# Examples Manual

# Mefisto 2.8

# Volume 2: Aerodynamics

## Contents

1 Introduction.....2
2 Steady Aerodynamics.....3
2.1 Gull Wing.....3
2.2 Glider Wing.....12
3 Trim Analysis.....28
3.1 Glider.....28

# 1 Introduction

The examples presented in this manual demonstrate how to define the models, how to use the Mefisto functions and how to postprocess the results. The examples also show how to use Gmsh[1] for pre- and post-processing.

If you are new to Mefisto or to Gmsh, you are recommended to first study some of the examples of solid mechanics as described in Volume 1 of the Examples Manual.

The supporting files of all the examples can be found in directory `exa` of your Mefisto installation. The directory has the following subdirectories:

| | | |
|---|---|---|
| `aero` | `statresp` | Steady aerodynamics |
| | `trim` | Trim analysis |

Some of the examples use script `iniplot.m` located in directory `exa`, so this directory must be included in the search path. If you installed Mefisto using script `install.m` nothing need be done. Otherwise, you have to add a corresponding **`addpath`** command to file `.octaverc`.

Script `iniplot.m` defines six colours (red, green, blue, magenta, brown, light blue) to be used with the plots. It also defines the default format and extension of the files created with the **`print`** command.

1 http://www.geuz.org/gmsh

# 2 Steady Aerodynamics

## 2.1 Gull Wing

Summary

| | |
|---|---|
| Directory: | `exa/aero/statresp/gull_wing` |
| Objectives: | • learn how to define a simple aerodynamic model<br>• learn how to run a steady aerodynamic analysis<br>• learn how to access the results for further computations<br>• learn how to postprocess the results using Gmsh |
| Method: | Vortex-Lattice |
| Functions: | **`mfs_airfoil, mfs_new, mfs_export, mfs_statresp, mfs_results, mfs_getresp`** |

Problem Description

Compute the pressure, the lift coefficient and the moment coefficient with respect to the leading edge of the gull wing shown in Figure 2.1-1. The flap chord ratio of the aileron is 20 %. The wing has a NACA 43012 airfoil.

Consider the following configurations:

1. Angle of attack $\alpha = 0°$, aileron deflection $\eta = 0°$
2. Angle of attack $\alpha = 0°$, aileron deflection $\eta = 5°$
3. Angle of attack $\alpha = 2°$, aileron deflection $\eta = 0°$

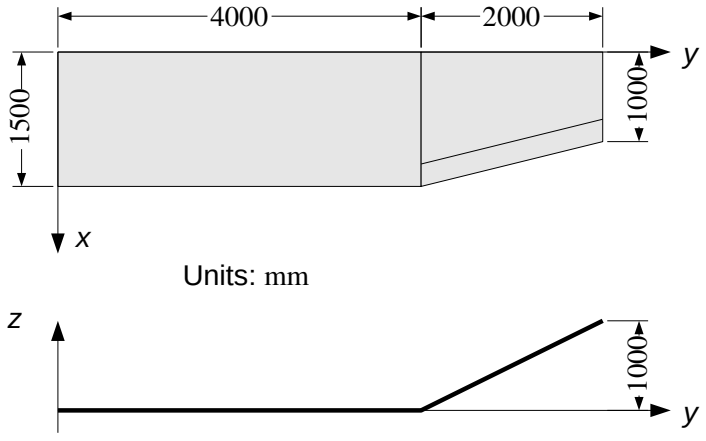



Figure 2.1-1: Gull Wing Geometry

4. Angle of attack $\alpha = 2°$, aileron deflection $\eta = 5°$

Also compute the derivatives $c_{L\alpha} = \partial c_L / \partial \alpha$ and $c_{L\eta} = \partial c_L / \partial \eta$.

Data: Flight velocity $v = 144$ km/h, mass density of air $\rho = 1.21$ kg/m$^3$

Model Definition

First, we open the output file and define the model data. Note that consistent units have to be used. We use mm as length units. Thus, the flight velocity has to be defined in mm/s. As we want the pressure to be in MPa = N/mm$^2$, the independent units are N, mm and s. The unit of the mass has to be expressed in terms of these units:

$$1\,\mathrm{N} = 1\,\frac{\mathrm{kg\,m}}{\mathrm{s}^2} \;\rightarrow\; 1\,\mathrm{kg} = 1\,\frac{\mathrm{N\,s}^2}{\mathrm{m}} = 10^{-3}\,\frac{\mathrm{N\,s}^2}{\mathrm{mm}} \;\rightarrow\; 1\,\frac{\mathrm{N\,s}^2}{\mathrm{mm}} = 10^3\,\mathrm{kg} = 1\,\mathrm{t}$$

The mass density is

$$\rho = 1{,}21\,\frac{\mathrm{kg}}{\mathrm{m}^3} = \frac{1{,}21\cdot 10^{-3}\,\mathrm{N\,s}^2}{10^9\,\mathrm{mm}^3\cdot\mathrm{mm}} = 1{,}21\cdot 10^{-12}\,\frac{\mathrm{t}}{\mathrm{mm}^3}\,.$$

The inner wing has a uniform discretisation of 25 panels in the spanwise direction and a uniform discretisation of 50 panels in the chordwise direction. The outer wing a has a discretisation of 30 panels in the spanwise direction, with a cosine refinement towards the wing tip, and a uniform discretisation of 40 panels in the chordwise direction. The aileron has the same discretisation in the spanwise direction as the outer wing and a uniform discretisation of 10 panels in the chordwise direction.

These data are defined by the following commands in file `wing.m`:

```
# Example: Gull wing
#
# ------------------------------------------------------------

  fid  = fopen("wing.res", "wt");

# Data

  ya = 4000;  % y-position of aileron in mm
  yt = 6000;  % y-position of wing tip in mm
  zt = 1000;  % z-position of wing tip in mm

  cr = 1500;  % Chord length at wing root
  ct = 1000;  % Chord length at wing tip
  fr =  0.2;  % Flap chord ratio of aileron

  nxi =  50;  % No. of panels in x-direction, inner wing
  nxo =  40;  % No. of panels in x-direction, outer wing
  nxa =  10;  % No. of panels in x-direction, aileron
```

```
  nyi =  25;  % No. of panels in y-direction, inner wing
  nyo =  30;  % No. of panels in y-direction, outer wing

  alpha =   {0, 0, 2, 2};   % Angles of attack in degrees
  eta   =   {0, 5, 0, 5};   % Aileron angles in degrees
  v     =     40E3;         % Flight velocity in mm/s
  rho   = 1.21E-12;         % Mass density of air in t/mm^3
```

Next, we define the model type, the model subtype and the symmetry. In an aerodynamic analysis, the model type is **"aero"**. The subtype identifies the method used. We use the vortex-lattice method which is the only method currently available. So the subtype is **"vlm"**.

The model is symmetric with respect to the plane $y = 0$. Thus, field **symy** is assigned a value of zero. If this field is undefined, no symmetry is used.

```
# Model definition
# ----------------

# Model type, subtype and symmetry

  model = struct("type", "aero", "subtype", "vlm", "symy", 0);
```

The vortex-lattice method needs only the camber of the airfoil. In Mefisto, we can define the camber using a piecewise polynomial structure. In case of NACA 4 digit and NACA 5 digit airfoils, this structure is returned by function **mfs_airfoil**. The second argument defines the number of digits. For a NACA 5 digit airfoil, the third argument defines digits 2 and 3 of the airfoil. The camber does not depend on digits 4 and 5 that define the thickness.

The optional last two arguments of function **mfs_airfoil** define the start and end position when only a segment of the airfoil is requested. These positions are defined in fractions of the chord length. This is useful when defining wings with flaps.

```
# Airfoil data

  frc = 1 - fr;

  ppi = mfs_airfoil("NACA", 5, 30);
  ppo = mfs_airfoil("NACA", 5, 30, 0, frc);
  ppa = mfs_airfoil("NACA", 5, 30, frc, 1);
```

The first call of **mfs_airfoil** returns the camber of the inner wing. The second call returns the camber of the outer wing that ends at 80 % of the chord length. The third call returns the camber of the aileron that begins at 80 % of the chord.

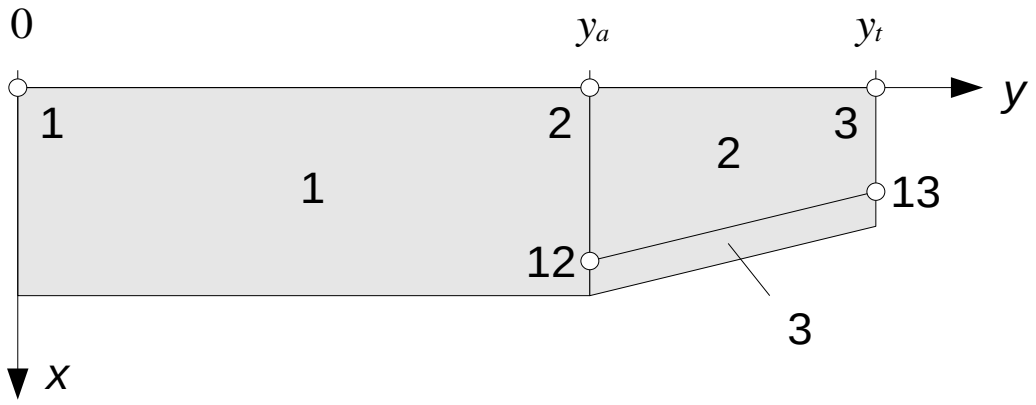



Figure 2.1-2: Lifting Surfaces

The aerodynamic model consists of three lifting surfaces, one for the inner wing, one for the outer wing and one for the aileron. Before we can define these lifting surfaces, we have to define the points on their leading edges.

Points are defined by structure **`points`** which has fields **`id`** for the point identifier and **`coor`** for the coordinates. The identifiers and coordinates of the points can be seen in Figure 2.1-2.

```
# Points on leading edge of wing

  points(1).id = 1; points(1).coor = [0,  0,  0];
  points(2).id = 2; points(2).coor = [0, ya,  0];
  points(3).id = 3; points(3).coor = [0, yt, zt];

# Points on leading edge of aileron

  points(4).id = 12; points(4).coor = [frc * cr, ya,  0];
  points(5).id = 13; points(5).coor = [frc * ct, yt, zt];
```

Now we can define the lifting surfaces. Lifting surfaces are defined by specifying their identifier, the two points of the leading edge, the chord length, the camber and the discretization parameters. The identifiers of the lifting surfaces and the points on the leading edge can be seen in Figure 2.1-2.

```
# Lifting surface of inner wing

  lsf(1).id = 1;     lsf(1).points = [1, 2];
  lsf(1).chord = cr; lsf(1).camber = ppi;
  lsf(1).nx = nxi;   lsf(1).ny = nyi;
  lsf(1).typex = "linear";

# Lifting surface of outer wing

  lsf(2).id = 2;                 lsf(2).points = [2, 3];
```

```
  lsf(2).chord = frc * [cr, ct]; lsf(2).camber = ppo;
  lsf(2).nx = nxo;                lsf(2).ny = nyo;
  lsf(2).typex = "linear";        lsf(2).typey = "cos>";

# Lifting surface of aileron

  lsf(3).id = 3;                 lsf(3).points = [12, 13];
  lsf(3).chord = fr * [cr, ct]; lsf(3).camber = ppa;
  lsf(3).nx = nxa;               lsf(3).ny = nyo;
  lsf(3).typex = "linear";       lsf(3).typey = "cos>";
```

Figure 2.1-3 shows the resulting discretization of the lifting surfaces.

Next, the aileron is declared a control surface. Control surfaces are defined by structure **`controls`** which has the fields **`name`** for the control surface name and **`ls`** for the identifiers of the lifting surfaces of the control surface. The name can be any string, so we call the control surface **`"aileron"`**. In this simple example, the control surface consists of only one lifting surface.

```
# Aileron is control surface

  controls = struct("name", "aileron", "ls", 3);
```

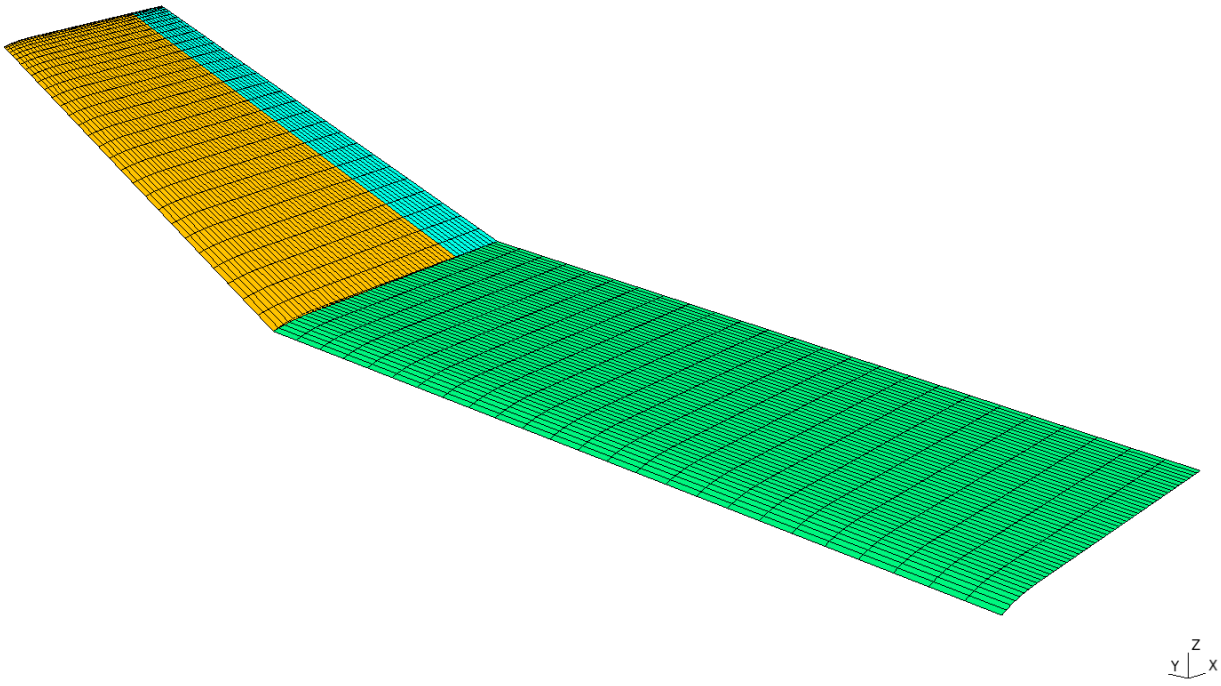



Figure 2.1-3: Discretization of Gull Wing

The next commands define the configurations. Configurations have a name and a set of configuration parameters defined. They are defined in structure **config**. Field **name** contains the name of the configuration that can be any arbitrary string. Here, we place the values of the angle of attack and the aileron deflection angle into this string.

The dynamic pressure is defined in field **qdyn**. The remaining fields define the values of the configuration parameters. Among the configuration parameters is the angle of attack and all control surface angles. The angle of attack is defined in field **alpha**. The names of the fields defining the deflections of the control surfaces equal the control surface names as defined in structure **controls**. Because variables **cfgnam**, **alpha** and **eta** are cell arrays, the **struct** command creates a structure array.

```
# Configurations

  qdyn = 0.5 * rho * v^2;

  for l = 1 : 4
      cfgnam{l} = sprintf("Conf. %d: alpha = %5.2f, eta = %5.2f",
                          l, alpha{l}, eta{l});
  endfor

  config = struct("name", cfgnam, "qdyn", qdyn,
                  "alpha", alpha, "aileron", eta);
```

Finally, we assign all these structures to the corresponding fields of structure **model**.

```
# Add definitions to the model

  model.points   = points;
  model.ls       = lsf;
  model.controls = controls;
  model.config   = config;
```

## Analysis

Like in solid mechanics, the analysis begins with the creation of the component. Next, we export the component so that it can be visualized with Gmsh. The keyword **"camber"** requests the geometry to be modified according to the camber. This allows to check if the definition of the airfoil is correct. The resulting surfaces should be smooth except where lifting surfaces are joined with kinks. Figure 2.1-3 shows that the aerodynamic model is correct.

```
# Analysis
# --------
```

```
# Create and export component

  wing = mfs_new(fid, model);
  mfs_export("wing.msh", "msh", wing, "mesh", "camber");
```

In a steady aerodynamic analysis, the primary results are computed by function **mfs_statresp**. For the vortex-lattice method, the primary results are the strengths of the vortices. Next, function **mfs_results** computes the panel results, i.e. the pressure and the force of each panel. Subsequently, the pressure is exported so that it can be postprocessed using Gmsh.

```
# Compute and export results

  wing = mfs_statresp(wing);
  wing = mfs_results(wing, "statresp", "panel");
  mfs_export("wing.pos", "msh", wing, "statresp", "pressure");
```

Up to now, the output file contains the following information:

```
Mefisto 2.7: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =  2938,  Number of panels =  2750
  Number of lifting surfaces =            3
  Number of control surfaces =            1
  Number of configurations   =            4
  Reference chord length     =  0.00000e+00
  Symmetry plane: y          =  0.00000e+00
```

To compute the aerodynamic coefficients, function **mfs_getresp** is used to retrieve the resulting forces and moments and the areas of the lifting surfaces.

```
# Aerodynamic coefficients

  [F, M] = mfs_getresp(wing, "statresp", "aeload");
  As     = mfs_getresp(wing, "mesh", "area", 1 : 3);
```

The arrays **F** and **M** consist of four columns, each of them containing the resulting force or moment vector of one configuration.

Array **As** contains the areas of the three lifting surfaces. If called without the last argument, function **mfs_getresp** would return the total area. However, we need the areas of the single lifting surfaces because the areas of lifting surfaces 2 and 3 have to be projected onto the *xy*-plane:

```
  tg = zt / (yt - ya);
  cs = 1 / sqrt(1 + tg^2);
  A  = As(1) + (As(2) + As(3)) * cs;
```

Now, the aerodynamic coefficients can be computed from

$$c_L = \frac{F_z}{q_\infty A}, \quad c_M = \frac{M_y}{q_\infty A\, c_{ref}},$$

where we use the chord length at the wing root as reference length $c_{ref}$.

In a linear analysis, the derivatives of the lift coefficient can be computed from the differences:

$$\frac{\partial c_L}{\partial \alpha} = \frac{\Delta c_L}{\Delta \alpha}, \quad \frac{\partial c_L}{\partial \eta} = \frac{\Delta c_L}{\Delta \eta}$$

Of course, the first term has to be evaluated for two configurations with the same aileron deflection, i.e. configurations 3 and 1, and the second term has to be evaluated for two configurations with the same angle of attack, i.e. configurations 2 and 1. In addition, the angles have to be converted from degrees to radians.

```
cL = F(3, :) / (qdyn * A);
cM = M(2, :) / (qdyn * A * cr);

dalpha = pi * (alpha{3} - alpha{1}) / 180;
deta   = pi * (eta{2} - eta{1}) / 180;

dcLda = (cL(3) - cL(1)) / dalpha;
dcLde = (cL(2) - cL(1)) / deta;

fprintf(fid, "\nWing area = %10.5e, ", A);
```

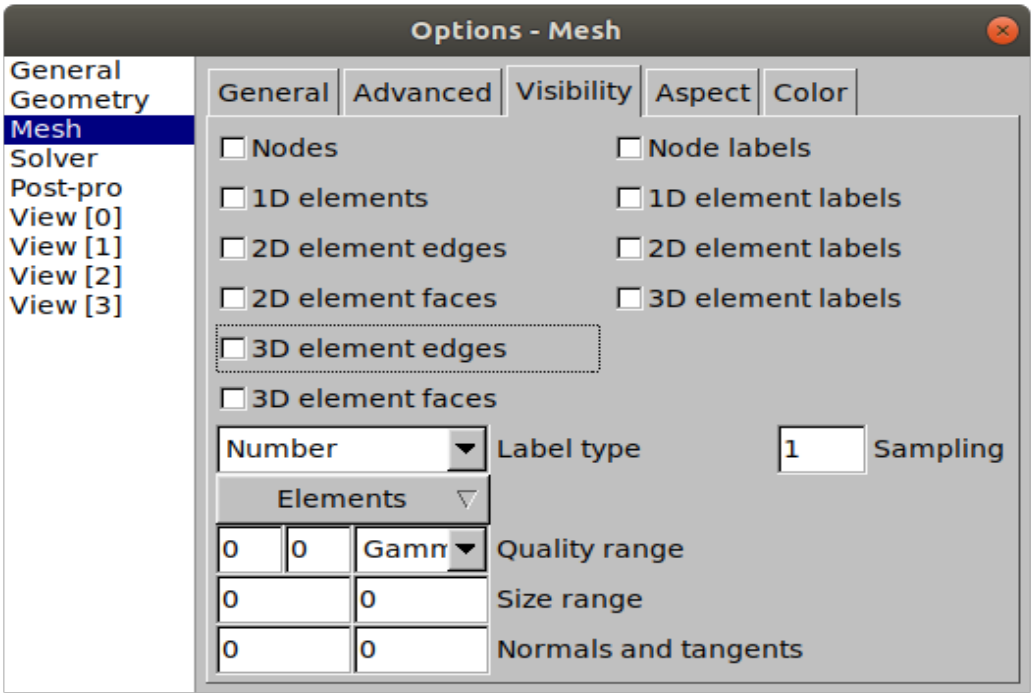



Figure 2.1-4: Gmsh: Mesh Visibility Options

```
fprintf(fid, "dcLda = %7.4f, dcLde = %7.4f\n\n",
             dcLda, dcLde);

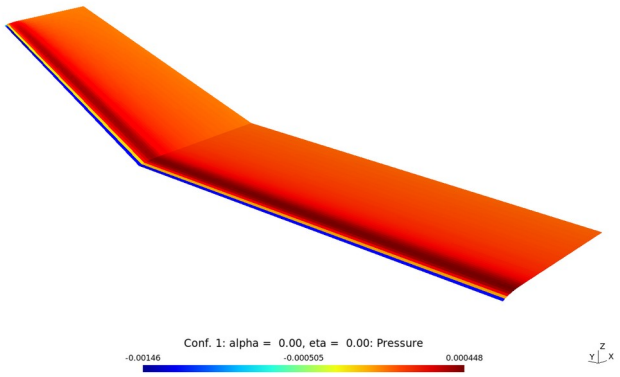
for n = 1 : length(config)
    fprintf(fid, "%s: ", config(n).name);
    fprintf(fid, "cL = %7.4f, cM = %7.4f\n", cL(n), cM(n));
endfor

fclose(fid);
```

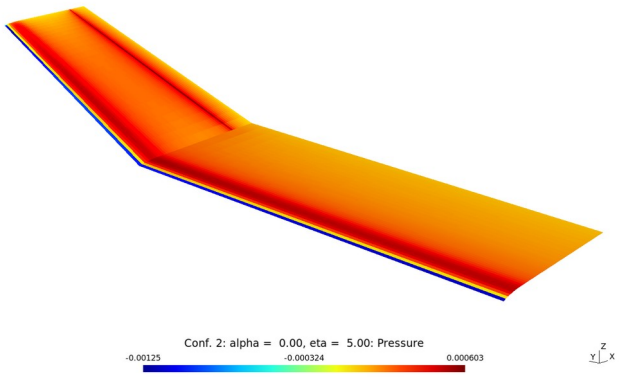
The results can be found in the output file:

```
Wing area = 8.50000e+06, dcLda =  4.7456, dcLde =  0.7200

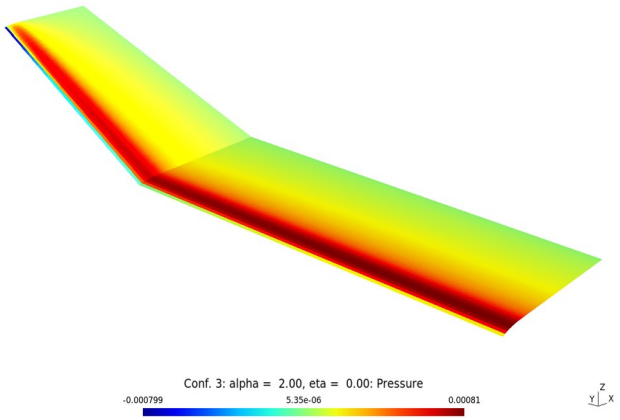
Conf. 1: alpha =  0.00, eta =  0.00: cL =  0.0939, cM = -0.0342
Conf. 2: alpha =  0.00, eta =  5.00: cL =  0.1568, cM = -0.0612
Conf. 3: alpha =  2.00, eta =  0.00: cL =  0.2596, cM = -0.0734
Conf. 4: alpha =  2.00, eta =  5.00: cL =  0.3224, cM = -0.1004
```



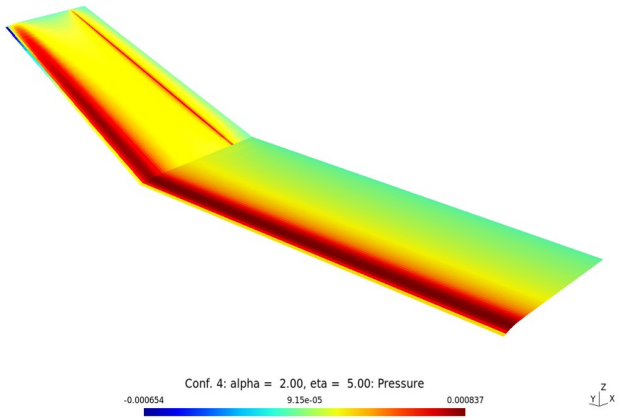

Configuration 1




Configuration 2




Configuration 3




Configuration 4

Figure 2.1-5: Pressure (in MPa)

Postprocessing

To visualize the pressure, we start Gmsh, open file **wing.msh** (File → Open …) and merge file **wing.pos** (File → Merge …). There are four different views corresponding to the four different configurations. To get a nice view, we switch off the visibility of all mesh items (Tools → Options → Mesh → Visibility), see Figure 2.1-4. The resulting picture can be seen in Figure 2.1-5.

## 2.2 Glider Wing

Summary

| | |
|---|---|
| Directory: | **exa/aero/statresp/glider_wing** |
| Objectives: | • learn how to work with arbitrary airfoils<br>• learn how to define aerodynamic models with more sophisticated control surfaces<br>• learn how to run a steady aerodynamic analysis<br>• learn how to access the results for further computations<br>• learn how to postprocess the results using Mefisto |
| Method: | Vortex-Lattice |
| Functions: | **mfs_airfoil, mfs_vortex2d, mfs_new, mfs_export, mfs_statresp, mfs_results, mfs_getresp, mfs_xydata** |

Problem Description

Examine an open class glider wing with winglet, flap and aileron (see Figure 2.2-1). The wing has a Horstmann-Quast HQ-17/14.38 airfoil the data of which can be found at Airfoil Tools.

The leading edge of the wing is straight and parallel to the *y*-axis. The length of the wing is $10\ \mathrm{m}$. The length of the inner wing with the flap is 70 % of the total length of the wing. The inner wing has a constant chord length of $1\ \mathrm{m}$. The outer wing with the aileron has a linearly decreasing chord length, the chord length at the wing tip being $80\ \mathrm{cm}$. The flap chord ratio of both the flap and the aileron is 20 %. The wing has a dihedral angle of 1°. The outer wing is twisted, with a linear decrease of the rigging angle of incidence from 0° to -2°.

The winglet has a height of $1\ \mathrm{m}$ and a chord length of $60\ \mathrm{cm}$. The upper leading edge point of the winglet is $20\ \mathrm{cm}$ behind the leading edge of the wing.

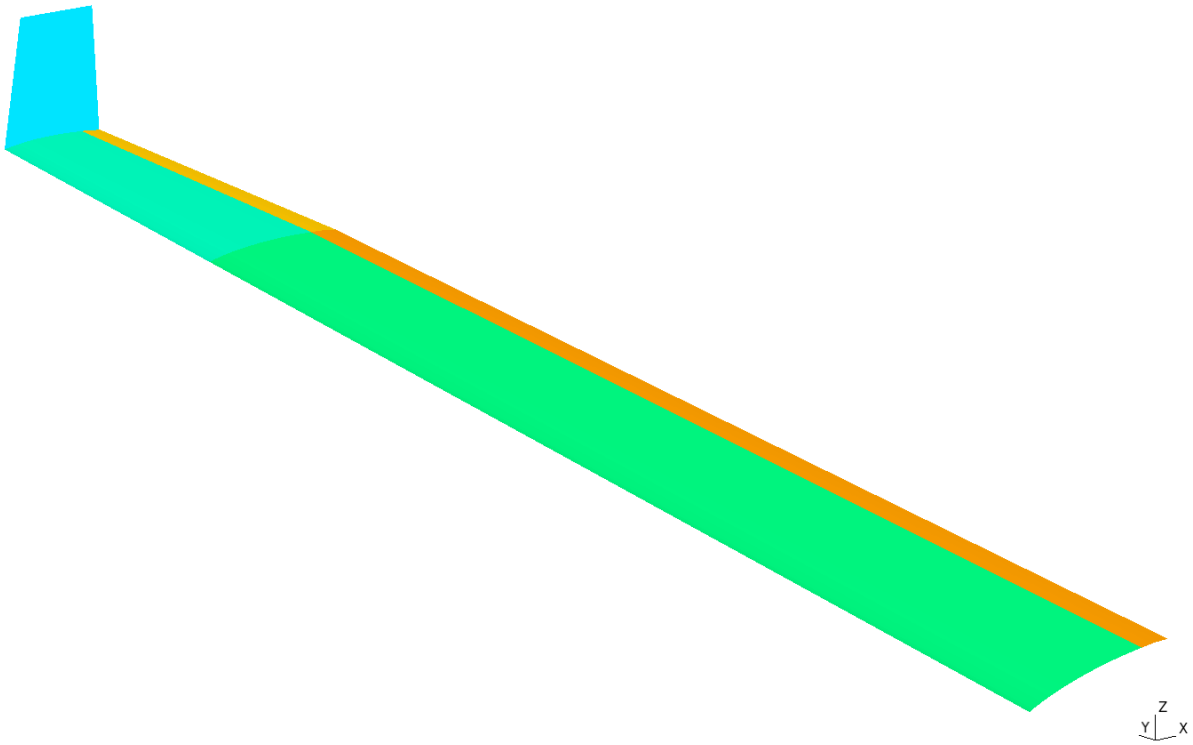



Figure 2.2-1: Glider Wing

The motions of the aileron and the flap are interconnected. The aileron follows the motion of the flap, and the flap angle follows 10 % of the aileron angle.

First, study how to best approximate the camber of the airfoil using a piecewise polynomial by comparing the camber and computing the derivative and the pressure coefficient.

Subsequently, use the approximation parameters found to compute the pressure coefficient, the lift coefficient and the moment coefficient of the wing for the configurations shown in Table 2.2-1. The moment coefficient is defined with respect to the leading edge.

Also determine the flight velocity needed for each of the configurations if the mass of the glider is $650\ \mathrm{kg}$ and the mass density of the air is $1.21\ \mathrm{kg/m^3}$.

Finally, visualize the pressure coefficient in some typical wing sections for some of the configurations, as well as the loads in the direction of span.

## Airfoil Study

The csv-files that can be downloaded from Airfoil Tools contain a pointwise description of the surface and the camber of the airfoil. The camber is com-

| Conf. | Angle of Attack | Flap Angle | Aileron Angle |
|---|---|---|---|
| 1 | 0° | -2° | -4° |
| 2 | 0° | -2° | 0° |
| 3 | 0° | -2° | 4° |
| 4 | 0° | 0° | -4° |
| 5 | 0° | 0° | 0° |
| 6 | 0° | 0° | 4° |
| 7 | 0° | 2° | -4° |
| 8 | 0° | 2° | 0° |
| 9 | 0° | 2° | 4° |
| 10 | 2° | -2° | -4° |
| 11 | 2° | -2° | 0° |
| 12 | 2° | -2° | 4° |
| 13 | 2° | 0° | -4° |
| 14 | 2° | 0° | 0° |
| 15 | 2° | 0° | 4° |
| 16 | 2° | 2° | -4° |
| 17 | 2° | 2° | 0° |
| 18 | 2° | 2° | 4° |

Table 2.2-1: Glider Wing Configurations

puted from the surface by using a linear interpolation between the points (cf. Airfoil data information). This procedure is sufficiently accurate for the camber, but the quality of its derivative may be poor. Therefore, instead of using a piecewise polynomial that goes exactly through the given points, the camber is approximated by a piecewise polynomial with a lower number of spline segments, the coefficients of which are determined such as to minimize the sum of the squared errors. Function **`mfs_airfoil`** can be used to compute this piecewise polynomial. This function also provides the possibility to smooth the data before computing the polynomial.

File `airfoil.m` contains the GNU Octave code to check the approximation of the camber. It begins with the definition of some data. The number of points to check the approximation is the number of points at which the approximating spline is evaluated when checking the accuracy. This number should be considerably larger than the number of data points.

```
# Airfoil test:
#
#   Airfoil: Horstmann and Quast HQ-17/14.38
#   Units:   mm
#
# The airfoil data are approximated by a spline. The approximation
# is tested by computing the pressure coefficient using the
# discrete vortex method.
#
# ------------------------------------------------------------

  [EXT, FORMAT] = iniplot();

  set(0, "defaultaxesfontsize", 10);

  fid = fopen("airfoil.res", "wt");

# Data

  ns    =  10;       % Number of spline segments
  ng    = 100;       % Number of points to check approximation
  nv    =  50;       % Number of discrete vortices
  alpha = [0,  2];   % Angles of attack
```

Next, the camber points are input from file `hq17.csv`. The camber is defined in columns A and B of rows 108 to 155.

```
# Read camber from csv file

  data   = dlmread("hq17.csv", ",", "A108:B155");
  c      = data(end, 1);
  data   = data / c;
```

Function **`mfs_airfoil`** is used to obtain the piecewise polynomial. Keyword **`"camber"`** indicates that array **`data`** contains the coordinates of the camber line. Subsequently, the camber line and its first derivative are plotted. The plot of the camber line also contains the data points.

```
# Generate the spline approximation

  zspp = mfs_airfoil("fit", "camber", data, ns);

# Graphical check of interpolation

  xg  = linspace(0, 1, ng + 1);
  zsa = ppval(zspp, xg);
  dzs = ppval(ppder(zspp), xg);

  figure(1, "position", [100, 600, 750, 750],
            "paperposition", [0, 0, 14, 10]);
  subplot(2, 1, 1)
     plot(data(:, 1), 100 * data(:, 2), "color", "green",
          "marker", "o", "linestyle", "none",
```

```
          xg, 100 * zsa, "color", "red");
    legend("Data", "Spline", "location", "south");
    grid;
    ylabel("z/c [%]");
  subplot(2, 1, 2)
    plot(xg, dzs, "color", "red");
    grid;
    ylabel("dz/dx");
    xlabel("x/c");
  print(["hq_z", EXT], FORMAT);
```

The resulting picture is shown in Figure 2.2-2. It can be seen that with 10 spline segments a good approximation of the camber is achieved and the first derivative is sufficiently smooth.

Next, function **mfs_vortex2d** is used to compute the pressure coefficient, the lift coefficient and the moment coefficient for two different angles of attack:

```
# Compute aerodynamic coefficients

  x = linspace(0, 1, nv + 1);

  [cp, xv, cL, cM] = mfs_vortex2d(x, zspp, alpha);

# Output results

  fprintf(fid, "Aerodynamic coefficients:\n\n");
```

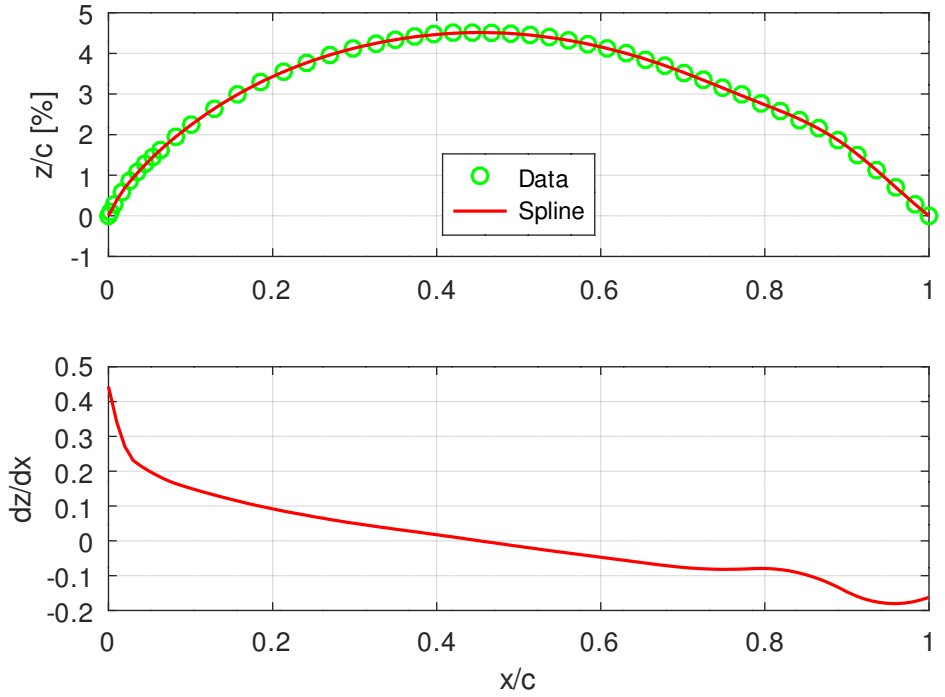



Figure 2.2-2: Camber and Derivative without Flap

```
  fprintf(fid, "  alpha      cL         cM\n");
  fprintf(fid, "  -------------------------\n")
  for n = 1 : length(alpha)
      fprintf(fid, "  %5.2f    %7.4f    %7.4f\n",
              alpha(n), cL(n), cM(n));
      txt{n} = sprintf("\\alpha = %4.1f\\deg", alpha(n));
  endfor

  figure(2, "position", [200, 500, 750, 500],
            "paperposition", [0, 0, 12, 8]);
  axes("outerposition", [0, 0.02, 1, 1]);
  plot(xv, cp);
  legend(txt, "location", "south");
  grid;
  xlabel('x/c');
  ylabel('\Delta c_p');
  print(["hq_cp", EXT], FORMAT);

  fclose(fid);
```

The lift and moment coefficients are written to the output file:

```
Aerodynamic coefficients:

  alpha      cL         cM
  -------------------------
   0.00    0.5674   -0.1332
   2.00    0.7867   -0.1332
```

The pressure coefficient is shown in Figure 2.2-3. Except at the leading edge it is sufficiently smooth for both angles of attack. With a cosine subdivision which is finer at the leading and the trailing edge slightly better results could be obtained. We used a uniform subdivision because it is more flexible for wings with flaps. In addition, the cosine subdivision leads to very large values

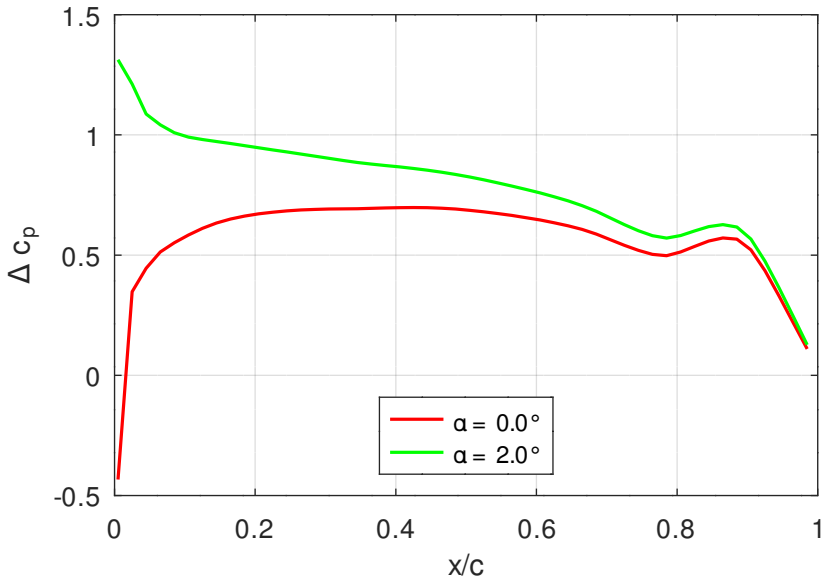



Figure 2.2-3: Pressure Coefficient

at the leading edge, as the analytical solution becomes infinite there.

Model Definition

GNU Octave script `wing.m` begins with the definition of the wing data, the discretization data and some additional data:

```
# Example: Wing with winglet and controls
#
# ------------------------------------------------------------

  [EXT, FORMAT] = iniplot();

  set(0, "defaultaxesfontsize", 10);

  file = mfilename();

# Wing data

  b     = 10000;      % Half span (mm)
  s     =   0.7;      % Aileron starts at 70 % of half span
  xw    =   200;      % x-coordinate of winglet tip (mm)
  h     =  1000;      % Winglet height (mm)

  c1    =  1000;      % Chord length at wing root
  c2    =   800;      % Chord length at wing tip
  cw    =   600;      % Chord length at winglet tip
  fr    =   0.2;      % Flap chord ratio

  alpha_tip = -2;     % Twist at wing tip (degrees)

  alpha   = [0, 2];     % Angle of attack (degrees)
  flap    = [-2, 0, 2]; % Flap angle (degrees)
  aileron = [-4, 0, 4]; % Aileron angle (degrees)

  delta  = 1;         % Dihedral angle (degrees)

# Discretization data

  nxw = 40;   % Number of wing panels in x-direction
  nxf = 10;   % Number of flap panels in x-direction
  nyi = 20;   % Number of inner wing panels in y-direction
  nyo = 15;   % Number of outer wing panels in y-direction
  nyw = 15;   % Number of winglet panels in y-direction

  ns  = 10;   % Number of spline segments

# Additional data

  mass   =  650;      % Mass (kg)
  g      = 9.81;      % Gravity acceleration (m/s^2)
  rho    = 1.21;      % Mass density of air (kg/m^3)

  d      = tand(delta);
```

```
  frc    = 1 - fr;
```

Next, the airfoil data are read from the csv-file and approximated by two piecewise polynomials:

```
# Airfoil data: Horstmann-Quast HQ17

  data   = dlmread("hq17.csv", ",", "A108:B155");

  airfoil_wing = mfs_airfoil("fit", "camber", data, ns, 0, frc);
  airfoil_flap = mfs_airfoil("fit", "camber", data, ns, frc, 1);
```

The model definition begins with the model type, the model subtype and the symmetry plane $y = 0$.

```
# Model definition
# ----------------

  model = struct("type", "aero", "subtype", "vlm", "symy", 0);
```

Subsequently, the points on the leading edge and the hinge of the flap and the aileron are defined. In this example, we use the option of defining structure arrays using cell arrays for the field values.

```
# Leading edge points

  ya = s * b; za = d * ya;
  zt = d * b;

  points(1 : 4) = struct("id",   {1, 2, 3, 4},
                         "coor", {[0, 0, 0], [0, ya, za], ...
                                  [0, b, zt], [xw, b, zt + h]});

# Hinge points

  xr = frc * c1; xt = frc * c2;

  points(5 : 7) = struct("id",   {11, 12, 13},
                         "coor", {[xr, 0, 0], [xr, ya, za], ...
                                  [xt, b, zt]});

  model.points = points;
```

Next, we define five lifting surfaces. The first three lifting surfaces model the inner and outer wings and the winglet. The division type in the *x*-direction of the inner and outer wing is **"linear"**. The division type in the *y*-direction of the inner wing is **"linear"**, of the outer wing it is **"cos>"** and of the winglet it is **"cos"**. Again, we use cell arrays to define the structure arrays. Field values that are constant need to be defined only once.

```
# Lifting surfaces of wing and winglet

  lsf(1 : 3) = struct("id",     {10, 20, 30},
                      "points", {[1, 2], [2, 3], [3, 4]},
                      "chord",  {frc * c1, frc * [c1, c2], ...
                                 [c2, cw]},
                      "camber", {airfoil_wing, airfoil_wing, []},
                      "nx",     nxw,
                      "ny",     {nyi, nyo, nyw},
                      "typex",  "linear",
                      "typey",  {"linear", "cos>", "cos"},
                      "alpha",  {0, [0, alpha_tip], 0});
```

The fourth lifting surface models the flap and the fifth the aileron. The division type in the *x*-direction of both surfaces is linear. The division type in the *y*-direction of the flap corresponds to that of the inner wing and that of the aileron that of the outer wing.

```
# Lifting surfaces of flap and aileron

  lsf(4 : 5) = struct("id",     {11, 21},
                      "points", {[11, 12], [12, 13]},
                      "chord",  {fr * c1, fr * [c1, c2]},
                      "camber", airfoil_flap,
                      "nx",     nxf,
                      "ny",     {nyi, nyo},
                      "typex",  "linear",
                      "typey",  {"linear", "cos>"},
                      "alpha",  {0, [0, alpha_tip]});

  model.ls = lsf;
```

The resulting discretization can be seen in Figure 2.2-4.

Then we define the control surfaces. As flap and aileron are interconnected, both consist of lifting surfaces 11 and 21, but with different factors. The aileron completely follows the motion of the flap but the flap motion is only 10 % of the aileron motion.

```
# Control surfaces

  model.controls = struct("name",    {"flap", "aileron"},
                          "ls",      {[11, 21], [11, 21]},
                          "factors", {[1, 1], [0.1, 1]});
```

Finally, the configurations are defined in a loop over the different angles of attack, the different flap angles and the different aileron angles. The names of the configurations are built from the names and values of the configuration parameters. The dynamic pressure is left undefined. Thus, the default value of 1 is used so that pressures actually are pressure coefficients.

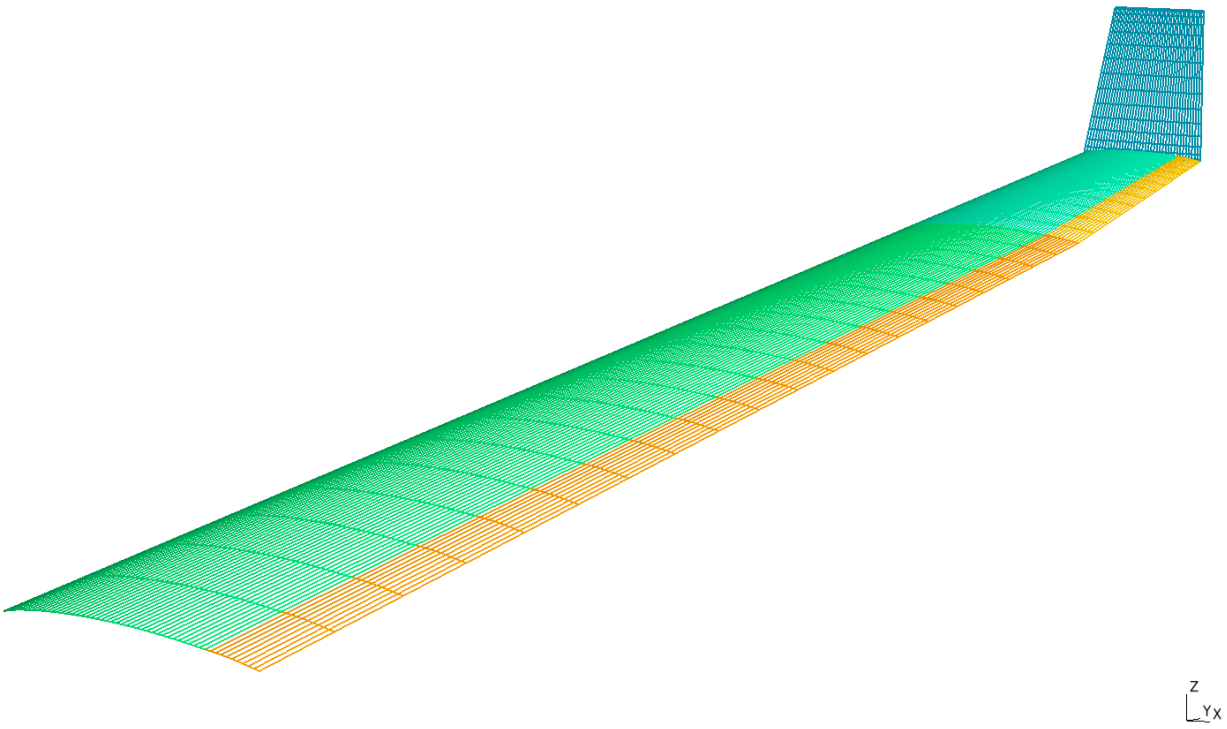



Figure 2.2-4: Discretization of Glider Wing

```
# Configurations

  nc = 1;
  for l = 1 : length(alpha)
    for m = 1 : length(flap)
      for n = 1 : length(aileron)
        name = sprintf("Conf. %2d: alpha = %4.1f, ",
                       nc, alpha(l));
        name = [name, sprintf("flap = %4.1f, ", flap(m))];
        name = [name, sprintf("ail. = %4.1f", aileron(n))];
        config(nc++) = struct("name",    name,
                              "alpha",   alpha(l),
                              "flap",    flap(m),
                              "aileron", aileron(n));
      endfor
    endfor
  endfor

  model.config   = config;
```

## Analysis

First, we create a new component and export the mesh. Subsequently, we retrieve information on the areas of the first four lifting surfaces and compute

their sum. This area does not include the area of the winglet. It will be needed later to compute the lift and moment coefficient.

```
# Analysis
# --------

  fid = fopen([file, ".res"], "wt");

# Create and export component

  wing = mfs_new(fid, model);
  mfs_export([file, ".msh"], "msh", wing, "mesh", "camber");

# Wing area

  A     = mfs_getresp(wing, "mesh", "area", [10, 11, 20, 21]);
  Awing = sum(A);
```

Next, we use functions **mfs_statresp** and **mfs_results** to compute the vortex strengths and the panel results, i.e. the panel pressure and forces.

```
# Compute and export results

  wing = mfs_statresp(wing);
  wing = mfs_results(wing, "statresp", "panel");
  mfs_export([file, ".pos"], "msh", wing, "statresp", "pressure");
```

The lift and moment coefficients can be computed from the resulting force and moment. These results are obtained with function **mfs_getresp**. As no reference point is specified, the moment refers to the origin of the coordinate system. Actually, because the results have been computed for a unit dynamic pressure, these forces and moments are already forces and moments per dynamic pressure.

```
# Aerodynamic coefficients

  [F, M] = mfs_getresp(wing, "statresp", "aeload");

  cL = F(3, :) / Awing;
  cM = M(2, :) / (Awing * c1);
```

The flight velocity can be computed from the condition that the lift must be in equilibrium with the weight:

$$m\,g = q_\infty c_L A = \frac{1}{2}\rho\, v_\infty^2 c_L A$$

From this equation,

$$v_\infty = \sqrt{\frac{2mg}{\rho c_L A}}$$

is obtained. Because the glider has two wings, one wing carries only half of the weight. Care has to be taken to convert the units to t and mm. The following code computes the flight velocity in m/s and subsequently converts it to km/h:

```
# Flight velocity

  v = (1e6 * mass * g / (rho * Awing)) ./ cL;
  v = 3.6 * sqrt(v);
```

The computation assumes that the contribution of the horizontal stabilizer is negligible. Besides, only the velocities computed for the configuration with no aileron deflection are meaningful because the ailerons on the two wings move in opposite direction so that the total lift remains unchanged.

The lift and moment coefficient, the flight velocity and the moment about the *x*-axis, computed for a unit dynamic pressure, are written to the output file:

```
# Output results

  fprintf(fid, "\nWing Area = %10.5e mm^2\n", Awing);

  fprintf(fid, "\n                                                    ");
  fprintf(fid, "             cL       cM     v [km/h] ");
  fprintf(fid, "Mx [Nmm/MPa]\n");
  for nc = 1 : length(config)
      fprintf(fid, "%s: ", config(nc).name);
      fprintf(fid, "%7.4f  %7.4f  %7.2f  %10.5e\n",
              cL(nc), cM(nc), v(nc), M(1, nc));
  endfor
```

The output file contains the following results:

```
Mefisto 2.7: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =  2580,  Number of panels =  2350
  Number of lifting surfaces =            5
  Number of control surfaces =            2
  Number of configurations   =           18
  Reference chord length     =  0.00000e+00
  Symmetry plane: y          =  0.00000e+00


Wing Area = 9.70148e+06 mm^2

                                                    cL       cM     v [km/h] Mx [Nmm/MPa]
Conf.  1: alpha =  0.0, flap = -2.0, ail. = -4.0:  0.3108  -0.1701   150.49  1.20049e+10
Conf.  2: alpha =  0.0, flap = -2.0, ail. =  0.0:  0.3824  -0.2010   135.68  1.68241e+10
Conf.  3: alpha =  0.0, flap = -2.0, ail. =  4.0:  0.4539  -0.2318   124.53  2.16433e+10
Conf.  4: alpha =  0.0, flap =  0.0, ail. = -4.0:  0.4186  -0.2179   129.68  1.69699e+10
Conf.  5: alpha =  0.0, flap =  0.0, ail. =  0.0:  0.4902  -0.2487   119.84  2.17890e+10
Conf.  6: alpha =  0.0, flap =  0.0, ail. =  4.0:  0.5617  -0.2796   111.95  2.66082e+10
```

```
Conf.  7: alpha =  0.0, flap =  2.0, ail. = -4.0:  0.5264  -0.2656   115.64  2.19348e+10
Conf.  8: alpha =  0.0, flap =  2.0, ail. =  0.0:  0.5979  -0.2965   108.51  2.67539e+10
Conf.  9: alpha =  0.0, flap =  2.0, ail. =  4.0:  0.6695  -0.3274   102.54  3.15731e+10
Conf. 10: alpha =  2.0, flap = -2.0, ail. = -4.0:  0.5084  -0.2181   117.67  2.10940e+10
Conf. 11: alpha =  2.0, flap = -2.0, ail. =  0.0:  0.5799  -0.2490   110.18  2.59131e+10
Conf. 12: alpha =  2.0, flap = -2.0, ail. =  4.0:  0.6515  -0.2798   103.95  3.07323e+10
Conf. 13: alpha =  2.0, flap =  0.0, ail. = -4.0:  0.6162  -0.2659   106.89  2.60589e+10
Conf. 14: alpha =  2.0, flap =  0.0, ail. =  0.0:  0.6877  -0.2967   101.18  3.08780e+10
Conf. 15: alpha =  2.0, flap =  0.0, ail. =  4.0:  0.7593  -0.3276    96.29  3.56972e+10
Conf. 16: alpha =  2.0, flap =  2.0, ail. = -4.0:  0.7240  -0.3136    98.61  3.10238e+10
Conf. 17: alpha =  2.0, flap =  2.0, ail. =  0.0:  0.7955  -0.3445    94.07  3.58429e+10
Conf. 18: alpha =  2.0, flap =  2.0, ail. =  4.0:  0.8670  -0.3754    90.11  4.06621e+10
```

Function **`mfs_xydata`** is used to obtain the pressure coefficient as a function of the *x*-coordinate in some selected wing sections. The positions of the wing sections are as follows:

1. through the middle of the first panel column of the inner wing
2. through the middle of a panel column at about 70 % of the inner wing
3. through the middle of the first panel column of the outer wing
4. through the middle of a panel column near the wing tip

Three configurations are considered:

- Configuration 5: Angle of attack, flap angle and aileron angle are zero.
- Configuration 8: Angle of attack and aileron angle are zero, flap angle is 2°
- Configuration 9: Angle of attack is zero, flap angle is 2° and aileron angle is 4°

The pressure coefficients can be plotted using the GNU Octave plotting functions.

```
# xy-plots of pressure coefficient

  cfg = [5, 8, 9];

  for k = 1 : 3
      lgtxt{k} = sprintf("Config. %2.0d", cfg(k));
  endfor

  [x1, cp1, y1] = ...
      mfs_xydata(wing, "statresp", "pressure", [10, 11], 1, cfg);
  [x2, cp2, y2] = ...
      mfs_xydata(wing, "statresp", "pressure", [10, 11],
                 floor(0.7 * nyi), cfg);
  [x3, cp3, y3] = ...
      mfs_xydata(wing, "statresp", "pressure", [20, 21], 1, cfg);
  [x4, cp4, y4] = ...
      mfs_xydata(wing, "statresp", "pressure", [20, 21],
                 floor(0.7 * nyo), cfg);

  figure(1, "position", [100, 500, 1000, 1000],
            "paperposition", [0, 0, 15, 15]);
```

```
subplot(2, 2, 1)
   plot(x1, cp1);
   ylim([-0.5, 1]);
   title(sprintf("y = %4.0f mm", y1));
   legend(lgtxt, "location", "south");
   grid;
   ylabel('\Delta c_p');
subplot(2, 2, 2)
   plot(x1, cp2);
   ylim([-0.5, 1]);
   title(sprintf("y = %4.0f mm ", y2));
   grid;
subplot(2, 2, 3)
   plot(x1, cp3);
   ylim([-0.5, 1.5]);
   title(sprintf("y = %4.0f mm ", y3));
   grid;
   ylabel('\Delta c_p');
```

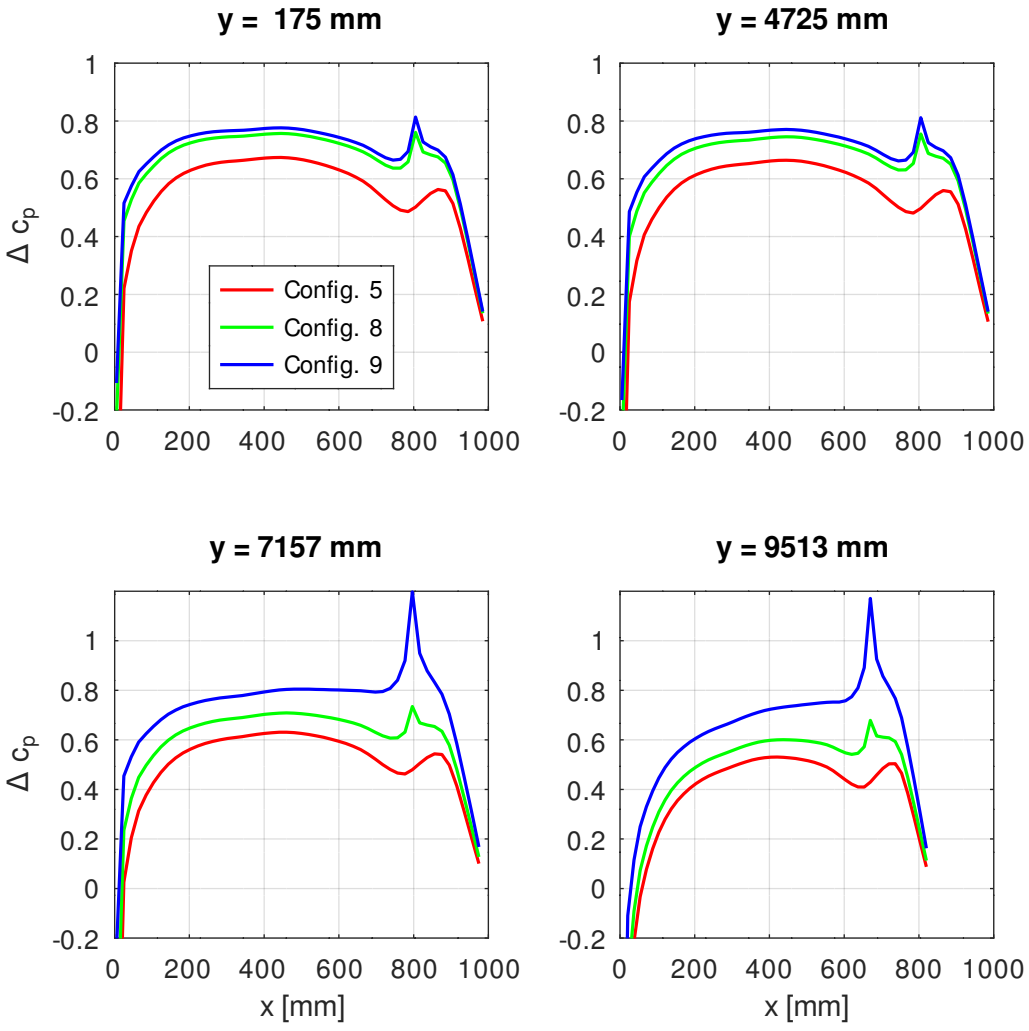



Figure 2.2-5: Glider Wing: Pressure Coefficient

```
    xlabel('x [mm]');
subplot(2, 2, 4)
    plot(x1, cp4);
    ylim([-0.5, 1.5]);
    title(sprintf("y = %4.0f mm ", y4));
    grid;
    xlabel('x [mm]');
print(["wing_cp", EXT], FORMAT);
```

Figure 2.2-5 shows the resulting diagrams. The effects of the flap and the aileron are clearly visible. In Configuration 5, all angles are zero. In Configuration 8, the flap is deflected, i.e. both the actual flap and the aileron have a deflection angle of 2°. In Configuration 9, both the flap and the aileron are deflected. The total flap angle is 2.4° and the total aileron angle is 6°.

Function **`mfs_xydata`** can also be used to obtain loads in the direction of the

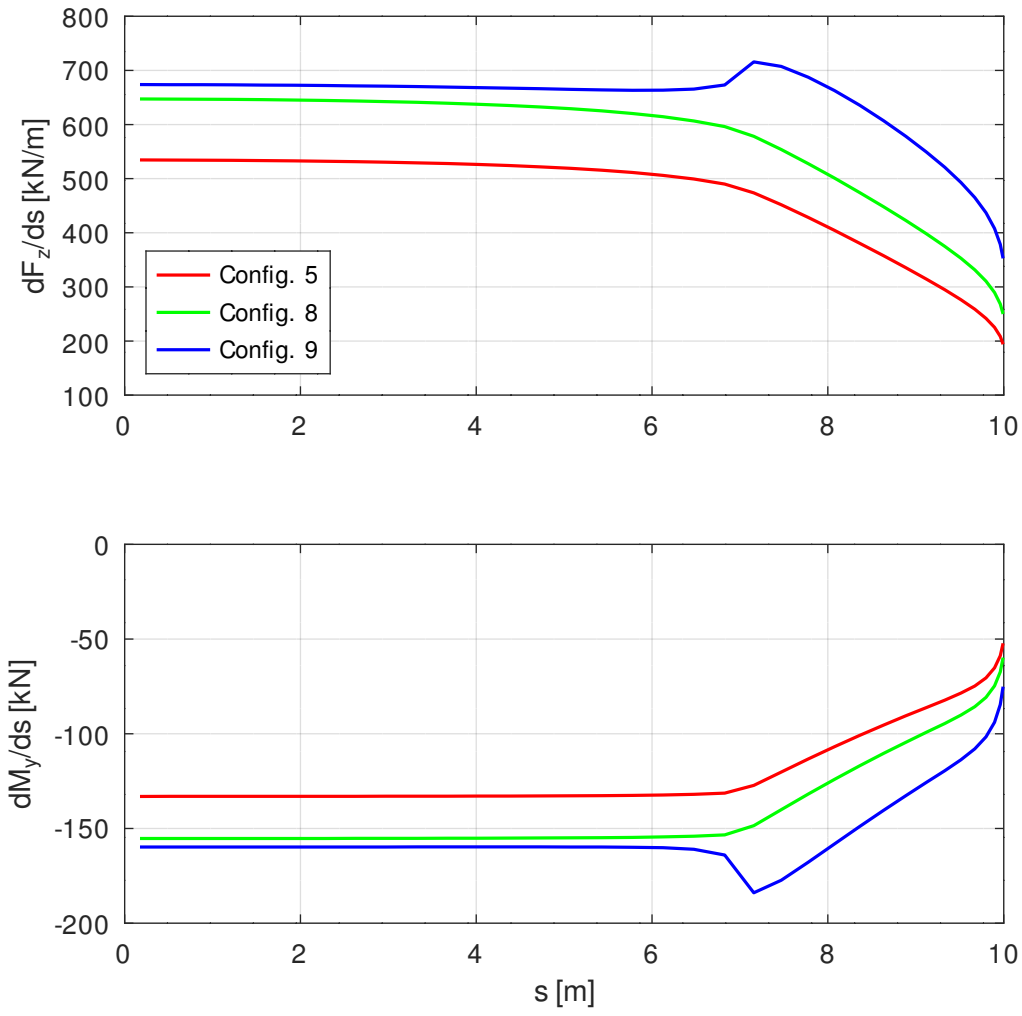



Figure 2.2-6: Glider Wing: Spanwise Loads

span. These loads are actually loads per length. The fourth argument of the function is a cell array with lifting surface groups. Lifting surfaces in each of the groups are aligned in *x*-direction. The fifth argument selects the configurations, and the sixth argument defines the reference point for computing the moments. The configurations considered are the same as for the pressure coefficient.

```
# xy-plots of spanwise loads

  refpnt = [0.25 * c1, 0, 0];
  [s, F, M] = mfs_xydata(wing, "statresp", "spanload",
                         {[10, 11], [20, 21]}, cfg, refpnt);
  s *= 1e-3;
  Fz = squeeze(F(3, :, :)); My = squeeze(M(2, :, :)) * 1e-3;

  figure(2, "position", [300, 500, 1000, 1000],
            "paperposition", [0, 0, 15, 15]);
  subplot(2, 1, 1)
     plot(s, Fz);
     legend(lgtxt, "location", "southwest");
     grid;
     ylabel("dF_z/ds [kN/m]");
  subplot(2, 1, 2)
     plot(s, My);
     grid;
     ylabel("dM_y/ds [kN]");
     xlabel("s [m]");
  print(["wing_FM", EXT], FORMAT);

  fclose(fid);
```

Figure 2.2-6 shows the lift per length and the torsional moment per length. Because of the winglet the loads are not zero at the wing tip.

# 3 Trim Analysis

## 3.1 Glider

Summary

| | |
|---|---|
| Directory: | `exa/aero/trim/glider` |
| Objectives: | • learn how to define control surfaces<br>• understand trim parameters<br>• learn how to define configurations<br>• learn how to perform a trim analysis of a rigid aircraft<br>• learn how to postprocess results of a trim analysis<br>• learn how to determine the aerodynamic centre |
| Method: | Vortex-Lattice |
| Functions: | **`mfs_airfoil, mfs_vortex2d, mfs_new, mfs_export, mfs_trim, mfs_results, mfs_getresp, mfs_xydata`** |

Problem Description

Perform a trim analysis of the standard class glider shown in Figure 3.1-1. Consider the following manoeuvres:

1. Straight flight
2. Truly banked left turn with a bank angle of 30°
3. Left turn with a bank angle of 30° and zero side slip
4. Sudden aileron deflection to 1°

The velocity of all manoeuvres is $30\ \mathrm{m/s}$.

Then carry out some additional trim analyses and use their results to determine the position of the aerodynamic centre or neutral point.

The wing has a FX 66-S-196 V1 airfoil the data of which can be found at Airfoil Tools. The rigging angle of incidence linearly decreases from 0° at the wing root to -2° at the wing tip. The dihedral angle is 2°.

The vertical and the horizontal stabilizer have symmetric airfoils. The rigging angle of incidence of the horizontal stabilizer is -3°.

The flap ratio of the aileron is 20 %. The depth of the rudder is $200\ \mathrm{mm}$ and

that of the elevator is $100\ \mathrm{mm}$.

The mass of the aircraft including the pilot is $309\ \mathrm{kg}$. The centre of mass has the coordinates $x_S = 1796\ \mathrm{mm}$, $y_S = 0$ mm and $z_S = 103\ \mathrm{mm}$. The inertia tensor with respect to the centre of mass is

$$[\boldsymbol{J}^S] = \begin{bmatrix} 2467 & 0 & -72.43 \\ 0 & 662.8 & 0 \\ -72.43 & 0 & 3083 \end{bmatrix} \mathrm{kgm}^2.$$

The mass density of the air is $1.21\ \mathrm{kg/m}^3$.

## Model Definition

Since we need the model definition in two different analyses, we use a function **`glider_aero`** to define the model. This function is contained in file `glider_aero.m`.

The definition begins with the definition of the geometry parameters and the discretization parameters. In this example, we use the units kg, m and s.

```
function model = glider_aero();
```

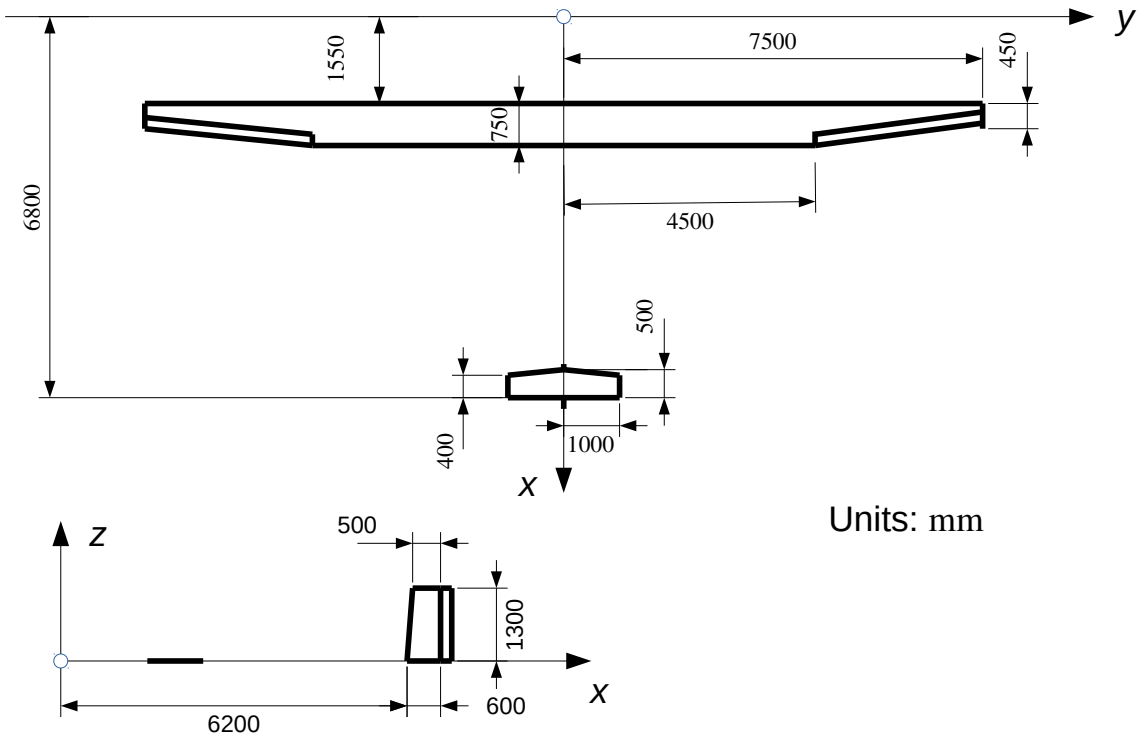



Figure 3.1-1: Geometry of Standard Class Glider

```
# usage: model = glider_aero();
#
# Input  none
#
# Output model   Structure containing the model definition
#
# This function defined the aerodynamic model of the glider.
#
# Units: kg, m, s
#
# ------------------------------------------------------------

# Data
# ----

# Geometry

  b     =   15;   % Wing span
  cr    = 0.75;   % Chord length of wing at wing root
  ct    = 0.45;   % Chord length of wing at wing tip
  delta =    2;   % Dihedral angle in degrees
  lr    =  0.4;   % Aileron length ratio
  fr    =  0.2;   % Aileron flap ratio
  at    =   -2;   % Angle of incidence at wing tip (in degrees)
  l     =  6.8;   % Length of fuselage
  be    =  2.0;   % Span of elevator
  h     =  1.3;   % Height of fin
  xle   = 1.55;   % x-position of leading edge

# Discretization

  nsa =   5;      % No. of spline segments to fit airfoil
  nxi =  50;      % No. of panels in x-direction of inner wing
  nxo =  40;      % No. of panels in x-direction of outer wing
  nxa =  10;      % No. of panels in x-direction of aileron
  nyi =  20;      % No. of panels in y-direction of inner wing
  nyo =  24;      % No. of panels in y-direction of outer wing
```

Next, the airfoil data are imported, and the airfoil is approximated by a piecewise polynomial. Of course, this approximation has been checked before by computing the pressure distribution of the 2-dimensional airfoil using function **mfs_vortex2d** (cf. Example 2.2).

```
# Airfoil: FX 66-S-196 V1

  zs   = dlmread("fx66s196v1.csv", ",", "A100:B143");
  cmbi = mfs_airfoil("fit", "camber", zs, nsa);
  cmbo = mfs_airfoil("fit", "camber", zs, nsa, 0, 1 - fr);
  cmba = mfs_airfoil("fit", "camber", zs, nsa, 1 - fr, 1);
```

The first polynomial is used for the inner wing. It applies to the complete airfoil. The second polynomial is used for the front part of the outer wing. It ap-

plies to the first 80 % of the airfoil. Finally, the third polynomial, which applies to the last 20 % of the airfoil, is used for the aileron.

Now we can define the aerodynamic model. First, we define the points and the lifting surfaces of the right and the left wing. We have to make sure that all lifting surfaces have the same orientation. The identifiers of the lifting surfaces of the right wing begin with 1 and those of the left wing with 2.

```
# Model Definition
# ----------------

# Model type and subtype

  model = struct("type", "aero", "subtype", "vlm");

# Some coordinates and lengths

  yt  = 0.5 * b;           % y-coordinate of wing tip
  ya  = (1 - lr) * yt;     % y-coordinate of aileron

  dihedral = tand(delta);
  zt = yt * dihedral;      % z-coordinate of wing tip
  za = ya * dihedral;      % z-coordinate at start of aileron

  ca1 = fr * cr;           % inner aileron chord length
  ca2 = fr * ct;           % outer aileron chord length
  co1 = cr - ca1;          % inner outer wing chord length
  co2 = ct - ca2;          % outer outer wing chord length

  aa = (1 - lr) * at;      % Angle of incidence at start of aileron

# Right wing: Inner wing, outer wing, aileron

  points(1 : 5) = struct("id", {1, 2, 3, 4, 5},
                         "coor", {[xle, 0, 0], ...
                                  [xle, ya, za], ...
                                  [xle, yt, zt], ...
                                  [xle + co1, ya, za], ...
                                  [xle + co2, yt, zt]});

  lsf(1 : 3) = struct("id",     {10, 11, 12},
                      "points", {[1, 2], [2, 3], [4, 5]},
                      "chord",  {cr, [co1, co2], [ca1, ca2]},
                      "camber", {cmbi, cmbo, cmba},
                      "nx",     {nxi, nxo, nxa},
                      "ny",     {nyi, nyo, nyo},
                      "typex",  "linear",
                      "typey",  {"linear", "cos>", "cos>"},
                      "alpha",  {[0, aa], [aa, at], [aa, at]});

# Left wing: Inner wing, outer wing, aileron

  points(6 : 10) = struct("id", {6, 7, 8, 9, 10},
                          "coor", {[xle, 0, 0], ...
```

```
                                    [xle, -ya, za], ...
                                    [xle, -yt, zt], ...
                                    [xle + co1, -ya, za], ...
                                    [xle + co2, -yt, zt]});

  lsf(4 : 6) = struct("id",     {20, 21, 22},
                      "points", {[7, 6], [8, 7], [10, 9]},
                      "chord",  {cr, [co2, co1], [ca2, ca1]},
                      "camber", {cmbi, cmbo, cmba},
                      "nx",     {nxi, nxo, nxa},
                      "ny",     {nyi, nyo, nyo},
                      "typex",  "linear",
                      "typey",  {"linear", "<cos", "<cos"},
                      "alpha",  {[aa, 0], [at, aa], [at, aa]});
```

We continue with the definition of the points and the lifting surfaces of the vertical and the horizontal stabilizer. The identifiers of the lifting surfaces of the vertical stabilizer begin with 3 and those of the horizontal stabilizer with 4. The orientation of the lifting surfaces of the vertical stabilizer is such that the normal vector points to the left (in opposite *y*-direction).

```
# Vertical stabilizer: Points

  points(11 : 14) = struct("id",   {31, 32, 33, 34},
                           "coor", {[l - 0.6, 0, 0], ...
                                    [l - 0.5, 0, h], ...
                                    [l, 0, 0], ...
                                    [l, 0, h]});

# Vertical stabilizer: Fin and rudder

  lsf(7 : 8) = struct("id",     {31, 32},
                      "points", {[31, 32], [33, 34]},
                      "chord",  {[0.6, 0.5], [0.2]},
                      "camber", [],
                      "nx",     {30, 10},
                      "ny",     8,
                      "typex",  "linear",
                      "typey",  "cos",
                      "alpha",  0);

# Horizontal stabilizer: Points

  points(15 : 18) = struct("id",   {41, 42, 43, 44},
                           "coor", {[l - 0.4, 0.5 * be, h], ...
                                    [l, 0.5 * be, h], ...
                                    [l - 0.4, -0.5 * be, h], ...
                                    [l, -0.5 * be, h]});

# Horizontal stabilizer: Fin

  lsf( 9 : 10) = struct("id",     {41, 42},
                        "points", {[32,41], [43, 32]},
                        "chord",  {[0.5, 0.4], [0.4, 0.5]},
```

```
                          "camber", [],
                          "nx",     30,
                          "ny",      8,
                          "typex",  "linear",
                          "typey",  {"cos>", "<cos"},
                          "alpha",  -3);

# Horizontal stabilizer: Elevator

  lsf(11 : 12) = struct("id",     {43, 44},
                          "points", {[34, 42], [44, 34]},
                          "chord",  0.1,
                          "camber", [],
                          "nx",     6,
                          "ny",     8,
                          "typex",  "linear",
                          "typey",  {"cos>", "<cos"},
                          "alpha",  -3);

  model.points = points;
  model.ls     = lsf;
```

The resulting discretization can be seen in Figure 3.1-2. The total number of panels is 5296.

Now we can define the control surfaces. The following sign conventions are

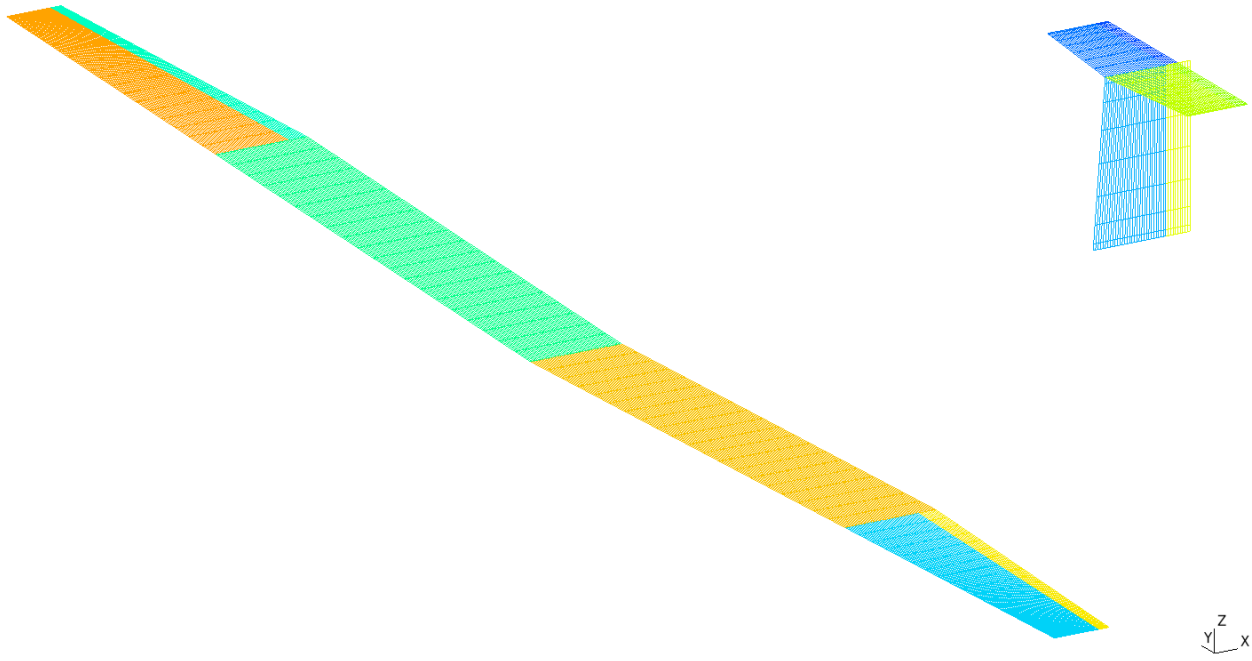



Figure 3.1-2: Discretization of the Glider

used:

1. A positive value of the aileron means the stick is moved to the right, i.e. the right wing aileron moves up and the left wing aileron moves down.
2. A positive value of the rudder means that the right pedal is pushed, i.e. the rudder moves to the right.
3. A positive value of the elevator means that the stick is moved backwards (pulled), i.e. the elevator moves upwards.

Please note that a positive angle always means a rotation of the control surface towards its negative side, i.e. opposite to its normal vector.

```
# Control surfaces
#    Positive aileron : stick to the right
#    Positive rudder  : right pedal
#    Positive elevator: pull
# ------------------------------------------------------------

  controls = struct("name",    {"aileron", "rudder", "elevator"},
                    "ls",      { [12, 22],    32,     [43, 44]},
                    "factors", { [-1, 1],      1,     [-1, -1]});
  model.controls = controls;
```

In a trim analysis, also the rigid body mass properties are needed, i.e. the mass, the centre of mass and the inertia tensor. In case of a rigid trim analysis, these data need be defined by the user. In case of an aeroelastic trim analysis they are computed from the mass matrix of the solid structure.

```
# Mass properties
# ---------------

  model.mass = struct("m",  309,
                      "cm", [1.796, 0., 0.103],
                      "JS", [  2467,     0, -72.43;
                                  0, 662.8,      0;
                             -72.43,    0,   3083]);

endfunction
```

Trim Analysis

In a trim analysis, the configurations define the manoeuvres to be analysed. Six of the trim parameters are determined from the dynamic equilibrium conditions. The remaining parameters have to be specified. They define the manoeuvre.

GNU Octave script `trim.m` contains the commands needed to run a trim analysis. The script begins with the definition of the parameters that are needed to define the manoeuvres. Then function **`glider_aero`** is called to

define the model.

```
# Example: Trim analysis of a standard class glider
#
# Manoeuvres:
#
#  1. Straight level flight
#  2. Truly banked turn (bank angle = 30°)
#  3. Turn with zero side slip (bank angle = 30°)
#  4. Sudden aileron deflection (deflection angle = 1°)
#
# ------------------------------------------------------------

  [EXT, FORMAT] = iniplot();

  set(0, "defaultaxesfontsize", 10);

  fid  = fopen("trim.res", "wt");

# Data (kg, m, s)
# ---------------

  v     =   30;   % Flight velocity
  rho   = 1.21;   % Mass density of air
  g     = 9.81;   % Gravity acceleration
  gamma =   30;   % Bank angle in degrees
  eta   =    1;   % Aileron deflection in degrees
  ds    =  1.0;   % Distance of yaw string from center of mass

# Model Definition
# ----------------

  model = glider_aero();
```

Next, we define the configurations. The dynamic pressure is the same for all three manoeuvres.

```
# Configurations

  qdyn = 0.5 * rho * v^2;
```

The first manoeuvre is a straight level flight. The following trim parameters define the manoeuvre:

1. The linear acceleration **ay** is zero. The linear acceleration **az** equals the gravity acceleration.
2. The angular accelerations **pacce**, **yacce** and **racce** (pitch, yaw and roll accelerations) are zero.
3. The angular velocities **pitch**, **yaw** and **roll** are zero.

The following six trim parameters are determined in the trim analysis:

1. the linear acceleration **ax**
2. the angle of attack **alpha** and the sideslip angle **beta**
3. the three control surface angles **aileron**, **rudder** and **elevator**

Thus, the definition of this manoeuvre is as follows:

```
% Straight level flight

config(1).name  = "Manoeuvre 1: Straight level flight";
config(1).qdyn  = qdyn;
config(1).ay    = 0;
config(1).az    = g;
config(1).pacce = 0;
config(1).yacce = 0;
config(1).racce = 0;
config(1).pitch = 0;
config(1).yaw   = 0;
config(1).roll  = 0;
```

The second manoeuvre is a truly banked turn, as shown in Figure 3.1-3. The vertical acceleration can be computed from the bank angle $\gamma$:

$$g = a_z \cos(\gamma) \;\rightarrow\; a_z = \frac{g}{\cos(\gamma)} \tag{3.1-1}$$

The radius $R$ of the turn can be computed from the bank angle and the flight velocity. The centrifugal force $Z_F$ is

$$Z_F = m \frac{v^2}{R} \quad . \tag{3.1-2}$$

Figure 3.1-3 shows that

$$\tan(\gamma) = \frac{Z_F}{m\,g} = \frac{v^2}{g\,R} \quad . \tag{3.1-3}$$

Thus, the radius of the turn is

$$R = \frac{v^2}{g \tan(\gamma)} . \tag{3.1-4}$$

The angular velocity $\omega$ about the $z_E$-axis of the earth coordinate system is

$$\omega = \frac{v}{R} = \frac{g}{v} \tan(\gamma) \tag{3.1-5}$$

from which the pitch and yaw rates are obtained:

$$q = \omega \sin(\gamma), \quad r = \omega \cos(\gamma) \tag{3.1-6}$$

For the truly banked turn we have to define the same trim parameters as for the straight level flight, but we have to give them the values computed above.

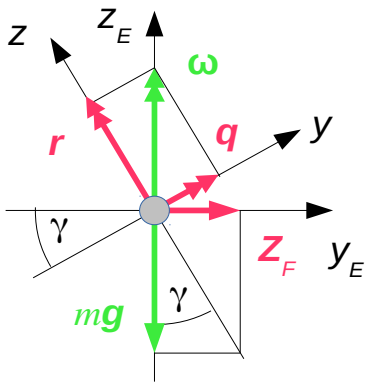



Figure 3.1-3: Truly Banked Turn

Please note that the pitch and the yaw rate have to be divided by the velocity.

```
% Truly banked turn

tg = tand(gamma);
cs = 1 / sqrt(1 + tg^2); sn = tg * cs;
w  = tg * g / v;

config(2).name  = "Manoeuvre 2: Truly banked turn";
config(2).qdyn  = qdyn;
config(2).ay    = 0;
config(2).az    = g / cs;
config(2).pacce = 0;
config(2).yacce = 0;
config(2).racce = 0;
config(2).pitch = w * sn / v;
config(2).yaw   = w * cs / v;
config(2).roll  = 0;
```

The six trim parameters that are determined in the trim analysis are the same as in the first manoeuvre.

The third manoeuvre is a left turn as flown by glider pilots. Glider pilots usually try to keep the yaw string parallel to the longitudinal axis of the aircraft, i.e. the side slip angle at the point where the yaw string is attached to the canopy must be zero. The definition of this manoeuvre is a little more complicated.

First, as with the truly banked turn, we have (cf. Figure 3.1-3)

$$\omega = \frac{v}{R} \tag{3.1-7}$$

and

$$q = \omega \sin(\gamma), \quad r = \omega \cos(\gamma). \tag{3.1-8}$$

The components of the acceleration with respect to the earth fixed coordinate system must fulfil the conditions

$$a_{y_E} = a_y \cos(\gamma) - a_z \sin(\gamma) = -\frac{v^2}{R} = -\omega v \tag{3.1-9}$$

and

$$a_{z_E} = a_y \sin(\gamma) + a_z \cos(\gamma) = g. \tag{3.1-10}$$

Equation 3.1-10 is already a linear relation between the trim variables $a_y$ and $a_z$. From the Equation 3.1-9 we get

$$\omega = -\frac{1}{v}\left(\cos(\gamma)\, a_y - \sin(\gamma)\, a_z\right). \tag{3.1-11}$$

Insertion of Equation 3.1-11 into Equations 3.1-8 yields two further linear relations between the trim variables $q$, $r$, $a_y$ and $a_z$:

$$v^2 \frac{q}{v} + \sin(\gamma)\cos(\gamma)\, a_y - \sin^2(\gamma)\, a_z = 0 \tag{3.1-12}$$

$$v^2 \frac{r}{v} + \cos^2(\gamma)\, a_y - \sin(\gamma)\cos(\gamma)\, a_z = 0 \tag{3.1-13}$$

If the position of the yaw string is a distance $d_S$ in front of the centre of mass and $\beta$ is the side slip angle at the centre of mass, then the condition that the side slip angle at the position of the yaw string is zero reads

$$\beta + d_S \frac{r}{v} = 0. \tag{3.1-14}$$

Equations 3.1-10 and 3.1-12 to 3.1-14 are four linear relations between the trim variables. In addition, all three angular accelerations and the roll rate are zero. This manoeuvre is thus defined by specifying the values of four trim parameters and four linear relations.

```
% Turn with zero side slip

config(3).name  = "Manoeuvre 3: Turn with zero side slip";
config(3).qdyn  = qdyn;
config(3).pacce = 0;
config(3).yacce = 0;
config(3).racce = 0;
config(3).roll  = 0;

ss = sn * sn; cc = cs * cs; sncs = sn * cs; vv = v * v;

lincon{1} = struct("ay", sn, "az", cs, "rhs", g);
lincon{2} = struct("ay", sncs, "az", -ss, "pitch", vv);
lincon{3} = struct("ay", cc, "az", -sncs, "yaw", vv);
lincon{4} = struct("beta", 1, "yaw", ds);

config(3).lincon = lincon;
```

The following ten trim parameters are determined in the trim analysis:

1. the linear accelerations **ax**, **ay** and **az**
2. the angle of attack **alpha** and the sideslip angle **beta** at the centre of mass
3. the pitch rate **pitch** and the yaw rate **yaw**, divided by the velocity
4. the three control surface angles **aileron**, **rudder** and **elevator**

The last manoeuvre is a sudden aileron deflection. This is a quasi-steady manoeuvre, i.e. the results only apply immediately at the start of the manoeuvre.

The following trim parameters define the manoeuvre:

1. The linear acceleration **ay** is zero. The linear acceleration **az** equals the gravity acceleration.
2. The angular accelerations **pacce** and **yacce** (pitch and yaw accelerations) are zero.
3. The angular velocities **pitch**, **yaw** and **roll** are zero.
4. The aileron is deflected.

The following six trim parameters are determined in the trim analysis:

1. the linear acceleration **ax**
2. the angle of attack **alpha** and the sideslip angle **beta**
3. the roll acceleration **racce**
4. the control surface angles **rudder** and **elevator**

This results in the following definition of the manoeuvre:

```
% Sudden aileron deflection

config(4).name  = "Manoeuvre 4: Sudden aileron deflection";
config(4).qdyn  = qdyn;
config(4).ay    = 0;
config(4).az    = g;
config(4).pacce = 0;
config(4).yacce = 0;
config(4).pitch = 0;
config(4).yaw   = 0;
config(4).roll  = 0;
config(4).aileron = eta;

model.config = config;
```

The model, including the configurations, is now completely defined. We continue by creating a new component **glider** from the model definition and exporting the mesh for post-processing with Gmsh. Next, we perform the trim analysis. The trim analysis determines the values of the six trim parameters that are not specified and computes the vortex strengths. The trim parameters are written to the output file. Subsequently, the panel results, i.e. the panel pressure and forces, are computed and exported to Gmsh for post-processing.

```
# Analysis
# --------

# Create and export component

  glider = mfs_new(fid, model);
  mfs_export("glider.msh", "msh", glider, "mesh",
             "mesh", "normals");

# Perform the trim analysis

  glider = mfs_trim(glider);
  mfs_print(fid, glider, "trim", "params");

# Compute and export results

  glider = mfs_results(glider, "trim", "panel");
  mfs_export("trim.pos", "msh", glider, "trim", "pressure");
```

Next, we retrieve the resulting forces and moments and write them to the output file:

```
# Get load resultants

  [F, M] = mfs_getresp(glider, "trim", "aeload", model.mass.cm);
  [n, nconf] = size(F);

  fprintf(fid,
          "\nLoad resultants with respect to center of mass:\n");
  for k = 1 : nconf
      fprintf(fid, "\n  Configuration %2d:\n", k)
      fprintf(fid, "     F = [%10.3e, %10.3e, %10.3e] kN\n",
              F(:, k) / 1000);
      fprintf(fid, "     M = [%10.3e, %10.3e, %10.3e] kNm\n",
              M(:, k) * 1e-3);
  endfor
```

Then we retrieve the pressure as a function of the *x*-coordinate in two wing sections of the left wing and two wing sections of the right wing and plot it using the GNU Octave plot routines. Two of the sections are approximately in the middle of the inner wing. The other two sections are through the outer

wing with the aileron. The positions of the sections are symmetric with respect to the *xz*-plane.

```
# Get pressure in some wing sections

  nyi   = model.ls(1).ny;   nyo  = model.ls(2).ny;
  ycmi  = floor(0.5 * nyi); ycmo = floor(0.5 * nyo);
  ycoli = [ycmi, nyi + 1 - ycmi];
  ycolo = [ycmo, nyo + 1 - ycmo];

  [x1, p1, y(1)] = mfs_xydata(glider, "trim", "pressure",
                              [21, 22], ycolo(1));
  [x2, p2, y(2)] = mfs_xydata(glider, "trim", "pressure",
                              20, ycoli(1));
  [x3, p3, y(3)] = mfs_xydata(glider, "trim", "pressure",
                              10, ycoli(2));
  [x4, p4, y(4)] = mfs_xydata(glider, "trim", "pressure",
                              [11, 12], ycolo(2));

  for k = 1 : 4
      tittxt{k} = sprintf("y = %6.3f m", y(k));
  endfor

  figure(1, "position", [200, 200, 1000, 1000],
            "paperposition", [0, 0, 15, 15]);
  subplot(2, 2, 1);
     plot(x1, p1 * 1e-3);
     title(tittxt{1});
     grid;
     ylim([0, 0.4]);
     ylabel('\Delta p [kPa]');
  subplot(2, 2, 2);
     plot(x4, p4 * 1e-3);
     legend("Man. 1", "Man. 2", "Man. 3", "Man. 4",
            "location", "south");
     title(tittxt{4});
     grid;
     ylim([-0.2, 0.4]);
  subplot(2, 2, 3);
     plot(x2, p2 * 1e-3);
     title(tittxt{2});
     grid;
     ylim([0.1, 0.5]);
     xlabel('x [m]'); ylabel('\Delta p [kPa]');
  subplot(2, 2, 4);
     plot(x3, p3 * 1e-3);
     title(tittxt{3});
     grid;
     ylim([0.1, 0.5]);
     xlabel('x [m]');
  print(["p_wing", EXT], FORMAT);
```

We also plot the pressure in a section through the vertical stabilizer.

```
# Plot pressure in vertical stabilizer section

  [xv, pv, ~, zv] = mfs_xydata(glider, "trim", "pressure",
                               [31, 32], 4);

  figure(2, "position", [500, 200, 500, 500],
            "paperposition", [0, 0, 10, 10]);
  plot(xv, pv * 1e-3)
  title(sprintf("z = %6.3f m", zv));
  legend("Man. 1", "Man. 2", "Man. 3", "Man. 4");
  grid;
  xlabel('x [m]'); ylabel('\Delta p [kPa]');
  print(["p_vert", EXT], FORMAT);
```

Next, we compute the radius of the turn for the second and third manoeuvre and the yaw string angle for all manoeuvres.

For the truly banked turn (configuration 2), the radius of the turn is obtained from Equation 3.1-4. For the turn with zero sideslip angle (configuration 3) we use Equations 3.1-7 and 3.1-11 to get

$$R = \frac{v}{\omega} = \frac{v^2}{\sin(\gamma) a_z - \cos(\gamma) a_y} \quad . \tag{3.1-15}$$

The yaw string angle is calculated according to Equation 3.1-14.

```
# Compute radius of turn and yaw string angle

  tp = mfs_getresp(glider, "trim", "params");

  R2 = vv / (g * tg);
  R3 = vv / (sn * tp.az(3) - cs * tp.ay(3));

  fprintf(fid, "\nRadius of truly banked turn       : %6.2f m\n",
          R2);
  fprintf(fid, "Radius of turn with zero side slip: %6.2f m\n",
          R3);

  ysa = (tp.beta + ds * tp.yaw) * 180 / pi;

  fprintf(fid, "\nYaw string angles:\n");
  fprintf(fid, "  %6.3f, %6.3f, %6.3f, %6.3f deg.\n", ysa);
```

Finally, we access the loads in span direction of the right wing and plot the lift per length along the wing. By integrating the lift per length and the torsional moment per length, we get the resulting force in *z*-direction and the resulting moment about the *y*-axis respectively. These values should agree with the aerodynamic load resultants of the right wing which are obtained with function **mfs_getresp**.

```
# Get loads along right wing
```

```
  xle = model.points(1).coor(1);
  cr  = model.ls(1).chord;
  refpnt = [xle + 0.25 * cr, 0, 0];
  [s, f, m, xy, w] = mfs_xydata(glider, "trim", "spanload",
                                {10, [11, 12]}, [], refpnt);

  figure(3, "position", [800, 100, 750, 500],
            "paperposition", [0, 0, 15, 10]);
  plot(s, squeeze(f(3, :, :)));
  legend("Man. 1", "Man. 2", "Man. 3", "Man. 4",
         "location", "southwest");
  grid;
  xlabel('s [m]');
  ylabel('dF_z/ds [N/m]');
  print(["Fz", EXT], FORMAT);

# Compare loads on right wing

  [F, M] = mfs_getresp(glider, "trim", "aeload", refpnt, 10 : 12);
  fdisp(fid, "\nFref"); fdisp(fid, F);
  fdisp(fid, "\nMref"); fdisp(fid, M);

  Fz = sum(squeeze(f(3, :, :)) .* w');
  fdisp(fid, "\nFz"); fdisp(fid, Fz);
  My = sum(squeeze(m(2, :, :)) .* w');
  fdisp(fid, "\nMy"); fdisp(fid, My);

  fclose(fid);
```

## Results of the Trim Analysis

The trim parameters, the load resultants, the radius of the turn as well as the values of the yaw string angle can be found in the output file:

```
Mefisto 2.8: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =  5804,  Number of panels =  5296
  Number of lifting surfaces =          12
  Number of control surfaces =           3
  Number of configurations   =           4
  Reference chord length     =  0.00000e+00


------------------------------------------------------------------------------

Component "glider"

Results of rigid trim analysis (Angles are in degrees)

  Configuration   1: Manoeuvre 1: Straight level flight
  Configuration   2: Manoeuvre 2: Truly banked turn
  Configuration   3: Manoeuvre 3: Turn with zero side slip
  Configuration   4: Manoeuvre 4: Sudden aileron deflection

  Configuration                1            2            3            4

  qdyn                =   5.4450e+02    5.4450e+02    5.4450e+02    5.4450e+02
  ax                  =   0.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00
  ay                  =   0.0000e+00    0.0000e+00   -7.7215e-03    0.0000e+00
```

```
  az                    =    9.8100e+00    1.1328e+01    1.1332e+01    9.8100e+00
  racce                 =    0.0000e+00    0.0000e+00    0.0000e+00   -1.4192e-01
  pacce                 =    0.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00
  yacce                 =    0.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00
  alpha                 =    1.8250e+00    2.6059e+00    2.6083e+00    1.8250e+00
  beta                  =   -4.5857e-15    1.3213e+00   -3.1275e-01   -2.9327e+00
  pitch                 =    0.0000e+00    3.1466e-03    3.1515e-03    0.0000e+00
  yaw                   =    0.0000e+00    5.4500e-03    5.4586e-03    0.0000e+00
  roll                  =    0.0000e+00    0.0000e+00    0.0000e+00    0.0000e+00
  aileron               =   -7.7314e-15   -1.4043e-01    5.2010e-02    1.0000e+00
  elevator              =   -1.1974e+00    1.2829e+00    1.2881e+00   -1.1974e+00
  rudder                =    9.1480e-15   -4.2086e-01   -2.7641e+00   -4.3188e+00

Load resultants with respect to center of mass:

  Configuration  1:
     F = [ 0.000e+00,  1.339e-16,  3.031e+00] kN
     M = [-6.212e-15,  3.408e-16,  2.181e-17] kNm

  Configuration  2:
     F = [ 0.000e+00,  9.940e-18,  3.500e+00] kN
     M = [ 2.345e-15,  1.843e-16,  3.056e-17] kNm

  Configuration  3:
```

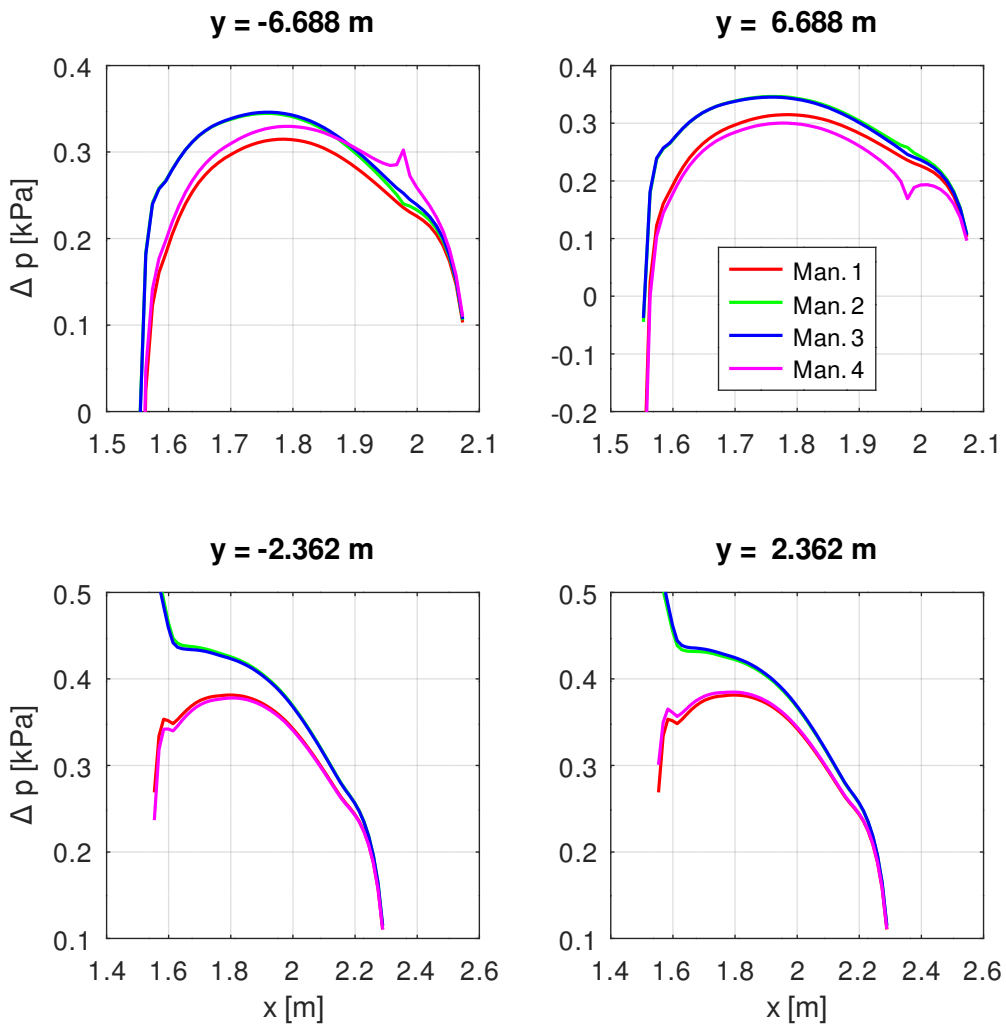



Figure 3.1-4: Glider: Pressure in Selected Wing Sections

```
     F = [ 0.000e+00, -2.386e-03,  3.502e+00] kN
     M = [-1.335e-17, -1.502e-17,  1.156e-17] kNm

  Configuration  4:
     F = [ 0.000e+00, -4.392e-17,  3.031e+00] kN
     M = [-3.501e-01,  6.849e-16,  1.028e-02] kNm

Radius of truly banked turn       : 158.90 m
Radius of turn with zero side slip: 158.65 m

Yaw string angles:
  -0.000,  1.634,  0.000, -2.933 deg.
```

Angles are in degrees, and the pitch, yaw and roll rates are rates divided by the flight velocity.

Because there are no forces in *x*-direction, the acceleration **ax** is zero for all three manoeuvres.

Manoeuvres 1 and 4 have the same angle of attack and the same elevator angle. Remember that a positive elevator angle means pull.

As expected, the angle of attack and the elevator angle for the turns are larger than for the straight level flight. The turns need a higher lift, as can be seen from the load resultants.

For the truly banked turn, the sideslip angle **beta** is not zero but its value of 1.32° is small. The yaw string angle has a value of 1.63°. This means that the tip of the yaw string points to the right. For the turn with zero yaw string angle the lateral acceleration **ay** is not zero but its value is small. This explains why

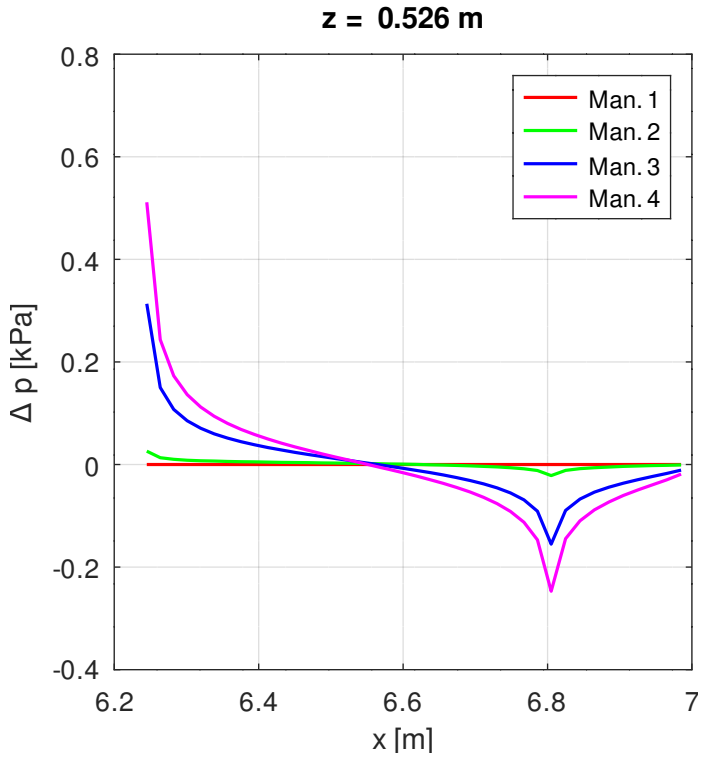



Figure 3.1-5: Glider: Pressure in Section through Vertical Stabilizer

it feels more comfortable to fly a turn with the yaw string pointing slightly in the opposite direction. The difference of the radii of the two turns is very small.

For the first two manoeuvres, all load resultants except the force in vertical direction are zero. For the third manoeuvre, there is also a small resulting force in lateral direction.

For the fourth manoeuvre, there is a resulting negative moment about the *x*-axis. This corresponds to a negative roll acceleration as is to be expected when the stick is moved to the right. Both this moment and the roll acceleration can be regarded as a measure of the effectiveness of the aileron.

Figure 3.1-4 shows the pressure in the wing sections. The upper two diagrams show the pressure in the wing sections containing the aileron.

The lower two diagrams show the pressure in the wing sections of the inner wing. It can be seen that the pressures of manoeuvres 1 and 4 are slightly different, indicating that the aileron deflection also influences the pressure of the inner wing.

Figure 3.1-5 shows the pressure in a section through the vertical stabilizer. It can be seen that the pressure difference for the turn with zero yaw string angle is much greater than for the truly banked turn. For the truly banked turn, the pressure difference is almost zero. Since pressure differences result in induced drag, a truly banked turn is actually more efficient than a turn with zero

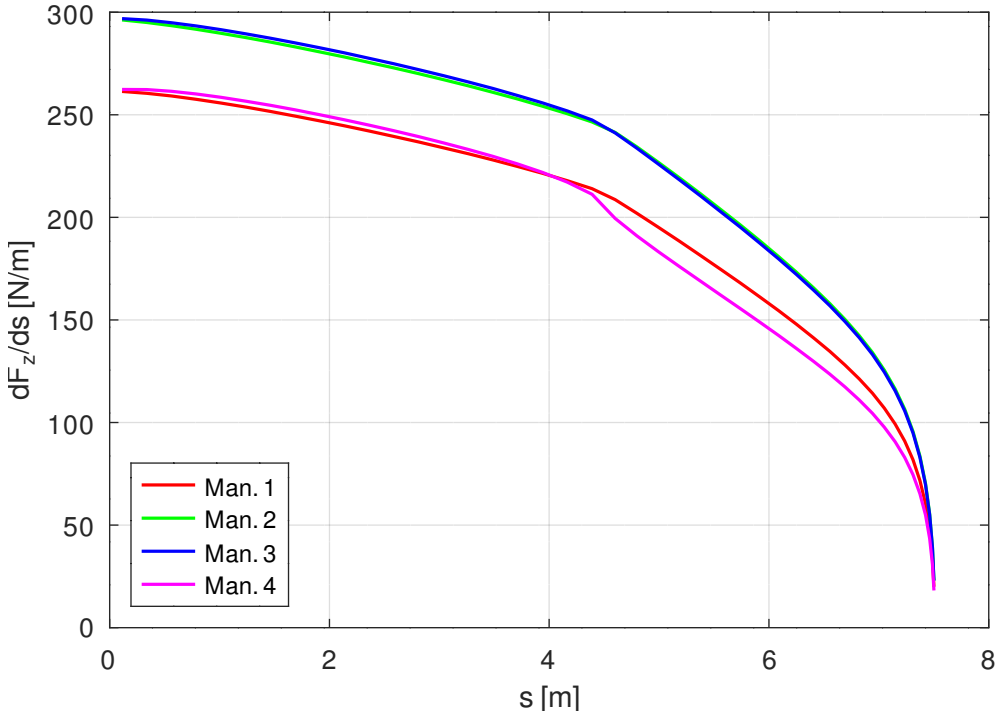



Figure 3.1-6: Glider: Lift per Length along Right Wing

yaw string angle.

The output file ends with a comparison of the aerodynamic load resultants:

```
Fref
             0             0             0             0
  -5.3910e+01  -6.1843e+01  -6.2010e+01  -5.3075e+01
   1.5438e+03   1.7709e+03   1.7757e+03   1.5199e+03

Mref
   5.0030e+03   5.7817e+03   5.7822e+03   4.8255e+03
  -2.1978e+02  -2.1987e+02  -2.1867e+02  -2.1432e+02
  -7.6749e+00  -7.6780e+00  -7.6361e+00  -7.4841e+00

Fz
   1543.8   1770.9   1775.7   1519.9

My
  -219.78  -219.87  -218.67  -214.32
```

**`Fref`** and **`Mref`** are the load resultants obtained with function **`mfs_getresp`** whereas **`Mz`** and **`My`** are the load resultants obtained by integrating the line loads. It can be seen that the values are identical. The lift per length along the right wing is shown in Figure 3.1-6.

Aerodynamic centre

The aerodynamic centre or neutral point is very important for the dynamics of flight. The moment with respect to the neutral point, resulting from all aerodynamic forces on the aircraft, does not depend on the angle of attack.

Figure 3.1-7 shows the free body diagram of the aircraft. *S* denotes the centre of mass and *N* the neutral point. ***L*** is the resulting aerodynamic force and ***M*** the resulting aerodynamic moment. The balance of moments with respect to the neutral point reads

$$\sum M_y^N = 0 \;:\; M - (x_N - x_S)\, m\, g = 0 \qquad (3.1\text{-}16)$$

With the dynamic pressure $q_\infty$, the moment coefficient $c_M$ with respect to the neutral point, the reference surface $S$ and the reference length $c$, the aerodynamic moment is

$$M = q_\infty S\, c\, c_M \quad . \qquad (3.1\text{-}17)$$

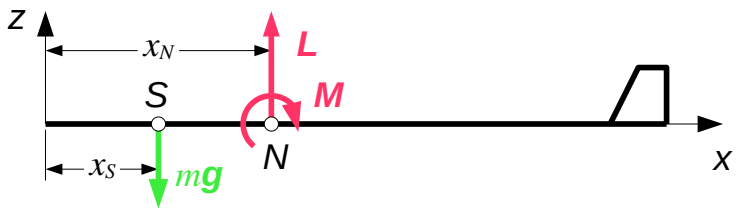



Figure 3.1-7: Free Body Diagram of Glider

For small deflections of the elevator, the moment coefficient $c_M$ depends linearly on the deflection angle $\eta_E$:

$$c_M = c_{M0} + \frac{\partial c_M}{\partial \eta_E}\eta_E \tag{3.1-18}$$

Inserting Equations 3.1-17 and 3.1-18 into Equation 3.1-16 we get

$$q_\infty S c\left(c_{M0} + \frac{\partial c_M}{\partial \eta_E}\eta_E\right) = (x_N - x_S) mg \tag{3.1-19}$$

which can be solved for the deflection angle $\eta_E$:

$$\eta_E = \frac{(x_N - x_S) m g}{S c \partial c_M / \partial \eta_E}\frac{1}{q_\infty} - \frac{c_{M0}}{\partial c_M / \partial \eta_E} \tag{3.1-20}$$

Equation 3.1-20 shows that the deflection angle $\eta_E$ is proportional to the reciprocal of the dynamic pressure $q_\infty$. With

$$a = \frac{(x_N - x_S) m g}{S c \partial c_M / \partial \eta_E}, \quad \eta_{E\infty} = \frac{c_{M0}}{\partial c_M / \partial \eta_E} \tag{3.1-21}$$

Equation 3.1-20 reads

$$\eta_E = a\frac{1}{q_\infty} - \eta_{E\infty} \quad . \tag{3.1-22}$$

The proportionality constant $a$ itself is proportional to $x_N$ - $x_S$,

$$a = -b(x_N - x_S) \tag{3.1-23}$$

with

$$b = \frac{-m g}{S c \partial c_M / \partial \eta_E} \quad . \tag{3.1-24}$$

Once $a$ and $b$ are known, Equation 3.1-23 can be solved for $x_N$:

$$\frac{a}{b} = x_S - x_N \;\rightarrow\; x_N = x_S - \frac{a}{b} \tag{3.1-25}$$

The coefficients $a$ and $b$ can be computed from two suitable trim analyses. In a first trim analysis, the constant $a_A$ is be determined for $x_S = x_{SA}$. The trim analysis is carried out for two straight level flights with different velocities. Equation 3.1-22 is used to calculate $a_A$ from the elevator angles and the dynamic pressures:

$$a_A = \frac{\eta_{E2} - \eta_{E1}}{1/q_{\infty 2} - 1/q_{\infty 1}} \tag{3.1-26}$$

In the same way, the second trim analysis determines the constant $a_B$ for $x_S = x_{SB}$. Equation 3.1-23 then yields

$$b = \frac{a_B - a_A}{x_{SB} - x_{SA}} \quad . \tag{3.1-27}$$

Now we can compute $x_N$ either from

$$x_N = x_{SA} - \frac{a_A}{b} \quad \text{or} \quad x_N = x_{SB} - \frac{a_B}{b} \quad . \tag{3.1-28}$$

Script `neutral.m` contains the commands to compute the position of the neutral point. First, the data required to define the manoeuvres are specified. We want to compute the elevator deflections for flight velocities of $25$ m/s and $50$ m/s. The calculations are carried out for two different positions of the centre of mass, namely $x_S = 1.7$ m and $x_S = 1.9$ m.

Next, function **`glider_aero`** returns the definition of the model.

```
# Example: Trim analysis of a standard class glider
#          Determine the position of the neutral point
#
# ------------------------------------------------------------

  fid = fopen("neutral.res", "wt");

# Data (kg, m, s)

  v   = [25, 50];    % Flight velocities
  rho = 1.21;        % Mass density of the air
  g   = 9.81;        % Gravity acceleration
  xs  = [1.7, 1.9];  % x-coordinate of center of mass

# Model definition

  model = glider_aero();
```

Subsequently the two configurations are defined. The manoeuvres are two straight level flights at different flight velocities.

```
# Configurations

  qdyn     = 0.5 * rho * v.^2;
  cname{1} = sprintf("v = %2.0f m/s", v(1));
  cname{2} = sprintf("v = %2.0f m/s", v(2));

  model.config = struct("name", cname, "qdyn", num2cell(qdyn),
                        "ay", 0, "az", g,
                        "pacce", 0, "yacce", 0, "racce", 0,
                        "pitch", 0, "yaw", 0, "roll", 0);
```

The trim parameters are now determined in a loop over the two positions of the centre of mass and stored in structure array **`tp`**. Actually, we only need the elevator angles.

```
# Trim analysis for two different positions of center of mass

  for n = 1 : 2

      fprintf(fid, "xS = %4.1f m\n\n", xs(n));

      model.mass.cm(1) = xs(n);

      glider = mfs_new(fid, model);
      glider = mfs_trim(glider);
      mfs_print(fid, glider, "trim", "params");

      tp(n) = mfs_getresp(glider, "trim", "params");

      fprintf(fid, "\n");
      for k = 1 : 8
          fprintf(fid, "==========");
      endfor
      fprintf(fid, "\n");

  endfor
```

Finally, we use Equation 3.1-26 to calculate $a$, Equation 3.1-27 to calculate $b$ and Equation 3.1-28 to compute $x_N$. Variable **elev** is a matrix whose first column contains the elevator angles of the first position of the centre of mass and the second column whose of the second position. The two rows correspond to the flight velocities. Variable **a** contains the proportionality constants $a_A$ and $a_B$. The variable **xN** is therefore also an array containing two values of the position of the neutral point, computed from the first and second of the Equations 3.1-28. The values should be identical.

```
# Determination of neutral point

  elev = reshape([tp.elevator], 2, 2);
  qinv = 1 ./ qdyn';

  a  = (elev(2, :) - elev(1, :)) ./ (qinv(2) - qinv(1));
  b  = (a(2) - a(1)) / (xs(2) - xs(1));
  xN = xs - a / b;

  fprintf(fid, "\na = [%7.3f, %7.3f] Pa\n", a);
  fprintf(fid, "b = %7.2f Pa/m\n", b);
  fprintf(fid,
     "\nPosition of neutral point: xN = %7.3f %7.3f m\n", xN);

  fclose(fid);
```

The output file contains the following information:

```
xS =  1.7 m

Mefisto 2.8: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm
```

```
  Number of nodes  =  5804,  Number of panels =  5296
  Number of lifting surfaces =           12
  Number of control surfaces =            3
  Number of configurations   =            2
  Reference chord length     =  0.00000e+00


--------------------------------------------------------------------------------

Component "glider"

Results of rigid trim analysis (Angles are in degrees)

  Configuration   1: v = 25 m/s
  Configuration   2: v = 50 m/s

  Configuration                 1             2

  qdyn                =   3.7812e+02    1.5125e+03
  ax                  =   0.0000e+00    0.0000e+00
  ay                  =   0.0000e+00    0.0000e+00
  az                  =   9.8100e+00    9.8100e+00
  racce               =   0.0000e+00    0.0000e+00
  pacce               =   0.0000e+00    0.0000e+00
  yacce               =   0.0000e+00    0.0000e+00
  alpha               =   4.3538e+00   -1.5820e+00
  beta                =  -8.3908e-15   -4.5107e-15
  pitch               =   0.0000e+00    0.0000e+00
  yaw                 =   0.0000e+00    0.0000e+00
  roll                =   0.0000e+00    0.0000e+00
  aileron             =  -8.2964e-15   -6.5518e-15
  elevator            =   5.8436e+00   -4.0367e+00
  rudder              =   1.0803e-14    1.2083e-14

================================================================================
xS =  1.9 m

Mefisto 2.8: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =  5804,  Number of panels =  5296
  Number of lifting surfaces =           12
  Number of control surfaces =            3
  Number of configurations   =            2
  Reference chord length     =  0.00000e+00


--------------------------------------------------------------------------------

Component "glider"

Results of rigid trim analysis (Angles are in degrees)

  Configuration   1: v = 25 m/s
  Configuration   2: v = 50 m/s

  Configuration                 1             2

  qdyn                =   3.7812e+02    1.5125e+03
  ax                  =   0.0000e+00    0.0000e+00
  ay                  =   0.0000e+00    0.0000e+00
  az                  =   9.8100e+00    9.8100e+00
  racce               =   0.0000e+00    0.0000e+00
  pacce               =   0.0000e+00    0.0000e+00
  yacce               =   0.0000e+00    0.0000e+00
  alpha               =   4.0222e+00   -1.6649e+00
  beta                =  -9.1057e-16   -2.5917e-15
  pitch               =   0.0000e+00    0.0000e+00
  yaw                 =   0.0000e+00    0.0000e+00
  roll                =   0.0000e+00    0.0000e+00
  aileron             =  -8.8273e-15   -6.6902e-15
  elevator            =  -3.2034e+00   -6.2985e+00
```

```
  rudder               =   5.4914e-15    1.0829e-14

================================================================================

a = [ 86.940,  27.235] Pa
b = -298.53 Pa/m

Position of neutral point: xN =   1.991   1.991 m
```

It can be seen that both values for the position of the neutral point are identical. The neutral point is at $1.991\ \mathrm{m}$ and therefore behind the centre of mass, as required for stable flight.