AP-266 Aerostructural Optimization
Final Exam

Cap Eng **Ney** Rafael Secco
ney@ita.br

June 14th, 2019

Instructions

- Due date: July 7th, 2019. 11:59 PM.
- Send the solutions to ney@ita.br.
- Send the source code as well. The code should have comments.
- You may discuss solution approaches with other students, but you must write your own solution. Do NOT share your code.
- Late delivery penalty: 20% of the total score per late day.
- You will have to use Tapenade for this assignment. So do not start it too close to the due date!

1 Introduction

It is finally time to run an aerostructural optimization. We developed the aerodynamic shape optimization tool during the first semester and we studied structural optimization during the second semester. For this exercise, I will provide the structural optimization code and the codes that transfer displacements and forces between the aerodynamic and structural modules. You will have to tie these codes together and then run the aerostructural optimization. Our main goal here, besides running an aerostructural optimization, is to understand how different objective functions impact the outcome of the optimization problem.

2 Problem description

One important metric regarding the performance of an airplane is how much fuel it uses for a given mission. This is what we will call **fuel burn** during this text.

We can compute the fuel burn FB of an airplane flying for a time T with:

$$FB = W_0 \cdot \left(1 - \exp\left(-\frac{T \cdot C}{L/D}\right)\right) \tag{1}$$

Where W_0 is the initial weight of the airplane, C is the thrust-specific fuel consumption of the engine and L/D is the aerodynamic efficiency of the airplane during this flight phase. The aerodynamic efficiency (L/D) depends on which lift coefficient (C_L) the airplane is flying with. If we enforce level flight:

$$L = W_0 \tag{2}$$

In reality, the airplane weight varies as fuel is consumed, but this approximation is enough for our studies.

We can break the initial airplane weight (W_0) into three contributions:

$$W_0 = W_s + W_f + FB \tag{3}$$

where W_s is the wing structure weight and W_f is a fixed weight accounting for payload, systems, and the remaining structures. We can substitute Eq. (3) into Eq. (1) to remove the implicit relationship between FB and W_0, giving:

$$FB = (W_s + W_f) \cdot \left(\exp\left(\frac{T \cdot C}{L/D}\right) - 1\right) \tag{4}$$

For this exercise, assume that we have an airplane with rectangular wing whose area and span has already been determined. We have freedom to select the **twist distribution** along the wing.

This wing will also use a tubular beam to support aerodynamic loads. The radius of this tube is already determined by the airfoil thickness, but we are allowed to vary the **thickness distribution** along the tube.

Once we apply the aerodynamic forces to the structural model, we can compute Von-Mises stresses σ along the beam elements. Therefore, we can compute the safety margin of the i-th section with:

$$m_i = 1 - \frac{\sigma_i}{\sigma_Y} \tag{5}$$

where σ_Y is the yield stress of the beam material. The structure fails if $m_s < 0$, so we need to make sure that the smallest value of σ_i of the structure is positive. Instead of creating constraints for every single element, we can use the KS function to aggregate all safety margins in a single value. The KS function estimates the minimum value of a set of margins m_i with:

$$m_{\text{KS}} = -\frac{1}{\rho_{\text{KS}}} \ln \left(\sum_i e^{-\rho_{\text{KS}} \cdot m_i} \right) \tag{6}$$

where ρ_{KS} controls the quality of the estimate. Increasing ρ_{KS} improves the estimate of the minimum value, but this may also lead to numerical issues. If $m_{\text{KS}} \geq 0$, it follows that all $m_i \geq 0$. Thus we only need to set a single constraint for m_{KS}.

Even though we compute the lift distribution for level flight ($L = W_0$) we need to take into account that the structure should also support maneuvering conditions. One way to define maneuvering loads is by using the load factor n_f:

$$n_f = \frac{L}{W_0} \tag{7}$$

The load factor indicates how many times the lift is greater than the airplane weight. Regulatory agencies define the limit load factors that the structure should stand, and this value is usually around 2.5.

We usually solve the aerostructural analysis for a load factor of 1.0 to satisfy Eq. (2). However, we need to enforce the stress constraints for a load factor of 2.5. One way is to run a second aerostructural analysis for this condition, but we can use the fact that our models are (roughly) linear to avoid this additional analysis. If we increase the lift by a factor n_f, then the structural stresses will also increase by a factor n_f. So the new safety margin m_i^* will be:

$$m_i^* = 1 - \frac{n_f \cdot \sigma_i}{\sigma_Y} \tag{8}$$

we can manipulate this expression to get:

$$m_i^* = n_f \left(1 - \frac{\sigma_i}{\sigma_Y} \right) + 1 - n_f \tag{9}$$

We can use Eq. (5) to correlate the original margin (m_i) with the margin modified by the load factor (m_i^*):

$$m_i^* = n_f \cdot m_i + 1 - n_f \tag{10}$$

The same process can be used to correct the aggregated margin used for the failure constraint:

$$m_{\text{KS}}^* = n_f \cdot m_{\text{KS}} + 1 - n_f \tag{11}$$

Now we have all pieces necessary to define an aerostructural optimization problem:

Table 1: Aerostructural optimization problem. n_p is the number of panels.

	Variable/function	Description	Quantity
minimize	FB	Fuel burn	
with respect to	$\alpha_{0,i}$	Twist angle of panels [deg]	n_p
	$t_i \geq 0.001$	Beam thickness distribution [m]	n_p
		Total design variables	$2n_p$
subject to	$L = W_0$	Lift constraint	1
	$m_{KS}^* \geq 0$	Structural feasibility	1
		Total constraints	**2**

3 Description of modules and subroutines

We have four important modules and subroutines that will be used in this problem. This section explains each one of them.

The code of these modules and subroutines will be given to you. Check their headers for more information about the inputs and outputs of each one of them.

3.1 Lifting line theory module (llt_module)

We will use the same Lifting Line Theory (LLT) module for aerodynamic analysis in this exercise. The inputs and outputs of this module are described in Fig. 1.

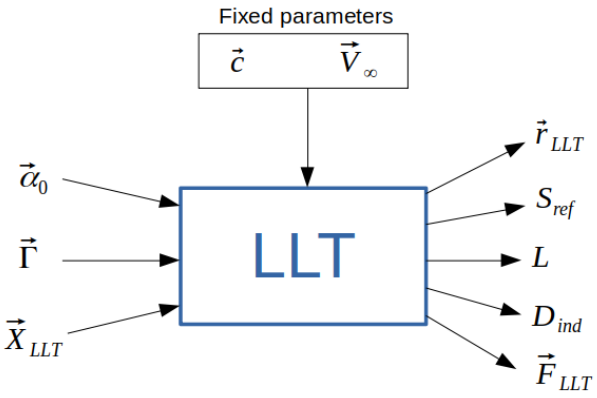

Figure 1: Inputs and outputs of the LLT module. Some fixed parameters are not mentioned for simplicity.

The variables used by this module are:
Fixed Parameters:

- $\vec{c}$: array of panel chords
- $\vec{V}_\infty$: free-stream velocity vector

Input Parameters:

- $\vec{\alpha}_0$: array of panel incidence angles
- $\vec{\Gamma}$: array of panel circulations
- $\vec{X}_{\text{LLT}}$: array with coordinates of the horseshoe vortices kinks

Output Parameters:

- $\vec{r}_{\text{LLT}}$: residuals of the LLT formulation
- S_{ref}: reference area obtained by summing areas of all panels
- L: total lift force
- D_{ind}: total induced drag force
- $\vec{F}_{\text{LLT}}$: load distribution (force per unit span) on every panel

An aerodynamic analysis consists of finding which value of $\vec{\Gamma}$ results in $\vec{r}_{\text{LLT}} = 0$.

Remember that the lifting line theory only predicts the induced drag of the wing. We need to include additional drag contributions to find the total drag of the airplane. Assuming that the value of the parasite drag coefficient of the airplane (C_{D0}) is given to us, we can estimate the total drag with:

$$D = D_{\text{ind}} + \frac{\rho_{\text{air}} \cdot \left|\vec{V}_\infty\right|^2}{2} \cdot S_{\text{ref}} \cdot C_{D0} \tag{12}$$

where ρ_{air} is the air density at the desired flight altitude.

3.2 Beam analysis module (fem_module)

You will receive the Finite Element Method (FEM) module for structural analysis in this exercise. The inputs and outputs of this module are described in Fig. 2.

Tubular beam elements are distributed along the bound vortices. This way, we can use the same kink coordinates of the horseshoe vortices as nodes of the beam elements.

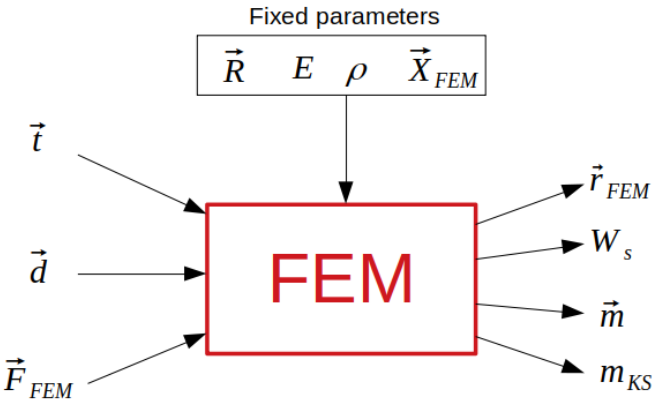

Figure 2: Inputs and outputs of the FEM module.

The variables used by this module are:

Fixed Parameters:

- $\vec{R}$: array with the radii of each beam element
- E: Young's module of the beam material
- ρ: density of the beam material
- $\vec{X}_{\text{FEM}}$: array with nodal coordinates of the undeformed structure

Input Parameters:

- $\vec{t}$: array with the thicknesses of each beam element
- $\vec{d}$: array with nodal displacements and rotations
- $\vec{F}_{\text{FEM}}$: array with nodal forces and moments

Output Parameters:

- $\vec{r}_{\text{FEM}}$: residuals of the FEM formulation
- W_s: weight of the structure
- $\vec{m}$: array of failure margins computed at the nodes of each element
- m_{KS}: critical failure margin estimated by the KS function

An structural analysis consists of finding which value of $\vec{d}$ results in $\vec{r}_{\text{FEM}} = 0$.

3.3 Displacement transfer subroutine (transfer_disps)

The displacements should be transferred to the aerodynamic model. You will receive the subroutine that executes this task. The inputs and outputs of this module are described in Fig. 3.

The aerodynamic mesh coincides with the structural mesh, we can. This way, we just take the FEM nodes and apply the vertical displacements $\vec{d}$ to generate the deformed structure for the LLT module.

The variables used by this module are:

Input Parameters:

- $\vec{X}_{\text{FEM}}$: array with nodal coordinates of the undeformed structure
- $\vec{d}$: array with nodal displacements and rotations

Output Parameters:

- $\vec{X}_{\text{LLT}}$: array with coordinates of the horseshoe vortices kinks

Figure 3: Inputs and outputs of the displacement transfer subroutine.

3.4 Load transfer subroutine (transfer_forces)

The aerodynamic forces should be transferred to the structural model. You will receive the subroutine that executes this task as well. The inputs and outputs of this module are described in Fig. 4.

Figure 4: Inputs and outputs of the force transfer subroutine.

Since the aerodynamic mesh coincides with the structural mesh, we can take the constant lift distribution along each bound vortex and convert it into consistent forces and moments applied at the end nodes (Fig. 5).

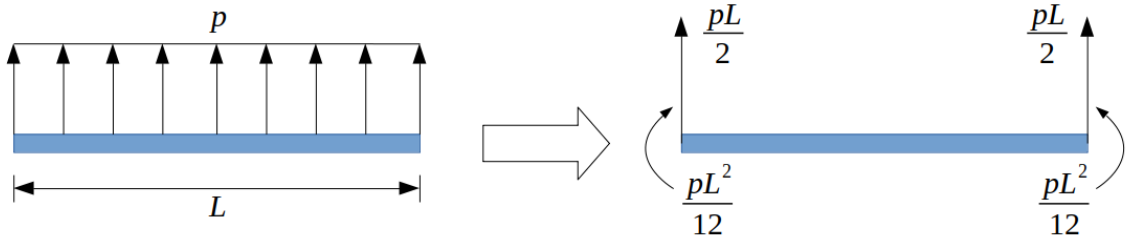

Figure 5: Conversion of distributed aerodynamic forces of each bound vortex into consistent nodal forces and moments.

The variables used by this module are:
Input Parameters:

- $\vec{X}_{\text{FEM}}$: array with nodal coordinates of the undeformed structure
- $\vec{F}_{\text{LLT}}$: load distribution (force per unit span) on every panel

Output Parameters:

- $\vec{F}_{\text{FEM}}$: array with nodal forces and moments

4 Aerostructural analysis module

Now that we have introduced all components, we need to couple them in an ordered manner for the aerostructural analysis. In the end, we want an Aerostructural Analysis (ASA) module with the interface presented in Fig. 6.
Fixed Parameters:

- $\vec{c}$: array of panel chords
- $\vec{V}_{\infty}$: free-stream velocity vector
- $\vec{R}$: array with the radii of each beam element
- E: Young's module of the beam material
- ρ: density of the beam material

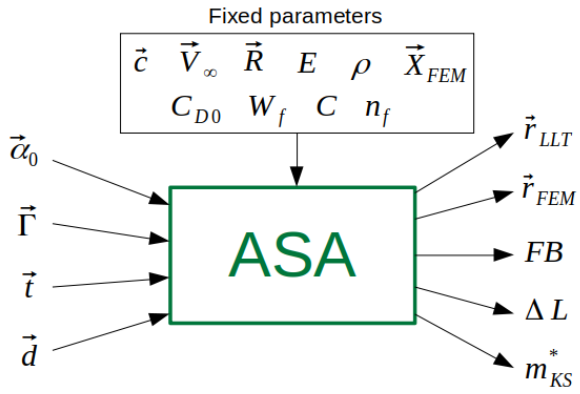

Figure 6: Inputs and outputs of the ASA module. Some fixed parameters are not mentioned for simplicity.

- $\vec{X}_{\text{FEM}}$: array with nodal coordinates of the undeformed structure
- C_{D0}: parasite drag of the airplane
- W_f: fixed weight component
- C: specific fuel consumption of the engine
- n_f: load factor for structural sizing

Input Parameters:

- $\vec{\alpha}_0$: free-stream velocity vector
- $\vec{\Gamma}$: array of panel circulations
- $\vec{t}$: array with the thicknesses of each beam element
- $\vec{d}$: array with nodal displacements and rotations

Output Parameters:

- $\vec{r}_{\text{LLT}}$: residuals of the LLT formulation
- $\vec{r}_{\text{FEM}}$: residuals of the FEM formulation
- FB: fuel burn
- ΔL: lift excess given by:

$$\Delta L = \frac{L}{W_0} - 1 \tag{13}$$

- m^*_{KS}: critical failure margin at the load factor n_f estimated by the KS function

The ASA module should call the modules and subroutines from Sec 3 to compute all required outputs.

5 Activities

This sections describes the activities necessary to implement the aerostructural optimization code.

5.1 Starting point

You will receive the following files in the package:

- `llt_module.f90`: File containing the LLT module.
- `fem_module.f90`: File containing the FEM module.
- `asa_module.f90`: File where we will implement the ASA module. It already contains the displacement and force transfer subroutines.

- `build_f2py_asa.sh`: Script that compiles the ASA module and generates its Python interface.
- `build_f2py_diff_asa.sh`: Script that compiles the ASA module and its differentiated versions. It also generates the Python interface for them.
- `f2py_f2cmap`: Auxiliary file for the Python-Fortran interface.
- `ADFirstAidKit`: Folder with auxiliary files for the differentiated code.

5.2 Implementing the ASA module

Our first step is the implementation of the ASA module.

The `asa_module.f90` file contains the definition of the `asa_analysis` subroutine. We will add code to this subroutine to compute the outputs described in its header.

Draw a block diagram connecting the inputs and outputs of the ASA module while using the blocks presented in Sec. 3. You may need some additional equations to complete all outputs shown in Fig. 6. You can create new blocks in you diagram to represent these equations. If you use equations from this document, use the equation number as the block name.

Once you have the full picture in mind we can start writing the code. Follow these steps to complete the `asa_analysis` subroutine:

1. Make calls to the modules and subroutines following the order you established in your block diagram. Remember that the blocks correspond to the following subroutines:
   - LLT block ⇒ `tapenade_main` subroutine from `llt_module.f90`
   - FEM block ⇒ `fem_main` subroutine from `fem_module.f90`
   - DISPS block ⇒ `transfer_disps` subroutine from the `asa_module.f90`
   - FORCES block ⇒ `transfer_forces` subroutine from the `asa_module.f90`
2. Use Eq. (10) to convert the margins given by the FEM module into margins corrected by the load factor n_f.
3. Use Eq. (11) to convert the KS margin given by the FEM module into a KS margin corrected by the load factor n_f.
4. Compute the magnitude of `Vinf` (variable represented by $\vec{V}_\infty$ in this document).
5. Compute the total drag of the aircraft with Eq. (12).
6. Compute the fuel burn with Eq. (4).
7. Compute the total weight of the aircraft with Eq. (3).
8. Compute the lift excess with Eq. (13).

All necessary subroutines from other modules are already imported for your convenience (check the `use` statements before the variables definition).

Once you complete the subroutine, you can compile and generate its Python interface with:

```
$ bash build_f2py_asa.sh
```

Check if you can import the `asa_module` in Python and then test it with the following inputs:

- **Geometric parameters:**
- $\mathbf{X}_{\text{FEM}} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 4.0 & 6.0 & 8.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$ [m]
- **Aerodynamic parameters:**
- `Gama = [80.0 80.0 80.0 80.0]`
- `alpha0 = [0.0 0.0 0.0 0.0]` [rad]
- `chords = [1.0 1.0 1.0 1.0]` [m]
- `cl0 = [0.0 0.0 0.0 0.0]`

- `cla = [6.283 6.283 6.283 6.283]` [1/rad]
- `Vinf = [Vinfm*cos(alpha) 0.0 Vinfm*sin(alpha)]` with `Vinfm=60.0` [m/s] and `alpha=5*pi/180` [rad]
- `rhoAir = 1.225` [kg/m^3]
- **Structural parameters:**
- `R = [0.1 0.1 0.1 0.1]` [m]
- `t = [0.005 0.005 0.005 0.005]` [m]
- `E = 73.1e9` [Pa]
- `rhoMat = 2780.0` [kg/m^3]
- `sigmaY = 324.0e6` [Pa]
- `pKS = 200.0`
- `conIDs = [5, 6]` (constraint the displacement and rotation of the middle node)
  ATTENTION: Remember to use `dtype='int32'` when defining this array, so that Fortran recognizes its integers. That is: `conIDs = np.array([5,6], dtype='int32')`
- `d = [0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001]` [m and rad]
- **Additional parameters:**
- `CD0 = 0.0270`
- `fixedMass = 700.0` [kg]
- `g = 9.8` [m/s^2]
- `Endurance = 4.0*60.0*60.0` [s]
- `TSFC = 0.5/3600.0` [1/s]
- `loadFactor = 3.0*1.5`

You should get the following results:

- `reslt = [1.63465176e+08 1.19342261e+08 1.19342261e+08 1.63465176e+08]`
- `resfem = [-2.47501176e+03  1.47150031e+03 -1.18063931e+04  7.47468644e+03`
  `1.00000000e-03  1.00000000e-03 -8.36163678e+03  4.00783471e+03`
  `-9.36452445e+03  5.41801238e+03]`
- `liftExcess = 4.114355893146916`
- `margins = [0.69088168 0.69088168 0.69088168 0.69088168 0.69088168 0.69088168`
  `0.69088168 0.69088168]`
- `KSmargin = 0.6440942430587695`
- `FB = 1652.9227499511073` [N]
- `Weight = 9197.639151986312` [N]
- `Sref = 8.0` [m^2]
- `CL = 2.6666666666666656`

5.3 Differentiating the code

We have to differentiate the `asa_analysis` subroutine for optimization. Adapt the `send_to_tapeande.py` script used in previous homework to obtain the differentiated code in both forward and reverse modes. The `asa_analysis` subroutine should be differentiated according to the following parameters:

- **Input variables:** `Gama`, `alpha0`, `d`, and `t`
- **Output variables:** `resllt`, `resfem`, `liftExcess`, `margins`, `KSmargin`, `FB`, and `Weight`

ATTENTION: Remember to include all Fortran files (`llt_module.f90`, `fem_module.f90`, and `asa_module.f90`) in the `files` list of the `send_to_tapenade.py` script since the `asa_analysis` subroutine calls subroutines from all these files. Another important thing is to check if the `download_directory` variable defined in this Python script is correctly pointing to the folder where Firefox downloads files by default.

Once you get the differentiated code in the `TapenadeResults_d` and `TapenadeResults_b` folders, you can compile and generate their Python interface with:

```
$ bash build_f2py_asa_diff.sh
```

Check the differentiated code using methods you learned in this course. It is not necessary to use complex step.

5.4 Running an ASA

Now let's find the state variables that solve the aerostructural problem stated in Sec. 5.2. In other words, we need to find circulations `Gama` and displacements `d` that give `resllt = 0` and `resfem = 0`.

We will use Scipy's `root` function for this task. The problem is that we expect the values of `Gama` to be in the order of 10^1–10^2, while the values of `d` should be around 10^{-3}–10^{-2}. Therefore, we need to normalize these variables separately to improve convergence.

The following steps may help you implement the aerostructural analysis procedure:

1. Create a function `resfunc` that receives an array called `stateVars` of length $n_p + 2(n_p + 1)$. This is the size of `Gama` plus the size of `d`.
2. Take the first n_p elements of `stateVars`, divide them by 0.1, and assign the results to `Gama`.
3. Take the remaining elements of `stateVars`, divide them by 100.0, and assign the results to `d`.
4. Call the Fortran subroutine `asa_analysis` with these values of `Gama` and `d` to obtain `resllt` and `resfem`.
5. Concatenate `resllt` and `resfem` is a single array `res`. You can use the `numpy.hstack` function for this task: `res = np.hstack([resllt, resfem])`.
6. Use `res` as the return value of the `resfunc` function.
7. Use the `scipy.optimize.root` function to find the root of the `resfunc` function. The values of `Gama` and `d` shown in Sec. 5.2 can be used as initial guesses.
8. Take the solution given by the `root` function and apply the same normalization procedure described in steps 2 and 3 to find the values of `Gama` and `d` that satisfy the aerostructural equations.

If you apply this procedure for the parameters given in Sec. 5.2, you should get the following state variables:

- `Gama = [12.99803609 14.28790619 14.28790619 12.99803609]`
- ```
  d = [ 2.69493469e-02 -8.96491635e-03  9.57299675e-03 -7.85795127e-03
        4.85780076e-33 -3.27034245e-49  9.57299675e-03  7.85795127e-03
        2.69493469e-02  8.96491635e-03]
  ```
  ```

5.5 Adjoint problem

Let's solve an adjoint problem to compute the total derivatives of `KSmargin` with respect to the design variables `Gama` and `d`.

The following steps may help you implement the aerostructural analysis procedure:

1. Create a function `adjfunc` that receives an array called `resb` of length $n_p + 2(n_p + 1)$. This is the size of `Gama` plus the size of `d`.
2. Take the first n_p elements of `resb` and assign them to `reslltb`.
3. Take the remaining elements of `resb` and assign them to `resfemb`.
4. Initialize the output variables of the `asa_analysis` subroutine (namely `resllt`, `resfem`, `liftExcess`, `margins`, `KSmargin`, `FB`, `Weight`, `Sref`, and `CL`) to any value, since the Fortran code will overwrite them.
5. Initialize all derivative seeds used by the `asa_analysis_b` subroutine. I will let you think about the values of these seeds.
6. Call the Fortran subroutine `asa_analysis_b`. Remember to use `resllt.copy()` and `resfem.copy()` when calling the subroutine to avoid modifications to these seeds on the Python side.
7. Concatenate `Gamab` and `db` is a single array `stateVarsb`. You can use the `numpy.hstack` function for this task: `stateVarsb = np.hstack([Gamab, db])`. Explain why `Gamab` and `db` represent the residuals of the adjoint equation.
8. Use `stateVarsb` as the return value of the `adjfunc` function.
9. Use the `scipy.optimize.root` function to find the root of the `adjfunc` function. The initial guess for the aerodynamic adjoint variables should be around 10^6, while the initial guess for the structural adjoint variables should be around 10^0.
10. Take the solution given by the `root` and split them according to steps 2 and 3 to obtain `psi` (the aerodynamic adjoint state variables) and `lambda` (the structural adjoint state variables).
11. Use the adjoint variables to compute the total derivatives calling once again the `asa_analysis_b` subroutine. You may have to reinitialize derivative seeds for this call.

If you get lost, remember you can reuse the adjoint code of the `llt_module` developed in homework 3. The code structure should be the same.

The adjoint variables you should get for the case described in Sec. 5.2 are:

- `psi = [25.57826006 -8.62015154 -8.61876256 25.57847327]`
- ```
  lambda = [ 1.76989719e-04 -4.41232719e-05  8.87470713e-05 -4.41174285e-05
            -1.53738245e+02  1.16173737e-10  8.87470713e-05  4.41174285e-05
             1.76989719e-04  4.41232719e-05]
  ```

The total derivatives of `KSmargin` with respect to the design variables are:

- `dKSmargin_dalpha0 = [-2.63697857 -1.16166158 -1.16166158 -2.63697857]`
- `dKSmargin_dt = [1.11410770e-03 6.61735633e+01 6.61735633e+01 1.11410770e-03]`

5.6 Fuel burn minimization

Now that we have the capability to compute gradients, we can finally run an optimization.

Set up an optimization script to solve the optimization problem defined in Table 2.

Once again, you can use the scripts you developed for homework 3 as starting point since the steps are the same. One important difference is that we have to normalize the design variables since they have different magnitudes. I also suggest that you create two auxiliary functions to avoid code repetition:

`solve_asa` function:

1. Create a function `solve_asa` that receives an array `desVars` of length $2n_p$. This is the size of `alpha0` plus the size of `t`.
  ```

Table 2: Aerostructural optimization problem. n_p is the number of panels.

	Variable/function	Description	Quantity
minimize	`FB`	Fuel burn	
with respect to	`alpha0`	Twist angle of panels [deg]	n_p
	`t` ≥ 0.001	Beam thickness distribution [m]	n_p
		Total design variables	$2n_p$
subject to	`liftExcess` $= 0$	Lift constraint	1
	`KSmargin` ≥ 0	Structural feasibility	1
		Total constraints	**2**

2. Take the first n_p elements of `desVars` and assign them to `alpha0`.
3. Take the remaining elements of `desVars`, divide them by 100.0, and assign them to `t`.
4. Solve the ASA problem according to the steps described in Sec. 5.4.
5. Return the output variables given by the `asa_analysis` subroutine for the solved condition.

`compute_grads` function:

1. Create a function `compute_grads` that receives an array `desVars` of length $2n_p$, and the derivative seeds `liftExcessb`, `KSmarginb`, `FBb`, and `Weightb`.
2. Take the first n_p elements of `desVars` and assign them to `alpha0`.
3. Take the remaining elements of `desVars`, divide them by 100.0, and assign them to `t`.
4. Solve the ASA problem according to the steps described in Sec. 5.4. You may even use the `solve_asa` function you just implemented.
5. Solve the aerostructural adjoint and compute the total derivatives according to the steps described in Sec. 5.5.
6. Correct the derivatives by dividing them by the same normalizing factors used in steps 2 and 3.
7. Concatenate the derivatives in a single array using the `np.hstack` function.
8. Return the array with total derivatives.

These two functions can be used to implement the objective function, the constraint functions, and their gradients.

We will use the same parameters specified in Sec. 5.2, with the exception of the nodal coordinates `Xfem` and `chords`. We will define these variables in function of a desired wing area `Sref` and aspect ratio `AR`. We will also parameterize the discretization according to the number of panels n_p. Use the following steps to compute these variables at the beginning of the optimization script:

1. For user-defined values of `Sref` and `AR`, compute the wingspan `bref` with:

$$b_{\text{ref}} = \sqrt{\text{AR} \cdot S_{\text{ref}}} \tag{14}$$

2. Assuming a rectangular wing, compute the reference chord `cref` with:

$$c_{\text{ref}} = \frac{S_{\text{ref}}}{b_{\text{ref}}} \tag{15}$$

3. Initialize an array `chords` with n_p elements having the same value `cref`.
4. Initialize an array `Xfem` of size 3 by n_p. The first and third rows should be set to zero, while the second row should vary linearly from zero to `bref`. This creates a flat wing of span `bref`. ATTENTION: Remember to use `order='F'` when defining `Xfem`.
5. You have to update `conIDs` so that we restrain the two degrees of freedom (displacement and rotation) of the center node of the wing. The first node of the wing has degrees of freedom 1 and 2, the second node has degrees of freedom 3 and 4, and so on. If we have n_p elements, the degrees of freedom of the center node will be $n_p + 1$ and $n_p + 2$. Therefore we should use: `conIDs = np.array([n_panels+1, n_panels+2], dtype='int32')`.

6. The change in chords will determine how much thickness is available for the internal structure. Therefore, set the radius (`R`) of the tubular beam as $0.05 \cdot c_{\text{ref}}$ (that is, 5% of the chord).

Solve this optimization problem for $n_p = 20$, $AR = 6.0$, and $S_{\text{ref}} = 16.0$ m^2. Show in your report:

- Comparison between the baseline and optimized design in terms of design variables, objective, and constraints.
- Plot with the lift distribution of the optimized design and comparison with the elliptical lift distribution. Remember that the `asa_analysis` subroutine outputs the `CL` value required to generate the elliptical lift distribution.
- Plot with the failure margins of the FEM nodes in the optimized design. I do not want just the KS value, but the values of all nodes (use the `margins` variable).
- Weight distribution of the optimized aircraft in terms of structural weight, fixed weight, and fuel weight. You can show these results as fractions of the total weight.
- Optimization history showing design variables, objective, and constraints.

5.7 Weight minimization

Fuel burn is not the only important metric in aircraft design. The total weight of the aircraft (W_0) is another important figure, since it is related to production and operating costs of the vehicle.

Modify (or make a copy of) the fuel burn minimization script so that the objective becomes the minimization of the aircraft total weight, as shown in Table 3.

Table 3: Weight minimization problem. n_p is the number of panels.

	Variable/function	Description	Quantity
minimize	`Weight`	Total aircraft weight	
with respect to	`alpha0`	Twist angle of panels [deg]	n_p
	`t` ≥ 0.001	Beam thickness distribution [m]	n_p
		Total design variables	$2n_p$
subject to	`liftExcess` $= 0$	Lift constraint	1
	`KSmargin` ≥ 0	Structural feasibility	1
		Total constraints	**2**

Solve this optimization problem for $n_p = 20$, $AR = 6.0$, and $S_{\text{ref}} = 16.0$ m^2. Show in your report:

- Comparison between the baseline and optimized design in terms of design variables, objective, and constraints.
- Plot with the lift distribution of the optimized design and comparison with the elliptical lift distribution. Remember that the `asa_analysis` subroutine outputs the `CL` value required to generate the elliptical lift distribution.
- Plot with the failure margins of the FEM nodes in the optimized design. I do not want just the KS value, but the values of all nodes (use the `margins` variable).
- Weight distribution of the optimized aircraft.
- Optimization history showing design variables, objective, and constraints.

5.8 Aspect ratio effect

Now that you have all the optimization scripts, solve these two additional optimization cases:

1. Fuel burn minimization for $n_p = 20$, $AR = 10.0$, and $S_{\text{ref}} = 16.0$ m^2.
2. Weight minimization for $n_p = 20$, $AR = 10.0$, and $S_{\text{ref}} = 16.0$ m^2.

Once you have the results, show in your report:

- Comparison among the four optimized designs in terms of design variables, objective, and constraints.
- Compare the shape of lift distributions of the optimized designs. Remember to show the elliptical lift distribution as reference.

- Weight distribution of the optimized designs.
- Discussion about the design trends. Do you best to find physical and rational reasons to explain the decisions taken by the optimizer.

6 Grading

You should submit a report containing at least the topics I have highlighted in red. This time I do not want just the answers. You have to write some text to discuss and analyze the results. On the other hand, this is not a thesis, so go straight to the point.

The way you present the results and the clarity and effectiveness of the plots will be taken into account for the grade.

Feel free to show additional analyses beyond the ones I asked. You may even get extra points if you surprise me!

7 Troubleshooting

The structure of the optimization problem is very similar to what we have done for the aerodynamic shape optimization in Homework 3. You can save some time by reusing codes whenever you can.

You only need to write Fortran code inside the `asa_analysis` subroutine. If you are changing the Fortran code somewhere else, you are in the wrong path.

List of Acronyms

ASA Aerostructural Analysis

FEM Finite Element Method

LLT Lifting Line Theory