correction factor (if one is desired)
if isfield(sw,'tc')
    scale=abs(blade.tc(blade.r(i))/blade.tmax);
    if scale>1
        v(2,:) = v(2,:).*scale;
    end
end

%prepare the rotation matrix
R = [cosd(theta) -sind(theta); sind(theta) cosd(theta)];
%get rotated coordinates
vo = R*(v - center) + center;
%get new coordinates
x_rotated = vo(1,:);
y_rotated = vo(2,:);

%size coordinates based on geometry of blade
X3d(i,:) = blade.LE(blade.r(i))+x_rotated;%chord direction
Y3d(i,:) = blade.x(i).*ones(size(airfoil.x(airfoil.x<=1))');%span
Z3d(i,:) = y_rotated;%thickness

B.7.1 Interpolation of \( C_P \)

As with the other coefficients discussed in Section B.6.2, if XFOIL did not converge, there will be no \( c_p \) values assigned to that blade element. Another issue at hand is that the total number of pressure points data points may not be equal to the number of surface geometric nodes. To overcome these possible sources of computational error, the \( c_p \) gets inter/extrapolated from the known data-points to the number of physical nodes available. The scheme is similar to the one discussed in Section B.6.2. However, the separation point now must be for the upper and lower surfaces. Listing B.15 shows the section of the script which performs this calculation. This process is nested within a for loop, being repeated for each blade element until the \( c_p \) distribution on the surface is assembled.

Listing B.15.Blade Plotting function detail: \( c_p \) interpolation

%find the separation point between upper and lower surface of CP
upxcpI=find(airfoil.xcp{1}==min(airfoil.xcp{1}),1);
%get the x and z coordinates of the CP so it corresponds to the plotted airfoil coordinates. Must separate into upper and lower
%because interp can't have identical x's
if length(airfoil.xcp{1}(1:upxcpI))==length(airfoil.x(1:upxI))
    v=[airfoil.x(airfoil.x<=1)';...
After the calculations have been preformed, the data is graphically plotted and presented to the user. The 3D surface node coordinates, the $c_p$ distribution, and the $\Delta P$ on the surface is then exported into the blade result structure. Now there is sufficient data to to perform the aeroacoustic estimation.

### B.8 WOPWOP Function

PSU-WOPWOPv3 (Hennes et al., 2017) is a powerful and useful tool. Part of the challenge of this research was the ability to use the code within the MATLAB environment. Since it takes input commands from ASCII case files, a similar approach used in Section B.6 was used. The inputs to the function are the same result structures, along with some optional switches to output plots or to customize the observer position and time for the analysis. The function also looks for a separate input structure, simply called `wop`, where the customized observer positions will be defined. This structure is not a mandatory input however, and if not present the code will default to the standard observer distance and time frame. More detail and a thorough description is available in the function preamble, seen in Appendix C.7, lines 2-35.

A worthy note is that the units must be consistent when feeding it to WOPWOP. Since the desired pressure has to be gauge pressure, and the samples provided were all in Pascals, it was decided to convert all the coordinates and pressures to SI units. This is done internally within the function however, so the user is not required to do this conversion beforehand. As with the XFOIL interface, the code has distinct sections, each performing a specific task. The first section sets-up the instruction command
file. The next ones assemble the geometry and loading binary input files. After these inputs files are generated, the executable is called using the windows command line function. Once the execution has been completed, the output files are read and decoded to the MATLAB work-space. These sections are discussed individually next.

B.8.1 Case Files

WOPWOP takes inputs from a case text file. The file is assembled based on the rotor parameters, blade position, flight characteristics, and observer position and time window. There are two files created. The first is saved to the same directory where WOPWOP resides. This file simply tells the program where the data and instruction files are located. The second file is located within a separate directory, where all the input files are saved and where the results files will be written. This directory can be renamed as per the user’s selection, but it will default to \scripts\tools\wopwop\caseEFRC\ and \scripts\tools\wopwop\caseEFRC\results\.

Lines 107-251 in Appendix C.7 is where these files are being assembled. For more information on the commands and the structure use, refer the PSU-WOPWOPv3 User Manual (Hennes et al., 2017)

B.8.2 Geometry Input File

The geometry input file is a structured little-endian binary file, where the node surfaces are assembled and fed to the software. The encoding and structure are based on the sample files provided by Dr. Brentner and the information found in the User Manual. There are two key components to be discussed about this section of the tool. The first involves the grid assembly, while the second involves the calculation of the area vectors which the code uses to calculate the pressure forces.

The node coordinates have be assembled in a specific structure: XYZ coordinates for the node location, ordered by chord-wise to span-wise distribution. A sample of the assembling structure is shown in Listing B.16. It should be noticeable that there
is an overlap of the first and last points between the upper and lower surfaces. This
has be coded this way because the surface nodes must match at their junction point
in order for the program to understand there is a continuous surface. Since the
coordinates do not repeat at these locations, this fix must be applied.

Listing B.16.PSU-WOPWOP Geometry File assembling MATLAB function
detail

```
280 iMax = [1+size(X3d,2)/2;1+size(X3d,2)/2;size(X3d,2)/2;size(X3d,2)/2];
281 jMax = [size(X3d,1);size(X3d,1);2;2];
282 %predefine a temporary cell structure with the coordinates to use a
283 %loop later
284 tempCoord={ ...
285 % Zone 1 – Upper
286 X3d(:,1:iMax(1)) ,...
287 Y3d(:,1:iMax(1)) ,...
288 Z3d(:,1:iMax(1))...
289 },{...
290 % Zone 2 – Lower (overlap first and last point)
291 X3d(:,1,1:end:-1:iMax(1)) ,...
292 Y3d(:,1,1,end:-iMax(1)) ,...
293 Z3d(:,1,1,end:-1:iMax(1))...
294 },{...
295 % Zone 3 – Root
296 [X3d(1,1,iMax(1)-1);X3d(1,end:-1:iMax(1))] ,...
297 [Y3d(1,1,iMax(1)-1);Y3d(1,iMax(1):end)] ,...
298 [Z3d(1,1,iMax(1)-1);Z3d(1,end:-1:iMax(1))]
299 };
```

The next issue arises with the use of node-centered area-vectors. If we calculate
the area of each cell, we end up with less vectors than the total number of nodes.
This crashes the execution. The tool only takes node-centered area vectors. To
overcome this complication, this concept had to be thought of as it was not inherently
understood by the author. The calculation scheme can be visualized in Figure B.10,
where a sample 3D surface is shown with different vector area calculations shown.

Listing B.17.Node-centered are vector calculation scheme

```
300 %perform calculations for vertex points to perform Node Centerd Vector
301 %Area
```
for zdx=1:2
    % get normal unit vectors for each surface
    [U{zdx}, V{zdx}, W{zdx}]=surfNorm(tempCoord{zdx}{1}, tempCoord{zdx}{2}, tempCoord{zdx}{3});
    % for each set of coordinates
    for cdx=1:size(tempCoord{zdx}, 2)
        % setup area vertex points empty array
        mid{cdx}=zeros(size(tempCoord{zdx}{cdx})+1);
        % set mid points (node + half distance to next node)
        mid{cdx}(2:end-1,end-1)=tempCoord{zdx}{cdx}(1:end-1,1:end-1)+
        ((tempCoord{zdx}{cdx}{2:end,2:end})-tempCoord{zdx}{cdx}{1:end-1,1:end-1})/2;
        % set corner points
        mid{cdx}(1,1)=tempCoord{zdx}{cdx}(1,1);
        mid{cdx}(1,end)=tempCoord{zdx}{cdx}(1,end);
        mid{cdx}(end,1)=tempCoord{zdx}{cdx}(end,1);
        mid{cdx}(end,end)=tempCoord{zdx}{cdx}(end,end);
        % set boundary points
        mid{cdx}(1,2:end-1)=tempCoord{zdx}{cdx}(1,1:end-1)+((tempCoord{zdx}{cdx}(1,2:end)-tempCoord{zdx}{cdx}(1,1:end-1)))/2;
        mid{cdx}(end,2:end-1)=tempCoord{zdx}{cdx}(end,1:end-1)+((tempCoord{zdx}{cdx}(end,2:end)-tempCoord{zdx}{cdx}(end,1:end-1)))/2;
        mid{cdx}(2:end-1,1)=tempCoord{zdx}{cdx}(1:end-1,1)+((tempCoord{zdx}{cdx}(2:end,1)-tempCoord{zdx}{cdx}(1:end-1,1))/2;
        mid{cdx}(2:end-1,end)=tempCoord{zdx}{cdx}(1:end-1,end)+((tempCoord{zdx}{cdx}(2:end,end)-tempCoord{zdx}{cdx}(1:end-1,end))/2;
    end
end

for i=1:size(Xmid{zdx}, 1)-1
    for j=1:size(Xmid{zdx}, 2)-1
        Amid{zdx}(i, j, :)=cross(...
        [Xmid{zdx}(i+1,j)-Xmid{zdx}(i, j),
        Ymid{zdx}(i+1,j)-
        Ymid{zdx}(i, j),
        Zmid{zdx}(i+1,j)-Zmid{zdx}(i, j)],
        ...
        [Xmid{zdx}(i, j+1)-Xmid{zdx}(i, j),
        Ymid{zdx}(i, j+1)-
        Ymid{zdx}(i, j),
        Zmid{zdx}(i, j+1)-Zmid{zdx}(i, j)])
        Nmid{zdx}(i, j, :)=abs(norm([Amid{zdx}(i, j, 1), Amid{zdx}(i, j, 2), Amid{zdx}(i, j, 3)]));
    end
end

for i=1:size(Xmid{zdx}, 1)-1
for j=1:size(Xmid{zdx}, 2)-1
    % get normal unit vectors for each surface
    [U{zdx}, V{zdx}, W{zdx}]=surfNorm(tempCoord{zdx}{1}, tempCoord{zdx}{2}, tempCoord{zdx}{3});
    % for each set of coordinates
    for cdx=1:size(tempCoord{zdx}, 2)
        % setup area vertex points empty array
        mid{cdx}=zeros(size(tempCoord{zdx}{cdx})+1);
        % set mid points (node + half distance to next node)
        mid{cdx}(2:end-1,end-1)=tempCoord{zdx}{cdx}(1:end-1,1:end-1)+
        ((tempCoord{zdx}{cdx}{2:end,2:end})-tempCoord{zdx}{cdx}{1:end-1,1:end-1})/2;
        % set corner points
        mid{cdx}(1,1)=tempCoord{zdx}{cdx}(1,1);
        mid{cdx}(1,end)=tempCoord{zdx}{cdx}(1,end);
        mid{cdx}(end,1)=tempCoord{zdx}{cdx}(end,1);
        mid{cdx}(end,end)=tempCoord{zdx}{cdx}(end,end);
        % set boundary points
        mid{cdx}(1,2:end-1)=tempCoord{zdx}{cdx}(1,1:end-1)+((tempCoord{zdx}{cdx}(1,2:end)-tempCoord{zdx}{cdx}(1,1:end-1)))/2;
        mid{cdx}(end,2:end-1)=tempCoord{zdx}{cdx}(end,1:end-1)+((tempCoord{zdx}{cdx}(end,2:end)-tempCoord{zdx}{cdx}(end,1:end-1)))/2;
        mid{cdx}(2:end-1,1)=tempCoord{zdx}{cdx}(1:end-1,1)+((tempCoord{zdx}{cdx}(2:end,1)-tempCoord{zdx}{cdx}(1:end-1,1))/2;
        mid{cdx}(2:end-1,end)=tempCoord{zdx}{cdx}(1:end-1,end)+((tempCoord{zdx}{cdx}(2:end,end)-tempCoord{zdx}{cdx}(1:end-1,end))/2;
    end
end

for i=1:size(Xmid{zdx}, 1)-1

The areas are calculated by doing the cross multiplication of the vectors to the midpoints on each cell. Each cell’s midpoint is calculated and placed on the geometry. Then, a vector is created in the i-th direction (\( \mathbf{a}_{i,j}^- \), \( \mathbf{a}_{i+1,j}^- \)) and another one in the j-th
Figure B.10 Node-centered area-vector calculation scheme and demonstration

direction $\vec{a}_{i,j}$, $\vec{a}_{i,j+1}$. This vector originate from the surface midpoints and end at the midpoints, except at the boundaries. In the boundary nodes, the origin / end are the midpoints between the two nodes. This calculation is iterated for each node, and then the resultant magnitude is obtained. This resultant magnitude is assigned to the
unit vector corresponding the surface node location, which is conveniently available through a MATLAB embedded function. This is exemplified in Listing B.17.

To ensure this calculated area vector was correct, these were computed for the sample case geometry provided with the code. The vectors were then compared to the extracted input values from the same geometry and ensured they match. Figure B.11 shows that these match.

![Figure B.11 Upper surface node-centered area-vector comparison between calculated and provided results](image)

Figure B.11: Upper surface node-centered area-vector comparison between calculated and provided results.

After each run, a decoding function has been created to visualize what was the input to WOPWOP. With this visual tool, the user can be assured that the correct input were fed to the code prior to the execution. This function can be found in Appendix C.8. Figure B.12 depicts the sample output from this decoder, showing a correct geometric representation and correct area vector magnitude and direction.
Figure B.12 Decoded output plot shown to the user as means to ensure the correct inputs were fed to WOPWOP for execution. Units are in SI, thus the distance is in meters and gauge pressure in Pascals.

B.8.3 Loading Input File

The loading input file is simpler than the geometric one, where the surface pressure values are assigned to each node. Since the pressure has already been calculated before, these values are readily available. The only task at hand is to assemble the data in the desired structure, and include the correct commands. The full file writing script can be found in Appendix C.7, lines 451-529. As with the above mentioned geometry patch file writing script, the file must be little-endian binary encoded, where each value is assigned four bits.

B.8.4 Running WOPWOP

Running WOPWOP now is similar to the way XFOIL was executed. The main difference is that, since there is no visual output when being executed, a visual aid was coded to show the user the code is still executing and preventing any interaction that could jeopardize the results. Whether the execution is successful or not, the user
will be notified. If successful, plots are generated with the sound pressure variation and sound pressure frequency distribution. If unsuccessful, the user can access a debugging file in `\scripts\tools\wopwop\resultWOPWOP.out` where the reason for the crash will be reported. This usually is related to erroneous time step or time-frame selection by the user.

B.8.5 Reading WOPWOP results

The results are extracted from the files and saved into the work-space in a similar fashion as discussed in Section B.6. These extracted values are saved into a structure names `noise`, and into a separate output file. Sample plots are depicted in Figure B.13. The plots are not saved. However, these can be generated easily from the saved data.

![Sample result extraction from WOPWOP’s execution](image)

*Figure B.13* Sample result extraction from WOPWOP’s execution
C. Appendix - Source Code Listings

C.1 runProp.m script

Full \runProp.m script

```matlab
% runPROP Function caller to perform blade acoustic analysis using a
% modified Blade Element Momentum Theory based on Leishman (2006) and
% McCormick (1995) books. Implemented use of XFOIL (Drela, 2001) to
% get
% aero properties of blade, and PSU-WOPWOPv3 (Brentner, 2017) to
% perform
% aeroacoustic approximation of the blade. Current release only
% performs
% static/hover case, with no unsteady loads, no chordwise distribution
% , and
% only considers thickness and loading noise.
%
%% Author: Xavier Santacruz (santacrx@gmail.com)
%% Release: April/01/2019
%% Purpose: MSc. Aerospace Engineering Thesis − ERAU EFRC

%%% Initialization [DO NOT EDIT]
close all
clear
clc
load_paths; %load the necessary paths for subscripts

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%BEGIN USER INPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Setup Switches [0=Off, 1=On; unless specified]
sw.geom = 0; % Use m-file inputs (0) or open geometry editor (1)
sw.aero = 1; % Use Xfoil in all elements (1) or just for general Cla
% (0)
sw.perf = 0; % Calculate hover performance curves (1), advancing
% flight
sw.noise = 1; % Run WOPWOP after all blade calculations
sw.bemt = 4; % BEMT Version. 0=Leishman (element by element),
% 1=Leishman Mod (array calcs),
% 2=McCormick BEVT, 3=McCormick BEVT P–C
% 4=Thesis BEMT P–C method
sw.bias = 1; % Tip element bias. 1=sine weighted. 0=equally spaced
% elements

%%%% Blade Input − Override with preloaded geometry (uncomment your
% choice)
```
C.2 geometry.m script

Full \scripts\tools\xfoil\geometry.m script

1 % geometry.m: User editable input file for custom blades. The user can also
2 % copy this file, rename it, and call as an input on "run_Prop". This file
3 % open in edit mode if sw.geom is set to 1.
4 % STANDALONE INPUT FILE. NO COMPUTATIONS PERFORMED
5 % Xavier Santacruz (santacrx@gmail.com)
6
7 % flight data
8 flight.Vclimb = 2; [%ft/s] - Climb/Axial Velocity
9 flight.Vfwd = 0; [%ft/s] - Forward flight speed
10 flight.alt = 0;
11
12 % rotor data
13 rotor.Nr = 2; %Number of rotors
14 rotorNb = 3; %Number of blades
15 rotor.macht = 0; %Tip Mach, overrides rpm. 0 uses rpm
16 rotor.rpm = 1400;
17
18 % blade data
19 blade.el = 100; %number of blade elements
20 blade.pn = 50; %number of chordwise surface panels (TOTAL i.e.: Upper + Lower)
21 % blade geometry
22 blade.R = 4; [%ft] - Blade length / rotor radius
23 blade.root = 0.5; [%ft] - Root cut-out / hub radius
24 blade.LE = @(r) 0.5 + 0.375*r; [%ft] - Leading Edge function wrt non-dim r
25 blade.TE = @(r) -0.5 + 0.375*r; [%ft] - Trailing Edge function wrt non-dim r
26 % blade angles
27 blade.twist = @(r) -25*r + 25; [%deg] - Linear twist slope. 0 at tip.
28 blade.tw_center = 0.25; [%c] Center of Twist Rotation
29 blade.c_e = 8; [%deg]
30 % blade airfoil
31 blade.airfoil = 'e339'; % airfoil name for Xfoil
blade.Cla = @(alpha) 3.33;
blade.Cd0 = @(alpha) 0.011;

%% WOP custom commands
%Microphone position
wop.obs.x = 3*blade.R;%[ft]
wop.obs.y = 0; %[ft]
wop.obs.z = 0; %[ft]
%time to read and points to calculate within the timeframe
wop.t.n = 4192;
wop.t.Min = 0; %[sec]
wop.t.Max = 1/(rotor.rpm*60); %[sec]
%file directory name to save data
wop.fileDir = 'sampleName';

C.3 blade_calculations.m script

Full \scripts\tools\xfoil\blade_calculations.m script

% blade_calculations: perform blade calcs to load to BEMT
% not a standalone script, must be run by run_Prop.m
% XSantacruz

%start timer
propTime=tic;
% Check switches
if sw.geom ==1 %if edit geometry is selected, open geometry file and
    wait
    clear flight rotor blade wop
    edit geometry.m
    waitfor(msgbox('Save file and click "OK" when done to proceed.'));
    geometry;
end

%change precision to more than 32 to avoid rounding off error
digitsOld=digits;
digits(64);

%% Pre-BET Calculations
%atmospheric properties
[flight.T, flight.a, flight.P, flight.rho] = atmosisa(flight.alt*0.3048);
flight.a = flight.a.*3.28084;%[ft/s]
flight.rho = flight.rho.*0.06242796057614516;%[lb/ft^3]
flight.T = ((flight.T - 273.15).*9/5) + 32; %[F]
flight.nu = 1.602e-4; %[ft^2/s] Kinematic Viscosity of Air
%rotor calculations
rotor.D=blade.R+2; %[ft]
rotor.A=(blade.R^2-blade.root^2)*pi;%Disk Area [ft^2]
%element length (tip weighted distribution)
if sw.bias==1
    rWeight = sin(0:pi/(2*blade.el):pi/2);
else
rWeight = linspace(0,1,blade.el+1);
end
rWalls = (rWeight.*(blade.R-blade.root))+blade.root;
blade.dr = rWalls(2:end)-rWalls(1:end-1);
%element coordinates
blade.x = rWalls(1:end-1)+(blade.dr./2); %center x-coord of element
blade.xi = rWalls(1:end-1); %inner x-coord of element
blade.xo = rWalls(2:end); %outer x-coord of element
clear rWeight rWalls %remove temporary vars
%element characteristics
blade.r = blade.x/blade.R; %non dimensionalized radius
blade.theta = blade.twist(blade.r); %[deg] element twist (zero collective)
blade.beta = blade.theta+blade.collective; %[deg] element angle with TPP
blade.c = blade.LE(blade.r)-blade.TE(blade.r); %[ft]
if sum(blade.c<0)>1; error('Negative chord length detected. Stopping. '); end %check that all chord lengths are positive
blade.dS = blade.c.*blade.dr;%element area [ft^2]
blade.Ssingle = sum(blade.dS);%single blade area [ft^2]
rotor.sigma = blade.Ssingle .* (rotor.Nb / rotor.A); %rotor solidity
blade.dSigma = rotor.Nb .* blade.dS ./((pi*(blade.xo.^2 - blade.xi.^2)) ; %element solidity
%element dynamics
if rotor.macht ~= 0 %if mach tip is not zero, calculate the rpm
rotor.rpm = 60*(rotor.macht*flight.a)/(blade.R*2*pi);
end
if isfield(rotor, 'J') %if J is defined, us this to get axial speed
flight.Vclimb=(rotor.rpm/60)*rotor.D*rotor.J;
end
blade.v_plane = blade.x.*(rotor.rpm/60);
blade.Mach = blade.v_plane./flight.a; %Local blade section Mach
blade.Re = blade.c.*(blade.x.*(rotor.rpm/60))./flight.nu; %Local blade section Reynolds
% BEMT
%perform BEMT based on McCormick's method
%use Xfoil for aero parameters
%return blade CP distribution on surface and dT dQ for rotor
switch sw.bemt
  case 0
  %Leishman's Original element per element
  [coeffs,airfoil,blade,rotor] = BEMTv0(flight,rotor,blade,sw);
  case 1
  %Leishman's, vector calculations
  [coeffs,airfoil,blade,rotor] = BEMTv1(flight,rotor,blade,sw);
  case 2
  %McCormick's BEMT
  [coeffs,airfoil,blade,rotor] = BEMTv3b(flight,rotor,blade,sw);
  case 3
tempAero = sw.aero;
sw.aero = 0;
bemtCount = 0;
while bemtCount < 5
    [~,~,blade,rotor] = BEMTv3b1(flight,rotor,blade,sw);
    bemtCount = bemtCount+1;
end
sw.aero = tempAero;
clear bemtCount tempAero
[coeffs,airfoil,blade,rotor] = BEMTv3b(flight,rotor,blade,sw);

% Thesis way
[coeffs,airfoil,blade,rotor] = BEMTv0b(flight,rotor,blade,sw);

end
%return precision to what it was before
digits(digitsOld);
clear digitsOld

% BEMT Post-processing
% plot and get 3d coordinates and pressure distribution
if sw.perf==0
    blade = blade_plotting(airfoil,blade,rotor,flight,coeffs,sw,0);
    save([pwd '/output/results_bemt.mat'],'airfoil','blade','rotor','flight','coeffs','sw');
end
% if noise switch is activated, perform noise analysis using WopWop
if sw.noise==1 && sw.perf==0
    if exist('mic','var')
        wop.obs = thomasMics(mic);
    end
    if exist('wop','var')
        if isfield(wop,'fileDir')
            noise = wopwop(blade,rotor,flight,true,wop,wop.fileDir);
        else
            noise = wopwop(blade,rotor,flight,true,wop);
        end
    else
        noise = wopwop(blade,rotor,flight,true);
    end
    readPatchTest;
    save([pwd '/output/results_noise.mat'], 'noise');
end
tEnd=toc(propTime);
fprintf('\nTotal runtime: %d min %d sec\n', floor(tEnd/60), ceil(rem(tEnd,60)));
clear propTime tEnd
function [coeffs, airfoil, blade, rotor] = BEMT_v0(flight, rotor, blade, sw)
% BEMT V3
% Based on Leishman’s Book, Modified in Thesis Formulation
% Xavier Santacruz (santacrx@gmail.com)
% Eagle Flight Research Center
% ERAU
%
% 2019/03/11 − Rev 0
a = flight.a; [% ft/s]
rho = flight.rho; [% lb/ft^3]
nu = flight.nu; [% ft^2/s]
rotor.Vaxial = flight.Vclimb; [% ft/s]

% number of airfoil panels
if isfield(blade, 'pn')
    nPane = blade.pn;
else
    nPane = 50;
end

% Initial Calculations
omega = rotor.rpm *(2*pi/60); [% rad/s]
omega_r = omega.*blade.x; [% ft/s]
V = rotor.Vaxial.*ones(size(blade.x)); [% ft/s]
VT = blade.R.*omega; [% ft/s] Tip Speed (SCALAR)
x = blade.x; [% ft]
r = x/blade.R; [% nd] Non-dimensionalized element location
dr = blade.dr/blade.R;
lambda_c = rotor.Vaxial./omega_r;
lambda = V./VT; [% nd] See bottom of pg302 (V + VT)/omegar
beta = (blade.collective + blade.twist(blade.xi./blade.R)).*(pi/180); [% rad]
c = blade.c; [% ft]
Nb = rotor.Nb; [% nd]
D = rotor.D; [% ft]
A = rotor.A; [% ft^2]
n = rotor.rpm/60; [% rev/s]
J = rotor.Vaxial/(n*D);

% run Xfoil once from 0 to 10 deg at tip mach and re to get average Cla,
% Cd0, and t_max/c of airfoil to use in equations
temp.coeff.aoa = [0:2:10];
[temp.coeff, temp.foil] = aerocalcs(blade.airfoil, temp.coeff.aoa, ...
    blade.Re(fix(blade.el.*.75)).*ones(size(temp.coeff.aoa)), ...
    blade.Mach(fix(blade.el.*.75)).*ones(size(temp.coeff.aoa)), false);
% convert aoa to rads for calculations
temp.coeff.aoa = temp.coeff.aoa.*(pi/180);
linear fit the Cl curve to get Cla and Cl0

\[ f = \text{fit} \left( \text{temp.coef.aoa}', \text{temp.coef.Cl}', '\text{poly1}' \right); \]

fit in the form \( f(x) = p1*x + p2 \)

\[ \text{Cla} = f.p1; \] \[ \text{Cl0} = f.p2; \]

% quadratic fit the Cd curve to get Cd0, Cd1 and Cd2

\[ f = \text{fit} \left( \text{temp.coef.aoa}', \text{temp.coef.Cd}', '\text{poly2}' \right); \]

fit in the form \( f(x) = p1*x^2 + p2*x + p3 \)

\[ \text{Cd0} = f.p3; \] \[ \text{Cd1} = f.p2; \] \[ \text{Cd2} = f.p1; \]

get the max thickness over c for the airfoil section

\[ [\text{blade.tmax,}^*] = \text{findtmax} \left( \text{temp.foil} \right); \]

if isfield(sw, 'tc')

\[ \text{tmax} = \text{abs} \left( \text{blade.tc}(x)./\text{blade.tmax} \right); \]

else

\[ \text{tmax} = \text{blade.tmax}.*\text{ones}(\text{size}(x)); \]

end

Calculate inflow for each blade element

\[ \text{Nr} = \text{blade.el}; \]

for \( i = 1: \text{Nr} \)

% define placeholders and counters

\[ m = 0; \]

\[ \text{sigma_r} = \text{blade.dSigma}(i); \]

% BLADE TWIST CALCULATIONS

\[ \text{dtheta_r}(i) = \text{beta}(i); \]

% PRANDTL TIP LOSS CALCULATIONS

% Predefine 1st F as 1

\[ F = 1; \]

% Iterate for 5 times to get modified inflow

\[ \text{while } m < 10 \]

\[ \begin{align*}
\text{lambda}(i) &= \sqrt{((\text{sigma_r} * \text{Cla} / (16*F)) - (\text{lambda_c}(i)/2))^2 + ...} \\
& \quad ((\text{sigma_r} * \text{Cla} * \text{dtheta_r}(i) * r(i)/(8*F))^2) - ...} \\
& \quad ((\text{sigma_r} * \text{Cla} / (16*F)) - (\text{lambda_c}(i)/2)); \quad \text{Eq.3.126 modified with 3.58} \\
\text{f} &= (\text{Nb}/2)*((1 - r(i))/(\text{lambda}(i) - (\text{lambda_c}(i)/\text{Nr}))); \quad \text{Eq.3.121 [NOTE: MAY NEED TO MODIFY]} \\
F &= (2/\pi())*\text{acos} (\exp(-f)); \quad \text{Eq.3.120} \\
\text{m} &= m+1;
\end{align*} \]

end

\[ \begin{align*}
\text{lambda}(i) &= \sqrt{((\text{sigma_r} * \text{Cla} / (16*F)) - (\text{lambda_c}(i)/2))^2 + ...} \\
& \quad ((\text{sigma_r} * \text{Cla} * \text{dtheta_r}(i) * r(i)/(8*F))^2) - ...} \\
& \quad ((\text{sigma_r} * \text{Cla} / (16*F)) - (\text{lambda_c}(i)/2)); \quad \text{Eq.3.126 modified with 3.58} \\
\end{align*} \]

end

calculate element velocity vectors, angles, and params

\[ \text{UT} = \text{omega_r}; \]

\[ \text{UP} = \text{lambda}.*\text{UT}. / r; \]

\[ \text{U} = \text{real} (\text{sqrt} ( \left( \text{UP} .^2 \right) + (\text{UT} .^2) )); \]

\[ \text{phi} = \text{atan}(\text{UP}/\text{UT}); \]
Mach = real(U./a);
Re = real(c.*U./nu);  \%Local blade section Reynolds
AoA = real((dtheta_r - atan(lambda_r./r)).*(180/pi)); \%[deg]
disp('Thesis BEMT');

\%perform P-C iteration
if sw.aero==1
    coeffs.err = zeros(size(UT));
    \%call xfoil using solved alpha to get Cd and Cl
    [res, airfoil]=aerocalcs(blade.airfoil,AoA,Re,Mach,nPane);
dCl = res.Cl;
dCd = res.Cd;
\%new equation
for i=1:Nr-1
    fprintf('Element %d
',i);
    \%define placeholders and counters
    m = 0;
    sigma_r = blade.dSigma(i);
    origL = lambda(i);
    tempA = AoA(i).*pi()/180;
    err = 1;
    \% Iterate for 20 times to get modified inflow if twist angle is
    \% higher than 5deg
    while m <= 20 && err>0.001 && err<2
        UP(i) = lambda(i).*UT(i)./r(i); \%[ft/s]
        U(i) = real(sqrt((UP(i).^2)+(UT(i).^2))); \%[ft/s]
        Mach(i) = real(U(i)./a);
        Re(i) = real(c(i).*U(i)./nu); \%Local blade section Reynolds
        phi(i) = atan(UP(i)/UT(i)); \%[rad]
        AoA(i) = real((dtheta_r(i)-phi(i)).*(180/pi)); \%[deg]
        fprintf('M=%.4f\tAoA=%.3f\tRe=%.3e\n',Mach(i),AoA(i),Re(i));
        f = (Nb/2)*((1-r(i))/(r(i)).*(phi(i))); \%Eq.3.23 detail
        F = (2/pi())*acos(exp(-f)); \%Eq.3.120
        [pol,˜] = xfoil(blade.airfoil,AoA(i),Re(i),Mach(i),nPane,\'
                        oper\//iter+500 '\');
        if pol.err==0
            dCl(i)=pol.CL;
            dCd(i)=pol.CD;
        end
        Caero = dCl(i).*cos(phi(i)) - dCd(i).*sin(phi(i));
        aidC = sigma_r.*Caero;
        aidA = (8.*F.*r(i))-(aidC);
        aidB = 4.*F.*lambda_c(i); 
        lambda(i) = (r(i)/aidA).*...
                        ((aidB)+sqrt((aidB.^2)+(aidC.*aidA)))); \%Eq.3.23 MY
THESIS
        m = m+1;
        err = abs(AoA(i) - tempA).*pi()/180;
        tempA = AoA(i);
end
if err > 0.001
    lambda(i) = origL;
    UP(i) = lambda(i) .* UT(i) ./ r(i);
    U(i) = real(sqrt((UP(i).^2 + (UT(i).^2))) ;% [ft/s]
    Mach(i) = real(U(i) ./ a);
    Re(i) = real(c(i) .* U(i) ./ nu); % Local blade section Reynolds
    phi(i) = atan(UP(i) / UT(i));
    AoA(i) = real((dtheta_r(i) − phi(i)) .* (180 / pi)) ;% [deg]
    coeffs.err(i) = 1;
end
end
% the last element has NaN, so use half of the previous one
U(Nr) = U(Nr−1) ./ 2;
UT(Nr) = UT(Nr−1) ./ 2;
UP(Nr) = UP(Nr−1) ./ 2;
dCl(Nr) = dCl(Nr−1) ./ 2;
dCd(Nr) = dCd(Nr−1) ./ 2;
phi(Nr) = phi(Nr−1) ./ 2;
Mach(Nr) = Mach(Nr−1) ./ 2;
Re(Nr) = Re(Nr−1) ./ 2;
AoA(Nr) = AoA(Nr−1) ./ 2;
else
    airfoil = temp.foil;
    dCl = Cla.* AoA.* (pi / 180) + Cl0;
    dCd = Cd0 + Cd1.* (AoA.* (pi / 180)) + Cd1.* ((AoA.* (pi / 180)) .^ 2);
end
% calculate element lift and drag forces
dL = 0.5 .* rho .* (U .* ^ 2) .* c .* dCl .* blade.dr;
dD = 0.5 .* rho .* (U .* ^ 2) .* c .* dCd .* blade.dr;
% calculate forces parallel and perpendicular to rotor disk
dFx = dL .* sin(phi) + dD .* cos(phi);
dFz = dL .* cos(phi) − dD .* sin(phi);
% calculate thrust, torque and power element contributions
dT = dFz;
dQ = dFx .* x;
dP = dFx .* omega_r;
% calculate element coefficients
dCT = dT ./ (rho .* A .* (VT^2));
dCQ = dQ ./ (rho .* A .* (VT^2) .* blade.R);
dCP = dP ./ (rho .* A .* (VT^3));
% return numbers of interest
blade. Mach = Mach;
blade. Re = Re;
blade. AoA = AoA; % [deg]
blade. inflow = UP; % [ft/s]
blade. lambda = lambda;
blade. Vt = UT; % [ft/s]
blade. Ve = U; % [ft/s]
%return coefficients of interest
coeffs.dCl = dCl;
coeffs.dCd = dCd;
coeffs.dCT = real(dCT);
coeffs.dCP = real(dCP);
coeffs.dCQ = real(dCQ);

%get total rotor coefficients
CT = sum(real(dCT));
CP = sum(real(dCP));
CQ = sum(real(dCQ));

%return rotor parameters
rotor.J = J;
rotor.CT = CT;
rotor.CP = CP;
rotor.CQ = CQ;
rotor.eta = CT.*J/CP;
rotor.Thrust = sum(dT); [%lbf]
rotor.Torque = sum(dQ); [%f t . l b s ]
rotor.Power = sum(dP); [%f t ° 2 . l b s / s ]

end

C.5 xfoil.m script
Full \scripts\tools\xfoil\xfoil.m script

function [pol,foil] = xfoil(coord, alpha, Re, Mach, Np, varargin)
% Run Xfoil and return the results.
% [polar,foil] = xfoil(coord, alpha, Re, Mach, Np, {extra commands})
% Xfoil.exe needs to be in the same directory as this m function.
% For more information on Xfoil visit these websites;
% http://web.mit.edu/drela/Public/web/xfoil
%
% Inputs:
% coord: Normalised foil co-ordinates (n by 2 array, of x & y
% from the TE-top passed the LE to the TE bottom)
% or a filename of the Xfoil co-ordinate file
% or a NACA 4 or 5 digit descriptor (e.g. ‘NACA0012’)
% alpha: Angle-of-attack, can be a vector for an alpha polar
% Re: Reynolds number (use Re=0 for inviscid mode)
% Mach: Mach number
% Np: Number of airfoil panels (defaults to 50)
% extra commands: Extra Xfoil commands
% The extra Xfoil commands need to be proper xfoil commands
% in a character array. e.g. ‘oper iter+150 ’ leave a space
% for every new line command to return to base menu after
% command
% The transition criterion Ncrit can be specified using the
% ‘extra commands’ option as follows,
% foil = xfoil(‘NACA0012’,10,1e6,0.2,’oper vpar+n+12 ’)


% Situation   Ncrit
% sailplane     12–14
% motorglider  11–13
% clean wind tunnel  10–12
% average wind tunnel  9 <= standard "e`9 method"
% dirty wind tunnel  4–8

% A flap deflection can be added using the following command,
% 'gdes flap {xhinge} {yhinge} {flap_deflection} exec'

% Outputs:
% polar: structure with the polar coefficients (alpha,CL,CD,CDp,CM,
  Top_Xtr,Bot_Xtr)
% foil: structure with the specific aoa values (s,x,y,UEVinf,
  Dstar,Theta,Cf,H,cp) each column corresponds to a
different
% angle-of-attack.
% If only one left hand operator is specified, only the polar
  will be parsed and output
% If there are different sized output arrays for the different
  incidence
% angles then they will be stored in a structured array, foil(1),foil
  (2)...
% If the output array does not have all alphas in it, that indicates a
  convergence failure in Xfoil.
% In that event, increase the iteration count with 'oper iter ##;

% Examples:
% % Single AoA with a different number of panels
% [pol foil] = xfoil('NACA0012',10,1e6,0.0,'panels n 330')
% % Change the maximum number of iterations
% [pol foil] = xfoil('NACA0012',5,1e6,0.2,'oper/iter 50')
% % Deflect the trailing edge by 20deg at 60% chord and run
% multiple incidence angles
% [pol foil] = xfoil('NACA0012',[-5:15],1e6,0.2,'oper iter 150','
gdes flap 0.6 0 5 exec ')
% % Deflect the trailing edge by 20deg at 60% chord and run
% multiple incidence angles and only
% parse or output a polar.
% pol = xfoil('NACA0012',[-5:15],1e6,0.2,'oper iter 150','gdes flap
  0.6 0 5 exec ')
% % Plot the results
% figure;
% plot(pol.alpha,pol.CL); xlabel('alpha [circ]'); ylabel('C_L'); title(pol.name);

figure; subplot(3,1,[1 2]);
plot(foil(1).xcp(:,end).foil(1).cp(:,end)); xlabel('x');
ylabel('C_p'); title(sprintf('%s@%g\circ',pol.name,foil(1).alpha(end)));
set(gca,'ydir','reverse');
subplot(3,1,3);
I = (foil(1).x(:,end)<=1);
plot(foil(1).x(I,end),foil(1).y(I,end)); xlabel('x');
ylabel('y'); axis('equal');

% Some default values
if ~exist('coord','var'); coord = 'NACA0012'; end
if ~exist('alpha','var'); alpha = 0; end
if ~exist('Re','var'); Re = 1e6; end
if ~exist('Mach','var'); Mach = 0.2; end
if ~exist('Np','var'); Np = 50; end
Nalpha = length(alpha); % Number of alphas swept
% default foil name
foil_name = mfilename;
% default filenames
wd = fileparts(which(mfilename)); % working directory, where xfoil.exe needs to be
fname = mfilename;
file_coord= [foil_name '.dat'];

%delete all files in polar folder to avoid overwriting error
warning off
delete([pwd '\scripts\tools\Xfoil\polar\*.dat']); %path relative to XS BEMT
warning on

% Save coordinates
if ischar(coord) % Either a NACA string or a filename
    if isempty(regexp(coord,'^[0-9]{4,5}$')) % Check if a NACA string
        % foil_name = coord; % some redundant code removed to go green (~ isempty if uncommented)
    else % Filename supplied
        % set coord file
        file_coord = strcat('airfoils\',coord,'.dat');
    end
else % Write foil ordinate file
    if exist(file_coord,'file'); delete(file_coord); end
    fid = fopen(file_coord,'w');
    if (fid<=0)
        error([mfilename ':io'],'Unable to create file %s',file_coord);
    else
118 fprintf(fid,'%s\n',foil_name);
119 fprintf(fid,'%9.5f %9.5f\n',coord);
120 fclose(fid);
121 end
122 end
123 %pause(1);
124 % Write xfoil command file
125 fid = fopen([wd filesep fname '.inp'], 'w');
126 if (fid <= 0)
127 error([mfilename ':io'], 'Unable to create xfoil.inp file');
128 else
129 if ischar(coord)
130 if ~isempty(regexp(coord, '^[0-9]\{4,5\}$')) % NACA string supplied
131 fprintf(fid,'naca %s\n', coord(5:end));
132 else % filename supplied
133 fprintf(fid,'load %s\n', file_coord);
134 end
135 else % Coordinates supplied, use the default filename
136 fprintf(fid,'load %s\n', file_coord);
137 end
138 fprintf(fid,'ppar\n'); %load airfoil data
139 fprintf(fid,'n\n'); %change panel number
140 fprintf(fid,'%d\n\n',floor(Np)); %set to 240 panels and get back to main menu
141 fprintf(fid,'pane\n'); %make sure no panels have sharp edges
142 % Extra Xfoil commands
143 for ii = 1:length(varargin)
144     txt = varargin{ii};
145     txt = regexprep(txt, '\[/]+', '\\n');
146     txt = regexprep(txt, '[+]+', ' ');
147     fprintf(fid,'%s\n', txt);
148 end
149 fprintf(fid,'oper\n');
150 % change the iteration steps to 100
151 fprintf(fid,'iter 500\n');
152 % set Reynolds and Mach
153 fprintf(fid,'re %g\n', Re);
154 fprintf(fid,'mach %g\n', Mach);
155 % Switch to viscous mode
156 if (Re>0)
157     fprintf(fid,'visc\n');
158 end
159 % Polar accumulation
160 fprintf(fid,'pacc\n');
161 fprintf(fid,'\n');
162 fprintf(fid,'\n');
163 % Xfoil alpha calculations
164 [file_dump, file_cpwr] = deal(cell(1,Nalpha)); % Preallocate cell arrays
\texttt{for ii = 1:Nalpha}
\texttt{\% Individual output filenames}
\texttt{file_dump{ii} = sprintf('\%s_a\%06.3f_dump.dat',fname,\alpha(ii));}
\texttt{file_cpwr{ii} = sprintf('\%s_a\%06.3f_cpwr.dat',fname,\alpha(ii));}
\texttt{\% Commands}
\texttt{fprintf(fid,'alpha %g\n',\alpha(ii));}
\texttt{fprintf(fid,'dump polar/%s\n',file_dump{ii});}
\texttt{fprintf(fid,'cpwr polar/%s\n',file_cpwr{ii});}
\texttt{end}
\texttt{\% Polar output filename}
\texttt{file_pwrt = sprintf('\%s_pwrt.dat',fname);}
\texttt{fprintf(fid,'pwrt\n');}
\texttt{fprintf(fid,'polar/%s\n',file_pwrt);}
\texttt{fprintf(fid,'plis\n');}
\texttt{fprintf(fid,'quit\n');}
\texttt{fclose(fid);}
\texttt{\%pause(1):}
\texttt{\% execute xfoil}
\texttt{cmd = sprintf('cd %s && xfoil.exe < xfoil.inp > xfoil.out',wd);}
\texttt{[status,result] = system(cmd);}
\texttt{if (status\&\&0)}
\texttt{\quad disp(result);}
\texttt{\quad error([mfilename ':system'],'Xfoil execution failed! %s',cmd);}
\texttt{end}
\texttt{\%pause(1);}
\texttt{\% Read dump file}
\texttt{\% # s x y Ue/Vinf Dstar Theta Cf H}
\texttt{jj = 0;}
\texttt{ind = 1;}
\texttt{\% Note that}
\texttt{foil.alpha = zeros(1,Nalpha); \% Preallocate alphas}
\texttt{\% Find the number of panels with an initial run}
\texttt{\% only = nargout; \% Number of outputs checked. If only one left hand}
\texttt{\% operator then only do polar}
\texttt{\% only >1 \% Only do the foil calculations if more than one left hand}
\texttt{\% operator is specified}
\texttt{for ii = 1:Nalpha}
\texttt{\quad jj = jj + 1;}
\texttt{\quad fid = fopen(file_dump{ii},'r');}
\texttt{\quad if (fid\&\&0)}
\texttt{\quad \quad error([mfilename ':io'],'Unable to read xfoil output file %s',}
\texttt{\quad \quad \quad file_dump{ii});}
\texttt{\quad else}
\texttt{\quad \quad fprintf(fid,'\%s\n',\alpha(ii));}
\texttt{\quad \quad fprintf(fid,'dump polar/%s\n',file_dump{ii});}
\texttt{\quad \quad fprintf(fid,'cpwr polar/%s\n',file_cpwr{ii});}
\texttt{\texttt{\quad end\texttt{fi});}
\texttt{end\texttt{fi});}
D = textscan(fid, '%f%f%f%f%f%f%f%f', 'Delimiter', '', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 1);
fclose(fid);
delete([pwd '\scripts\tools\Xfoil\polar\' file_dump{ii}]); % Mod
to work with file setup

if ii == 1 % Use first run to determine number of panels (so
that NACA airfoils work without vector input)
Npanel = length(D{1}); % Number of airfoil panels pulled from
the first angle tested
% Preallocate Outputs
[foil.s, foil.x, foil.y, foil.UeVinf, foil.Dstar, foil.Theta,
foil.Cf, foil.H] = deal(zeros(Npanel, Nalpha));
end

% store data
if ((jj>1) && (size(D{1},1)-length(foil(ind).x)) && sum(abs(
    foil(ind).x(:,1)-size(D{1},1)))>1e-6 ),
    ind = ind + 1;
jj = 1;
end
foil.s(:,jj) = D{1}(:,1);
foil.x(:,jj) = D{1}(:,2);
foil.y(:,jj) = D{1}(:,3);
foil.UeVinf(:,jj) = D{1}(:,4);
foil.Dstar(:,jj) = D{1}(:,5);
foil.Theta(:,jj) = D{1}(:,6);
foil.Cf(:,jj) = D{1}(:,7);
foil.H(:,jj)= D{1}(:,8);
end
foil.alpha(1,jj) = alpha(jj);

% Read cp file
% # x y Cp
fid = fopen(file_cpwr{ii}, 'r');
if (fid<=0)
    error(['fprintf( ":io")', 'Unable to read xfoil output file %s',
          file_cpwr{ii}]);
else
    C = textscan(fid, '%10f%9f%f', 'Delimiter', '', 'WhiteSpace', '',
                 'HeaderLines', 3, 'ReturnOnError', false);
fclose(fid);
delete([pwd '\scripts\tools\Xfoil\polar\' file_cpwr{ii}]); % Mod
to work with file setup
% store data
if ii == 1 % Use first run to determine number of panels (so
that NACA airfoils work without vector input)
NCp = length(C{1}); % Number of points Cp is listed for
pulled from the first angle tested
% Preallocate Outputs
[foil.xcp, foil.cp] = deal(zeros(NCp, Nalpha));
foi1.xcp = C1(:,1);
end
foi1.cp(:,jj) = C3(:,1);
end
end

if only <= 1% clear files for default run
for ii=1:Nalpha % Clear out the xfoil dump files not used
    delete(fullfile('scripts\tools\Xfoil\polar\' file_dump{ii}));
    delete(fullfile('scripts\tools\Xfoil\polar\' file_cpwr{ii}));
end
end

% Read polar file
%     XFOIL     Version 6.96
% Calculated polar for: NACA 0012
% 1 1 Reynolds number fixed     Mach number fixed
% xtrf = 1.000 (top) 1.000 (bottom)
% Mach = 0.000     Re = 1.000 e 6     Ncrit = 12.000
% alpha    CL    CD    CDp    CM    Top_Xtr    Bot_Xtr
fid = fopen(file_pwrt,'r');
if (fid <=0)
    error([mfilename ': io'], 'Unable to read xfoil polar file %s', file_pwrt);
else
    % Header
    % Calculated polar for: NACA 0012
    P = textscan(fid,' Calculated polar for: %[^\n]+\'', 'Delimiter',' ', 'MultipleDelimsAsOne', true, 'HeaderLines',3);
pol.name = strtrim(P{1}{1});
% xtrf = 1.000 (top) 1.000 (bottom)
P = textscan(fid, '%*[\s]*[\s]*%*[\s]*%*[\s]*%*[\s]*%*[\s]*', 1, 'Delimiter',' ','MultipleDelimsAsOne', true, 'HeaderLines',2, 'ReturnOnError', false);
pol.xtrf.top = P{1}{1};
pol.xtrf.bot = P{2}{1};
% Mach = 0.000     Re = 1.000 e 6     Ncrit = 12.000
P = textscan(fid, '%*[\s]*%*[\s]*%*[\s]*%*[\s]*%*[\s]*%*[\s]*%*[\s]*', 1, 'Delimiter',' ','MultipleDelimsAsOne', true, 'HeaderLines',0, 'ReturnOnError', false);
pol.Re = P{2}{1} * 10^P{3}{1};
pol.Ncrit = P{4}{1};
% data
P = textscan(fid, '%f%f%f%f%f%f%f% %s% %s% %s% %s% %s% %s', 'Delimiter', ' ', 'MultipleDelimsAsOne', true, 'HeaderLines', 4, 'ReturnOnError', false);
fclose(fid);
delete(fullfile(pwd '\scripts\tools\Xfoil\tools\polar\', file_pwt));%file path relative to XS BEAM

% store data
pol.alpha = P{1}(:,1);
pol.err = 0; %error flag
pol.CL = P{2}(:,1);
pol.CD = P{3}(:,1);
pol.CDP = P{4}(:,1);
pol.Cm = P{5}(:,1);
pol.Top_xtr = P{6}(:,1);
pol.Bot_xtr = P{7}(:,1);
end
if length(pol.alpha) ~= Nalpha % Check if xfoil failed to converge
    warning('One or more alpha values failed to converge. Last converged was alpha = %f. Rerun with ' oper/iter ##' command \n', pol.alpha(end))
    warning('Alpha = %f failed to converge. Rerun with ' oper/iter## command.\n', alpha);
    pol.alpha = alpha;
pol.err = 1; %error flag
pol.CL = 0;
pol.CD = 0;
pol.CDP = 0;
pol.Cm = 0;
end
end

C.6 aerocalcs.m script

Full \scripts\tools\Xfoil\aerocalcs.m script

function [coeff, airfoil]=aerocalcs(foil, aoa, Re, Mach, varargin)
% AEROCALCS: Uses Xfoil to calculate airfoil aero coefficients and
% exports them in two structures.
% inputs: foil = String of the airfoil name
% aoa = Array of angles to test [Degrees]
% Re = Array of Reynolds to test with each aoa (size must
%      match)
% Mach = Array of Mach # to test with each aoa (size
%       match)
% [plot] = true or false. If empty, defaults to false
% [Np] = number of airfoil panels (defaults to 50)
% [print] = print iterations to workspace (default to true)
% outputs: coeff = String of the airfoil name
% airfoil = Array of angles to test [Degrees]

% set default for plot switch
if isempty(varargin)
    varargin{1} = false;
    varargin{2} = 50;
    varargin{3} = true;
else
    if length(varargin) < 2
        varargin{2} = 50;
    end
    if length(varargin) < 3
        varargin{3} = true;
    end
end

% pre-Allocate Vectors to Fill
coeff.Cl = zeros(1, length(aoa));
coeff.Cd = zeros(1, length(aoa));
coeff.Cm = zeros(1, length(aoa));
fprintf('Starting Xfoil iterations...
');
tic;
for k=1:length(aoa)
    if varargin{3}
        fprintf('M=%.4f\tAoA=%.3f\tRe=%.3e\n', Mach(k), aoa(k), Re(k));
    end
    [pol, foilDat] = xfoil(foil, aoa(k), ...
        Re(k), Mach(k), varargin{2}, 'operator+100 ');
    coeff.Cl(k) = pol.CL; % Cl
    coeff.Cd(k) = pol.CD; % Cd
    coeff.Cm(k) = pol.Cm; % Cm
    err(k) = pol.err; % error flag
    airfoil.xcp{k} = foilDat.xcp;
    airfoil.cp{k} = foilDat.cp;
end
fprintf('Finished %d iterations in %f seconds\n', length(aoa), toc);

airfoil.x=foilDat.x;
airfoil.y=foilDat.y;
if sum(err)>0 % if there were errors
    errC=1;% error counter
    errI=find(err==1); % get indices where xfoil failed
    fprintf('Non convergance on elements:\n');
    noErrI=find(err==0); % get indices where xfoil didn’t fail
    maxI=find(aoa==max(aoa)); % find index of maximum AOA location
    % if there are more than one max AOA (say, all the same for example )
    if max(size(maxI))>1
        maxI=maxI(1); % use first value
    end
    for j=1:length(errI) % for each error
        fprintf('Error %d, ', errI(j));
        % find which data set to use for interpolation: above or below max aoa
    end
if errI(j)<=maxI
    interI=find(noErrI<=maxI);
else
    interI=find(noErrI>maxI);
end
try
    coeff.Cl(errI(j)) = interp1(aoa(noErrI(interI)),...
        coeff.Cl(noErrI(interI)),aoa(errI(j)),'pchip','extrap'
    );%Cl
    coeff.Cd(errI(j)) = interp1(aoa(noErrI(interI)),...
        coeff.Cd(noErrI(interI)),aoa(errI(j)),'pchip','extrap'
    );%Cd
    coeff.Cm(errI(j)) = interp1(aoa(noErrI(interI)),...
        coeff.Cm(noErrI(interI)),aoa(errI(j)),'pchip','extrap'
    );%Cm
%for CP do the same interpolation of converged elements
for all
    try
        coeff.err(errI(j)) = 0;
        catch ME
            warning('Interpolation Failed due to Max &/or Min > 1. 0
                assigned');
            %rethrow(ME)
            coeff.Cl(errI(j)) = NaN;
            coeff.Cd(errI(j)) = NaN;
            coeff.Cm(errI(j)) = NaN;
            for p=1:length(airfoil.cp{j})
                airfoil.cp{errI(j)}(p) = interp1(aoa(noErrI(interI)),...
                    tempCP(p,noErrI(interI)),aoa(errI(j)),'pchip','extrap');
            end
            coeff.err(errC) = ME;
            errC=errC+1;
        end
    end
end
if sum(isnan(coeff.Cl))>0
    try
        coeff.Cl = interp1(Mach(not(isnan(coeff.Cl))),...
            coeff.Cl(not(isnan(coeff.Cl))),Mach,'pchip','extrap');
        coeff.Cd = interp1(Mach(not(isnan(coeff.Cd))),...
            coeff.Cd(not(isnan(coeff.Cd))),Mach,'pchip','extrap');
        coeff.Cm = interp1(Mach(not(isnan(coeff.Cm))),...
            coeff.Cm(not(isnan(coeff.Cm))),Mach,'pchip','extrap');
        for p=1:length(airfoil.cp{j})
            tempCP=[airfoil.cp{:}];
            airfoil.cp{:}(p) = interp1(Mach(not(isnan(coeff.Cl))),...
tempCP(p, not(isnan(coef.Cl)), Mach, 'pchip', 'extrap');

    end
    catch ME
        warning('Tried to remove the NaN. Couldn't do it');
        coef.errMsg=ME;
    end
end
fprintf('
');
end
coef.aoa = aoa;

%% plot
if varargin{1}
    figure('Name', 'Airfoil Characteristics');
    subplot(2,2,1);
    hold on;
    plot(aoa, coef.Cl, '*b', 'DisplayName', 'Converged Solution');
    if sum(err)>0; plot(aoa(errI), coef.Cl(errI), 'xr', 'DisplayName', 'Interpolated Solution'); end;
    hold off;
    legend('−DynamicLegend', 'Location', 'NorthWest');
    grid on;
    xlabel('\alpha (deg)');
    ylabel('Cl');
    title('Cl\alpha from Xfoil');
 subplot(2,2,2);
    hold on;
    plot(aoa, coef.Cd, '*b', 'DisplayName', 'Converged Solution');
    if sum(err)>0; plot(aoa(errI), coef.Cd(errI), 'xr', 'DisplayName', 'Interpolated Solution'); end;
    hold off;
    legend('−DynamicLegend', 'Location', 'NorthWest');
    grid on;
    xlabel('\alpha (deg)');
    ylabel('Cd');
    title('Cd\alpha from Xfoil');
 subplot(2,2,3);
    hold on;
    plot(aoa, coef.Cm, '*b', 'DisplayName', 'Converged Solution');
    if sum(err)>0; plot(aoa(errI), coef.Cm(errI), 'xr', 'DisplayName', 'Interpolated Solution'); end;
    hold off;
    legend('−DynamicLegend', 'Location', 'NorthWest');
    grid on;
    xlabel('\alpha (deg)');
    ylabel('Cm');
    title('Cm\alpha from Xfoil');
 subplot(2,2,4);
C.7 wopwop.m script

Full `scripts\tools\wopwop\wopwop.m` script

```matlab
function noise = wopwop(blade, rotor, flight, varargin)

% WOPWOP: This function finds the location of the filepath containing
% WOPWOP executable as defined in the load_paths, writes the geometry and
% loading
% patch files inputs as well as the case instructions, and then
% executes it
% in the background. It will save the cmd output onto a file named
% 'resultWOPWOP.out' saved in the same directory as WOPWOP. It will
% then
% grab the output data in the /results/ folder and grab the pressure
% and
% sound files to be plotted in MATLAB.
%
% INPUTS: blade structure, containing
% .X3d = X coordinates of blade element node (ft)
% .Y3d = Y coordinates of blade element node (ft)
% .Z3d = Z coordinates of blade element node (ft)
% .Pz = Pressure of element (Pa)
% rotor structure, containing
% .Nb = Number of Blades
% .rpm = RPM
% flight parameters structure, containing:
% .a = speed of sound (ft/s)
% .rho = density (slug/ft^3)
% [plot] = True/False (default if empty: T)
% [wop] = WOPWOP custom inputs:
% .obs = Observer structure (obs.x, obs.y, obs.z)
% Defaults to x=60.728, y=0, z=5.313 [m]
% .t = Time structure (t.n, t.Min, t.Max)
% Defines custom number of times to store
data
% at (n), beginning observer time (Min)[sec
```
ending observer time (Max) [sec]. Defaults to
n=35001, Min=0, Max=30.

[case dir] = Case directory name (default: 'caseEFRC')

OUTPUTS: noise = structure containing the SPL, Pressure and dBA
Plots for each unless specified
written by Xavier Santacruz for the EFRC/ERAU research purposes only

grab parameters from inputs
X3d = blade.X3d.*3048; [% ft to m]
Y3d = blade.Y3d.*3048; [% ft to m]
Z3d = blade.Z3d.*3048; [% ft to m]
Pz = blade.Pz; [% Pa]
Nb = rotor.Nb;
rpm = rotor.rpm;
omega = rpm*2*pi()/60;
a = flight.a.*3048; [% ft/s to m/s]
rho = flight.rho./0.06242796057614516; [% lb/ft^3 to kg/m^3]

set default for plot switch
if isempty(varargin)
    disp('Defaults for optional input args');
    plot
    varargin{1} = true;
    observer location and times
    obs.x = 60.72803;
    obs.y = 0;
    obs.z = 5.313015;
    nt=35001;
    tMin=0;
    tMax=30;
    case file name
    caseFileName = 'caseEFRC';
else
    if nargin<5
        disp('Default Observer Location and Time');
        obs.x = 60.72803;
        obs.y = 0;
        obs.z = 5.313015;
        nt=35001;
        tMin=0;
        tMax=30;
    else
        if isfield(varargin{2},'obs')
            obs.x = varargin{2}.obs.x.*3048; [% ft to m]
            obs.y = varargin{2}.obs.y.*3048; [% ft to m]
            obs.z = varargin{2}.obs.z.*3048; [% ft to m]
        else
            disp('Default Observer Location');
            obs.x = 60.72803;
obs.y = 0;
obs.z = 5.313015;
end
if isfield(varargin{2}, 't')
    nt = varargin{2}.t.n; % 35001 for FAA
    tMin = varargin{2}.t.Min; % time to start measuring [sec]
    tMax = varargin{2}.t.Max; % time to end measuring [sec]
else
    disp('Default Observer Time');
    nt = 35001;
    tMin = 0;
    tMax = 30;
end
end
if nargin < 6
    disp('Default Folder for Cases');
    caseFileName = 'caseEFRC';
else
    caseFileName = varargin{3};
end
end
[i, j] = size(Pz);
%dTau = (tMax) / ((i * j) - 1);
dTau = (2 * pi / omega) / 128;
clear i j
if ~isfield(blade, 'name')
    blade.name = 'Unnamed Blade';
end
% start timer counter
wopTime = tic; % time count
% get the executable directory address
wop_wd = which('PSUWOPWOP.exe');
wop_wd(end - 13:end) = [];
% prepare the case file
fid = fopen([wop_wd '\cases.nam'], 'w');
fprintf(fid, ' &casename\n');
fprintf(fid, ' globalFolderName=\'/\%s/\%
', caseFileName);
fprintf(fid, ' caseNameFile=\'case.nam\'
');
fprintf(fid, ' /\n');
fclose(fid);
clear fid
warning off
mkdir(wop_wd, caseFileName);
mkdir([wop_wd '\ results'], 'results');
warning on
\% Refer to PSU-WOPWOP v.3. User Manual for further details on the structure
\% or content of the case file. See Section 7.1–7.4 in Pages 38–59
\%
\% Open file and save in caseEFRC file, relative to WOPWOP’s location
fid = fopen([wop.wd ‘\’ caseFileName ‘\case.nam’],’w’);
\% BEGIN file content. Set calculation environment
fprintf(fid,’ &EnvironmentIn\n’);
fprintf(fid,’ nbObserverContainers = 1\n’);
fprintf(fid,’ nbSourceContainers = 1\n’);
\% location to save results from WOPWOP; relative to WOPWOP’s location
fprintf(fid,’ pressureFolderName=’/results/’\n’);
fprintf(fid,’ SPLFolderName =’/results/’\n’);
fprintf(fid,’ sigmaFolderName =’/sigma/’\n’);
fprintf(fid,’ audioFolderName =’/audio/’\n’);
\% output flags
fprintf(fid,’ debugLevel = 1\n’);
fprintf(fid,’ ASCIIOutputFlag=.true.\n’);
fprintf(fid,’ OASPLdBFlag =.true.\n’);
fprintf(fid,’ OASPLdBAFlag =.true.\n’);
fprintf(fid,’ spectrumFlag =.true.\n’);
fprintf(fid,’ SPLdBFlag =.true.\n’);
fprintf(fid,’ SPLdBAFlag =.true.\n’);
fprintf(fid,’ acousticPressureFlag=.true.\n’);
fprintf(fid,’ thicknessNoiseFlag=.true.\n’);
fprintf(fid,’ loadingNoiseFlag=.true.\n’);
fprintf(fid,’ totalNoiseFlag=.true.\n’);
fprintf(fid,’ EPNLFlag =.true.\n’);
fprintf(fid,’ forceEPNL =.true.\n’);
fprintf(fid,’ PNLFlag =.true.\n’);
fprintf(fid,’ PNLFlag =.true.\n’);
fprintf(fid,’ audioFolderName =’/\n’);
% END &EnvironmentIn
\%
\% BEGIN Environmental Constants (in metric)
fprintf(fid,’ &EnvironmentConstants\n’);
fprintf(fid,’ rho = %8f \n’,rho);
fprintf(fid,’ c = %8f\n’,a);
fprintf(fid,’ windowFunction=’‘Hanning Window’\n’);
% END &EnvironmentConstants
\%
fprintf(fid,’ &ObserverIn\n’);
fprintf(fid,’ nt = %d ! was 256, 35001 FAA\n’,nt);
fprintf(fid,’ tMin = %8f \n’,tMin);
fprintf(fid,’ tMax = %8f \n’,tMax);
fprintf(fid,’ xloc = %8f \n’,obs.x);
fprintf(fid,’ yloc = %8f \n’,obs.y);
fprintf(fid,’ zloc = %8f \n’,obs.z);
fprintf(fid,’ nbFreqRanges = 2\n’);
\% END &ObserverIn
\%
fprintf(fid,’ &RangeIn\n’);
fprintf(fid,’ Title=’‘Low’\n’);
minFrequency = 0.01 ! To small, \textsc{PSU-WOP} will default to min possible\(n\).
maxFrequency = 30.\(n\);

Title=’MidToHigh’\(n\);
minFrequency = 30.\(n\);
maxFrequency = 100000.0 ! Way too big – \textsc{WOP} will default to max possible\(n\);

Title=’\textsc{EFRC} %s Run’\(n\), datetrim(now, ’yyyymmddHHMM’));

nbContainer=1\(n\);

nbBase=0\(n\);
dTau = \(\approx e\(n\), dTau);

Title=’Main Rotor’\(n\);

nbBase = 1\(n\);

nbContainer = %d ! two blades in main rotor\(n\), Nb);

Title=’Constant Rotation’\(n\);

Omega= %8f ! radians/s was 30.04796\(n\), omega);

AxisValue = 0.0, 0.0, 1.0\(n\);

\%Blade definition, repeat for each blade with according name and position
for idx=1:Nb
    Title=’Blade %d,\(n\),idx);
    patchGeometryFile=’geometry.dat’\(n\);
    patchLoadingFile=’functional.dat’\(n\);
    nbBase=3\(n\);
    Title=’Constant Rotation’\(n\);
    AngleType=’\textquotesingle TimeIndependent’\(n\);
    angleValue= %8f (3.141592\(n\), (idx - 1)*2*pi() / Nb);
    axisValue = 0.0, 0.0, 1.0\(n\);
    Title=’\textquotesingle Constant translation’\(n\);
    translationType=’\textquotesingle TimeIndependent’\(n\);
    translationValue = 0.0, 0.0, 0.0\(n\);
fprintf(fid, \&CB\n');
fprintf(fid, 'Title="Pitch"\n');
fprintf(fid, 'AngleType="TimeIndependent"\n');
fprintf(fid, 'AngleValue= 0.0 \n');
fprintf(fid, 'AxisValue = 0.0, 1.0, 0.0\n');
end
fclose(fid);

%f%% prepare the pathfiles
fclose(fid);

% HELPER FUNCTIONS

% pad text to specified byte length, and convert ascii to bit values
so it
% can be read by a text editor.
binTxt = @(T, N) int8(char(pad(T,N)));
% convert single precision floats to 4 byte unsigned binary
floatBin = @(F) typecast(single(F),'uint8');
% write numbers in spec bytes. NOT to be used for floating point
% values
% used in the coordinates. modified from source:
% https://www.mathworks.com/matlabcentral/answers/286647−matlab−to−write−binary−files
RowV = @(C) reshape(C, 1, []);
FirstN = @(C, N) C(1:min(N,end));
ZPadTo = @(C, N) [C, zeros(1, N − length(C), 'int8')];
CString = @(C, N) ZPadTo( FirstN(RowV(uint8(C)), N−1), N ) ;

% % writeGeoPatch ;
% Pressure Units (TEXT)
unitsTxt=’Pa’; %32 chars max
% Comments (TEXT)
commentTxt = [char(10), ‘Geometry input file for PSU-WOPWOP (Format v1.0)’ , char(10), ...
’_______________________________________________________________’, char(10), ‘
’‘Created by MATLAB (written by Xavier Santacruz)’, char(10), ‘
’‘for research purposes at Embry–Riddle’s EFRC’, char(10), ‘
’‘Creation date: ’ , date , char(10), ‘
’‘Creation time: ’ , datestr(now , ’HH:MM:SS’), char(10), ‘
’‘Blade: ’ , blade.name , char(10), ‘
’‘Units: m, Pa’, char(10), ‘
’‘Format: Structured grid, node−centered, unit area vectors’, ...
char(10), char(10)]; %1024 chars max

%Number of Surfaces (must match loading file too). Note this value
%also determine the total size of file and output, so be sure it
matches
Zones = 4; %upper, lower, and tips
zoneNames = { ’Upper Surface’;...
’Lower Surface’;...
'Root Surface';...
'Tip Surface';

iMax = [1 + size(X3d, 2) / 2; 1 + size(X3d, 2) / 2; size(X3d, 2) / 2; size(X3d, 2) / 2];
jMax = [size(X3d, 1); size(X3d, 1); 2; 2];

%predefine a temporary cell structure with the coordinates to use a
loop later

tempCoord={
    ... % Zone 1 - Upper
    X3d(:,1:iMax(1)),...
    Y3d(:,1:iMax(1)),...
    Z3d(:,1:iMax(1))...
},
    ... % Zone 2 - Lower (overlap first and last point)
    X3d(:,[2,1,end:-1:iMax(1)]),...
    Y3d(:,[1,end:-1:iMax(1)]),...
    Z3d(:,[1,end:-1:iMax(1)]),...

    },
    ... % Zone 3 - Root
    [X3d(1,1:iMax(1)-1);X3d(1,[end:-1:iMax(1)])],... %XZ need to flip
    [Y3d(1,1:iMax(1)-1);Y3d(1,[iMax(1):end])],... %Y's are equal
    [Z3d(1,1:iMax(1)-1);Z3d(1,[end:-1:iMax(1)])]...
},
    ... % Zone 4 - Tip
    [X3d(end,1:iMax(1)-1);X3d(end,[end:-1:iMax(1)])],...
    [Y3d(end,1:iMax(1)-1);Y3d(end,[iMax(1):end])],...
    [Z3d(end,1:iMax(1)-1);Z3d(end,[end:-1:iMax(1)])]...
};

%perform calculations for vertex points to perform Node Centerd Vector
Area

for zdX=1:2
    %get normal unit vectors for each surface
    [U{zdX},V{zdX},W{zdX}]=surfnorm(tempCoord{zdX}{1},tempCoord{zdX}
    }{2},tempCoord{zdX}{3});

    %for each set of coordinates
    for cdX=1:size(tempCoord{zdX},2)
        %setup area vertex points empty array
        mid{cdX}=zeros(size(tempCoord{zdX}{cdX})+1);
        %set mid points (node + half distance to next node)
        mid{cdX}(2:end-1,2:end-1)=tempCoord{zdX}{cdX}(1:end-1,1:end-1)
        +...((tempCoord{zdX}{cdX}(2:end,2:end)--tempCoord{zdX}{cdX}(1:
        end-1,1:end-1))/2);

        %set corner points
        mid{cdX}(1,1)=tempCoord{zdX}{cdX}(1,1);
        mid{cdX}(1,end)=tempCoord{zdX}{cdX}(1,end);
        mid{cdX}(end,1)=tempCoord{zdX}{cdX}(end,1);
        mid{cdX}(end,end)=tempCoord{zdX}{cdX}(end,end);

        %set boundary points
        mid{cdX}(1,2:end-1)=tempCoord{zdX}{cdX}(1,1:end-1)+((tempCoord
        {zdX}{cdX}(1,2:end)--tempCoord{zdX}{cdX}(1,1:end-1))/2);
        mid{cdX}(end,2:end-1)=tempCoord{zdX}{cdX}(end,1:end-1)+((
        tempCoord{zdX}{cdX}(end,2:end)--tempCoord{zdX}{cdX}(end,1:
        end-1))/2);
        mid{cdX}(2:end-1,1)=tempCoord{zdX}{cdX}(1:end-1,1)+((tempCoord
        {zdX}{cdX}(2:end,1)--tempCoord{zdX}{cdX}(1:end-1))/2);

        for...
mid\{cdx\}(2:\text{end}-1,\text{end})=\text{tempCoord}\{cdx\}\{1:\text{end}-1,\text{end}\}+((\text{tempCoord}\{cdx\}\{2:\text{end},\text{end}\}-\text{tempCoord}\{cdx\}\{1:\text{end}-1,\text{end}\})\div 2);\]

end

%extract vertex points for ease of calculations
Xmid\{cdx\}=\text{mid}\{1\};
Ymid\{cdx\}=\text{mid}\{2\};
Zmid\{cdx\}=\text{mid}\{3\};
clear\ cdx\ mid
%calculate area vectors (cross product) and the are norm (magnitude) for each original node
for\ i=1:\text{size}(Xmid\{cdx\},1)-1
  for\ j=1:\text{size}(Xmid\{cdx\},2)-1
    Amid\{cdx\}(i,j,:)=\text{cross}(...
      [Xmid\{cdx\}(i+1,j)-Xmid\{cdx\}(i,j), Ymid\{cdx\}(i+1,j)-Ymid\{cdx\}(i,j), Zmid\{cdx\}(i+1,j)-Zmid\{cdx\}(i,j)], ...
      [Xmid\{cdx\}(i,j+1)-Xmid\{cdx\}(i,j), Ymid\{cdx\}(i,j+1)-Ymid\{cdx\}(i,j), Zmid\{cdx\}(i,j+1)-Zmid\{cdx\}(i,j)]);
    Nmid\{cdx\}(i,j,:)=\text{abs}(\text{norm}([Amid\{cdx\}(i,j,1), Amid\{cdx\}(i,j,2), Amid\{cdx\}(i,j,3)]));
  end
end
clear\ i\ j
%set scale to be normalized against largest area (as is in Case 5)
AScale\{cdx\}=\text{Nmid}\{cdx\}\div\text{max}(\text{max}(\text{Nmid}\{cdx\}));
%size unit vectors in accordance with area magnitude
U\{cdx\}=U\{cdx\}\times\text{Nmid}\{cdx\}\{(:, :)\};
V\{cdx\}=V\{cdx\}\times\text{Nmid}\{cdx\}\{(:, :)\};
W\{cdx\}=W\{cdx\}\times\text{Nmid}\{cdx\}\{(:, :)\};
clear\ zdx\ cdx
%predefine temporary structure to hold the vector areas for each node
tempNorms={
  {...
    % Zone 1 – Upper
    -1.*U\{1\}, ...
    -1.*V\{1\}, ...
    -1.*W\{1\}, ...
  },
  {...
    % Zone 2 – Lower (overlap first and last point)
    U\{2\}, ...
    V\{2\}, ...
    W\{2\}, ...
  },
  {...
    % Zone 3 – Root
    \text{zeros}(\text{size}(\text{tempCoord}\{3\}\{1\})), ...
    \text{zeros}(\text{size}(\text{tempCoord}\{3\}\{2\})), ...
    \text{zeros}(\text{size}(\text{tempCoord}\{3\}\{3\})), ...
  },
  {...
    % Zone 4 – Tip
    \text{zeros}(\text{size}(\text{tempCoord}\{4\}\{1\})), ...
    \text{zeros}(\text{size}(\text{tempCoord}\{4\}\{2\})), ...
    \text{zeros}(\text{size}(\text{tempCoord}\{4\}\{3\})), ...
  });
% Note:
i\text{Max} (chordwise nodes) goes from TE to LE (1=TE, end=LE)
if exist('wop_wd\caseFileName\geometry.dat', 'file') == 2
    delete(['wop_wd\caseFileName\geometry.dat']);
end

fid = fopen('wop_wd\caseFileName\geometry.dat', 'w-', 'ieee-le');
fwrite(fid, CString(42,4)); %42 [4] The magic number
fwrite(fid, CString(1,4)); fwrite(fid, CString(0,4)); %1 0 [8] The
    version numbers
fwrite(fid, binTxt(unitsTxt,32)); %ASCII [32] The pressure units
fwrite(fid, binTxt(commentTxt,1024)); %ASCII [1024] A comment string

%load function switches (see Pg.60, section 7.5.1 of WOPWOP3 UM)
fwrite(fid, CString(1,4)); % 1 [4] This is a geometry file
fwrite(fid, CString(Zones,4)); % [4] number of zones (surfaces)
fwrite(fid, CString(1,4)); % 1 [4] They are structured
fwrite(fid, CString(1,4)); % 1 [4] They are constant
fwrite(fid, CString(1,4)); % 1 [4] 1= normals and areas are node–
    centered, 2=face–centered
fwrite(fid, CString(1,4)); % 1 [4] Data is single–precision
fwrite(fid, CString(0,4)); % 0 [4] IBLANK data NOT included
fwrite(fid, CString(0,4)); % 0 [4] Unused in this version

%one set per zone
for zdx=1:Zones
    fwrite(fid, binTxt(zoneNames{zdx},32)); % [32] The zone patch name
    fwrite(fid, CString(iMax(zdx),4)); fwrite(fid, CString(jMax(zdx),4))
); % [4] iMax, [4] jMax
end

%coordinates, one set per zone:
%
% (Zone 1 X coordinates)[iMax1*jMax1*4] The node data
%
% (Zone 1 Y coordinates)[iMax1*jMax1*4]
%
% (Zone 1 Z coordinates)[iMax1*jMax1*4]
%
% (Zone 1 Normal X coordinates)[iMax1*jMax1*4] The normal data ALL
% ONES
%
% (Zone 1 Normal Y coordinates)[iMax1*jMax1*4]
%
% (Zone 1 Normal Z coordinates)[iMax1*jMax1*4]
%
% for each individual 4–byte coordinate, use: typecast(single(##COORD
##),'uint8')
for zdx=1:Zones
    fwrite coordinates
    for cdx=1:3
        for jdx=1:jMax(zdx)
            for idx=1:iMax(zdx)
                %write ith coords per jth step
                fwrite(fid, floatBin(tempCoord{zdx}{cdx}{jdx},idx)); %
[4]
            end
        end
    end
%write normals
end
for cdx=1:3
for jdx=1:jMax(zdx)
    for idx=1:iMax(zdx)
        %write ith coords per jth step
        fwrite(fid, float Bin(tempNorms{zdx}{cdx}{jdx, idx})); % [4]
        if zdx==1
            fwrite(fid, float Bin(1.00)); % [4]
        elseif zdx==2
            fwrite(fid, float Bin(-1.00)); % [4]
        else
            fwrite(fid, float Bin(0)); % [4]
        end
    end
end
fclose(fid);

% comments (TEXT)
commentTxt = [char(10),...
    'Functional input file for PSU-WOPWOP (Format v1.0)\',char(10),...
    '==================================================',char(10),...
    'Created by MATLAB (written by Xavier Santacruz)\',char(10),...
    'for research purposes at Embry-Riddle’ s EFRC',char(10),...
    'Creation date: ',datestr(now,'HH:MM:SS'),char(10),...
    'Creation time: ',char(10),...
    'Blade: ',blade.name,char(10),...
    'Units: m, Pa',char(10),...
    'Format: Structured grid, node-centered, unit area vectors',...
    char(10),char(10)]; %1024 chars max

% also determine the total size of file and output, so be sure it matches
Zones = 4; %upper, lower, and tips
dataZones = 2; %
zoneId=[1:2]; %
zoneNames = {'Upper Surface';...
    'Lower Surface'};
iMax = [1+size(X3d,2)/2;1+size(X3d,2)/2];
jMax = [size(X3d,1);size(X3d,1)];
%
% predefine a temporary cell structure with the pressure to use a loop later
tempP= {...
    Pz(:,1:iMax(1))...
% Zone 2 − Lower (overlap first and last point)
Pz (:,[1,end:−1:iMax(1)]);%

% WRITE FILE
if exist ([wop wd '\caseFileName 'functional.dat'], 'file')==2
    delete ([wop wd '\caseFileName 'functional.dat']);
end
fid = fopen ([wop wd '\caseFileName 'functional.dat'], 'w', 'ieee−le ');
fwrite (fid, CString (42,4)); % The magic number
fwrite (fid, CString (1,4)); fwrite (fid, CString (0,4)); % The version numbers
fwrite (fid, CString (42,4)); fwrite (fid, CString (1,4)); fwrite (fid, CString (0,4)); % The number of zones with data
for zdx=1:datZones
    fwrite (fid, CString (zoneId (zdx),4));
end
for zdx=1:datZones
    fwrite (fid, binTxt (zoneNames{zdx},32)); % The zone patch name
    fwrite (fid, CString (1,4)); % time information
    fwrite (fid, CString (iMax (zdx),4)); fwrite (fid, CString (jMax (zdx),4));
end
end
% One set per zone
% Coordinates, one set per zone:
%(Zone 1 Pressure)[iMax1*jMax1*4] The node data
% for each individual 4−byte coordinate, use: typecast (single(##COORD ##), 'uint8 ')
for zdx=1:datZones
    %write pressure values at nodes
    for jdx=1:jMax(zdx)
        for idx=1:iMax(zdx)
            fwrite (fid, floatBin (tempP{zdx}{1}{jdx, idx})); %
        end
    end
end
end
%
fclose(fid);
%
% run

fprintf('Running PSU-WOPWOP.\n');
%
% generate waiting dialogue
busyFig = dialog;
scrsz = get(groot,'ScreenSize'); %get screen dimensions
set(busyFig,'position',[100,100,120,120]); % set internal size
wdwsz = get(busyFig,'OuterPosition'); %get outer dimensions to set
   position
set(busyFig,'OuterPosition',[(scrsz(3)-wdwsz(3))/2,(scrsz(4)-wdwsz(4))
   /2,wdwsz(3),wdwsz(4)]);
%
% insert text
uicontrol('Parent',busyFig,...
   'Style','text',...
   'Position',[10 70 100 30],...
   'String','PSU-WOPWOP');
%
% get wait gif and set location
iconsClassName = 'com.mathworks.widgets.BusyAffordance$AffordanceSize' ;
iconsSizeEnums = javaMethod('values',iconsClassName);
SIZE_32x32 = iconsSizeEnums(2); % (1) = 16x16, (2) = 32x32
jObj = com.mathworks.widgets.BusyAffordance(SIZE_32x32,'Running...');
   % icon, label
%
% set parameters
jObj.setPaintsWhenStopped(true); % default = false
jObj.useWhiteDots(false); % default = false (true is good for
   dark backgrounds)
javacomponent(jObj.getComponent, [20,30,80,50], busyFig);
jObj.start;
%
% Do work

cmd = sprintf('cd %s & & PSUWOPWOP.exe > resultWOPWOP.out',wop_wd);

if (status!=0)
   % Work is done
   jObj.stop;
   jObj.setBusyText('Error!!');
   pause(1)
   % close dialogue
   close(busyFig);
   disp(result);
error([mfilename ' : system '], 'WOPWOP execution failed! %s', cmd);
end
% Work is done
jObj.stop;
jObj.setBusyText('All done!');
pause(1)
% close dialogue
close(busyFig);
% clear unused vars
clear jObj iconsClassName iconsSizeEnums SIZE_32x32 scrsz busyFig

fprintf('Done.\n');
%
%% Import data from WOPWOP pressure file.
try
% Initialize variables.
filename = [wop_wd '\' caseFileName '\results\pressure.tec'];
delimiter = ', ';
%VariableNames extractor
nameDelimiter = {', ', '='};
nameStartRow = 2;
nameEndRow = 2;
%DataExtractor
startRow = 4;

% Format for each line of text:
formatSpec = '%15f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f%16f\n\n';

% Open the text file.
fileID = fopen(filename, 'r');

% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this code.
% It may be necessary to regenerate the code from the Import Tool.
%Names
textscan(fileID, '%\n\n', nameStartRow-1, 'WhiteSpace', '', 'ReturnOnError', false);
varNameArray = textscan(fileID, nameFormatSpec, nameEndRow-
nameStartRow+1, 'Delimiter', nameDelimiter, 'TextType', 'string',
'ReturnOnError', false, 'EndOfString', '\r\n');
origNameArray = varNameArray;
%format names for appropriate table name
for i=1:size(varNameArray,2)
%remove signs that cannot be used in variable table names
remove_sign = {'\', '!', '"', '$', '-', '/', '(', ')', '%', '<', '>'};
for idx_rem = 1:length(remove_sign)
    varNameArray(:,i)=strrep(varNameArray(:,i),remove_sign{idx_rem},''); 
end
%restructure array to load correctly to the table variable name
varTestArray{i}=sprintf('%s',varNameArray(:,i));
noise.pressureNames{i}=sprintf('%s',origNameArray(:,i));

% Data
textscan (fileID, '%\n\r', startRow−1, 'WhiteSpace', '', ',
    ReturnOnError', false, 'EndOfLine', '\r\n');
dataArray = textscan(fileID, formatSpec, 'Delimiter', ',', ',
    WhiteSpace', ',', 'TextType', 'string', 'EmptyValue', NaN, ',
    ReturnOnError', false);

% Close the text file.
fclose (fileID);

% Create output variable
noise.pressure = table (dataArray{1:end−1}, 'VariableNames', [ 
    varTestArray]);%{'TITLEPSUWOP', 'WOP', 'VarName3', 'VarName4', 'VarName5', 'VarName6', 'VarName7', 'VarName8', 'VarName9', 'VarName10', 'VarName11', 'VarName12', 'VarName13'}

% Clear temporary variables
clearvars filename startRow formatSpec fileID dataArray ans
catch ME
    msgbox({'An error occurred while reading the pressure files.';...
        'Probably an error during execution.';...
        'Opening resultWOPWOP.out to aid debugging'});
end

% Import data from WOPWOP SPL file.
clear
clc

% Initialize variables.
filename = [wop_wd ' \caseFileName ' \results\spl_spectrum.tec ];
delimiter = ', ';
VariableNames extractor = {'\',',','='};
nameDelimiter = {'\',',','='};
nameStartRow = 2;
nameEndRow = 2;
formatSpec = '15f16f16f16f16f16f%*[\n\r]';
nameFormatSpec = '*q%q%q%q%q%q%q%*[\n\r]';

% Open the text file.
fileID = fopen(filename, 'r');

% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this code. If an error occurs for a different file, try regenerating the code from the Import Tool.
Names

textscan (fileID, '%*[\n\r]', nameStartRow - 1, 'WhiteSpace', '', 'ReturnOnError', false);
varNameArray = textscan (fileID, nameFormatSpec, nameEndRow - nameStartRow + 1, 'Delimiter', 'WhiteSpace', 'ReturnOnError', false, 'EndOfLine', '\r\n');

origNameArray = varNameArray;
%format names for appropriate table name
for i = 1:size (varNameArray, 2)
  remove_sign = {'', '[', ']', '$', '\', '/', '(', ')', '%', '<', '>'};
  for idx_rem = 1:length (remove_sign)
    varNameArray(:, i) = strrep (varNameArray(:, i), remove_sign{idx_rem}, '');
    varNameArray(:, i) = strrep (varNameArray(:, i), '*', '');
    origNameArray(:, i) = strrep (origNameArray(:, i), '*', '');
    origNameArray(:, i) = strrep (origNameArray(:, i), '_', '');
  end
end
%restructure array to load correctly to the table variable name
varTestArray{i} = sprintf ('%s', varNameArray(:, i));
noise.splNames{i} = sprintf ('%s', origNameArray(:, i));
end

% Data

textscan (fileID, '%*[\n\r]', startRow - 1, 'WhiteSpace', '', 'ReturnOnError', false, 'EndOfLine', '\r\n');
dataArray = textscan (fileID, formatSpec, 'Delimiter', '', 'WhiteSpace', '', 'WhiteSpace', 'WhiteSpace', 'WhiteSpace', 'WhiteSpace', 'WhiteSpace', 'ReturnOnError', false);

% Close the text file.
fclose (fileID);

% Create output variable
noise.splSpectrum = table (dataArray{1:end-1}, 'VariableNames', [varTestArray]);

%DataExtractor
startRow = 4;

% Format for each line of text:

% Open the text file.

% This call is based on the structure of the file used to generate this code. If an error occurs for a different file, try regenerating the code from the Import Tool.

% from the Import Tool.

% Names

% Data

% Close the text file.

% Create output variable

noise.splSpectrum = table (dataArray{1:end-1}, 'VariableNames', [varTestArray]);

% DataExtractor
startRow = 4;

% Format for each line of text:

% Open the text file.

% This call is based on the structure of the file used to generate this code. If an error occurs for a different file, try regenerating the code from the Import Tool.

% from the Import Tool.

% Names

% Data

% Close the text file.

% Create output variable

noise.splSpectrum = table (dataArray{1:end-1}, 'VariableNames', [varTestArray]);

% DataExtractor
startRow = 4;

% Format for each line of text:

% Open the text file.

% This call is based on the structure of the file used to generate this code. If an error occurs for a different file, try regenerating the code from the Import Tool.

% from the Import Tool.

% Names

% Data

% Close the text file.

% Create output variable

noise.splSpectrum = table (dataArray{1:end-1}, 'VariableNames', [varTestArray]);

% DataExtractor
startRow = 4;

% Format for each line of text:

% Open the text file.

% This call is based on the structure of the file used to generate this code. If an error occurs for a different file, try regenerating the code from the Import Tool.

% from the Import Tool.

% Names

% Data

% Close the text file.

% Create output variable

noise.splSpectrum = table (dataArray{1:end-1}, 'VariableNames', [varTestArray]);
VarName5’, ‘VarName6’, ‘VarName7’, ‘VarName8’, ‘VarName9’, ‘VarName10’, ‘VarName11’, ‘VarName12’, ‘VarName13’ });

% Clear temporary variables

clearvars filename startRow formatSpec fileID dataArray ans delimiter nameDelimiter nameStartRow nameEndRow nameFormatSpec varNameArray varTestArray remove_sign idx rem i origNameArray;

if varargin{1}
    figure;
    clf;
end

grid minor
title( ‘WOPWOP Pressure Output Post Processing’ );
xlabel(noise.pressureNames{1});
ylabel(‘\Delta \text{P} \text{(Acoustic)} (\text{Pa})’);
hold on
for i = 2:4
    plot(noise.pressure{:,1},noise.pressure{:,i},’displayName’, noise.pressureNames{i});
end
hold off
legend(’–DynamicLegend’);

datLength = size(noise.splNames,2)−1;

figure;
clf;
for i = 2:4
    semilogx(noise.splSpectrum{:,1},noise.splSpectrum{:,i},’displayName’,noise.splNames{i});
    hold on
end
hold off

%set(gca,’yscale’,’log’)
grid on
title( ‘WOPWOP SPL Output Post Processing’ );
xlabel(noise.splNames{1});
%xlim([0 5000])
ylabel(‘dB’);
legend(’–DynamicLegend’);

figure;
clf;
for i = 5:7
    semilogx(noise.splSpectrum{:,1},noise.splSpectrum{:,i},’displayName’,noise.splNames{i});
    hold on
end
hold off
%semilogx(noise.splSpectrum{1},noise.splSpectrum{7},'
displayName',noise.splNames(7));

grid on
title('WOPWOP SPL Output Post Processing');
xlabel(noise.splNames{1});
xlim([10 1000])
ylabel('dBA');
legend('−DynamicLegend');
%
end
tEnd=toc(wopTime);
fprintf('WopWop runtime: %d min %d sec\n', floor(tEnd/60), ceil(rem(tEnd,60)));
%
end

C.8 readBinPatch.m script
Full \scripts\tools\xfoil\readBinPatch.m script

function varargout = readBinPatch(fileLoc, varargin)
% readBinPath: This function reads the binary input patch file used by
% WOPWOP and decodes it to regular floating numbers to be used in
% MATLAB.
% It is also good to doublecheck that the Patch Writing scripts are
% working
% correctly
%
% Inputs: fileLoc = File Name (and location if not in the same dir
% [raw out] = True to output raw binary. False to output
% decoded data. [Optional. Defaults to FALSE]
% % Outputs: Data = Decoded Data or Raw Data. See [raw out] input
% [comTxt] = Comment String [Optional]
% [config] = Config Numbers [Optional]
%
% Written by Xavier Santacruz
% R2018a − 2019/02/16
%
% check to see what the user wants to output
if nargin<2 %if no second parameter, set default
    raw_out = false;
end
if nargin==2
    raw_out = varargin{1};
end

% get data from file
rawData=memmapfile(fileLoc).Data;
% check if file type is geometry or loading
if rawData(1069)==1 % if it's a geometry file, then grab the config
    disp(' Geometry Patch File.');
    % clear comment string
    clear comTxt
    % comment and pressure units
    comTxt.comment=char(rawData(45:1068)'); % Comment String
    comTxt.units=char(rawData(13:44)'); % Pressure Units
    % initialize config array
    config=zeros(8,1);
    % 1: Type: Geometry (1) or Loading (2)
    % 2: Number of zones
    % 3: structured (1) or unstructured (2)
    % 4: constant (1), periodic (2), aperiodic (3) geometry
    % 5: node- (1) or face-centered (2) area normal vectors
    % 6: single (1) or double (2) precision
    % 7: isblank included (1) or not (0)
    % 8: ALWAYS 0
    for idx=0:7
        config(idx+1)=rawData(1069+(4*idx));
    end
    clear idx
    % bit 1100 up to here. Now the fun part
    bitCount=1100;
    if config(3)==1 && config(4)==1 % if structured and constant
        disp(' Structured Constant Data Found');
        disp(' Extracting data. ');
        for zdx=1:config(2) % for the amount of zones
            % get the zone name
            zones{zdx}.Name = char(rawData(bitCount+1:bitCount+32)');
            % add the bit count to the end of the string
            bitCount=bitCount+32;
            % get the zones iMax and jMax, adjusting the bit count as we go
            zones{zdx}.iMax = rawData(bitCount+1);
            bitCount=bitCount+4;
            zones{zdx}.jMax = rawData(bitCount+1);
            bitCount=bitCount+4;
        end
        clear zdx
        % Get the coordinates
        for zdx=1:config(2) % for each zone
            for cdx=1:3 % for every coordinate axis X, Y, Z
                for jdx=1:zones{zdx}.jMax % for every j
                    for idx=1:zones{zdx}.iMax % for every i
                        % get the node vector coordinate
                        zones{zdx}.Coords{cdx}(jdx,idx)=typecast(
                            rawData(bitCount+1:bitCount+4), 'single');
                        bitCount=bitCount+4;
                    end
                end
            end
        end
for cdx = 4:6 % for every coordinate axis X, Y, Z
    for jdx = 1:zones{zdx}.jMax % for every j
        for idx = 1:zones{zdx}.iMax % for every i
            % get the normal vector coordinate
            zones{zdx}.Coords{cdx}(jdx,idx) = typecast(
                rawData(bitCount + 1:bitCount + 4), 'single');
            bitCount = bitCount + 4;
        end
    end
end

else % done decoding constant structured data
    disp('Non-structured or non-constant geometry detected. No
         decoding scheme yet.');
    disp('Returning the raw bin data');
    raw_out = true;
end

elseif rawData(1037) == 2 % if its a function file, then grab the config
    disp('Loading Patch File Detected. ');
    % clear comment string
    clear comTxt
    % add pressure units to comment and format it nicely
    comTxt.comment = char(rawData(13:1036)); % Comment String in ASCII
    % initialize config array
    config = zeros(11, 1);
    % 1: Type: Geometry (1) or Loading (2)
    % 2: Number of zones
    % 3: structured (1) or unstructured (2)
    % 4: constant (1), periodic (2), aperiodic (3) geometry
    % 5: node- (1) or face-centered (2) area normal vectors
    % 6: surface pressure (1), loading vector (2), or flow parameters (3)
    % 7: stationary (1), rotating (2), patch (3) fixed frame
    % 8: single (1) or double (2) precision
    % 9: ALWAYS 0 (Unused)
    % 10: ALWAYS 0 (Unused)
    % 11: Number of Zones with Data
    for idx = 0:10
        config(idx+1) = rawData(1037+(4*idx));
    end
    clear idx
    % bit 1100 up to here. Now the fun part
    bitCount = 1080;

    if config(3) == 1 & config(4) == 1 % if structured and constant
        disp('Structured Constant Data Found');
        disp('Extracting data.');
        for zdx = 1:config(11) % for the amount of zones
            zones{zdx}.Id = rawData(bitCount + 1);
            bitCount = bitCount + 4;
end
for zdx=1:config(11) %for the amount of zones
  %get the zone name
  zones{zdx}.Name = char(rawData(bitCount+1:bitCount+32)');
  %add the bit count to the end of the string
  bitCount=bitCount+32;
  %get the zones iMax and jMax, adjusting the bit count as
  %we go
  zones{zdx}.iMax = rawData(bitCount+1);
  bitCount=bitCount+4;
  zones{zdx}.jMax = rawData(bitCount+1);
  bitCount=bitCount+4;
end

clear zdx

if config(5)==1 && config(6)==1 %if data is node centered and
  %surface pressure
  %Get the pressure values for each node
  for zdx=1:config(11) %for each zone
    for jdx=1:zones{zdx}.jMax %for every j
      for idx=1:zones{zdx}.iMax %for every i
        %get the node vector coordinate
        zones{zdx}.pressure(jdx,idx)=typecast(rawData(
          bitCount+1:bitCount+4),'single');
        bitCount=bitCount+4;
      end
    end
  end
elseif config(5)==1 && config(6)==2 %if data is node centered
  %and a vector
  %Get the pressure vector coordinates
  for zdx=1:config(11) %for each zone
    for cdx=1:3 %for every coordinate axis X, Y, Z
      for jdx=1:zones{zdx}.jMax %for every j
        for idx=1:zones{zdx}.iMax %for every i
          %get the node vector coordinate
          zones{zdx}.pressVcoords{cdx}(jdx,idx)=
            typecast(rawData(bitCount+1:bitCount+4)
            ,'single');
          bitCount=bitCount+4;
        end
      end
    end
  end
end
%done decoding constant structured data
else
  disp(' Non-structured or non-constant loading detected. No
  decoding scheme yet.');
  disp(' Returning the raw bin data');
  raw_out=true;
end
else
disp(' File type not recognized. ');
disp(' No decoding available. ');
disp(' Returning the raw bin data');
raw_out=true;
config=[];
comTxt=[];
end

% if user wants raw data, give it to him (unless there wasn’t a
decoding
%scheme set up for it
if raw_out
    varargout{1} = rawData; % output raw data
else
    varargout{1} = zones; % output decoded data
end
if nargout>1 % if more than one output, make second one the comment
    varargout{2} = comTxt;
end
if nargout>2 % if more than two outputs, make the third the config
    varargout{3} = config;
end
disp(' Done. ');
end

C.9 blade_plotting.m script

Full \scripts\tools\xfoil\blade_plotting.m script

function bladeDat = blade_plotting(airfoil, blade, rotor, flight, coeffs, sw, varargin)

% blade_plotting: This function performs all the calculations to
% convert
% BEMT outputs into 3d cartesian coordinates and pressures, plots them
% and returns the calculated values into structure elements. Designed to
% work with the outputs of "run_Prop.m" as the main caller function.

% Inputs:
% - airfoil structure
% - blade structure
% - rotor structure
% - flight structure
% - coefficient structure
% - [show CP(0) or P(1)] Defaults to CP
% Outputs:
% - structure with XYZ 3d Components, Pressure and C,P for each
%  surface
%  
% Xavier Santacruz (santacrx@gmail.com)
%  ERAU EFRC Thesis

if isempty(varargin)
    plotCP=0;
else
    plotCP=varargin{1};
end

bladeDat = blade;

n=blade.el;

scrsz = get(groot,'ScreenSize');

figure('Name','Blade Performance Window');
set(gcf,'position',[scrsz(3)*.1 scrsz(3)*.05 ... 
                 scrsz(3)*.8 scrsz(3)*.4]);

for i=n:-1:1
    %get Angle of blade Element
    theta=blade.beta(i);
    %scale center to local element chord
    center = repmat([blade.tw.center*blade.c(i); 0], 1, length(airfoil.x(airfoil.x<=1')));
    %reset unscaled airfoil coordinates
    v=[airfoil.x(airfoil.x<=1')';airfoil.y(airfoil.x<=1')'];
    %scale them to current element chord
    v(1,:) = blade.c(i).*v(1,:);%scale to element chord
    v(2,:) = blade.c(i).*v(2,:);%scale to element chord
    %get thickness correction factor (if one is desired)
    if isfield(sw,'tc')
        scale=abs(blade.tc(blade.r(i))/blade.tmax);
        if scale>1
            v(2,:) = v(2,:).*scale;
        end
    end
\begin{verbatim}
end
%prepare the rotation matrix
R = [cosd(theta) -sind(theta); sind(theta) cosd(theta)];
%get rotated coordinates
vo = R*(v - center) + center;
%get new coordinates
x_rotated = vo(1,:);
y_rotated = vo(2,:);
%size coordinates based on geometry of blade
X3d(i,:) = -blade.LE(blade.r(i))+x_rotated;%chord direction
Y3d(i,:) = blade.x(i).*ones(size(airfoil.x(airfoil.x<=1)));
Z3d(i,:) = y_rotated;%thickness
end
% Get Pressure Distribution of blade for colorscheme
if sw.aero==1
%initialize arrays
% Get Pressure Distribution of blade for colorscheme
%initialize arrays
CPz = zeros(n,length(airfoil.x(airfoil.x<=1)));
Pz = zeros(n,length(airfoil.x(airfoil.x<=1)));
%find separation point between upper and lower surface of airfoil
%coords
upx=find(airfoil.x(airfoil.x<=1)==min(airfoil.x(airfoil.x<=1)));
for i=n:-1:1
%find the separation point between upper and lower surface of CP
%coords
upxcp=find(airfoil.xcp{i}==min(airfoil.xcp{i}),1);
%get the x and z coordinates of the CP so it corresponds to the
%plotted airfoil coordinates. Must separate into upper and lower
%because interp can't have identical x's
if length(airfoil.xcp{i})(1:upxcpI)==length(airfoil.x(1:upxI))
  v=[airfoil.x(airfoil.x<=1)';...
    [...%Upper Surface interpolation
     airfoil.cp{i}(1:upxcpI)',...
     ... %Lower Surface interpolation
     airfoil.cp{i}(upxcpI+1:end)',...
    ];
else
  v=[airfoil.x(airfoil.x<=1)';...
    [...%Upper Surface interpolation
     interp1(airfoil.xcp{i})(1:upxcpI)',...
     airfoil.cp{i}(1:upxcpI)',...
     airfoil.x(1:upxI)','.pchip'),...
     ... %Lower Surface interpolation
     interp1(airfoil.xcp{i}(upxcpI+1:end)',...
     airfoil.cp{i}(upxcpI+1:end)',...
     airfoil.x(upxI+1:length(airfoil.x(airfoil.x<=1)))','.pchip')
\end{verbatim}
end

% fill the resulting array with CP values corresponding to the
% plotted airfoil coordinates. Get Delta P in Pa from
V_resultant

CPz(i,:) = v(2,:);
Pz(i,:) = 0*flight.P + (CPz(i,:)*...  
( (blade.Mach(i)*a).^2 ).* ... /cosd(blade.AoA(i))).*...  
 rho.*.5);

end
hold on
if plotCP==1
    h=surf(X3d,Y3d,Z3d,Pz) ;
c=colorbar('east', 'TickDirection', 'out');
c.Label.String=\Delta Pressure (Pa); 
c.Label.FontSize=12;
    fill3(X3d(1,:),Y3d(1,:),Z3d(1,:),Pz(1,:),...
    'edgecolor',[128/255 128/255 128/255]) ;%
    fill3(X3d(end,:),Y3d(end,:),Z3d(end,:),Pz(end,:),...
    'edgecolor',[128/255 128/255 128/255]) ;%
else
    h=surf(X3d,Y3d,Z3d,CPz);
c=colorbar('east', 'TickDirection', 'out');
c.Label.String=Pressure Coefficient;
c.Label.FontSize=12;
    fill3(X3d(1,:),Y3d(1,:),Z3d(1,:),CPz(1,:),...
    'edgecolor',[128/255 128/255 128/255]) ;%
    fill3(X3d(end,:),Y3d(end,:),Z3d(end,:),CPz(end,:),...
    'edgecolor',[128/255 128/255 128/255]) ;%
end

plot3(X3d(:,max(X3d(1,:)<=1)),Y3d(:,max(X3d(1,:)<=1)),Z3d(:,max(X3d(1,:)<=1)),...
plot3(blade.R.*cos(linspace(0,2*pi)),blade.R.*sin(linspace(0,2*pi)) ),zeros(size(linspace(0,2*pi)))),...
    ':k','HandleVisibility', 'off')
set(h, 'edgecolor', 'none');
colormap('jet');%'gray');%'winter');
title('Blade Pressure Distribution');
bladeDat.CPz=CPz;
bladeDat.Pz=Pz;
else
hold on
h=surf(X3d,Y3d,Z3d);
    fill3(X3d(1,:),Y3d(1,:),Z3d(1,:), 'k', ...  
    'edgecolor',[128/255 128/255 128/255]) ;%
    fill3(X3d(end,:),Y3d(end,:),Z3d(end,:), 'k', ...  
    'edgecolor',[128/255 128/255 128/255]) ;%
plot3(X3d(:,max(X3d(1,:))<=1)),Y3d(:,max(X3d(1,:))<=1)),Z3d(:,max(X3d(1,:))<=1)),...
plot3(blade.R.*cos(linspace(0,2*pi)),blade.R.*sin(linspace(0,2*pi))
    ),zeros(size(linspace(0,2*pi))));...
    ':k','HandleVisibility','off');
set(h,'edgecolor','none');
colormap('gray');%'winter');
title('3D Blade');
end
rotate3d on;
hold off;
grid on;
view([225,60]);%135,45];
xlim([-blade.R/2,blade.R/2]);
ylim([0,blade.R]);
zoom([-blade.R/2,blade.R/2]);
xlabel('Chord (ft)');
ylabel('Span (ft)');
zlabel('Thickness (ft)');

% Plot thrust and power loading along blade
% Thrust
subplot(2,5,9);
plot(blade.r(1:n),coeffs.dCT,'Linewidth',1);
title('Element C_T Distribution');
xlabel('Radial Position');
ylabel('C_T');
grid minor;
xlim([0 1]);%blade.R]);

% Power
subplot(2,5,10);
plot(blade.r(1:n),coeffs.dCP,'Linewidth',1);
title('Element C_P Distribution');
xlabel('Radial Position');
ylabel('C_P');
grid minor;
xlim([0 1]);%blade.R]);

% Plot Angle of Attack along Blade
subplot(2,5,4);
plot(blade.r(1:n),blade.AoA,'Linewidth',1);
title('Element Angle of Attack');
xlabel('Radial Position');
ylabel('\alpha (deg)');
grid minor;
xlim([0 1]);%blade.R]);
% Plot inflow
subplot(2,5,5),
plot(blade.r(1:n), blade.lambda,'Linewidth',1);
title('Local Induced Velocity');
xlabel('Radial Position');
ylabel('\lambda_i');
grid minor;
xlim([0 1]);

% Plot entire rotor
angl = 360/rotor.Nb;  % Angle Between Blades
figure('Name','Rotor 3D Render');
set(gcf,'position',[scrsz(3)*.25 scrsz(3)*.05 ...
    scrsz(3)*.5 scrsz(3)*.45]);
hold on
for k = 1:rotor.Nb
    for j=1:n
        v=[X3d(j,:) ; Y3d(j,:) ; Z3d(j,:) ];
        R = [ cosd(angl*(k-1)) -sind(angl*(k-1)) 0; 
            sind(angl*(k-1)) cosd(angl*(k-1)) 0; 0 0 1];
        vo=R*v;
        X3dFull(j,:)=vo(1,:);
        Y3dFull(j,:)=vo(2,:);
        Z3dFull(j,:)=vo(3,:);
    end
    fill3(X3dFull(1,:), Y3dFull(1,:), Z3dFull(1,:), 'k' , ...
        'edgecolor',[128/255 128/255 128/255]); %'none';
    fill3(X3dFull(end,:), Y3dFull(end,:), Z3dFull(end,:), 'k' , ...%
        'edgecolor',[128/255 128/255 128/255]); %'none';
    plot3(X3dFull(:,max(X3d(1,:)<=1)), Y3dFull(:,max(X3d(1,:)<=1)),
        Z3dFull(:,max(X3d(1,:)<=1)) , ...%
        'Color',[128/255 128/255 128/255]); %'none';
    h=surf(X3dFull, Y3dFull, Z3dFull); %'edgcolor','none';
    set(h,'edgecolor','none');
colormap('gray'); %'winter';
end
hold off
grid on;
rotate3d on;
view([45.60]);
xlim([-blade.R blade.R]);
ylim([-blade.R blade.R]);
zlim([-blade.R blade.R]);
title('Rotor 3D Render');
xlabel('(ft)');
ylabel('(ft)');
zlabel('(ft)');
Present Results in Command Window

```
fprintf('Vehicle Performance (%d Rotors)
', rotor.Nr);
fprintf('Thrust: %.1f lb
', rotor.Thrust*rotor.Nr/32.2);
fprintf('Power: %.1f hp
', rotor.Power*rotor.Nr/(550*32.2));
fprintf('Rotor Performance
');
fprintf('Thrust: %.1f lb
', rotor.Thrust/32.2);
fprintf('Power: %.1f hp
', rotor.Power/(550*32.2));
fprintf('Torque: %.1f lb-ft
', rotor.Power*5252/(rotor.rpm));
fprintf('RPM: %.1f RPM
', rotor.rpm);
fprintf('Power Loading: %.1f lb/hp
', rotor.Thrust/(rotor.Power/550));
fprintf('Solidity: %.4f
', rotor.sigma);
fprintf('J: %.3f
', rotor.J);
fprintf('eta: %.3f
', rotor.eta);
fprintf('Ct: %.3e
', rotor.CT);
fprintf('Cp: %.3e
', rotor.CP);
```

Output data

```
bladeDat.X3d=X3d;
bladeDat.Y3d=Y3d;
bladeDat.Z3d=Z3d;
if sw.aero ==1
    bladeDat.Pz=Pz;
end
end
```