Pralines 1.0

(Prandtl’s Lifting Line Solver)



**User Manual**

November 2013

Josh Hodson

# Table of Contents

[Table of Contents i](#_Toc372814597)

[Table of Figures ii](#_Toc372814598)

[Table of Tables ii](#_Toc372814599)

[Introduction 2](#_Toc372814600)

[Feature Summary 2](#_Toc372814601)

[System Requirements 4](#_Toc372814602)

[Installation 5](#_Toc372814603)

[Software Overview 6](#_Toc372814604)

[Menus 6](#_Toc372814605)

[User Input 6](#_Toc372814606)

[Numerical Input 6](#_Toc372814607)

[Character Input 7](#_Toc372814608)

[Execution 7](#_Toc372814609)

[Planform Design Menu 7](#_Toc372814610)

[Wing Parameters 8](#_Toc372814611)

[Aileron Parameters 8](#_Toc372814612)

[Output and Plotting Options 8](#_Toc372814613)

[Execution Commands 9](#_Toc372814614)

[Operating Conditions Menu 9](#_Toc372814615)

[Operating Conditions 9](#_Toc372814616)

[Plotting Options 11](#_Toc372814617)

[Execution Commands 12](#_Toc372814618)

[Examples 13](#_Toc372814619)

[Example 1: Tapered Planform 13](#_Toc372814620)

[Example 2: Elliptic Planform 16](#_Toc372814621)

[Example 3: Combination Planform 19](#_Toc372814622)

[References 22](#_Toc372814623)

[Appendix A – *Pralines* Source Code 23](#_Toc372814624)

# Table of Figures

[Figure 1 – Examples of planform geometries supported. 3](#_Toc372814625)

[Figure 2 – Example Menu item 6](#_Toc372814626)

[Figure 3 – Planform Design Menu 7](#_Toc372814627)

[Figure 4 – Wing Type Selection submenu 8](#_Toc372814628)

[Figure 5 – Operating Conditions Menu 10](#_Toc372814629)

[Figure 6 – Planform geometry plot for Example 1 13](#_Toc372814630)

[Figure 7 – Dimensionless Washout Distribution for Example 1 14](#_Toc372814631)

[Figure 8 – Section Lift Distribution for Example 1 14](#_Toc372814632)

[Figure 9 – Normalized Section Lift Coefficient profile for Example 1 14](#_Toc372814633)

[Figure 10 – Example 1 Results 15](#_Toc372814634)

[Figure 11 – Planform geometry plot for Example 2 16](#_Toc372814635)

[Figure 12 – Dimensionless Washout Distribution for Example 2 17](#_Toc372814636)

[Figure 13 – Section Lift Distribution for Example 2 17](#_Toc372814637)

[Figure 14 – Normalized Section Lift Coefficient profile for Example 2 17](#_Toc372814638)

[Figure 15 – Example 2 Results 18](#_Toc372814639)

[Figure 16 – Planform geometry plot for Example 3 19](#_Toc372814640)

[Figure 17 – Dimensionless Washout Distribution for Example 3 20](#_Toc372814641)

[Figure 18 – Section Lift Distribution for Example 3 20](#_Toc372814642)

[Figure 19 – Normalized Section Lift Coefficient profile for Example 3 20](#_Toc372814643)

[Figure 20 – Example 2 Results 21](#_Toc372814644)

# Table of Tables

[Table 1: System Requirements 4](#_Toc372814645)

[Table 2: *Pralines* Installation Inventory 5](#_Toc372814646)

# Introduction

*Pralines* is a numerical analysis tool for examining the aerodynamic characteristics and performance parameters of finite wings in incompressible flow. The physics being modeled are based on Prandtl’s Classical Lifting-Line Theory, as presented in (Phillips, 2010).

## Feature Summary

The following features are included within the software:

1. Planform geometry – Three types of planform geometries are supported: elliptic, tapered, and combination (tapered with elliptic tips). Examples of these planform geometries are shown in Figure 1.
2. Washout / Twist Distribution – Two types of washout distribution are supported: linear and optimum. The optimum washout distribution is the distribution that results in the minimum possible induced drag for a given lift coefficient and aspect ratio. Since elliptic planforms are already optimized through their geometry, the optimum washout distribution option is only available for tapered and combination planforms.
3. Aileron Controls – The definition of a single set of symmetric ailerons along the trailing edge of the planform is supported for all planform geometries. Examples of ailerons can be seen in the planforms shown in Figure 1.

|  |
| --- |
| Elliptic    Tapered (RT = 1.0)    Tapered (RT = 0.5)    Combination (Tapered with Elliptic Tip) |

Figure 1 – Examples of planform geometries supported.

1. Aileron Controls – The definition of a single set of symmetric ailerons along the trailing edge of the planform is supported for all planform geometries. Examples of ailerons can be seen in the planforms shown in Figure 1.
2. Lift and Drag Coefficients – The program calculates total lift and drag coefficients for defined geometry and operating conditions, along with other parameters necessary for the lift and drag coefficient calculations. The coefficients are displayed to the user through the standard console window. The user also has the option to write these parameters to an output file.
3. Post Processing – The program integrates with ES-Plot, a freeware plotting utility written by Dr. Steven Folkman. Plots of planform geometry (such as those displayed in Figure 1), washout distribution, normalized section lift distribution, and normalized section lift coefficient can all be generated from within *Pralines*.

# System Requirements

The minimum system requirements for *Pralines* are listed in the following table. Other configurations have not been tested and may cause unexpected behavior.

Table 1: System Requirements

|  |  |
| --- | --- |
|  | **Supported** |
| Processor | Pentium® 2 GHz or faster |
| Architecture | 32 or 64-bit |
| Operating System | Windows® 7 |
| Memory (Depends on problem size) | 1 GB or more |
| Disk Space (Depends on problem size) | 10 MB or more |
| Documentation | Adobe Reader® XI |
| Post processing | ES-Plot 1.3c |

# Installation

*Pralines* is a stand-alone executable and does not require any customization of the operating environment. To run properly, the executable should be placed in a folder on the computer’s local hard drive where the user has read, write, and execute privileges. Additional template files are necessary if plotting is desired. These template files should be included in the same folder as the main executable.

To enable plotting capabilities within the software, ES-Plot must be installed on the local machine and accessible to the user. ES-Plot is a freeware plotting utility written by Dr. Steven Folkman. The software and installation instructions are included with *Pralines*. They can also be downloaded directly at <http://www.neng.usu.edu/mae/faculty/stevef/prg/ESPlot/>. In order to work properly with *Pralines*, ES-Plot must be installed in the default installation directory (C:\Program Files (x86)\ESPlot v1.3c).

Table 2 gives an inventory of the files that should be included as part of the *Pralines* software package.

Table 2: *Pralines* Installation Inventory

|  |  |
| --- | --- |
| **Item** | **Description** |
| Pralines.exe | Main executable |
| PralinesUser.docx | User documentation |
| planform.qtp | ES-Plot template used for plotting planform geometry |
| washout.qtp | ES-Plot template used for plotting washout distribution |
| liftdistribution.qtp | ES-Plot template used for plotting the dimensionless wing section lift distribution |
| liftcoefficient.qtp | ES-Plot template used for plotting the local section lift coefficient normalized by the wing lift coefficient |
| Problem1p34b\_results.txt | Result file for solver testing |
| Example1\_results.out | Result file for Example 1 |
| Example2\_results.out | Result file for Example 2 |
| Example3\_results.out | Result file for Example 3 |

# Software Overview

*Pralines* is a menu-driven command line program in which selections are made from a list of options to modify parameters, perform calculations, display results, and navigate between menus. *Pralines* can be executed by double-clicking on Pralines.exe within Windows Explorer. This will open a command window in which the software will display output. The user responds to prompts and inputs data by typing into this window.

## Menus

The primary method for gathering input from the user is through menus displayed in the command window. Each menu provides a collection of options the user can select from. The text displayed for each menu option is broken into three parts, as shown in Figure 2: a key, a description, and (if applicable) the current value for the parameter associated with that option. The key is a series of one or two characters. A menu option is invoked when the user types the key associated with that menu option into the command window and presses the **<ENTER>** key. Note that keys are case-insensitive: either “**RA**” or “**ra**” can be used to invoke the option shown in Figure 2. The key and the description are separated by a hyphen. If there is a parameter associated with the menu option, the current value is displayed in parenthesis after the description.

|  |
| --- |
|  |

Figure 2 – Example Menu item

## User Input

After selecting an option from one of the menus, the user will typically be prompted for further input relating to the option selected. For example, typing “**RA**” and pressing **<ENTER>** will display a prompt asking the user to enter a new aspect ratio for the current planform. At this point the user must provide a response, or press **<ENTER>** to accept the current value and return to the previous menu. Only two types of input are used:

### Numerical Input

Numerical parameters can be modified by selecting the appropriate menu item key and, when prompted, typing in a new value. Numerical data in both floating point format and scientific notation can be accepted. In addition, simple formulas containing multiplication and division operations can be included. The keyword “**PI**” can also be used to include the value of π in the input. For example, typing “**3\*PI/4**” at the prompt for aspect ratio will store a value of 2.35619449019234 in the aspect ratio parameter, to an accuracy of 15 significant digits.

If any other form of input is received when prompted for a numerical value, an error message will be displayed and the user will be asked to try again. In most cases, valid numerical inputs are limited to a range of numbers; entering a value outside this range will also cause an error message to be displayed, and the user will be asked to try again. In all cases, pressing the **<ENTER>** key by itself will accept the current value and return control to the previous menu.

### Character Input

Character input is used for specifying the name of the output file for saved data and for selecting options from the menu. As described previously, menu items can be invoked by typing the appropriate key and pressing **<ENTER>**. Menu item keys are case-insensitive. When specifying an output file, no additional checks are made to ensure the input is valid and that the output file can indeed be created and written to. It is up to the user to ensure that this file path is accessible.

# Execution

Execution of the software is broken into two main phases, or loops, each controlled by a menu. The first menu provides the user with options for designing the planform, while the second menu provides the user with options for specifying operating conditions. The following sections describe each of these menus and their various options in detail.

## Planform Design Menu

The Planform Design Menu is shown in Figure 3. This menu is displayed when the program is first launched. It contains options for specifying wing parameters, aileron parameters, and output and plotting options. It also contains three execution commands. A description of each available menu option follows. Note that the menu is customized for the current selection, meaning only options relevant to the current planform are displayed. For example, the wing type selected in Figure 3 is “Tapered”, so options for specifying a transition point and transition chord (relevant only to the “Combination” wing type) are not displayed. On the other hand, an option for specifying taper ratio is displayed.

|  |
| --- |
|  |

Figure 3 – Planform Design Menu

### Wing Parameters

**WT** – Edit the wing type of the planform. Selecting this option brings up a submenu listing the three wing types supported, as shown in Figure 4. Enter the appropriate key from the submenu for the desired wing type.

**N** – Edit the number of nodes per semispan, including the root node. The total number of nodes used in the analysis will be calculated from the formula

. (1)

**RA** – Edit the aspect ratio.

**RT** – Edit the taper ratio of the wing. Available for Tapered planforms only.

**S** – Edit the section lift slope.

**TZ** – Edit z/b at the transition point from tapered to elliptic. Available for Combination planforms only.

**TC** – Edit c/croot at the transition point from tapered to elliptic. Available for Combination planforms only.

**WD** – Toggle the washout distribution type between Linear and Optimum. Available for Tapered and Combination planforms only.

|  |
| --- |
|  |

Figure 4 – Wing Type Selection submenu

### Aileron Parameters

**ZR** – Edit z/b at the aileron root.

**ZT** – Edit z/b at the aileron tip.

**PH** – Toggle the parallel hinge line constraint on or off. If **PH** is set to True, the aileron hinge line will be constrained to be parallel to the planform quarter-chord line, overriding any user-specified value for the section flap fraction (cf/c) at the aileron root. The user-specified value for cf/c at the aileron tip will still be used to determine the location of the hinge line. If **PH** is set to False, the user specified values for cf/c at both the aileron tip and root will be used to define a straight hinge line between these two points on the planform.

**CR** – Edit cf/c at the aileron root. If **PH** is set to True, the value displayed for **CR** will be the value that makes the hinge line parallel with the quarter-chord line. If **PH** is set to False, the value displayed for **CR** will be the user-specified value. If **CR** is selected from the menu, **PH** will automatically be switched to False and the user will be prompted to enter a new value for **CR**.

**CT** – Edit cf/c at the aileron tip.

**HE** – Edit the aileron hinge efficiency.

### Output and Plotting Options

**C** – Toggle whether C matrix, C-1 matrix, and Fourier coefficients are written to the output file.

**F** – Edit the output file name.

**PP** – Plot the planform geometry using ES-Plot. This command generates an ASCII text file named “planform.dat”, containing the coordinates (z/b, c/b) for each node on the planform and the location of each aileron. ES-Plot is then launched and the contents of the file are displayed. The “planform.qtp” template is used to format the ES-Plot display.

### Execution Commands

**A** – Advance to the Operating Conditions Menu. Selecting this option will cause a summary of the planform parameters to be written to the output file, along with intermediate parameters used in calculating the lift, drag, rolling moment, and yawing moment coefficients. In addition, the C matrix, C-1 matrix, and Fourier coefficients will be calculated and (if the **C** option is set to True) written to the output file. Execution control will then be passed to the Operating Conditions Menu. Note that, when writing the planform design summary to the output file, any data previously written to that output file will be deleted.

**T** – Test the *Pralines* solver against the solution to Problem 1.34b from (Phillips, 2010). Selecting this option will run an automated test and compare the output (“Problem1p34b\_work.txt”) to the expected results (“Problem1p34b\_results.txt”). A report is displayed to the user indicating whether the test passed successfully or failed.

**Q** – Quit. This command ends execution of the *Pralines* software. Any data not saved to the output file when this command is executed will be lost, and all parameters will be reset to their original default values the next time the software is launched.

## Operating Conditions Menu

The Operating Conditions Menu is shown in Figure 5. This menu is displayed when option **A** is selected from the Planform Design Menu. When the Operating Conditions Menu is displayed, a summary of the current planform is also included. This summary lists all of the planform parameters specified in the Planform Design Menu, along with intermediate parameters used for calculating lift, drag, rolling moment, and yawing moment coefficients. After the planform design summary, a summary of the current operating conditions is displayed, along with the lift, drag, rolling moment, and yawing moment coefficients resulting from the current operating conditions. Note that the planform design summary will have been written to the output file at this point, but the operating conditions summary will not. The operating conditions summary is only written to the output file when the user makes an explicit request by selecting the **S** option (see below).

### Operating Conditions

**AA** – Edit the root aerodynamic angle of attack. The root aerodynamic angle of attack is related to the lift coefficient and washout through the following relationship:

, (2)

where is the wing lift coefficient, is the section lift slope, is the root aerodynamic angle of attack, is the washout effectiveness, and is the washout. The section lift slope and washout effectiveness are fixed by the planform design. By default, the root aerodynamic angle of attack is also fixed and is used to calculate the wing lift coefficient. If a desired lift coefficient is specified using option **CL** from the menu, the user-specified angle of

|  |
| --- |
|  |

Figure 5 – Operating Conditions Menu

attack will be disabled and a new angle of attack will be calculated from Equation 2 using the desired lift coefficient.

**CL** – Edit the coefficient of lift. As described in the previous section, the root aerodynamic angle of attack is fixed by default and the value displayed for is obtained by calculation. Changing the angle of attack will cause the lift coefficient to be recalculated automatically using Equation 2. On the other hand, selecting option **CL** from the Operating Conditions Menu will cause to become fixed, and Equation 2 will be used to calculate a new angle of attack.

**OW** – Toggle optimum washout on or off. If **OW** is set to True, the optimum total washout is calculated from the relationship:

, (3)

where is the lift-washout contribution to induced drag and is the washout contribution to induced drag. Note that both Equations 2 and 3 include dependencies on and . Therefore, if the root aerodynamic angle of attack is fixed and optimum total washout is specified, Equations 2 and 3 must be solved simultaneously to obtain values for and .

**W** – Edit the total washout to be used, specified in degrees. If **OW** is set to True, the optimum washout will be displayed here and used in the calculations. Otherwise, the user-specified washout will be displayed and used. If the **W** option is selected from the menu, **OW** will automatically be switched to False and the user will be prompted to enter a new value for **W**.

**AD** – Edit the aileron deflection, specified in degrees.

**SR** – Toggle Steady Dimensionless Rolling Rate on or off. If set to True, the dimensionless rolling rate is calculated from the relationship:

(4)

where is the dimensionless rolling rate, is the change in rolling moment coefficient with respect to aileron deflection, is the change in rolling moment coefficient with respect to dimensionless rolling rate, and is the aileron deflection.

**R** – Edit the dimensionless rolling rate. If **SR** is set to True, the steady dimensionless rolling rate is displayed here and used in the calculations. If **SR** is set to False, the user-specified dimensionless rolling rate will be displayed and used. If the **R** option is selected from the menu, **SR** will automatically be switched to False and the user will be prompted to enter a new value for **R**.

### Plotting Options

**PP** – Plot the planform geometry using ES-Plot. This command generates an ASCII text file named “planform.dat”, containing the coordinates (z/b, c/b) for each node on the planform and the location of each aileron. ES-Plot is then launched and the contents of the file are displayed. The “planform.qtp” template is used to format the ES-Plot display.

**PW** – Plot the dimensionless washout distribution using ES-Plot. This command generates an ASCII text file named “washout.dat”, containing the washout distribution ω at each node on the planform. ES-Plot is then launched and the contents of the file are displayed. The “washout.qtp” template is used to format the ES-Plot display.

**PL** – Plot the dimensionless wing section lift distribution using ES-Plot. This command generates an ASCII text file named “liftdistribution.dat”, containing the dimensionless wing section lift coefficient for each node on the planform. The dimensionless wing section lift distribution is defined by the following relationship:

(5)

ES-Plot is then launched and the contents of the file are displayed. The “liftdistribution.qtp” template is used to format the ES-Plot display.

**PN** – Plot the local section lift coefficient divided by the wing lift coefficient using ES-Plot. This command generates an ASCII text file named “liftcoefficient.dat”, containing the local section lift coefficient calculated at each node, normalized by the wing lift coefficient. The local section lift coefficient is defined by the following relationship:

(6)

ES-Plot is then launched and the contents of the file are displayed. The “liftcoefficient.qtp” template is used to format the ES-Plot display.

### Execution Commands

**S** – Save the current operating conditions and flight coefficients to the output file. When this option is selected, a summary of the current operating conditions is appended to the end of the output file specified on the Planform Design Menu, followed by the corresponding lift, induced drag, rolling moment, and yawing moment coefficients. Note that, because the data is appended to the file, the planform design summary as well as any previously-saved operating condition summaries will still be maintained within the output file.

Advance to the Operating Conditions Menu. Selecting this option will cause a summary of the planform parameters to be written to the output file, along with intermediate parameters used in calculating the lift, drag, rolling moment, and yawing moment coefficients. In addition, the C matrix, C-1 matrix, and Fourier coefficients will be calculated and (if the **C** option is set to True) written to the output file. Execution control will then be passed to the Operating Conditions Menu.

**B** – Go back to the Planform Design Menu. Selecting this option will return control to the Planform Design Menu. Note that operating condition summaries will not be written to the output file unless explicitly requested by the user from the Operating Conditions Menu.

**Q** – Quit. This command ends execution of the *Pralines* software. Any data not saved to the output file when this command is executed will be lost, and all parameters will be reset to their original default values the next time the software is launched.

# Examples

This section contains step-by-step examples of the solution process for evaluating flight conditions on each of the three planform geometries supported within *Pralines*.

## Example 1: Tapered Planform

1. Execute a new instance of *Pralines*. The Planform Design Menu should be displayed, with identical parameters to those shown in Figure 3.
2. Select option **RT** and set the taper ratio to **1/2**
3. Select option **WD**; this will automatically change the washout distribution type to **OPTIMUM**
4. Select option **C** to disable output of the C matrix, C-1 matrix, and Fourier coefficients.
5. Select option **F** and enter **Example1.out** for the output file name
6. Select option **PP** to view the planform design in ES-Plot. The planform geometry should match the plot shown in Figure 6. When you are finished viewing the planform geometry, exit ES-Plot to return control to the Planform Design Menu.
7. Select option **A** to advance to the Operating Conditions Menu. Once this selection is made, the file **Example1.out** will be created and contain the planform summary. Note that the C matrix, C-1 matrix, and Fourier coefficients will not be saved (see step 5).
8. Select option **CL** and press **<ENTER>** to accept the default value of 0.4. This will cause the root aerodynamic angle of attack and optimum washout to be recalculated.
9. Select option **AD** and set the aileron deflection to **3** degrees. This will cause the steady dimensionless rolling rate to be recalculated.
10. Select option **PW** to view the dimensionless washout distribution in ES-Plot.
11. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The washout distribution should match the plot shown in Figure 7. When you are finished viewing the washout distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.

|  |
| --- |
|  |

Figure 6 – Planform geometry plot for Example 1

|  |
| --- |
|  |

Figure 7 – Dimensionless Washout Distribution for Example 1

1. Select option **PL** to view the section lift distribution in ES-Plot.
2. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The section lift distribution should match the plot shown in Figure 8. When you are finished viewing the section lift distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.
3. Select option **PN** to view the normalized section lift coefficient in ES-Plot.
4. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The normalized section lift coefficient profile should match the plot shown in Figure 9. When you are finished viewing the normalized section lift coefficient plot, exit ES-Plot to return control to the Operating Conditions Menu.
5. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example1.out**.
6. Select option **CL** and set the lift coefficient to **0.5**. This will cause the root aerodynamic angle of attack and optimum washout to be recalculated.
7. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example1.out**.
8. Select option **Q** to exit the program.
9. Compare the contents of **Example1.out** with the results contained in Figure 10 and **Example1\_results.out**. The output should be identical.

|  |
| --- |
|  |

Figure 8 – Section Lift Distribution for Example 1

|  |
| --- |
|  |

Figure 9 – Normalized Section Lift Coefficient profile for Example 1

|  |
| --- |
| Planform Summary:  Wing type = Tapered  Number of nodes = 99 (50 nodes per semispan)  Airfoil section lift slope = 6.283185307179586  Aspect Ratio = 5.560000000000000  Taper Ratio = 0.500000000000000  Washout Distribution = Optimum  z/b at aileron root = 0.253000000000000  z/b at aileron tip = 0.438000000000000  cf/c at aileron root = 0.373828647925033  cf/c at aileron tip = 0.250000000000000  Hinge Efficiency = 0.850000000000000  Deflection Efficiency = 1.000000000000000  Lift Coefficient Parameters:  KL = 0.012656624734665  CL,a = 4.563212108702364  EW = -0.065219158642620  Drag Coefficient Parameters:  KD = 0.010492888814473  KDL = 0.030259580066700  KDW = 0.021815779286399  es = 0.989616068622924  Rolling Moment Coefficient Parameters:  Cl,da = -0.239513350217710  Cl,pb = -0.470979956137708  Optimum washout = 3.483165718785543 degrees (Eq. 1.8.37)  Optimum washout = 3.483165718785545 degrees (Eq. 1.8.42)  Washout used in calculations = 3.483165718785543 degrees  Aileron deflection = 3.000000000000000 degrees  Steady dimensionless rolling rate = -0.026627225910389  Dimensionless rolling rate used = -0.026627225910389  Root aerodynamic angle of attack = 4.795238600484883 degrees  Flight Coefficients:  CL = 0.400000000000000 (Eq. 1.8.24)  CL = 0.400000000000000 (Eq. 1.8.5)  CDi = 0.009159996724713 (Eq. 1.8.25)  CDi = 0.009309619640318 (Eq. 1.8.6)  CDi = 0.009309619640318 (Exact)  Croll = -0.000000000000000  Cyaw = 0.001331361295519  Optimum washout = 4.353957148481928 degrees (Eq. 1.8.37)  Optimum washout = 4.353957148481931 degrees (Eq. 1.8.42)  Washout used in calculations = 4.353957148481928 degrees  Aileron deflection = 3.000000000000000 degrees  Steady dimensionless rolling rate = -0.026627225910389  Dimensionless rolling rate used = -0.026627225910389  Root aerodynamic angle of attack = 5.994048250606104 degrees  Flight Coefficients:  CL = 0.500000000000000 (Eq. 1.8.24)  CL = 0.500000000000000 (Eq. 1.8.5)  CDi = 0.014312494882365 (Eq. 1.8.25)  CDi = 0.014462117797969 (Eq. 1.8.6)  CDi = 0.014462117797969 (Exact)  Croll = -0.000000000000000  Cyaw = 0.001664201619399 |

Figure 10 – Example 1 Results

## Example 2: Elliptic Planform

1. Execute a new instance of *Pralines*. The Planform Design Menu should be displayed, with identical parameters to those shown in Figure 3.
2. Select option **WT**. From the Wing Type Menu, select option **E** to create an Elliptic planform design.
3. Select option **CR** and press **<ENTER>** to accept the default value of 0.28. This will also disable the **PH** option so that the hinge line is no longer constrained to be parallel with the quarter-chord line.
4. Select option **C** to disable output of the C matrix, C-1 matrix, and Fourier coefficients.
5. Select option **F** and enter **Example2.out** for the output file name
6. Select option **PP** to view the planform design in ES-Plot. The planform geometry should match the plot shown in Figure 11. When you are finished viewing the planform geometry, exit ES-Plot to return control to the Planform Design Menu.
7. Select option **A** to advance to the Operating Conditions Menu. Once this selection is made, the file **Example2.out** will be created and contain the planform summary. Note that the C matrix, C-1 matrix, and Fourier coefficients will not be saved (see step 5).
8. Select option **CL** and set the lift coefficient to **1/2**. This will also cause the root aerodynamic angle of attack and optimum washout to be recalculated.
9. Select option **W** and set the total amount of washout to **2** degrees. This will cause the root aerodynamic angle of attack to be recalculated.
10. Select option **AD** and set the aileron deflection to **5** degrees. This will cause the steady dimensionless rolling rate to be recalculated.
11. Select option **PW** to view the dimensionless washout distribution in ES-Plot.
12. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The washout distribution should match the plot shown in Figure 12. When you are finished viewing the washout distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.

|  |
| --- |
|  |

Figure 11 – Planform geometry plot for Example 2

|  |
| --- |
|  |

Figure 12 – Dimensionless Washout Distribution for Example 2

1. Select option **PL** to view the section lift distribution in ES-Plot.
2. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The section lift distribution should match the plot shown in Figure 13. When you are finished viewing the section lift distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.
3. Select option **PN** to view the normalized section lift coefficient in ES-Plot.
4. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The normalized section lift coefficient profile should match the plot shown in Figure 14. When you are finished viewing the normalized section lift coefficient plot, exit ES-Plot to return control to the Operating Conditions Menu.
5. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example2.out**.
6. Select option **R** and set the dimensionless rolling rate to **-0.2**. This will also disable the **SR** option so that the steady dimensionless rolling rate is no longer used.
7. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example2.out**.
8. Select option **Q** to exit the program.
9. Compare the contents of **Example2.out** with the results contained in Figure 15 and **Example2\_results.out**. The output should be identical.

|  |
| --- |
|  |

Figure 13 – Section Lift Distribution for Example 2

|  |
| --- |
|  |

Figure 14 – Normalized Section Lift Coefficient profile for Example 2

|  |
| --- |
| Planform Summary:  Wing type = Elliptic  Number of nodes = 99 (50 nodes per semispan)  Airfoil section lift slope = 6.283185307179586  Aspect Ratio = 5.560000000000000  z/b at aileron root = 0.253000000000000  z/b at aileron tip = 0.438000000000000  cf/c at aileron root = 0.280000000000000  cf/c at aileron tip = 0.250000000000000  Hinge Efficiency = 0.850000000000000  Deflection Efficiency = 1.000000000000000  Lift Coefficient Parameters:  KL = -0.000000000000000  CL,a = 4.620966971946893  EW = 0.424304130856835  Drag Coefficient Parameters:  KD = 0.000000000000000  KDL = -0.000000000000000  KDW = 0.088934224521291  es = 1.000000000000000  Rolling Moment Coefficient Parameters:  Cl,da = -0.230298562976715  Cl,pb = -0.456779684988474  Optimum washout = -0.000000000000001 degrees (Eq. 1.8.37)  Washout used in calculations = 2.000000000000000 degrees  Aileron deflection = 5.000000000000000 degrees  Steady dimensionless rolling rate = -0.043997886931603  Dimensionless rolling rate used = -0.043997886931603  Root aerodynamic angle of attack = 7.048152627743041 degrees  Flight Coefficients:  CL = 0.500000000000000 (Eq. 1.8.24)  CL = 0.500000000000000 (Eq. 1.8.5)  CDi = 0.014444967071628 (Eq. 1.8.25)  CDi = 0.014798217233319 (Eq. 1.8.6)  CDi = 0.014798217233319 (Exact)  Croll = 0.000000000000000  Cyaw = 0.002595346795979  Optimum washout = -0.000000000000001 degrees (Eq. 1.8.37)  Washout used in calculations = 2.000000000000000 degrees  Aileron deflection = 5.000000000000000 degrees  Steady dimensionless rolling rate = -0.043997886931603  Dimensionless rolling rate used = -0.200000000000000  Root aerodynamic angle of attack = 7.048152627743041 degrees  Flight Coefficients:  CL = 0.500000000000000 (Eq. 1.8.24)  CL = 0.500000000000000 (Eq. 1.8.5)  CDi = 0.014444967071628 (Eq. 1.8.25)  CDi = 0.024100720631155 (Eq. 1.8.6)  CDi = -0.004402717794812 (Exact)  Croll = 0.071258596064919  Cyaw = 0.006250268463806 |

Figure 15 – Example 2 Results

## Example 3: Combination Planform

1. Execute a new instance of *Pralines*. The Planform Design Menu should be displayed, with identical parameters to those shown in Figure 3.
2. Select option **WT**. From the Wing Type Menu, select option **C** to create a Combination (tapered with elliptic tip) planform design.
3. Select option **TZ** and set the transition point z/b to **0.35**.
4. Select option **TC** and set the chord ratio c/croot at the transition point to **2/3**.
5. Select option **WD**; this will automatically change the washout distribution type to **OPTIMUM**
6. Select option **C** to disable output of the C matrix, C-1 matrix, and Fourier coefficients.
7. Select option **F** and enter **Example3.out** for the output file name
8. Select option **PP** to view the planform design in ES-Plot. The planform geometry should match the plot shown in Figure 16. When you are finished viewing the planform geometry, exit ES-Plot to return control to the Planform Design Menu.
9. Select option **A** to advance to the Operating Conditions Menu. Once this selection is made, the file **Example3.out** will be created and contain the planform summary. Note that the C matrix, C-1 matrix, and Fourier coefficients will not be saved (see step 5).
10. Select option **AA** and set the root aerodynamic angle of attack to **0**. This will also cause the lift coefficient and optimum washout to be recalculated.
11. Select option **W** and set the total amount of washout to -**2** degrees. This will cause the lift coefficient to be recalculated.
12. Select option **PW** to view the dimensionless washout distribution in ES-Plot.
13. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The washout distribution should match the plot shown in Figure 17. When you are finished viewing the washout distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.

|  |
| --- |
|  |

Figure 16 – Planform geometry plot for Example 3

|  |
| --- |
|  |

Figure 17 – Dimensionless Washout Distribution for Example 3

1. Select option **PL** to view the section lift distribution in ES-Plot.
2. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The section lift distribution should match the plot shown in Figure 18. When you are finished viewing the section lift distribution plot, exit ES-Plot to return control to the Operating Conditions Menu.
3. Select option **PN** to view the normalized section lift coefficient in ES-Plot.
4. In ES-Plot, select the “Reset X and Y Axes to view all data” button (). The normalized section lift coefficient profile should match the plot shown in Figure 19. When you are finished viewing the normalized section lift coefficient plot, exit ES-Plot to return control to the Operating Conditions Menu.
5. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example3.out**.
6. Select option **AA** and set the root aerodynamic angle of attack to **2.0**. This will cause the lift coefficient and optimum washout to be recalculated.
7. Select option **S** to save a summary of the current operating conditions and flight coefficients to **Example3.out**.
8. Compare the contents of **Example3.out** with the results contained in Figure 20 and **Example3\_results.out**. The output should be identical.

|  |
| --- |
|  |

Figure 18 – Section Lift Distribution for Example 3

|  |
| --- |
|  |

Figure 19 – Normalized Section Lift Coefficient profile for Example 3

|  |
| --- |
| Planform Summary:  Wing type = Tapered with elliptic tip  Number of nodes = 99 (50 nodes per semispan)  Airfoil section lift slope = 6.283185307179586  Aspect Ratio = 5.560000000000000  Transition Point z/b = 0.350000000000000  Transition Point c/c = 0.666666666666667  Washout Distribution = Optimum  z/b at aileron root = 0.253000000000000  z/b at aileron tip = 0.438000000000000  cf/c at aileron root = 0.423758432235187  cf/c at aileron tip = 0.250000000000000  Hinge Efficiency = 0.850000000000000  Deflection Efficiency = 1.000000000000000  Lift Coefficient Parameters:  KL = 0.001178266114320  CL,a = 4.615528650937818  EW = -0.076357684216684  Drag Coefficient Parameters:  KD = 0.001595552910306  KDL = 0.004668324226696  KDW = 0.003414686367463  es = 0.998406988823313  Rolling Moment Coefficient Parameters:  Cl,da = -0.256018648559314  Cl,pb = -0.462284274859193  Optimum washout = 0.000000000000000 degrees (Eq. 1.8.37)  Optimum washout = 0.000000000000000 degrees (Eq. 1.8.42)  Washout used in calculations = -2.000000000000000 degrees  Aileron deflection = 0.000000000000000 degrees  Steady dimensionless rolling rate = -0.000000000000000  Dimensionless rolling rate used = -0.000000000000000  Root aerodynamic angle of attack = 0.000000000000000 degrees  Flight Coefficients:  CL = -0.012302165437540 (Eq. 1.8.24)  CL = -0.012302165437540 (Eq. 1.8.5)  CDi = 0.000013222892772 (Eq. 1.8.25)  CDi = 0.000013222892772 (Eq. 1.8.6)  CDi = 0.000013222892772 (Exact)  Croll = 0.000000000000000  Cyaw = -0.000000000000000  Optimum washout = 0.000000000000000 degrees (Eq. 1.8.37)  Optimum washout = 0.000000000000000 degrees (Eq. 1.8.42)  Washout used in calculations = -2.000000000000000 degrees  Aileron deflection = 0.000000000000000 degrees  Steady dimensionless rolling rate = -0.000000000000000  Dimensionless rolling rate used = -0.000000000000000  Root aerodynamic angle of attack = 2.000000000000000 degrees  Flight Coefficients:  CL = 0.148810177920454 (Eq. 1.8.24)  CL = 0.148810177920454 (Eq. 1.8.5)  CDi = 0.001281275219520 (Eq. 1.8.25)  CDi = 0.001281275219520 (Eq. 1.8.6)  CDi = 0.001281275219520 (Exact)  Croll = 0.000000000000000  Cyaw = -0.000000000000000 |

Figure 20 – Example 3 Results

# References

Phillips, W. F. (2010). *Mechanics of Flight* (2nd ed.). Hoboken, NJ: John Wiley & Sons, Inc.

# Appendix A – *Pralines* Source Code

module Utilities

    implicit none

    real\*8, parameter :: pi = acos(-1.0d0)

    real\*8, parameter :: zero = 1.0d-10

contains

    integer function Compare(a, b, tol) result(eq)

    ! Comparison function

    ! Inputs:

    !   a = First argument to compare

    !   b = Second argument to compare

    !   tol = Relative tolerance for comparison

    ! Return Value (eq):

    !   -1 = a < (b - tol)

    !    0 = a == b (within tolerance)

    !    1 = a > (b + tol)

        real\*8, intent(in) :: a, b, tol

        if (abs(a) < tol .and. abs(b) < tol) then

            eq = 0

        else if (abs(a - b) / max(abs(a), abs(b)) < tol) then

            eq = 0

        else if (a < b) then

            eq = -1

        else

            eq = 1

        end if

    end function Compare

    real\*8 function Residual(oldVal, newVal) result(res)

        real\*8, intent(in) :: oldVal

        real\*8, intent(in) :: newVal

        res = dabs(oldVal - newVal) / max(dabs(oldVal), dabs(newVal), zero)

    end function Residual

    integer function CompareFiles(a, b) result(badline)

        character\*80, intent(in) :: a, b  ! Filenames of files to compare

        integer :: i, ios1, ios2

        character\*5000 :: results\_line, work\_line

        open(unit=11, file=a)

        open(unit=12, file=b)

        badline = 0

        ios1 = 0

        i = 0

        do while (badline == 0 .and. ios1 == 0)

            i = i + 1

            results\_line(1:5000) = " "

            read(11, '(A)', iostat=ios1, end=99) results\_line

            work\_line(1:5000) = " "

            read(12, '(A)', iostat=ios2, end=99) work\_line

            if (ios1 == 0) then

                if (work\_line /= results\_line) then

                    badline = i

                end if

            else if (len(trim(work\_line)) /= 0) then

                badline = i

            end if

        end do

        close(unit=11)

        close(unit=12)

99  continue

    end function CompareFiles

    character\*2 function GetCharacterInput(def) result(inp)

        character\*2, intent(in) :: def  ! Default value if invalid input

        integer :: i

        character :: a

        read(5, '(a)') inp

        if (len(inp) > 2 .or. len(inp) < 1) then

            inp = def

        else

            do i = 1, 2

                a = inp(i:i)

                if(iachar(a) >= iachar('a') .and. iachar(a) <= iachar('z')) then

                    inp(i:i) = char(iachar(a) - 32)

                end if

            end do

        end if

    end function GetCharacterInput

    character\*80 function GetStringInput(def) result(inp)

        character\*80, intent(in) :: def  ! Default value if invalid input

        integer :: i, ios

        character :: a

        read(5, '(a)', iostat=ios) inp

        if (len(trim(inp)) < 1 .or. ios /= 0) then

            inp = def

        end if

    end function GetStringInput

    integer function GetIntInput(mn, mx, def) result(inp)

        integer, intent(in) :: mn   ! Minimum accepted value

        integer, intent(in) :: mx   ! Maximum accepted value

        integer, intent(in) :: def  ! Default value if invalid input

        logical :: cont

        character\*80 :: inp\_str

        integer :: ios

        integer :: len\_mn, len\_mx

        character\*80 :: msg\_fmt

        cont = .true.

        do while (cont)

            read(5, '(a)', iostat=ios) inp\_str

            if (ios == 0 .and. trim(inp\_str) /= "") then

                read(inp\_str, \*, iostat=ios) inp

                if (ios /= 0 .or. inp < mn .or. inp > mx) then

                    len\_mn = int(log10(real(abs(mn)))) + 1

                    if (mn < 0) len\_mn = len\_mn + 1

                    len\_mx = int(log10(real(abs(mx)))) + 1

                    if (mx < 0) len\_mx = len\_mx + 1

                    write(msg\_fmt, '(a, i1, a, i1, a)') "(a, a, i", len\_mn, &

                        & ", a, i", len\_mx, ", a)"

                    write(6, \*)

                    write(6, msg\_fmt) "Invalid input. Please ", &

                        & "specify an integer between ", mn, " and ", mx, ","

                    write(6, '(a)') "or press <ENTER> to accept the default value."

                else

                    cont = .false.

                end if

            else

                inp = def

                cont = .false.

            end if

        end do

    end function GetIntInput

    real\*8 function GetRealInput(mn\_orig, mx\_orig, dflt\_orig) result(inp)

        real\*8, intent(in) :: mn\_orig   ! Minimum accepted value

        real\*8, intent(in) :: mx\_orig   ! Maximum accepted value

        real\*8, intent(in) :: dflt\_orig ! Default value for input

        logical :: cont

        character\*80 :: inp\_str

        integer :: ios

        integer :: len\_mn, ndec\_mn

        integer :: len\_mx, ndec\_mx

        character\*80 :: msg\_fmt

        real\*8 :: mn, mx, dflt

        if (Compare(mn\_orig, 0.0d0, zero) == 0) then

            mn = 0.0d0

        else

            mn = mn\_orig

        end if

        if (Compare(mx\_orig, 0.0d0, zero) == 0) then

            mx = 0.0d0

        else

            mx = mx\_orig

        end if

        if (Compare(dflt\_orig, 0.0d0, zero) == 0) then

            dflt = 0.0d0

        else

            dflt = dflt\_orig

        end if

        cont = .true.

        do while (cont)

            read(5, '(a)', iostat=ios) inp\_str

            if (ios == 0 .and. trim(inp\_str) /= "") then

                ios = ParseFormula(trim(inp\_str), inp)

                if (ios /= 0 .or. inp < mn .or. inp > mx) then

                    write(6, \*)

                    write(6, '(a, a, a, a, a, a)') "Invalid input. Please ", &

                        & "enter a number between ", trim(FormatReal(mn, 5)), &

                        & " and ", trim(FormatReal(mx, 5)), ","

                    write(6, '(a)') "or press <ENTER> to accept the default value."

                else

                    cont = .false.

                end if

            else

                inp = dflt

                cont = .false.

            end if

        end do

    end function GetRealInput

    integer function ParseFormula(inp\_str, num) result(estat)

        character\*80, intent(in) :: inp\_str

        real\*8, intent(out) :: num

        integer :: i, j, n\_oper, last\_ind, strlen, ios

        character\*40 :: operators, temp\_num

        real\*8, Dimension(41) :: numbers

        estat = 0

        strlen = len(trim(inp\_str))

        n\_oper = 0

        last\_ind = 0

        do i = 2, strlen

            if (inp\_str(i:i) == '\*' .or. inp\_str(i:i) == '/') then

                n\_oper = n\_oper + 1

                operators(n\_oper:n\_oper) = inp\_str(i:i)

                temp\_num = "                                        "

                temp\_num(1:i-last\_ind-1) = inp\_str(last\_ind+1:i-1)

                if ((temp\_num(1:1) == 'P' .or. temp\_num(1:1) == 'p') .and. &

                    & (temp\_num(2:2) == 'I' .or. temp\_num(2:2) == 'i')) then

                    numbers(n\_oper) = pi

                else

                    read(temp\_num, \*, iostat=ios) numbers(n\_oper)

                    if (ios /= 0) then

                        estat = 1

                    end if

                end if

                last\_ind = i

            end if

        end do

        temp\_num = "                                        "

        temp\_num(1:strlen-last\_ind) = inp\_str(last\_ind+1:strlen)

        if ((temp\_num(1:1) == 'P' .or. temp\_num(1:1) == 'p') .and. &

            & (temp\_num(2:2) == 'I' .or. temp\_num(2:2) == 'i')) then

            numbers(n\_oper + 1) = pi

        else

            read(temp\_num, \*, iostat = ios) numbers(n\_oper + 1)

            if (ios /= 0) then

                estat = 1

            end if

        end if

        num = numbers(1)

        do i = 1, n\_oper

            if (operators(i:i) == '\*') then

                num = num \* numbers(i + 1)

            else if (operators(i:i) == '/') then

                num = num / numbers(i + 1)

            else

                estat = 2

            end if

        end do

    end function ParseFormula

    character\*80 function FormatReal(r, ndigits) result(real\_str)

        real\*8, intent(in) :: r

        integer, intent(in) :: ndigits

        integer :: order, width, ndecimal

        character\*80 :: real\_fmt

        real\*8 :: r\_div\_pi

        integer :: num, denom

        if (Compare(r, 0.0d0, zero) /= 0 .and. (IsFactorOfPi(r, ndigits) &

            & .or. IsFractionOfPi(r, num, denom))) then

            if (Compare(r, pi, zero) == 0) then

                write(real\_str, '(a)') "PI"

            else if (IsFractionOfPi(r, num, denom)) then

                if (denom == 1) then

                    write(real\_str, '(a, a)') trim(FormatInteger(num)), "\*PI"

                else

                    write(real\_str, '(a, a, a, a)') trim(FormatInteger(num)), &

                        & "/", trim(FormatInteger(denom)), "\*PI"

                end if

            else

                r\_div\_pi = r / pi

                write(real\_str, '(a, a)') trim(FormatReal(r\_div\_pi, ndigits)), &

                    & "\*PI )"

            end if

        else

            if (Compare(r, 0.0d0, zero) == 0) then

                order = 1

            else

                ! Determine the location of the first non-zero digit in the number

                order = int(log10(real(abs(r), 8))) + 1

            end if

            ! Check for sizes that should use exponential format

            if (order <= -4 .or. order >= ndigits) then

                if (r < 0.0d0) then

                    width = ndigits + 6  ! e.g. -1.2345E+67 - 5 digits + 6 other

                else

                    width = ndigits + 5  ! e.g. 1.2345E-67 - 5 digits + 5 other

                end if

                write(real\_fmt, '(a, i2, a, i2, a)') "(ES", width, ".", &

                    & ndigits - 1, ")"

            else

                if (r < 0.0d0) then

                    width = ndigits + 2  ! e.g. -12.345 - 5 digits + 2 other

                else

                    width = ndigits + 1  ! e.g. 123.45 - 5 digits + 1 other

                end if

                if (order <= 0) then

                    width = width - order + 1  ! e.g. -0.012345 - additional for leading 0

                end if

                write(real\_fmt, '(a, i2, a, i2, a)') "(F", width, ".", &

                    & ndigits - order, ")"

            end if

            write(real\_str, real\_fmt) r

        end if

    end function FormatReal

    character\*80 function FormatInteger(i) result(int\_str)

        integer, intent(in) :: i

        integer :: len\_i

        character\*80 :: int\_fmt

        if (i == 0) then

            len\_i = 1

        else

            len\_i = int(log10(real(abs(i)))) + 1

            if (i < 0) then

                len\_i = len\_i + 1

            end if

        end if

        write(int\_fmt, '(a, i2, a)') "(i", len\_i, ")"

        write(int\_str, int\_fmt) i

    end function FormatInteger

    logical function IsFactorOfPi(r, ndigits) result(isFactor)

        real\*8, intent(in) :: r

        integer, intent(in) :: ndigits

        real\*8 :: rx, rx\_trunc

        rx = r / pi \* 10\*\*ndigits

        rx\_trunc = real(int(rx), 8)

        if (Compare(rx, rx\_trunc, zero) == 0) then

            isFactor = .true.

        else

            isFactor = .false.

        end if

    end function IsFactorOfPi

    logical function IsFractionOfPi(r, num, denom) result(isFraction)

        real\*8, intent(in) :: r

        integer, intent(out) :: num

        integer, intent(out) :: denom

        integer :: i, num2, denom2

        real\*8 :: r\_div\_pi, r\_div\_pi\_i

        isFraction = .false.

        r\_div\_pi = r / pi

        do i = 1, 360

            r\_div\_pi\_i = r\_div\_pi \* real(i, 8)

            if (Compare(r\_div\_pi\_i , real(int(r\_div\_pi\_i), 8), zero) == 0) then

                num = int(r\_div\_pi\_i)

                denom = i

                isFraction = .true.

                return

            end if

        end do

    end function IsFractionOfPi

end module Utilities

module class\_Planform

    use Utilities

    implicit none

    public :: Planform

    ! Supported wing types

    enum, bind(C)

        enumerator :: Tapered = 1, Elliptic = 2, Combination = 3

    end enum

    ! Supported washout distribution types

    enum, bind(C)

        enumerator :: Linear = 1, Optimum = 2

    end enum

    type Planform

        ! Wing Parameters

        integer :: WingType = Tapered ! Wing type

        integer :: WashoutDistribution = Linear ! Washout distribution type

        integer :: NNodes = 99 ! Total number of nodes

        real\*8 :: AspectRatio = 5.56d0 ! Aspect ratio

        real\*8 :: TaperRatio = 1.0d0 ! Taper ratio (tapered wing only)

        real\*8 :: TransitionPoint = 0.25d0 ! Transition point (Combination wing only)

        real\*8 :: TransitionChord = 1.0d0 ! c/croot at transtion point (Combination wing only)

        real\*8 :: SectionLiftSlope = 2.0d0 \* pi ! Section lift slope

        real\*8 :: AileronRoot = 0.253d0 ! Location of aileron root (z/b)

        real\*8 :: AileronTip = 0.438d0 ! Location of aileron tip (z/b)

        logical :: ParallelHingeLine = .true. ! Is the hinge line parallel to the

                                              ! quarter-chord line? When true,

                                              ! FlapFractionTip will be calculated

        real\*8 :: DesiredFlapFractionRoot = 0.28d0 ! Desired flap fraction at aileron root (cf/c)

        real\*8 :: FlapFractionRoot = 0.28d0 ! Flap fraction at aileron root (cf/c)

        real\*8 :: FlapFractionTip = 0.25d0 ! Flap fraction at aileron tip (cf/c)

        real\*8 :: HingeEfficiency = 0.85d0 ! Aileron hinge efficiency

        real\*8 :: DeflectionEfficiency = 1.0d0 ! Aileron deflection efficiency

        ! Coefficients for Tapered wing with elliptic tip

        real\*8 :: C1 = 0.0d0 ! Represents transition point

        real\*8 :: C2 = 0.0d0 ! Represents slope of tapered section

        real\*8 :: C3 = 0.0d0 ! Represents secondary axis of ellipse

        real\*8 :: C4 = 0.0d0 ! Represents ellipse center offset

        real\*8 :: C5 = 0.0d0 ! Represents croot/b

        ! Output Options

        logical :: OutputMatrices = .true.  ! Write C Matrix and Fourier coefficients to output file?

        character\*80 :: FileName = "Planform.out" ! Name of output file

        ! Operating Conditions

        real\*8 :: DesiredAngleOfAttack = pi / 36.0d0 ! Desired root aerodynamic angle of Attack

                                                     ! (alpha - alpha\_L0), in radians

                                                     ! When specified, a new LiftCoefficient is calculated

        real\*8 :: AngleOfAttack = pi / 36.0d0 ! Root Aerodynamic Angle of Attack

                                              ! (alpha - alpha\_L0), in radians

        real\*8 :: DesiredLiftCoefficient = 0.4d0 ! Desired lift coefficient

                                                 ! When specified, a new AngleOfAttack is calculated

        real\*8 :: LiftCoefficient = 0.4d0 ! Lift coefficient (user input, ignored if SpecifyAlpha == .true.)

        real\*8 :: DesiredWashout = 0.0d0 ! Desired total washout, in radians

        real\*8 :: OptimumWashout1 = 0.0d0 ! Optimum total washout, in radians (Eq. 1.8.37)

        real\*8 :: OptimumWashout2 = 0.0d0 ! Optimum total washout, in radians (Eq. 1.8.42)

        real\*8 :: Washout = 0.0d0 ! Total washout to use

        logical :: UseOptimumWashout = .true. ! Use the optimum total washout?

        real\*8 :: AileronDeflection = 0.0d0 ! Aileron deflection, in radians

        real\*8 :: DesiredRollingRate = 0.0d0 ! Desired dimensionless rolling rate (constant over wingspan)

        real\*8 :: RollingRate = 0.0d0 ! Dimensionless rolling rate (constant over wingspan)

        logical :: SpecifyAlpha = .true. ! Was alpha specified?

                                         ! .true.  = Use desired alpha to calculate CL

                                         ! .false. = Use desired CL to calculate alpha

        logical :: UseSteadyRollingRate = .true. ! Use the steady dimensionless rolling rate?

        ! Planform Calculations

        real\*8, allocatable, dimension(:,:) :: BigC, BigC\_Inv

        real\*8, allocatable, dimension(:) :: a, b, c, d, BigA

        real\*8, allocatable, dimension(:) :: Omega

        logical :: IsAllocated = .false.

        ! Lift Coefficient Calculations

        real\*8 :: KL  ! Lift slope factor

        real\*8 :: EW  ! Washout effectiveness (epsilon omega)

        real\*8 :: CLa ! Wing lift slope (derivative of CL with respect to alpha)

        real\*8 :: CL1 ! Lift Coefficient (Eq. 1.8.24)

        real\*8 :: CL2 ! Lift Coefficient (Eq. 1.8.5)

        ! Drag Coefficient Calculations

        real\*8 :: KD   ! Induced drag factor

        real\*8 :: KDL  ! Lift-washout contribution to induced drag

        real\*8 :: KDW  ! Washout contribution to induced drag

        real\*8 :: ES   ! Span efficiency factor

        real\*8 :: CDi1 ! Induced drag coefficient (Eq. 1.8.25)

        real\*8 :: CDi2 ! Induced drag coefficient (Eq. 1.8.6)

        real\*8 :: CDi3 ! Induced drag coefficient (Eq. 32, Wing Flapping paper)

        ! Roll/yaw calculations

        real\*8 :: CRM\_da   ! Change in rolling moment coefficient with respect to alpha

        real\*8 :: CRM\_pbar ! Change in rolling moment coefficient with respect to rolling rate

        real\*8 :: CRM      ! Rolling moment coefficient

        real\*8 :: CYM      ! Yawing moment coefficient

    end type Planform

    contains

        character\*80 function GetWingType(pf) result(name)

            type(Planform), intent(in) :: pf

            if (pf%WingType .eq. Tapered) then

                name = "Tapered"

            else if (pf%WingType .eq. Elliptic) then

                name = "Elliptic"

            else if (pf%WingType .eq. Combination) then

                name = "Tapered with elliptic tip"

            else

                name = "Unknown"

            end if

        end function GetWingType

        character\*80 function GetWashoutDistributionType(pf) result(name)

            type(Planform), intent(in) :: pf

            if (pf%WashoutDistribution .eq. Linear) then

                name = "Linear"

            else if (pf%WashoutDistribution .eq. Optimum) then

                name = "Optimum"

            else

                name = "Unknown"

            end if

        end function GetWashoutDistributionType

        real\*8 function theta\_i(i, nnodes) result(theta)

            integer, intent(in) :: i

            integer, intent(in) :: nnodes

            if (i < 1 .or. i > nnodes) then

                write(6, '(a, i3)') "ERROR: Function theta\_i called with i = ", i

                if (i < 1) then

                    theta = 0.0d0

                else

                    theta = pi

                end if

            else

                theta = real(i-1, 8) / real(nnodes - 1, 8) \* pi

            end if

        end function theta\_i

        real\*8 function theta\_zb(zb) result(theta)

            real\*8, intent(in) :: zb  ! z/b

            if (zb < -0.5d0 .or. zb > 0.5d0) then

                write(6, '(a, f7.4)') "ERROR: Function theta\_d called with z/b = ", zb

                if (zb < -0.5d0) then

                    theta = 0.0d0

                else

                    theta = pi

                end if

            else

                theta = acos(-2.0d0 \* zb)

            end if

        end function theta\_zb

        real\*8 function c\_over\_b\_i(pf, i) result(cb)

            type(Planform), intent(in) :: pf

            integer, intent(in) :: i

            real\*8 :: theta

            theta = theta\_i(i, pf%NNodes)

            cb = c\_over\_b(pf, theta)

        end function c\_over\_b\_i

        real\*8 function c\_over\_b\_zb(pf, zb) result(cb)

            type(Planform), intent(in) :: pf

            real\*8, intent(in) :: zb  ! z/b

            real\*8 :: theta

            theta = theta\_zb(zb)

            cb = c\_over\_b(pf, theta)

        end function c\_over\_b\_zb

        real\*8 function c\_over\_b(pf, theta) result(cb)

            type(Planform), intent(in) :: pf

            real\*8, intent(in) :: theta

            real\*8 :: zb, u

            if (pf%WingType == Tapered) then

                ! Calculate c/b for tapered wing

                cb = (2.0d0 \* (1.0d0 - (1.0d0 - pf%TaperRatio) \* &

                    & dabs(cos(theta)))) / (pf%AspectRatio \* (1.0d0 + pf%TaperRatio))

            else if (pf%WingType == Elliptic) then

                ! Calculate c/b for elliptic wing

                cb = (4.0d0 \* sin(theta)) / &

                    & (pi \* pf%AspectRatio)

            else if (pf%WingType == Combination) then

                ! Calculate c/b for combination wing

                zb = abs(z\_over\_b(theta))

                if (zb <= pf%TransitionPoint) then

                    cb = pf%C5 \* (1.0d0 - pf%C2 \* zb)

                else

                    u = (zb - pf%C4) / (0.5d0 - pf%C4)

                    cb = pf%C5 \* pf%C3 \* sqrt(1.0d0 - u\*\*2)

                end if

            else

                ! Unknown wing type!

                stop "\*\*\* Unknown Wing Type \*\*\*"

            end if

        end function c\_over\_b

        real\*8 function z\_over\_b\_i(i, nnodes) result(zb)

            integer, intent(in) :: i

            integer, intent(in) :: nnodes

            zb = z\_over\_b(theta\_i(i, nnodes))

        end function z\_over\_b\_i

        real\*8 function z\_over\_b(theta) result(zb)

            real\*8, intent(in) :: theta

            zb = -0.5d0 \* cos(theta)

        end function z\_over\_b

        real\*8 function cf\_over\_c\_i(pf, i) result(cfc)

            type(Planform), intent(in) :: pf

            integer, intent(in) :: i

            real\*8 :: zbi

            zbi = z\_over\_b\_i(i, pf%NNodes)

            if (Compare(dabs(zbi), pf%AileronRoot, zero) == -1 .or. &

                & Compare(dabs(zbi), pf%AileronTip, zero) == 1) then

                cfc = 0.0d0

            else

                cfc = 0.75d0 - y\_i(pf, i) / c\_over\_b\_i(pf, i)

            end if

        end function cf\_over\_c\_i

        real\*8 function y\_i(pf, i) result(y)

            type(Planform), intent(in) :: pf

            integer, intent(in) :: i

            real\*8 :: zb\_i, cb\_i

            real\*8 :: zb\_root, cfc\_root, theta\_root, cb\_root, y\_root

            real\*8 :: zb\_tip, cfc\_tip, theta\_tip, cb\_tip, y\_tip

            real\*8 :: slope, offst

            zb\_root = pf%AileronRoot

            cfc\_root = pf%FlapFractionRoot

            theta\_root = theta\_zb(zb\_root)

            cb\_root = c\_over\_b(pf, theta\_root)

            y\_root = (0.75d0 - cfc\_root) \* cb\_root

            zb\_tip = pf%AileronTip

            cfc\_tip = pf%FlapFractionTip

            theta\_tip = theta\_zb(zb\_tip)

            cb\_tip = c\_over\_b(pf, theta\_tip)

            y\_tip = (0.75d0 - cfc\_tip) \* cb\_tip

            slope = (y\_tip - y\_root) / (zb\_tip - zb\_root)

            offst = y\_root - slope \* zb\_root

            zb\_i = z\_over\_b\_i(i, pf%NNodes)

            y = slope \* dabs(zb\_i) + offst

        end function y\_i

        real\*8 function FlapEffectiveness(pf, i) result(eps\_f)

            type(Planform), intent(in) :: pf

            integer, intent(in) :: i

            real\*8 :: theta\_f, eps\_fi

            theta\_f = acos(2.0d0 \* cf\_over\_c\_i(pf, i) - 1.0d0)

            eps\_fi = 1.0d0 - (theta\_f - sin(theta\_f)) / pi

            eps\_f = eps\_fi \* pf%HingeEfficiency \* pf%DeflectionEfficiency

        end function FlapEffectiveness

        subroutine DeallocateArrays(pf)

            type(Planform), intent(inout) :: pf

            if (pf%IsAllocated) then

                deallocate(pf%BigC)

                deallocate(pf%BigC\_Inv)

                deallocate(pf%a)

                deallocate(pf%b)

                deallocate(pf%c)

                deallocate(pf%d)

                deallocate(pf%BigA)

                deallocate(pf%Omega)

                pf%IsAllocated = .false.

            end if

        end subroutine DeallocateArrays

        subroutine AllocateArrays(pf)

            type(Planform), intent(inout) :: pf

            if (pf%IsAllocated) call DeallocateArrays(pf)

            allocate(pf%BigC(pf%NNodes, pf%NNodes))

            allocate(pf%BigC\_Inv(pf%NNodes, pf%NNodes))

            allocate(pf%a(pf%NNodes))

            allocate(pf%b(pf%NNodes))

            allocate(pf%c(pf%NNodes))

            allocate(pf%d(pf%NNodes))

            allocate(pf%BigA(pf%NNodes))

            allocate(pf%Omega(pf%NNodes))

            pf%IsAllocated = .true.

        end subroutine AllocateArrays

end module class\_Planform

module LiftingLineSetters

    use class\_Planform

    implicit none

contains

    subroutine InitPlanform(pf)

        type(Planform), intent(inout) :: pf

        call SetParallelHingeLine(pf)

    end subroutine InitPlanform

    ! Planform Parameters

    subroutine SetWingType(pf, wingType)

        type(Planform), intent(inout) :: pf

        integer, intent(in) :: wingType

        if (pf%WingType /= wingType) then

            pf%WingType = wingType

            call DeallocateArrays(pf)

            if (pf%WingType == Combination) then

                call SetCombinationWingCoefficients(pf)

            else if (pf%ParallelHingeLine) then

                call SetParallelHingeLine(pf)

            end if

            if (pf%WingType == Elliptic) then

                pf%WashoutDistribution = Linear

            end if

        end if

    end subroutine SetWingType

    subroutine SetWashoutDistribution(pf, washoutDist)

        type(Planform), intent(inout) :: pf

        integer, intent(in) :: washoutDist

        if (pf%WingType /= Elliptic .and. pf%WashoutDistribution /= washoutDist) then

            pf%WashoutDistribution = washoutDist

            call DeallocateArrays(pf)

        end if

    end subroutine SetWashoutDistribution

    subroutine SetTransitionPoint(pf, tp)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: tp

        if (Compare(pf%TransitionPoint, tp, zero) /= 0) then

            pf%TransitionPoint = tp

            call DeallocateArrays(pf)

            call SetCombinationWingCoefficients(pf)

        end if

    end subroutine SetTransitionPoint

    subroutine SetTransitionChord(pf, tc)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: tc

        if (Compare(pf%TransitionChord, tc, zero) /= 0) then

            pf%TransitionChord = tc

            call DeallocateArrays(pf)

            call SetCombinationWingCoefficients(pf)

        end if

    end subroutine SetTransitionChord

    subroutine SetCombinationWingCoefficients(pf)

        type(Planform), intent(inout) :: pf

        real\*8 :: u, asin\_u

        pf%C1 = pf%TransitionPoint

        pf%C2 = (1.0d0 - pf%TransitionChord) / pf%C1

        pf%C4 = (pf%C1 - 0.25d0 \* pf%C2) / (pf%C1 \* pf%C2 - pf%C2 + 1.0d0)

        u = (pf%C1 - pf%C4) / (0.5d0 - pf%C4)

        asin\_u = asin(u)

        pf%C3 = (1.0d0 - pf%C1 \* pf%C2) / sqrt(1.0d0 - u\*\*2)

        pf%C5 = 1.0d0 / (pf%AspectRatio \* (2.0d0 \* pf%C1 - pf%C1\*\*2 \* pf%C2 + &

            & 0.5d0 \* pf%C3 \* (0.5d0 - pf%C4) \* (pi - 2.0d0 \* asin\_u - &

            & sin(2.0d0 \* asin\_u))))

        if (pf%ParallelHingeLine) then

            call SetParallelHingeLine(pf)

        end if

    end subroutine SetCombinationWingCoefficients

    logical function AreCombinationWingCoefficientsValid(pf) result(isValid)

        type(Planform), intent(in) :: pf

        real\*8 :: u, d1, d2

        u = (pf%C1 - pf%C4) / (0.5d0 - pf%C4)

        d1 = -pf%C2

        d2 = -(pf%C3 \* u) / (sqrt(1.0d0 - u\*\*2) \* (0.5d0 - pf%C4))

        if (Compare(pf%C1, 0.0d0, zero) /= 1 .and. &

            & Compare(pf%C1, 0.5d0, zero) /= -1) then

            isValid = .false.

        else if (Compare(pf%C1 \* pf%C2 - pf%C2 + 1.0d0, 0.0d0, zero) == 0) then

            isValid = .false.

        else if (Compare(pf%C4, 0.5d0, zero) == 0) then

            isValid = .false.

        else if (Compare(dabs(u), 1.0d0, zero) /= -1) then

            isValid = .false.

        else if (Compare(d1, d2, zero) /= 0) then

            isValid = .false.

        else

            isValid = .true.

        end if

    end function AreCombinationWingCoefficientsValid

    subroutine SetNNodes(pf, npss)

        type(Planform), intent(inout) :: pf

        integer, intent(in) :: npss

        integer :: nnodes

        nnodes = npss \* 2 - 1

        if (pf%NNodes /= nnodes) then

            pf%NNodes = nnodes

            call DeallocateArrays(pf)

        end if

    end subroutine SetNNodes

    subroutine SetAspectRatio(pf, ra)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: ra

        if (Compare(pf%AspectRatio, ra, zero) /= 0) then

            pf%AspectRatio = ra

            call DeallocateArrays(pf)

        end if

    end subroutine SetAspectRatio

    subroutine SetTaperRatio(pf, rt)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: rt

        if (Compare(pf%TaperRatio, rt, zero) /= 0) then

            pf%TaperRatio = rt

            call DeallocateArrays(pf)

            if (pf%ParallelHingeLine) then

                call SetParallelHingeLine(pf)

            end if

        end if

    end subroutine SetTaperRatio

    subroutine SetSectionLiftSlope(pf, cla\_sec)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: cla\_sec

        if (Compare(pf%SectionLiftSlope, cla\_sec, zero) /= 0) then

            pf%SectionLiftSlope = cla\_sec

            call DeallocateArrays(pf)

        end if

    end subroutine SetSectionLiftSlope

    subroutine SetAileronRoot(pf, ar)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: ar

        if (Compare(pf%AileronRoot, ar, zero) /= 0) then

            pf%AileronRoot = ar

            call DeallocateArrays(pf)

        end if

    end subroutine SetAileronRoot

    subroutine SetAileronTip(pf, at)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: at

        if (Compare(pf%AileronTip, at, zero) /= 0) then

            pf%AileronTip = at

            call DeallocateArrays(pf)

        end if

    end subroutine SetAileronTip

    subroutine SetParallelHingeLine(pf)

        type(Planform), intent(inout) :: pf

        real\*8 :: cfc\_root\_par

        pf%ParallelHingeLine = .true.

        cfc\_root\_par = ParallelRootFlapFraction(pf)

        if (Compare(pf%FlapFractionRoot, cfc\_root\_par, zero) /= 0) then

            pf%FlapFractionRoot = cfc\_root\_par

            call DeallocateArrays(pf)

        end if

    end subroutine SetParallelHingeLine

    subroutine SetFlapFractionRoot(pf, cfc\_root)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: cfc\_root

        pf%ParallelHingeLine = .false.

        pf%DesiredFlapFractionRoot = cfc\_root

        if (Compare(pf%FlapFractionRoot, cfc\_root, zero) /= 0) then

            pf%FlapFractionRoot = cfc\_root

            call DeallocateArrays(pf)

        end if

    end subroutine SetFlapFractionRoot

    subroutine SetFlapFractionTip(pf, cfc\_tip)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: cfc\_tip

        if (Compare(pf%FlapFractionTip, cfc\_tip, zero) /= 0) then

            pf%FlapFractionTip = cfc\_tip

            call DeallocateArrays(pf)

            if (pf%ParallelHingeLine) then

                call SetParallelHingeLine(pf)

            end if

        end if

    end subroutine SetFlapFractionTip

    subroutine SetHingeEfficiency(pf, eff\_hinge)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: eff\_hinge

        if (Compare(pf%HingeEfficiency, eff\_hinge, zero) /= 0) then

            pf%HingeEfficiency = eff\_hinge

            call DeallocateArrays(pf)

        end if

    end subroutine SetHingeEfficiency

    subroutine SetDeflectionEfficiency(pf, eff\_def)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: eff\_def

        if (Compare(pf%DeflectionEfficiency, eff\_def, zero) /= 0) then

            pf%DeflectionEfficiency = eff\_def

            call DeallocateArrays(pf)

        end if

    end subroutine SetDeflectionEfficiency

    subroutine ToggleOutputMatricies(pf)

        type(Planform), intent(inout) :: pf

        pf%OutputMatrices = .not. pf%OutputMatrices

    end subroutine ToggleOutputMatricies

    subroutine SetFileName(pf, filename)

        type(Planform), intent(inout) :: pf

        character\*80, intent(in) :: filename

        pf%FileName = trim(filename)

    end subroutine SetFileName

    ! Operating Conditions

    subroutine SetAngleOfAttack(pf, alpha)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: alpha

        pf%SpecifyAlpha = .true.

        pf%DesiredAngleOfAttack = alpha \* pi / 180.0d0

        pf%AngleOfAttack = pf%DesiredAngleOfAttack

        if (pf%UseOptimumWashout) then

            call SetOptimumWashout(pf)

        end if

        pf%LiftCoefficient = CL1(pf%CLa, pf%AngleOfAttack, pf%EW, pf%Washout)

    end subroutine SetAngleOfAttack

    subroutine SetLiftCoefficient(pf, cl)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: cl

        pf%SpecifyAlpha = .false.

        pf%DesiredLiftCoefficient = cl

        pf%LiftCoefficient = pf%DesiredLiftCoefficient

        if (pf%UseOptimumWashout) then

            call SetOptimumWashout(pf)

        end if

        pf%AngleOfAttack = RootAlpha(pf%CLa, pf%LiftCoefficient, pf%EW, pf%Washout)

    end subroutine SetLiftCoefficient

    subroutine SetAileronDeflection(pf, da)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: da

        pf%AileronDeflection = da \* pi / 180.0d0

    end subroutine SetAileronDeflection

    subroutine SetWashout(pf, washout)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: washout

        pf%DesiredWashout = washout \* pi / 180.0d0

        pf%Washout = pf%DesiredWashout

        pf%UseOptimumWashout = .false.

        if (pf%SpecifyAlpha) then

            pf%LiftCoefficient = CL1(pf%CLa, pf%AngleOfAttack, pf%EW, pf%Washout)

        else

            pf%AngleOfAttack = RootAlpha(pf%CLa, pf%LiftCoefficient, pf%EW, pf%Washout)

        end if

    end subroutine SetWashout

    subroutine SetOptimumWashout(pf)

        type(Planform), intent(inout) :: pf

        logical :: cont

        integer :: i

        real\*8 :: oldCL, newCL, resCL

        real\*8 :: oldOmega, newOmega, resOmega

        if (pf%SpecifyAlpha) then

            oldCL = pf%LiftCoefficient

            oldOmega = pf%Washout

            cont = .true.

            i = 0

            do while (i < 100 .or. (cont .and. i < 1000))

                i = i + 1

                newOmega = OptimumWashout1(pf%KDL, oldCL, pf%KDW, pf%CLa)

                newCL = CL1(pf%CLa, pf%AngleOfAttack, pf%EW, newOmega)

                resCL = Residual(oldCL, newCL)

                resOmega = Residual(oldOmega, newOmega)

                cont = (Compare(resCL, 0.0d0, zero) /= 0 .or. Compare(resOmega, 0.0d0, zero) /= 0)

                oldOmega = newOmega

                oldCL = newCL

            end do

            if (i >= 1000) then

                stop "\*\*\* Max number of convergence iterations reached! \*\*\*"

            else

                pf%LiftCoefficient = newCL

                pf%OptimumWashout1 = newOmega

            end if

        end if

        pf%OptimumWashout1 = OptimumWashout1(pf%KDL, pf%LiftCoefficient, &

            & pf%KDW, pf%CLa)

        if (pf%WashoutDistribution == Optimum) then

            pf%OptimumWashout2 = OptimumWashout2(pf, pf%LiftCoefficient, &

                pf%AspectRatio, pf%SectionLiftSlope)

        end if

        pf%Washout = pf%OptimumWashout1

        pf%UseOptimumWashout = .true.

        if (.not. pf%SpecifyAlpha) then

            pf%AngleOfAttack = RootAlpha(pf%CLa, pf%LiftCoefficient, pf%EW, pf%Washout)

        end if

    end subroutine SetOptimumWashout

    subroutine SetRollingRate(pf, rollingrate)

        type(Planform), intent(inout) :: pf

        real\*8, intent(in) :: rollingrate

        pf%DesiredRollingRate = rollingrate

        pf%RollingRate = rollingrate

        pf%UseSteadyRollingRate = .false.

    end subroutine

    subroutine SetSteadyRollingRate(pf)

        type(Planform), intent(inout) :: pf

        real\*8 :: steady\_pbar

        pf%RollingRate = SteadyRollingRate(pf)

        pf%UseSteadyRollingRate = .true.

    end subroutine SetSteadyRollingRate

    real\*8 function ParallelRootFlapFraction(pf) result(cfc\_root\_par)

        type(Planform), intent(in) :: pf

        real\*8 :: cb\_root, cfc\_root

        real\*8 :: cb\_tip, cfc\_tip

        cb\_tip = c\_over\_b\_zb(pf, pf%AileronTip)

        cfc\_tip = pf%FlapFractionTip

        cb\_root = c\_over\_b\_zb(pf, pf%AileronRoot)

        cfc\_root\_par = 0.75d0 - cb\_tip / cb\_root \* (0.75d0 - cfc\_tip)

    end function ParallelRootFlapFraction

    real\*8 function RootAlpha(cla, cl, ew, omega) result(alpha)

        real\*8, intent(in) :: cla

        real\*8, intent(in) :: cl

        real\*8, intent(in) :: ew

        real\*8, intent(in) :: omega

        alpha = cl / cla + ew \* omega

    end function RootAlpha

    real\*8 function SteadyRollingRate(pf) result(pbar\_steady)

        type(Planform), intent(in) :: pf

        ! Calculate steady dimensionless rolling rate (Eq. 1.8.59)

        pbar\_steady = -pf%CRM\_da / pf%CRM\_pbar \* pf%AileronDeflection

    end function SteadyRollingRate

    real\*8 function OptimumWashout1(kdl, cl, kdw, cla) result(ow1)

        real\*8, intent(in) :: kdl

        real\*8, intent(in) :: cl

        real\*8, intent(in) :: kdw

        real\*8, intent(in) :: cla

        ow1 = (kdl \* cl) / (2.0d0 \* kdw \* cla)

    end function OptimumWashout1

    real\*8 function OptimumWashout2(pf, cl, ra, ls) result(ow2)

        type(Planform), intent(in) :: pf

        real\*8, intent(in) :: cl

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: ls

        ow2 = (4.0d0 \* cl) / (pi \* ra \* ls \* c\_over\_b(pf, pi / 2.0d0))

    end function OptimumWashout2

    real\*8 function CL1(cla, alpha, ew, w) result(cl)

        real\*8, intent(in) :: cla

        real\*8, intent(in) :: alpha

        real\*8, intent(in) :: ew

        real\*8, intent(in) :: w

        cl = cla \* (alpha - ew \* w)  ! Eq. 1.8.24

    end function CL1

    real\*8 function CL2(ra, bigA1) result(cl)

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: bigA1

        cl = pi \* ra \* bigA1  ! Eq. 1.8.5

    end function CL2

    real\*8 function CDi1(pf) result(cdi)

        type(Planform), intent(in) :: pf

        cdi = (pf%CL1\*\*2 \* (1.0d0 + pf%KD) - pf%KDL \* pf%CL1 \* pf%CLa \* pf%Washout + &

            & pf%KDW \* (pf%CLa \* pf%Washout)\*\*2) / (pi \* pf%AspectRatio)

    end function CDi1

    real\*8 function CDi2(pf) result(cdi)

        type(Planform), intent(in) :: pf

        integer :: i

        cdi = 0.0d0

        do i = 1, pf%NNodes

            cdi = cdi + real(i, 8) \* pf%BigA(i)\*\*2

        end do

        cdi = cdi \* pi \* pf%AspectRatio

    end function CDi2

    real\*8 function CDi3(pf) result(cdi)

        type(Planform), intent(in) :: pf

        integer :: i

        real\*8 :: ri

        cdi = 0.0d0

        do i = 1, pf%NNodes

            ri = real(i, 8)

            cdi = cdi + ri \* pf%BigA(i)\*\*2

        end do

        cdi = (cdi - 0.5d0 \* pf%RollingRate \* pf%BigA(2)) \* pi \* pf%AspectRatio

    end function CDi3

end module LiftingLineSetters

module matrix

    implicit none

contains

    subroutine matinv\_gauss(n, mat, mat\_inv)

        integer, intent(in) :: n

        real\*8, intent(in) :: mat(n,n)

        real\*8, intent(out) :: mat\_inv(n,n)

        real\*8 :: b(n, n), c, d, temp(n)

        integer :: i, j, k, m, imax(1), ipvt(n)

        b = mat

        ipvt = (/ (i, i=1, n) /)

        do k = 1, n

            imax = maxloc(abs(b(k:n, k)))

            m = k - 1 + imax(1)

            if (m /= k) then

                ipvt( (/m, k/) ) = ipvt( (/k, m/) )

                b( (/m, k/), :) = b( (/k, m/), :)

            end if

            d = 1.0d0 / b(k, k)

            temp = b(:, k)

            do j = 1, n

                c = b(k, j) \* d

                b(:, j) = b(:, j) - temp \* c

                b(k, j) = c

            end do

            b(:, k) = temp \* (-d)

            b(k, k) = d

        end do

        mat\_inv(:, ipvt) = b

    end subroutine matinv\_gauss

    subroutine printmat(u, m, n, mat)

        integer, intent(in) :: u

        integer, intent(in) :: m

        integer, intent(in) :: n

        real\*8, intent(in) :: mat(m, n)

        integer :: i, j

        character\*80 :: format\_string

        if (u == 6) then

            write(format\_string, '(a, i10, a)') "(", n, "(F9.5, 2x))"

        else

            write(format\_string, '(a, i10, a)') "(", n, "(F24.15, 2x))"

        end if

        do i = 1, m

            write(u, format\_string) (mat(i, j), j=1, n)

        end do

    end subroutine printmat

end module matrix

module LiftingLineSolver

    use class\_Planform

    use LiftingLineSetters

    use LiftingLineOutput

    use matrix

    implicit none

contains

    subroutine ComputeCMatrixAndCoefficients(pf)

        type(Planform), intent(inout) :: pf

        write(6, '(a)') "Calculating C matrix and Fourier coefficients, please wait..."

        write(6, '(a, a, a)') "Estimated calculation time: ", &

            & trim(FormatReal(pf%NNodes\*\*2 \* 1.0d-5, 3)), " seconds"

        write(6, \*)

        if (.not. pf%IsAllocated) then

            if (pf%WingType == Combination) then

                call SetCombinationWingCoefficients(pf)

            end if

            call AllocateArrays(pf)

            call ComputeC(pf, pf%BigC)

            call ComputeCInverse(pf, pf%BigC\_Inv)

            call ComputeFourierCoefficients\_a(pf, pf%a)

            call ComputeFourierCoefficients\_b(pf, pf%b, pf%Omega)

            call ComputeFourierCoefficients\_c(pf, pf%c)

            call ComputeFourierCoefficients\_d(pf, pf%d)

            call ComputeLiftCoefficientParameters(pf)

            call ComputeDragCoefficientParameters(pf)

            call ComputeRollCoefficientParameters(pf)

            call ComputeFlightConditions(pf)

        end if

    end subroutine ComputeCMatrixAndCoefficients

    subroutine ComputeFourierCoefficients\_a(pf, a)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: a(pf%NNodes)

        real\*8 :: ones(pf%NNodes)

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        ones = (/ (1.0d0, i=1, nnodes) /)

        if (pf%WingType == Tapered .and. Compare(pf%TaperRatio, 0.0d0, zero) == 0) then

            ones(1) = 0.0d0

            ones(nnodes) = 0.0d0

        end if

        a = matmul(pf%BigC\_Inv, ones)

    end subroutine ComputeFourierCoefficients\_a

    subroutine ComputeFourierCoefficients\_b(pf, b, omega)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: b(pf%NNodes)

        real\*8, intent(out) :: omega(pf%NNodes)

        real\*8 :: croot\_over\_b, theta

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        if (pf%WashoutDistribution == Linear) then

            omega = (/ (dabs(cos(theta\_i(i, nnodes))), i=1, nnodes) /)

            if (pf%WingType == Tapered .and. Compare(pf%TaperRatio, 0.0d0, zero) == 0) then

                omega(1) = 0.0d0

                omega(nnodes) = 0.0d0

            end if

        else if (pf%WashoutDistribution == Optimum) then

            croot\_over\_b = c\_over\_b(pf, pi / 2.0d0)

            do i = 1, nnodes

                theta = theta\_i(i, nnodes)

                omega(i) = 1.0d0 - sin(theta) / (c\_over\_b(pf, theta) / croot\_over\_b)

            end do

            if (pf%WingType == Combination) then

                omega(1) = 1.0d0 - sqrt(1.0d0 - 2.0d0 \* pf%C4) / pf%C3

                omega(nnodes) = omega(1)

            else if (pf%WingType == Tapered .and. Compare(pf%TaperRatio, 0.0d0, zero) == 0) then

                omega(1) = 2.0d0

                omega(nnodes) = 2.0d0

            end if

        else

            write(6, '(a)') "Unknown washout distribution type!"

            stop

        end if

        b = matmul(pf%BigC\_Inv, omega)

    end subroutine ComputeFourierCoefficients\_b

    subroutine ComputeFourierCoefficients\_c(pf, c)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: c(pf%NNodes)

        real\*8 :: chi(pf%NNodes)

        real\*8 :: zbi

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        do i = 1, nnodes

            zbi = z\_over\_b\_i(i, nnodes)

            chi(i) = -sign(FlapEffectiveness(pf, i), zbi)

        end do

        if (pf%WingType == Tapered .and. Compare(pf%TaperRatio, 0.0d0, zero) == 0) then

            chi(1) = 0.0d0

            chi(nnodes) = 0.0d0

        end if

        c = matmul(pf%BigC\_Inv, chi)

    end subroutine ComputeFourierCoefficients\_c

    subroutine ComputeFourierCoefficients\_d(pf, d)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: d(pf%NNodes)

        real\*8 :: cos\_theta(pf%NNodes)

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        cos\_theta = (/ (cos(theta\_i(i, nnodes)), i=1, nnodes) /)

        if (pf%WingType == Tapered .and. Compare(pf%TaperRatio, 0.0d0, zero) == 0) then

            cos\_theta(1) = 0.0d0

            cos\_theta(nnodes) = 0.0d0

        end if

        d = matmul(pf%BigC\_Inv, cos\_theta)

    end subroutine ComputeFourierCoefficients\_d

    subroutine ComputeBigACoefficients(pf, bigA)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: bigA(pf%NNodes)

        integer :: i

        do i = 1, pf%NNodes

            bigA(i) = pf%a(i) \* pf%AngleOfAttack - pf%b(i) \* pf%Washout + &

                & pf%c(i) \* pf%AileronDeflection + pf%d(i) \* pf%RollingRate

        end do

    end subroutine ComputeBigACoefficients

    subroutine ComputeLiftCoefficientParameters(pf)

        type(Planform), intent(inout) :: pf

        pf%KL = Kappa\_L(pf%AspectRatio, pf%SectionLiftSlope, pf%a(1))

        pf%EW = Epsilon\_Omega(pf%a(1), pf%b(1))

        pf%CLa = C\_L\_alpha(pf%AspectRatio, pf%a(1))

    end subroutine ComputeLiftCoefficientParameters

    subroutine ComputeDragCoefficientParameters(pf)

        type(Planform), intent(inout) :: pf

        pf%KD = Kappa\_D(pf%NNodes, pf%a)

        pf%ES = SpanEfficiencyFactor(pf%KD)

        pf%KDL = Kappa\_DL(pf%NNodes, pf%a, pf%b)

        pf%KDW = Kappa\_DOmega(pf%NNodes, pf%a, pf%b)

    end subroutine ComputeDragCoefficientParameters

    subroutine ComputeRollCoefficientParameters(pf)

        type(Planform), intent(inout) :: pf

        pf%CRM\_da = CRM\_dAlpha(pf%AspectRatio, pf%c(2))

        pf%CRM\_pbar = CRM\_PBar(pf%AspectRatio, pf%d(2))

    end subroutine ComputeRollCoefficientParameters

    real\*8 function Kappa\_L(ra, cla\_section, a1) result(kl)

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: cla\_section

        real\*8, intent(in) :: a1

        kl = 1.0d0 / ((1.0d0 + pi \* ra / cla\_section) \* a1) - 1.0d0

    end function Kappa\_L

    real\*8 function Epsilon\_Omega(a1, b1) result(ew)

        real\*8, intent(in) :: a1

        real\*8, intent(in) :: b1

        ew = b1 / a1

    end function Epsilon\_Omega

    real\*8 function C\_L\_alpha(ra, a1) result(cla)

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: a1

        cla = pi \* ra \* a1

    end function C\_L\_alpha

    real\*8 function Kappa\_D(nnodes, a) result(kd)

        integer, intent(in) :: nnodes

        real\*8, intent(in) :: a(nnodes)

        integer :: i

        kd = 0.0d0

        do i = 2, nnodes

            kd = kd + real(i, 8) \* (a(i) / a(1))\*\*2

        end do

    end function Kappa\_D

    real\*8 function SpanEfficiencyFactor(kd) result(es)

        real\*8, intent(in) :: kd

        es = 1.0d0 / (1.0d0 + kd)

    end function SpanEfficiencyFactor

    real\*8 function Kappa\_DL(nnodes, a, b) result (kdl)

        integer, intent(in) :: nnodes

        real\*8, intent(in) :: a(nnodes)

        real\*8, intent(in) :: b(nnodes)

        integer :: i

        kdl = 0.0d0

        do i = 2, nnodes

            kdl = kdl + real(i, 8) \* a(i) / a(1) \* &

                & (b(i) / b(1) - a(i) / a(1))

        end do

        kdl = kdl \* 2.0d0 \* b(1) / a(1)

    end function Kappa\_DL

    real\*8 function Kappa\_DOmega(nnodes, a, b) result(kdw)

        integer, intent(in) :: nnodes

        real\*8, intent(in) :: a(nnodes)

        real\*8, intent(in) :: b(nnodes)

        integer :: i

        kdw = 0.0d0

        do i = 2, nnodes

            kdw = kdw + real(i, 8) \* (b(i) / b(1) - a(i) / a(1))\*\*2

        end do

        kdw = kdw \* (b(1) / a(1))\*\*2

    end function Kappa\_DOmega

    real\*8 function CRM\_dAlpha(ra, c2) result(crmda)

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: c2

        crmda = -pi \* ra / 4.0d0 \* c2

    end function CRM\_dAlpha

    real\*8 function CRM\_PBar(ra, d2) result(crmpbar)

        real\*8, intent(in) :: ra

        real\*8, intent(in) :: d2

        crmpbar = -pi \* ra / 4.0d0 \* d2

    end function CRM\_PBar

    subroutine ComputeFlightConditions(pf)

        type(Planform), intent(inout) :: pf

        ! Make sure planform characteristics have been computed

        if (.not. pf%IsAllocated) then

            call ComputeCMatrixAndCoefficients(pf)

        end if

        ! Compute root aerodynamic angle of attack, if necessary

        if (.not. pf%SpecifyAlpha) then

            pf%AngleOfAttack = RootAlpha(pf%CLa, pf%LiftCoefficient, pf%EW, pf%Washout)

        else

            pf%LiftCoefficient = CL1(pf%CLa, pf%AngleOfAttack, pf%EW, pf%Washout)

        end if

        ! Compute optimum total washout, if necessary

        if (pf%UseOptimumWashout) then

            call SetOptimumWashout(pf)

        else

            call SetWashout(pf, pf%DesiredWashout \* 180.0d0 / pi)

        end if

        ! Compute steady rolling rate, if necessary

        if (pf%UseSteadyRollingRate) then

            call SetSteadyRollingRate(pf)

        end if

        ! Compute BigA Fourier Coefficients

        call ComputeBigACoefficients(pf, pf%BigA)

        ! Compute lift coefficients

        call ComputeLiftCoefficients(pf)

        ! Compute drag coefficient

        call ComputeDragCoefficients(pf)

        ! Compute roll coefficient

        pf%CRM = CRoll(pf%CRM\_da, pf%CRM\_pbar, pf%AileronDeflection, pf%RollingRate)

        ! Compute yaw coefficient

        pf%CYM = CYaw(pf, pf%CL1, pf%BigA)

    end subroutine ComputeFlightConditions

    subroutine ComputeLiftCoefficients(pf)

        type(Planform), intent(inout) :: pf

        pf%CL1 = CL1(pf%CLa, pf%AngleOfAttack, pf%EW, pf%Washout)

        pf%CL2 = CL2(pf%AspectRatio, pf%BigA(1))

    end subroutine ComputeLiftCoefficients

    subroutine ComputeDragCoefficients(pf)

        type(Planform), intent(inout) :: pf

        pf%CDi1 = CDi1(pf)

        pf%CDi2 = CDi2(pf)

        pf%CDi3 = CDi3(pf)

    end subroutine ComputeDragCoefficients

    real\*8 function CRoll(crmda, crmpbar, da, pbar) result(crm)

        real\*8, intent(in) :: crmda

        real\*8, intent(in) :: crmpbar

        real\*8, intent(in) :: da

        real\*8, intent(in) :: pbar

        crm = crmda \* da + crmpbar \* pbar

    end function CRoll

    real\*8 function CYaw(pf, cl, bigA) result(cym)

        type(Planform), intent(in) :: pf

        real\*8, intent(in) :: cl

        real\*8, intent(in) :: bigA(pf%NNodes)

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        cym = cl / 8.0d0 \* (6.0d0 \* bigA(2) - pf%RollingRate) + &

            & pi \* pf%AspectRatio / 8.0d0 \* (10.0d0 \* bigA(2) - &

            & pf%RollingRate) \* bigA(3)

        do i = 4, nnodes

            cym = cym + 0.25d0 \* pi \* pf%AspectRatio \* &

                & (2.0d0 \* real(i, 8) - 1.0d0) \* bigA(i-1) \* bigA(i)

        end do

    end function CYaw

    subroutine ComputeC(pf, c)

        type(Planform), intent(in) :: pf

        real\*8, intent(inout) :: c(pf%NNodes, pf%NNodes)

        integer :: i

        integer :: nnodes

        nnodes = pf%NNodes

        ! Compute values for i=1, i=N

        call C1j\_Nj(c, pf)

        ! Compute values for i=2 to i=N-1

        do i = 2, nnodes-1

            call Cij(c, i, pf)

        end do

    end subroutine ComputeC

    subroutine ComputeCInverse(pf, c\_inv)

        type(Planform), intent(in) :: pf

        real\*8, intent(inout) :: c\_inv(pf%NNodes, pf%NNodes)

        call matinv\_gauss(pf%NNodes, pf%BigC, c\_inv)

    end subroutine ComputeCInverse

    subroutine C1j\_Nj(c, pf)

        real\*8, dimension(:,:), intent(inout) :: c

        type(Planform), intent(in) :: pf

        integer :: j

        integer :: jsq

        integer :: nnode

        real\*8 :: cb0

        nnode = pf%NNodes

        do j = 1, nnode

            jsq = j\*\*2

            c(1, j) = real(jsq, 8)

            c(nnode, j) = real((-1)\*\*(j + 1) \* jsq, 8)

        end do

        cb0 = c\_over\_b(pf, pi)

        if (dabs(cb0) < 1.0d-10) then

            call C1j\_Nj\_zero\_chord(c, pf)

        end if

    end subroutine C1j\_Nj

    subroutine Cij(c, i, pf)

        real\*8, dimension(:,:), intent(inout) :: c

        integer, intent(in) :: i

        type(Planform), intent(in) :: pf

        integer :: j

        integer :: nnode

        real\*8 :: theta

        real\*8 :: cb

        real\*8 :: sin\_theta

        nnode = pf%NNodes

        theta = theta\_i(i, nnode)

        cb = c\_over\_b\_i(pf, i)

        sin\_theta = sin(theta)

        do j = 1, nnode

            c(i, j) = (4.0d0 / (pf%SectionLiftSlope \* cb) + &

                & real(j, 8) / sin\_theta) \* sin(real(j, 8) \* theta)

        end do

    end subroutine Cij

    subroutine C1j\_Nj\_zero\_chord(c, pf)

        real\*8, dimension(:,:), intent(inout) :: c

        type(Planform), intent(in) :: pf

        integer :: j, n

        n = pf%NNodes

        if (pf%WingType == Tapered) then

            do j = 1, n

                c(1, j) = 2.0d0 \* pf%AspectRatio \* (1.0d0 + real(j, 8))

                c(n, j) = real((-1)\*\*(j + 1), 8) \* c(1, j)

            end do

        else if (pf%WingType == Elliptic) then

            do j = 1, n

                c(1, j) = c(1, j) + real(j, 8) \* pi \* &

                    & pf%AspectRatio / pf%SectionLiftSlope

                c(n, j) = c(n, j) + real((-1)\*\*(j + 1) \* j, 8) \* pi \* &

                    & pf%AspectRatio / pf%SectionLiftSlope

            end do

        else if (pf%WingType == Combination) then

            ! TODO: Add code for combination wing type

            do j = 1, n

                c(1, j) = c(1, j) + 4.0d0 \* real(j, 8) \* &

                    & sqrt(1.0d0 - 2.0d0 \* pf%C4) / &

                    & (pf%C3 \* pf%C5 \* pf%SectionLiftSlope)

                c(n, j) = c(n, j) + 4.0d0 \* real((-1)\*\*(j + 1) \* j, 8) \* &

                    & sqrt(1.0d0 - 2.0d0 \* pf%C4) / &

                    & (pf%C3 \* pf%C5 \* pf%SectionLiftSlope)

            end do

        else

            stop "\*\*\* Unknown Wing Type \*\*\*"

        end if

    end subroutine C1j\_Nj\_zero\_chord

end module LiftingLineSolver

module LiftingLineOutput

    use Utilities

    use class\_Planform

    use LiftingLineSetters

    use matrix

    implicit none

contains

    subroutine OutputHeader()

        integer :: i

        write(6, ('(80a)')) ("\*", i=1, 80)

        write(6, '(34x, a)') "Pralines v1.0"

        write(6, \*)

        write(6, '(28x, a)') "Author: Josh Hodson"

        write(6, '(28x, a)') "Release Date: 20 Nov 2013"

        write(6, \*)

        write(6, ('(80a)')) ("\*", i=1, 80)

    end subroutine OutputHeader

    subroutine OutputPlanform(pf)

        type(Planform), intent(in) :: pf

        ! Open a clean file for output

        open(unit=10, file=pf%FileName)

        ! Output the planform summary to output file

        call OutputPlanformSummary(10, pf)

        ! Output C matrix and fourier coefficients to output file

        if (pf%OutputMatrices) then

            call OutputC(10, pf%NNodes, pf%BigC)

            call OutputCInverse(10, pf%NNodes, pf%BigC\_Inv)

            call OutputFourierCoefficients(10, pf)

        end if

        ! Close the output file

        close(unit=10)

    end subroutine OutputPlanform

    subroutine OutputLiftCoefficientParameters(u, pf)

        integer, intent(in) :: u  ! Output unit

        type(Planform), intent(in) :: pf

        write(u, '(a)') "Lift Coefficient Parameters:"

        write(u, '(2x, a, f20.15)') "KL    = ", pf%KL

        write(u, '(2x, a, f20.15)') "CL,a  = ", pf%CLa

        write(u, '(2x, a, f20.15)') "EW    = ", pf%EW

        write(u, \*)

    end subroutine OutputLiftCoefficientParameters

    subroutine OutputDragCoefficientParameters(u, pf)

        integer, intent(in) :: u  ! Output unit

        type(Planform), intent(in) :: pf

        write(u, '(a)') "Drag Coefficient Parameters:"

        write(u, '(2x, a, f20.15)') "KD    = ", pf%KD

        write(u, '(2x, a, f20.15)') "KDL   = ", pf%KDL

        write(u, '(2x, a, f20.15)') "KDW   = ", pf%KDW

        write(u, '(2x, a, f20.15)') "es    = ", pf%ES

        write(u, \*)

    end subroutine OutputDragCoefficientParameters

    subroutine OutputRollCoefficientParameters(u, pf)

        integer, intent(in) :: u  ! Output unit

        type(Planform), intent(in) :: pf

        write(u, '(a)') "Rolling Moment Coefficient Parameters:"

        write(u, '(2x, a, f20.15)') "Cl,da = ", pf%Crm\_da

        write(u, '(2x, a, f20.15)') "Cl,pb = ", pf%Crm\_pbar

        write(u, \*)

    end subroutine OutputRollCoefficientParameters

    subroutine OutputFlightConditions(pf)

        type(Planform), intent(in) :: pf

        ! Open the file and append flight conditions to end

        open(unit=10, file=pf%FileName, access="append")

        ! Output flight conditions to output file

        call OutputOperatingConditions(10, pf)

        call OutputFlightCoefficients(10, pf)

        ! Close the output file

        close(unit=10)

    end subroutine OutputFlightConditions

    subroutine OutputOperatingConditions(u, pf)

        integer, intent(in) :: u  ! Output unit

        type(Planform), intent(in) :: pf

        write(u, '(a15, 19x, 1x, a1, f20.15, 1x, a)') "Optimum washout", &

            & "=", pf%OptimumWashout1 \* 180.0d0 / pi, "degrees (Eq. 1.8.37)"

        if (pf%WashoutDistribution == Optimum) then

            write(u, '(a15, 19x, 1x, a1, f20.15, 1x, a)') "Optimum washout", &

                & "=", pf%OptimumWashout2 \* 180.0d0 / pi, "degrees (Eq. 1.8.42)"

        end if

        write(u, '(a28, 6x, 1x, a1, f20.15, 1x, a)') "Washout used in calculations", &

            & "=", pf%Washout \* 180.0d0 / pi, "degrees"

        write(u, '(a18, 16x, 1x, a1, f20.15, 1x, a)') &

            & "Aileron deflection", "=", &

            & pf%AileronDeflection \* 180.0d0 / pi, "degrees"

        write(u, '(a33, 1x, 1x, a1, f20.15)') &

            & "Steady dimensionless rolling rate", "=", SteadyRollingRate(pf)

        write(u, '(a31, 3x, 1x, a1, f20.15)') &

            & "Dimensionless rolling rate used", "=", pf%RollingRate

        write(u, '(a32, 2x, 1x, a1, f20.15, 1x, a)') &

            & "Root aerodynamic angle of attack", "=", &

            & pf%AngleOfAttack \* 180.0d0 / pi, "degrees"

        write(u, \*)

    end subroutine OutputOperatingConditions

    subroutine OutputFlightCoefficients(u, pf)

        integer, intent(in) :: u  ! Output unit

        type(Planform), intent(in) :: pf

        write(u, '(a)') "Flight Coefficients:"

        write(u, '(2x, a, f20.15, a)') "CL    = ", pf%CL1, " (Eq. 1.8.24)"

        write(u, '(2x, a, f20.15, a)') "CL    = ", pf%CL2, " (Eq. 1.8.5)"

        write(u, '(2x, a, f20.15, a)') "CDi   = ", pf%CDi1, " (Eq. 1.8.25)"

        write(u, '(2x, a, f20.15, a)') "CDi   = ", pf%CDi2, " (Eq. 1.8.6)"

        write(u, '(2x, a, f20.15, a)') "CDi   = ", pf%CDi3, " (Exact)"

        write(u, '(2x, a, f20.15)') "Croll = ", pf%CRM

        write(u, '(2x, a, f20.15)') "Cyaw  = ", pf%CYM

        write(u, \*)

    end subroutine OutputFlightCoefficients

    subroutine OutputPlanformSummary(u, pf)

        integer, intent(in) :: u

        type(Planform), intent(in) :: pf

        character\*80 :: fmt\_str

        integer :: len\_nnodes

        len\_nnodes = int(log10(real(pf%NNodes))) + 1

        write(fmt\_str, '(a,i1,a,i1,a)') "(2x, a15, 11x, 1x, a1, 3x, i", &

            & len\_nnodes, ", 1x, a, i", len\_nnodes, ",a)"

        write(u, '(a)') "Planform Summary:"

        ! Wing type

        write(u, '(2x, a9, 17x, 1x, a1, 3x, a)') "Wing type", "=", &

            & trim(GetWingType(pf))

        ! Number of nodes

        write(u, fmt\_str) "Number of nodes", "=", pf%NNodes, " (", &

            & (pf%NNodes + 1) / 2, " nodes per semispan)"

        ! Section Lift Slope

        write(u, '(2x, a26, 1x, a1, f20.15)') &

            & "Airfoil section lift slope", "=", pf%SectionLiftSlope

        ! Aspect Ratio

        write(u, '(2x, a12, 14x, 1x, a1, f20.15)') &

            & "Aspect Ratio", "=", pf%AspectRatio

        ! Taper Ratio

        if (pf%WingType == Tapered) then

            write(u, '(2x, a11, 15x, 1x, a1, f20.15)') &

                & "Taper Ratio", "=", pf%TaperRatio

        end if

        ! Transition from tapered to elliptic

        if (pf%WingType == Combination) then

            write(u, '(2x, a20, 6x, 1x, a1, f20.15)') "Transition Point z/b", &

                & "=", pf%TransitionPoint

            write(u, '(2x, a20, 6x, 1x, a1, f20.15)') "Transition Point c/croot", &

                & "=", pf%TransitionChord

        end if

        ! Washout distribution type

        if (pf%WingType /= Elliptic) then

            write(u, '(2x, a20, 6x, 1x, a1, 3x, a)') "Washout Distribution", &

                & "=", trim(GetWashoutDistributionType(pf))

        end if

        ! Location of aileron root, tip

        write(u, '(2x, a19, 7x, 1x, a1, f20.15)') &

            & "z/b at aileron root", "=", pf%AileronRoot

        write(u, '(2x, a18, 8x, 1x, a1, f20.15)') &

            & "z/b at aileron tip", "=", pf%AileronTip

        ! Flap fraction at aileron root, tip

        write(u, '(2x, a20, 6x, 1x, a1, f20.15)') &

            & "cf/c at aileron root", "=", pf%FlapFractionRoot

        write(u, '(2x, a19, 7x, 1x, a1, f20.15)') &

            & "cf/c at aileron tip", "=", pf%FlapFractionTip

        ! Hinge Efficiency Factor

        write(u, '(2x, a16, 10x, 1x, a1, f20.15)') &

            & "Hinge Efficiency", "=", pf%HingeEfficiency

        ! Deflection efficiency factor

        write(u, '(2x, a21, 5x, 1x, a1, f20.15)') &

            & "Deflection Efficiency", "=", pf%DeflectionEfficiency

        write(u, \*)

        call OutputLiftCoefficientParameters(u, pf)

        call OutputDragCoefficientParameters(u, pf)

        call OutputRollCoefficientParameters(u, pf)

    end subroutine OutputPlanformSummary

    subroutine OutputFourierCoefficients(u, pf)

        integer, intent(in) :: u

        type(Planform), intent(in) :: pf

        integer :: i

        write(u, '(a)') "Fourier Coefficients:"

        write(u, '(a3, 4(2x, a20))') &

            & "i", "a(i)", "b(i)", "c(i)", "d(i)"

        do i = 1, pf%NNodes

            write(u, '(i3, 4(2x, f20.15))') &

                & i, pf%a(i), pf%b(i), pf%c(i), pf%d(i)

        end do

        write(u, \*)

    end subroutine OutputFourierCoefficients

    subroutine OutputC(u, nnodes, c)

        integer, intent(in) :: u

        integer, intent(in) :: nnodes

        real\*8, intent(in) :: c(nnodes, nnodes)

        write(u, \*) "[C] Matrix:"

        call printmat(u, nnodes, nnodes, c)

        write(u, \*)

    end subroutine OutputC

    subroutine OutputCInverse(u, nnodes, c\_inv)

        integer, intent(in) :: u

        integer, intent(in) :: nnodes

        real\*8, intent(in) :: c\_inv(nnodes, nnodes)

        write(u, \*) "[C]^-1 Matrix:"

        call printmat(u, nnodes, nnodes, c\_inv)

        write(u, \*)

    end subroutine OutputCInverse

    subroutine OutputHingeLine(u, pf)

        integer, intent(in) :: u

        type(Planform), intent(in) :: pf

        integer :: i

        do i = 1, pf%NNodes

            write(u, '(i3, 2x, f20.15, 2x, f20.15)') i, z\_over\_b\_i(i, pf%NNodes), y\_i(pf, i)

        end do

    end subroutine OutputHingeLine

    subroutine PlotPlanform(pf)

        type(Planform), intent(in) :: pf

        integer :: i

        ! Generate temporary text file for plotting

        open(unit=11, file='planform.dat')

        write(11, '(a)') "$ Planform Geometry"

        ! Write data points for planform

        write(11, '(a)') "! Wing"

        do i=1, pf%NNodes

            write(11, '(f22.15, a, 2x, f22.15)') &

                & z\_over\_b\_i(i, pf%NNodes), ";", 0.25d0 \* c\_over\_b\_i(pf, i)

            write(11, '(f22.15, a, 2x, f22.15)') &

                & z\_over\_b\_i(i, pf%NNodes), ";", -0.75d0 \* c\_over\_b\_i(pf, i)

            write(11, '(f22.15, a, 2x, f22.15)') &

                & z\_over\_b\_i(i, pf%NNodes), ";", 0.25d0 \* c\_over\_b\_i(pf, i)

        end do

        do i=pf%NNodes, 1, -1

            write(11, '(f22.15, a, 2x, f22.15)') &

                & z\_over\_b\_i(i, pf%NNodes), ";", -0.75d0 \* c\_over\_b\_i(pf, i)

        end do

        write(11, '(f22.15, a, 2x, f22.15)') &

            & z\_over\_b\_i(1, pf%NNodes), ";", 0.25d0 \* c\_over\_b\_i(pf, 1)

        ! Write data points for right aileron

        write(11, '(a)') "$"

        write(11, '(a)') "! Right Aileron"

        write(11, '(f22.15, a, 2x, f22.15)') pf%AileronRoot, ";", &

            & -0.75d0 \* c\_over\_b\_zb(pf, pf%AileronRoot)

        write(11, '(f22.15, a, 2x, f22.15)') pf%AileronRoot, ";", &

            & (-0.75d0 + pf%FlapFractionRoot) \* c\_over\_b\_zb(pf, pf%AileronRoot)

        write(11, '(f22.15, a, 2x, f22.15)') pf%AileronTip, ";", &

            & (-0.75d0 + pf%FlapFractionTip) \* c\_over\_b\_zb(pf, pf%AileronTip)

        write(11, '(f22.15, a, 2x, f22.15)') pf%AileronTip, ";", &

            & -0.75d0 \* c\_over\_b\_zb(pf, pf%AileronTip)

        ! Write data points for left aileron

        write(11, '(a)') "$"

        write(11, '(a)') "! Left Aileron"

        write(11, '(f22.15, a, 2x, f22.15)') -pf%AileronRoot, ";", &

            & -0.75d0 \* c\_over\_b\_zb(pf, pf%AileronRoot)

        write(11, '(f22.15, a, 2x, f22.15)') -pf%AileronRoot, ";", &

            & (-0.75d0 + pf%FlapFractionRoot) \* c\_over\_b\_zb(pf, pf%AileronRoot)

        write(11, '(f22.15, a, 2x, f22.15)') -pf%AileronTip, ";", &

            & (-0.75d0 + pf%FlapFractionTip) \* c\_over\_b\_zb(pf, pf%AileronTip)

        write(11, '(f22.15, a, 2x, f22.15)') -pf%AileronTip, ";", &

            & -0.75d0 \* c\_over\_b\_zb(pf, pf%AileronTip)

        ! Close the geometry file

        close(unit=11)

        ! System call to plot planform

        call system('"C:\Program Files (x86)\ESPlot v1.3c\esplot.exe" planform.dat planform.qtp')

    end subroutine PlotPlanform

    subroutine PlotWashout(pf)

        type(Planform), intent(in) :: pf

        integer :: i

        open(unit=11, file='washout.dat')

        write(11, '(a)') "$ Dimensionless Washout Distribution"

        ! Write washout distribution

        do i = 1, pf%NNodes

            write(11, '(f22.15, a, 2x, f22.15)') &

                & z\_over\_b\_i(i, pf%NNodes), ";", pf%Omega(i)

        end do

        close(unit=11)

        call system('"C:\Program Files (x86)\ESPlot v1.3c\esplot.exe" washout.dat washout.qtp')

    end subroutine PlotWashout

    subroutine PlotSectionLiftDistribution(pf)

        type(Planform), intent(in) :: pf

        integer :: i

        real\*8 :: zb, cl(pf%NNodes)

        call GetLiftDistribution(pf, cl)

        open(unit=11, file='liftdistribution.dat')

        write(11, '(a)') "$ Section Lift Distribution"

        do i = 1, pf%NNodes

            zb = z\_over\_b\_i(i, pf%NNodes)

            write(11, '(f22.15, a, 2x, f22.15)') zb, ";", cl(i)

        end do

        close(unit=11)

        call system('"C:\Program Files (x86)\ESPlot v1.3c\esplot.exe" liftdistribution.dat liftdistribution.qtp')

    end subroutine PlotSectionLiftDistribution

    subroutine PlotNormalizedLiftCoefficient(pf)

        type(Planform), intent(in) :: pf

        integer :: i

        real\*8 :: zb, cb, cl1, cl\_over\_cl, cl(pf%NNodes)

        ! Don't normalize if CL1 == 0

        if (Compare(pf%CL1, 0.0d0, zero) == 0) then

            cl1 = 1.0d0

        else

            cl1 = pf%CL1

        end if

        call GetLiftDistribution(pf, cl)

        open(unit=11, file='liftcoefficient.dat')

        write(11, '(a)') "$ Normalized Section Lift Coefficient"

        do i = 1, pf%NNodes

            zb = z\_over\_b\_i(i, pf%NNodes)

            cb = c\_over\_b\_zb(pf, zb)

            if (Compare(cb, 0.0d0, zero) == 0) then

                if (Compare(cl(i), 0.0d0, zero) == 0) then

                    if (pf%WingType == Elliptic) then

                        cl\_over\_cl = NLC\_ZeroChord\_Elliptic(pf, zb, cb, cl1)

                    else if (pf%WingType == Tapered) then

                        cl\_over\_cl = NLC\_ZeroChord\_Tapered(pf, zb, cb, cl1)

                    else if (pf%WingType == Combination) then

                        cl\_over\_cl = NLC\_ZeroChord\_Tapered(pf, zb, cb, cl1)

                    else

                        stop "\*\*\*Unknown Wing Type\*\*\*"

                    end if

                else

                    ! Finite lift from zero-chord section, should never happen

                    cl\_over\_cl = 1.0d0 / zero

                end if

            else

                cl\_over\_cl = cl(i) / cb / cl1

            end if

            write(11, '(f22.15, a, 2x, f22.15)') zb, ";", cl\_over\_cl

        end do

        close(unit=11)

        call system('"C:\Program Files (x86)\ESPlot v1.3c\esplot.exe" liftcoefficient.dat liftcoefficient.qtp')

    end subroutine PlotNormalizedLiftCoefficient

    subroutine GetLiftDistribution(pf, cl)

        type(Planform), intent(in) :: pf

        real\*8, intent(out) :: cl(pf%NNodes)

        integer :: i, j

        real\*8 :: zb, theta

        do i = 1, pf%NNodes

            zb = z\_over\_b\_i(i, pf%NNodes)

            theta = theta\_zb(zb)

            cl(i) = 0.0d0

            do j = 1, pf%NNodes

                cl(i) = cl(i) + pf%BigA(j) \* sin(real(j, 8) \* theta)

            end do

            cl(i) = cl(i) \* 4.0d0

        end do

    end subroutine GetLiftDistribution

    real\*8 function NLC\_ZeroChord\_Elliptic(pf, zb, cb, cl) result(cl\_over\_cl)

        type(Planform), intent(in) :: pf

        real\*8, intent(in) :: zb

        real\*8, intent(in) :: cb

        real\*8, intent(in) :: cl

        integer :: i

        real\*8 :: theta

        theta = theta\_zb(zb)

        cl\_over\_cl = 0.0d0

        do i = 1, pf%NNodes

            cl\_over\_cl = cl\_over\_cl + real(i, 8) \* pf%BigA(i) \* &

                & cos(real(i, 8) \* theta) / cos(theta)

        end do

        cl\_over\_cl = cl\_over\_cl \* pi \* pf%AspectRatio / cl

    end function NLC\_ZeroChord\_Elliptic

    real\*8 function NLC\_ZeroChord\_Tapered(pf, zb, cb, cl) result(cl\_over\_cl)

        type(Planform), intent(in) :: pf

        real\*8, intent(in) :: zb

        real\*8, intent(in) :: cb

        real\*8, intent(in) :: cl

        integer :: i

        real\*8 :: theta, cb2

        if (zb < 0) then

            theta = 1.0d-5

        else

            theta = pi - 1.0d-5

        end if

        cb2 = c\_over\_b(pf, theta)

        cl\_over\_cl = 0.0d0

        do i = 1, pf%NNodes

            cl\_over\_cl = cl\_over\_cl + pf%BigA(i) \* sin(real(i, 8) \* theta)

        end do

        cl\_over\_cl = 4.0d0 \* cl\_over\_cl / cb2 / cl

    end function NLC\_ZeroChord\_Tapered

end module LiftingLineOutput

module LiftingLineSolver\_Test

    use Utilities

    use class\_Planform

    use LiftingLineSetters

    use LiftingLineSolver

    use LiftingLineOutput

    implicit none

contains

    subroutine TestLiftingLineSolver()

        integer :: nerror

        nerror = 0

        nerror = nerror + TestProblem1p34b()

        write(6, \*)

        if (nerror == 0) then

            write(6, '(a)') "SUCCESS - All tests passed."

        else

            write(6, '(a, i3, a)') "FAIL - ", nerror, " tests failed."

        end if

        call system('pause')

    end subroutine TestLiftingLineSolver

    integer function TestProblem1p34b() result(fail)

        type(Planform) :: pf

        integer :: badline

        character\*80 :: fname\_work = "Problem1p34b\_work.txt"

        character\*80 :: fname\_results = "Problem1p34b\_results.txt"

call InitPlanform(pf)

        call SetWingType(pf, Elliptic)

        call SetNNodes(pf, 100)

        call SetSectionLiftSlope(pf, 2.0d0 \* pi)

        call SetAspectRatio(pf, 5.56d0)

        call SetAileronRoot(pf, 0.253d0)

        call SetAileronTip(pf, 0.438d0)

        call SetFlapFractionRoot(pf, 0.28d0)

        call SetFlapFractionTip(pf, 0.25d0)

        call SetHingeEfficiency(pf, 0.85d0)

        call SetFileName(pf, fname\_work)

        call ComputeCMatrixAndCoefficients(pf)

        call OutputPlanform(pf)

        call SetWashout(pf, 2.0d0)

        call SetAileronDeflection(pf, 5.0d0)

        call SetSteadyRollingRate(pf)

        call SetLiftCoefficient(pf, 0.5d0)

        call ComputeFlightConditions(pf)

        call OutputFlightConditions(pf)

        call SetWashout(pf, 2.0d0)

        call SetAileronDeflection(pf, 5.0d0)

        call SetRollingRate(pf, -0.02d0)

        call SetLiftCoefficient(pf, 0.5d0)

        call ComputeFlightConditions(pf)

        call OutputFlightConditions(pf)

        call SetWashout(pf, 2.0d0)

        call SetAileronDeflection(pf, 5.0d0)

        call SetRollingRate(pf, 0.0d0)

        call SetLiftCoefficient(pf, 0.5d0)

        call ComputeFlightConditions(pf)

        call OutputFlightConditions(pf)

        badline = CompareFiles(fname\_work, fname\_results)

        if (badline /= 0) then

            write(6, '(a)') "Test Failed - Problem1p34b"

            write(6, '(2x, a, i3, a)') "Comparison of line ", badline, " failed!"

            fail = 1

        else

            write(6, '(a)') "Test Successful - Problem1p34b"

            fail = 0

        end if

    end function TestProblem1p34b

end module LiftingLineSolver\_Test

module LiftingLineInterface

    use class\_Planform

    use LiftingLineSetters

    use LiftingLineSolver

    use LiftingLineOutput

    use LiftingLineSolver\_Test

    implicit none

contains

    subroutine BeginLiftingLineInterface()

        type(Planform) :: pf

        character\*2 :: inp = 'A'

        call InitPlanform(pf)

        do while(inp /= 'Q')

            inp = PlanformParamters(pf)

            if (inp == 'A') then

                call ComputeCMatrixAndCoefficients(pf)

                call OutputPlanform(pf)

                do while(inp /= 'Q' .and. inp /= 'B')

                    inp = OperatingConditions(pf)

                    if (inp /= 'Q') then

                        call UpdateOperatingConditions(pf, inp)

                    end if

                end do

            else if (inp /= 'Q') then

                call UpdatePlanformParameters(pf, inp)

            end if

        end do

    end subroutine BeginLiftingLineInterface

    character\*2 function PlanformParamters(pf) result(inp)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        ! Clear the screen and output the header

        call system('cls')

        call OutputHeader()

        ! Display options to user

        write(6, '(28x, a)') "Planform Design Menu"

        write(6, \*)

        write(6, '(a)') "Select from the following menu options:"

        write(6, \*)

        ! Wing parameters

        write(6, '(2x, a)') "Wing Parameters:"

        msg = "WT - Edit wing type"

        call DisplayMessageWithTextDefault(msg, GetWingType(pf), 4)

        msg = "N  - Edit number of nodes per semispan"

        call DisplayMessageWithIntegerDefault(msg, (pf%NNodes + 1) / 2, 4)

        msg = "RA - Edit aspect ratio"

        call DisplayMessageWithRealDefault(msg, pf%AspectRatio, 4)

        if (pf%WingType == Tapered) then

            msg = "RT - Edit taper ratio"

            call DisplayMessageWithRealDefault(msg, pf%TaperRatio, 4)

        end if

        msg = "S  - Edit section lift slope"

        call DisplayMessageWithRealDefault(msg, pf%SectionLiftSlope, 4)

        if (pf%WingType == Combination) then

            msg = "TZ - Edit z/b at the transition from tapered to elliptic"

            call DisplayMessageWithRealDefault(msg, pf%TransitionPoint, 4)

            msg = "TC - Edit c/croot at the transition from tapered to elliptic"

            call DisplayMessageWithRealDefault(msg, pf%TransitionChord, 4)

        end if

        if (pf%WingType /= Elliptic) then

            msg = "WD - Toggle washout distribution type"

            call DisplayMessageWithTextDefault(msg, GetWashoutDistributionType(pf), 4)

        end if

        ! Aileron parameters

        write(6, \*)

        write(6, '(2x, a)') "Aileron Parameters:"

        msg = "ZR - Edit z/b of aileron root"

        call DisplayMessageWithRealDefault(msg, pf%AileronRoot, 4)

        msg = "ZT - Edit z/b of aileron tip"

        call DisplayMessageWithRealDefault(msg, pf%AileronTip, 4)

        msg = "PH - Make hinge line parallel with quarter-chord line?"

        call DisplayMessageWithLogicalDefault(msg, pf%ParallelHingeLine, 4)

        msg = "CR - Edit cf/c of aileron root"

        call DisplayMessageWithRealDefault(msg, pf%FlapFractionRoot, 4)

        msg = "CT - Edit cf/c of aileron tip"

        call DisplayMessageWithRealDefault(msg, pf%FlapFractionTip, 4)

        msg = "HE - Edit aileron hinge efficiency"

        call DisplayMessageWithRealDefault(msg, pf%HingeEfficiency, 4)

        ! Output and Plotting options

        write(6, \*)

        write(6, '(2x, a)') "Output and Plotting Options:"

        msg = "C  - Output C matrix and Fourier Coefficients?"

        call DisplayMessageWithLogicalDefault(msg, pf%OutputMatrices, 4)

        msg = "F  - Edit output file name"

        call DisplayMessageWithTextDefault(msg, pf%FileName, 4)

        write(6, '(4x, a)') "PP - Plot planform in ES-Plot"

        ! Main Execution commands

        write(6, \*)

        write(6, '(2x, a)') "A - Advance to Operating Conditions Menu"

        write(6, '(2x, a)') "T - Test solver against Problem 1.34b solution"

        write(6, '(2x, a)') "Q - Quit"

        write(6, \*)

        write(6, '(a)') "Your selection: "

        inp = GetCharacterInput("  ")

        write(6, \*)

    end function PlanformParamters

    character\*2 function OperatingConditions(pf) result(inp)

        type(Planform), intent(inout) :: pf

        integer :: i

        character\*80 :: msg

        ! Clear the screen and output the header

        call system('cls')

        call OutputHeader()

        ! Output the Planform summary

        call OutputPlanformSummary(6, pf)

        call OutputOperatingConditions(6, pf)

        call OutputFlightCoefficients(6, pf)

        write(6, '(80a)') ("\*", i=1,80)

        ! Display options to user

        write(6, '(28x, a)') "Operating Conditions Menu"

        write(6, \*)

        write(6, '(a)') "Select from the following menu options:"

        ! Operating Conditions

        write(6, \*)

        write(6, '(2x, a)') "Operating Conditions:"

        msg = "AA - Edit root aerodynamic angle of attack"

        call DisplayMessageWithAngleDefault(msg, pf%AngleOfAttack, 4)

        msg = "CL - Edit coefficient of lift"

        call DisplayMessageWithRealDefault(msg, pf%LiftCoefficient, 4)

        msg = "OW - Use optimum total washout"

        call DisplayMessageWithLogicalDefault(msg, pf%UseOptimumWashout, 4)

        msg = "W  - Edit total amount of washout"

        call DisplayMessageWithAngleDefault(msg, pf%Washout, 4)

        msg = "AD - Edit aileron deflection"

        call DisplayMessageWithAngleDefault(msg, pf%AileronDeflection, 4)

        msg = "SR - Use steady dimensionless rolling rate"

        call DisplayMessageWithLogicalDefault(msg, pf%UseSteadyRollingRate, 4)

        msg = "R  - Edit dimensionless rolling rate"

        call DisplayMessageWithRealDefault(msg, pf%RollingRate, 4)

        ! Plotting options

        write(6, \*)

        write(6, '(2x, a)') "Plotting Options:"

        write(6, '(4x, a)') "PP - Plot Planform in ES-Plot"

        write(6, '(4x, a)') "PW - Plot Dimensionless Washout Distribution in ES-Plot"

        write(6, '(4x, a)') "PL - Plot Section Lift Distribution in ES-Plot"

        write(6, '(4x, a)') "PN - Plot Normalized Section Lift Coefficient in ES-Plot"

        ! Main Execution commands

        write(6, \*)

        msg = "S - Save Flight coefficients to output file"

        call DisplayMessageWithTextDefault(msg, pf%FileName, 2)

        write(6, '(2x, a)') "B - Back to Planform Design Menu"

        write(6, '(2x, a)') "Q - Quit"

        write(6, \*)

        write(6, '(a)') "Your selection: "

        inp = GetCharacterInput("  ")

        write(6, \*)

    end function OperatingConditions

    subroutine UpdatePlanformParameters(pf, input)

        type(Planform), intent(inout) :: pf

        character\*2, intent(in) :: input

        ! Process input command

        ! Wing parameters

        if (input == 'WT') then

            call EditWingType(pf)

        else if (input == 'N') then

            call EditNNodes(pf)

        else if (input == 'RA') then

            call EditAspectRatio(pf)

        else if (input == 'RT' .and. pf%WingType == Tapered) then

            call EditTaperRatio(pf)

        else if (input == 'S') then

            call EditLiftSlope(pf)

        else if (input == 'TZ' .and. pf%WingType == Combination) then

            call EditTransitionPoint(pf)

        else if (input == 'TC' .and. pf%WingType == Combination) then

            call EditTransitionChord(pf)

        else if (input == 'WD' .and. pf%WingType /= Elliptic) then

            call EditWashoutDistribution(pf)

        ! Aileron parameters

        else if (input == 'ZR') then

            call EditAileronRoot(pf)

        else if (input == 'ZT') then

            call EditAileronTip(pf)

        else if (input == 'PH') then

            call ToggleParallelHinge(pf)

        else if (input == 'CR') then

            call EditFlapFractionRoot(pf)

        else if (input == 'CT') then

            call EditFlapFractionTip(pf)

        else if (input == 'HE') then

            call EditHingeEfficiency(pf)

        ! Output options

        else if (input == 'C') then

            pf%OutputMatrices = .not. pf%OutputMatrices

        else if (input == 'F') then

            call EditFileName(pf)

        else if (input == 'PP') then

            call PlotPlanform(pf)

        ! Testing options

        else if (input == 'T') then

            call TestLiftingLineSolver()

        end if

    end subroutine UpdatePlanformParameters

    subroutine UpdateOperatingConditions(pf, input)

        type(Planform), intent(inout) :: pf

        character\*2, intent(in) :: input

        ! Operating Conditions

        if (input == 'AA') then

            call EditAngleOfAttack(pf)

        else if (input == 'CL') then

            call EditLiftCoefficient(pf)

        else if (input == 'OW') then

            call ToggleUseOptimumWashout(pf)

        else if (input == 'W') then

            call EditWashout(pf)

        else if (input == 'AD') then

            call EditAileronDeflection(pf)

        else if (input == 'SR') then

            call ToggleUseSteadyRollingRate(pf)

        else if (input == 'R') then

            call EditRollingRate(pf)

        ! Output and Plotting options

        else if (input == 'PP') then

            call PlotPlanform(pf)

        else if (input == 'PW') then

            call PlotWashout(pf)

        else if (input == 'PL') then

            call PlotSectionLiftDistribution(pf)

        else if (input == 'PN') then

            call PlotNormalizedLiftCoefficient(pf)

        else if (input == 'S') then

            call OutputFlightConditions(pf)

        end if

        call ComputeFlightConditions(pf)

    end subroutine UpdateOperatingConditions

    subroutine EditWingType(pf)

        type(Planform), intent(inout) :: pf

        logical :: cont

        character\*2 :: inp

        write(6, \*)

        write(6, '(a)') "Select from the following wing type options:"

        write(6, '(2x, a)') "T - Tapered"

        write(6, '(2x, a)') "E - Elliptic"

        write(6, '(2x, a)') "C - Combination (Tapered with elliptic tip)"

        write(6, \*)

        write(6, '(a)') "Your selection: "

        cont = .true.

        do while(cont)

            inp = GetCharacterInput("  ")

            write(6, \*)

            if (inp == "T") then

                call SetWingType(pf, Tapered)

                cont = .false.

            else if (inp == "E") then

                call SetWingType(pf, Elliptic)

                cont = .false.

            else if (inp == "C") then

                call SetWingType(pf, Combination)

                cont = .false.

            else

                write(6, '(a)') "Invalid input, please make a selection from the above menu."

            end if

        end do

    end subroutine EditWingType

    subroutine EditTransitionPoint(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        real\*8 :: tp\_old

        logical :: isValid

        tp\_old = pf%TransitionPoint

        msg = "Enter z/b at the transition point from tapered to elliptic"

        call DisplayMessageWithRealDefault(msg, pf%TransitionPoint, 0)

        call SetTransitionPoint(pf, GetRealInput(0.0d0, 0.5d0, pf%TransitionPoint))

        isValid = AreCombinationWingCoefficientsValid(pf)

        do while(.not. isValid)

            call SetTransitionPoint(pf, tp\_old)

            write(6, \*)

            write(6, '(a)') "The input provided results in invalid ellipse coefficients."

            write(6, '(a)') "Try a new value or press <ENTER> to accept default."

            call SetTransitionPoint(pf, GetRealInput(0.0d0, 0.5d0, pf%TransitionPoint))

            isValid = AreCombinationWingCoefficientsValid(pf)

        end do

    end subroutine EditTransitionPoint

    subroutine EditTransitionChord(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        real\*8 :: tc\_old

        logical :: isValid

        tc\_old = pf%TransitionChord

        msg = "Enter c/croot at the transition point from tapered to elliptic"

        call DisplayMessageWithRealDefault(msg, pf%TransitionChord, 0)

        call SetTransitionChord(pf, GetRealInput(0.0d0, 10.0d0, pf%TransitionChord))

        isValid = AreCombinationWingCoefficientsValid(pf)

        do while(.not. isValid)

            call SetTransitionChord(pf, tc\_old)

            write(6, \*)

            write(6, '(a)') "The input provided results in invalid ellipse coefficients."

            write(6, '(a)') "Try a new value or press <ENTER> to accept default."

            call SetTransitionChord(pf, GetRealInput(0.0d0, 2.0d0, pf%TransitionChord))

            isValid = AreCombinationWingCoefficientsValid(pf)

        end do

    end subroutine EditTransitionChord

    subroutine EditWashoutDistribution(pf)

        type(Planform), intent(inout) :: pf

        if (pf%WashoutDistribution == Linear) then

            call SetWashoutDistribution(pf, Optimum)

        else

            call SetWashoutDistribution(pf, Linear)

        end if

    end subroutine EditWashoutDistribution

    subroutine EditNNodes(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        integer :: npss

        character\*80 :: int\_str

        npss = (pf%NNodes + 1) / 2

        msg = "Enter number of nodes per semispan or press <ENTER> to accept default"

        call DisplayMessageWithIntegerDefault(msg, npss, 0)

        call SetNNodes(pf, GetIntInput(4, 1000, npss))

    end subroutine EditNNodes

    subroutine EditAspectRatio(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new aspect ratio or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%AspectRatio, 0)

        call SetAspectRatio(pf, GetRealInput(1.0d0, 100.0d0, pf%AspectRatio))

    end subroutine EditAspectRatio

    subroutine EditTaperRatio(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new taper ratio or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%TaperRatio, 0)

        call SetTaperRatio(pf, GetRealInput(0.0d0, 100.0d0, pf%TaperRatio))

    end subroutine EditTaperRatio

    subroutine EditLiftSlope(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new section lift slope or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%SectionLiftSlope, 0)

        call SetSectionLiftSlope(pf, GetRealInput(-100.0d0 \* pi, 100.0d0 \* pi, &

            & pf%SectionLiftSlope))

    end subroutine EditLiftSlope

    subroutine EditAileronRoot(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new z/b for aileron root or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%AileronRoot, 0)

        call SetAileronRoot(pf, GetRealInput(0.0d0, pf%AileronTip, pf%AileronRoot))

    end subroutine EditAileronRoot

    subroutine EditAileronTip(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new z/b for aileron tip or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%AileronTip, 0)

        call SetAileronTip(pf, GetRealInput(pf%AileronRoot, 0.5d0, pf%AileronTip))

    end subroutine EditAileronTip

    subroutine EditFlapFractionRoot(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        if (pf%ParallelHingeLine) then

            write(6, '(a)') "NOTE: Hinge is no longer constrained to be parallel with quarter-chord line."

        end if

        msg = "Enter new cf/c at aileron root or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%DesiredFlapFractionRoot, 0)

        call SetFlapFractionRoot(pf, GetRealInput(0.0d0, 1.0d0, pf%DesiredFlapFractionRoot))

    end subroutine EditFlapFractionRoot

    subroutine EditFlapFractionTip(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter new cf/c at aileron tip or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%FlapFractionTip, 0)

        call SetFlapFractionTip(pf, GetRealInput(0.0d0, 1.0d0, pf%FlapFractionTip))

    end subroutine EditFlapFractionTip

    subroutine ToggleParallelHinge(pf)

        type(Planform), intent(inout) :: pf

        if (pf%ParallelHingeLine) then

            call SetFlapFractionRoot(pf, pf%DesiredFlapFractionRoot)

        else

            call SetParallelHingeLine(pf)

        end if

    end subroutine ToggleParallelHinge

    subroutine EditHingeEfficiency(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter aileron hinge efficiency or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%HingeEfficiency, 0)

        call SetHingeEfficiency(pf, GetRealInput(0.0d0, 1.0d0, pf%HingeEfficiency))

    end subroutine EditHingeEfficiency

    subroutine EditDeflectionEfficiency(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter deflection efficiency or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%DeflectionEfficiency, 0)

        call SetDeflectionEfficiency(pf, GetRealInput(0.0d0, 1.0d0, &

            & pf%DeflectionEfficiency))

    end subroutine EditDeflectionEfficiency

    subroutine EditFileName(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter output file name or press <ENTER> to accept default"

        call DisplayMessageWithTextDefault(msg, pf%FileName, 0)

        call SetFileName(pf, GetStringInput(pf%FileName))

    end subroutine EditFileName

    subroutine EditAngleOfAttack(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        write(6, '(a, a)') "NOTE: This operation will calculate a new lift ", &

            & "coefficient and optimum washout."

        msg = "Enter angle of attack or press <ENTER> to accept default"

        call DisplayMessageWithAngleDefault(msg, pf%DesiredAngleOfAttack, 0)

        call SetAngleOfAttack(pf, GetRealInput(-12.0d0, 12.0d0, &

            & pf%DesiredAngleOfAttack \* 180.0d0 / pi))

    end subroutine EditAngleOfAttack

    subroutine EditLiftCoefficient(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        real\*8 :: mn, mx, dflt

        mn = CL1(pf%CLa, -12.0d0 \* pi / 180.0d0, pf%EW, pf%Washout)

        mx = CL1(pf%CLa,  12.0d0 \* pi / 180.0d0, pf%EW, pf%Washout)

        if (pf%DesiredLiftCoefficient < mn) then

            dflt = mn

        else if (pf%DesiredLiftCoefficient > mx) then

            dflt = mx

        else

            dflt = pf%DesiredLiftCoefficient

        end if

        write(6, \*)

        write(6, '(a, a)') "NOTE: This operation will calculate a new alpha ", &

            & "and optimum washout"

        msg = "Enter lift coefficient or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, dflt, 0)

        call SetLiftCoefficient(pf, GetRealInput(mn, mx, dflt))

    end subroutine EditLiftCoefficient

    subroutine EditWashout(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        if (pf%UseOptimumWashout) then

            write(6, '(a)') "NOTE: Use of optimum total washout has been disabled."

        end if

        msg = "Enter total washout or press <ENTER> to accept default"

        call DisplayMessageWithAngleDefault(msg, pf%DesiredWashout, 0)

        call SetWashout(pf, GetRealInput(-12.0d0, 12.0d0, pf%DesiredWashout \* 180.0d0 / pi))

    end subroutine EditWashout

    subroutine ToggleUseOptimumWashout(pf)

        type(Planform), intent(inout) :: pf

        if (pf%UseOptimumWashout) then

            call SetWashout(pf, pf%DesiredWashout \* 180.0d0 / pi)

        else

            call SetOptimumWashout(pf)

        end if

    end subroutine ToggleUseOptimumWashout

    subroutine EditAileronDeflection(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        msg = "Enter aileron deflection or press <ENTER> to accept default"

        call DisplayMessageWithAngleDefault(msg, pf%AileronDeflection, 0)

        call SetAileronDeflection(pf, GetRealInput(-12.0d0, 12.0d0, &

            & pf%AileronDeflection \* 180.0d0 / pi))

    end subroutine EditAileronDeflection

    subroutine ToggleUseSteadyRollingRate(pf)

        type(Planform), intent(inout) :: pf

        pf%UseSteadyRollingRate = .not. pf%UseSteadyRollingRate

        if (pf%UseSteadyRollingRate) then

            call SetSteadyRollingRate(pf)

        else

            call SetRollingRate(pf, pf%DesiredRollingRate)

        end if

    end subroutine ToggleUseSteadyRollingRate

    subroutine EditRollingRate(pf)

        type(Planform), intent(inout) :: pf

        character\*80 :: msg

        write(6, \*)

        if (pf%UseSteadyRollingRate) then

            write(6, '(a)') "NOTE: Use of steady rolling rate has been disabled."

        end if

        msg = "Enter dimensionless rolling rate or press <ENTER> to accept default"

        call DisplayMessageWithRealDefault(msg, pf%DesiredRollingRate, 0)

        call SetRollingRate(pf, GetRealInput(-100.0d0, 100.0d0, pf%DesiredRollingRate))

    end subroutine EditRollingRate

    subroutine DisplayMessageWithRealDefault(msg, dflt, tab)

        character\*80, intent(in) :: msg ! Message to be displayed

        real\*8, intent(in) :: dflt ! Default value to show in parenthesis

        integer, intent(in) :: tab ! Size of indentation to use

        call DisplayMessageWithTextDefault(msg, FormatReal(dflt, 5), tab)

    end subroutine DisplayMessageWithRealDefault

    subroutine DisplayMessageWithAngleDefault(msg, dflt, tab)

        character\*80, intent(in) :: msg ! Message to be displayed

        real\*8, intent(in) :: dflt ! Default value to show in parenthesis

        integer, intent(in) :: tab ! Size of indentation to use

        character\*80 :: dflt\_deg

        write(dflt\_deg, '(a, a)') trim(FormatReal(dflt \* 180.0d0 / pi, 5)), " degrees"

        call DisplayMessageWithTextDefault(msg, dflt\_deg, tab)

    end subroutine DisplayMessageWithAngleDefault

    subroutine DisplayMessageWithIntegerDefault(msg, dflt, tab)

        character\*80, intent(in) :: msg ! Message to be displayed

        integer, intent(in) :: dflt ! Default value to show in parenthesis

        integer, intent(in) :: tab ! Size of indentation to use

        call DisplayMessageWithTextDefault(msg, FormatInteger(dflt), tab)

    end subroutine DisplayMessageWithIntegerDefault

    subroutine DisplayMessageWithTextDefault(msg, dflt, tab)

        character\*80, intent(in) :: msg ! Message to be displayed

        character\*80, intent(in) :: dflt ! Default value to show in parenthesis

        integer, intent(in) :: tab ! Size of indentation to use

        character\*80 :: msg\_fmt

        if (tab == 0) then

            msg\_fmt = "(a, a, a, a)"

        else

            write(msg\_fmt, '(a, i1, a)') "(", tab, "x, a, a, a, a)"

        end if

        write(6, msg\_fmt) trim(msg), " ( ", trim(dflt), " )"

    end subroutine DisplayMessageWithTextDefault

    subroutine DisplayMessageWithLogicalDefault(msg, dflt, tab)

        character\*80, intent(in) :: msg ! Message to be displayed

        logical, intent(in) :: dflt ! Default value to show in parenthesis

        integer, intent(in) :: tab ! Size of indentation to use

        character\*80 :: tf

        if (dflt) then

            tf = "True"

        else

            tf = "False"

        end if

        call DisplayMessageWithTextDefault(msg, tf, tab)

    end subroutine DisplayMessageWithLogicalDefault

end module LiftingLineInterface

program PrandtlsLiftingLine

    use LiftingLineInterface

    implicit none

    !Begin execution

    call BeginLiftingLineInterface()

end program