4.2 Flügel endlicher Länge

Lösungen

Aufgabe 1

Die Funktion verwendet die im Skript angegebenen Formeln. Durch Verwendung geeigneter Arrays können die Geschwindigkeiten alle gleichzeitig berechnet werden.

Bei der Berechnung wird benutzt, dass die beiden freien Wirbel entlang der *x*-Achse keinen Beitrag zur Geschwindigkeitskomponente v_x liefern. Für die Kreuzprodukte mit $\boldsymbol{e}_x$ gilt:

$$\boldsymbol{e}_x \times \boldsymbol{r}_{AP} = \boldsymbol{e}_x \times \left((x_P - x_A)\boldsymbol{e}_x + (y_P - y_A)\boldsymbol{e}_y + (z_P - z_A)\boldsymbol{e}_z \right) = -(z_P - z_A)\boldsymbol{e}_y + (y_P - y_a)\boldsymbol{e}_z$$

Wenn der Punkt *P* auf der Geraden durch die Punkte *A* und *B* liegt, ist die vom gebundenen Wirbel induzierte Geschwindigkeit null. Dieser Fall wird gesondert behandelt.

Wenn der Punkt *P* auf einer der beiden Halbgeraden durch *A* oder *B* liegt, ist die induzierte Geschwindigkeit des entsprechenden freien Wirbels unendlich. Dieser Fall wird nicht überprüft.

GNU Octave-Code für die Funktion **horseshoe**

```
function v = horseshoe(rA, rB, rP)

# usage: v = horseshoe(rA, rB, rP)
#
# Input  rA(3)       Coordinates of point A
#        rB(3)       Coordinates of point B
#        rP(3, :)    Coordinates of points P
#
# Output v(3, :)     Velocity vectors at points P
#
# The function computes the velocities at a set of points P
# induced by a horseshoe vortex with circulation 1:
#
#        B------------>---------------- inf
#        |                        P2
#        ^        P1
#        |
#        A------------<---------------- inf       --> x
#                        P3
#
# ------------------------------------------------------------

# Check arguments
```

```
  if (nargin != 3 || nargout != 1)
     print_usage();
  end

  [nr, nc] = size(rA);
  if (nr != 3 || nc !=1)
     error("rA must be a 3-dim. column vector\n");
  end

  [nr, nc] = size(rB);
  if (nr != 3 || nc !=1)
     error("rB must be a 3-dim. column vector\n");
  end

  [nr, nP] = size(rP);
  if (nr != 3)
     error("rP must be an array of 3-dim. column vectors\n");
  end

# Initialize result

  v = zeros(3, columns(rP));

# Basic vectors and their norms

  rAP = rP - rA; rBP = rP - rB; rAB = rB - rA;
  nAP = norm(rAP, "cols"); nBP = norm(rBP, "cols");

# Contribution of leg inf-A

  v1 = [-rAP(3, :); rAP(2, :)];
  n1 = 4 * pi * dot(v1, v1);
  s  = -(1 + rAP(1, :) ./ nAP) ./ n1;
  v(2 : 3, :) = v1 .* s;

# Contribution of leg A-B

  v1 = cross(rAP, rBP);
  n1 = 4 * pi * dot(v1, v1);
  s  = (rAB' * (rAP ./ nAP - rBP ./ nBP)) ./ n1;
  v  += v1 .* s;

# Contribution of leg B-inf

  v1 = [-rBP(3, :); rBP(2, :)];
  n1 = 4 * pi * dot(v1, v1);
  s  =  (1 + rBP(1, :) ./ nBP) ./ n1;
  v(2 : 3, :) += v1 .* s;

end
```

GNU Octave-Code für den Test

```
# Übungsblatt 4.2, Aufgabe 1: Hufeisenwirbel
#
```

```
# ------------------------------------------------------------

  file = mfilename();

  rA = [0; -1; 0];    % Punkt A
  rB = [0;  1; 0];    % Punkt B

# 20 Punkte entlang der Mittellinie

  nP = 20;
  rP = [linspace(1, 20, nP); zeros(2, nP)];

# Geschwindigkeiten

  v = pi * horseshoe(rA, rB, rP);

# Graphische Darstellung

  set(0, "defaultlinelinewidth", 2);
  set(0, "defaultaxesfontname", "Arial");
  set(0, "defaultaxesfontsize", 12);

  figure(1, "position", [100, 500, 1000, 500],
            "paperposition", [0, 0, 14, 8]);
  plot(rP(1, :), v(3, :), "color", "red");
  grid;
  xlabel("x");
  ylabel("v_z");
  print([file, ".jpg"], "-djpg");
```

Ergebnis

Abbildung 1.1 zeigt, dass der Wert der Geschwindigkeit gegen -1 strebt. Dies entspricht dem Wert der Geschwindigkeit, die an Punkten in der Mitte zwi-

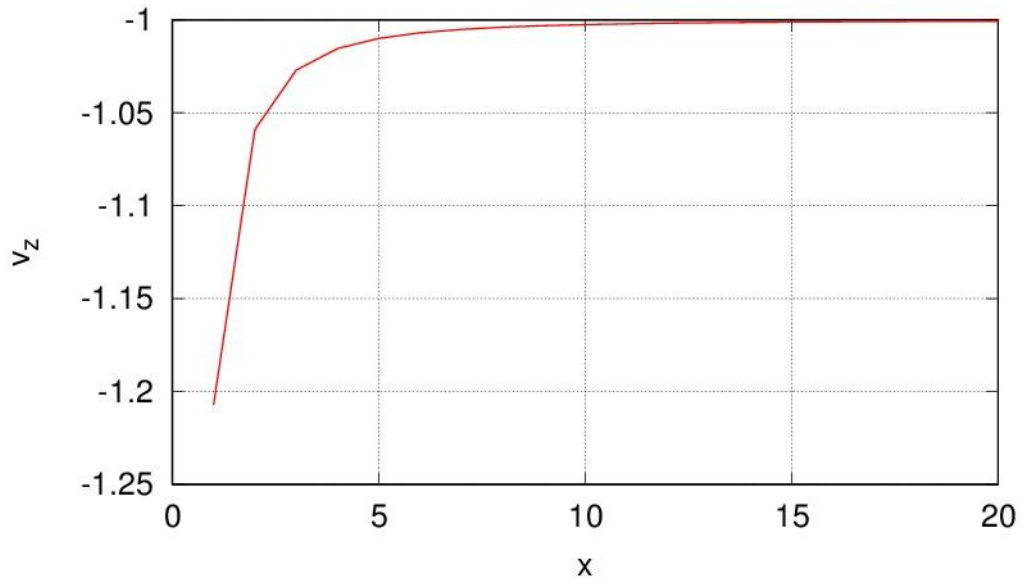

Abbildung 1.1: Geschwindigkeit v_z

schen zwei unendlich langen geraden Wirbelfäden mit dem Abstand 2 und den Wirbelstärken -π bzw. π induziert wird.

Aufgabe 2

Die Matrix des zu lösenden linearen Gleichungssystems wird in einem Loop über die Wirbel erstellt. Die Funktion **horseshoe** liefert den Beitrag eines Hufeisenwirbels zu den Geschwindigkeiten an allen Kontrollpunkten. Jede Spalte der rechten Seite entspricht einem Anstellwinkel.

GNU Octave-Code

```
function G = vlms(rA, rB, rC, dw, alpha)

# usage: G = vlms(rA, rB, rC, dw, alpha)
#
# Input  rA(3, :)  Coordinates of vortex points A
#        rB(3, :)  Coordinates of vortex points B
#        rC(3, :)  Coordinates of control points C
#        dw(:)     Slopes dz/dx
#        alpha(:)  Array with angles of attack
#
# Output G(:, :)   Vortex strengths: rows correspond to vortices,
#                  columns to angles of attack
#
# The function computes the strenghts of the horseshoe vortices of
# a wing in the xy-plane using the simplified vortex-lattice
# method. The function can process several angles of attack simul-
# taneously.
#
# ------------------------------------------------------------

# Check arguments

  if (nargin != 5 || nargout != 1)
     print_usage;
  end

  [nr, nc] = size(rA);

  if (nr != 3)
     error("rA, rB and rC must have 3 rows\n");
  end

  [nrB, ncB] = size(rB);
  if (nrB != nr || ncB != nc)
     error("Size of rB does not match size of rA\n");
  end

  [nrC, ncC] = size(rC);
  if (nrC != nr || ncC != nc)
     error("Size of rC does not match size of rA\n");
  end
```

```
  if (length(dw) != nc)
     error("Size of dw does not match size of rA\n");
  end

# Build right-hand side

  a = ones(nc, 1) * alpha;
  [nr, nx] = size(dw);
  if (nr == 1)
     rhs = dw' - a;
  else
     rhs = dw - a;
  end

# Build matrix of influence coefficients

  C = zeros(nc, nc, "double");
  for n = 1 : nc
      v       = horseshoe(rA(:, n), rB(:, n), rC);
      C(:, n) = v(3, :);
  end

# Solve for G

  G = C \ rhs;

end
```

Aufgabe 3

Aus

$$\frac{dz_S}{dx} = -4\frac{h}{c}\left(2\frac{x}{c} - 1\right)$$

folgt für die Steigung der Skelettlinie im Dreiviertelpunkt:

$$\frac{dz_S}{dx}\left(\frac{3}{4}c\right) = -4\frac{h}{c}\left(\frac{6}{4} - 1\right) = -2\frac{h}{c}$$

Der Auftriebsbeiwert wird berechnet, indem der auf den Staudruck bezogene Auftrieb durch die Flügelfläche geteilt wird.

Für die auf den Staudruck bezogene Kraft eines einzelnen gebundenen Wirbels gilt allgemein:

$$\frac{\boldsymbol{L}_n}{q_\infty} = 2\,\boldsymbol{e}_x \times \boldsymbol{r}_{ABn}\frac{\Gamma_n}{v_\infty}$$

Da der Flügel in der *xy*-Ebene liegt, gilt

$$\boldsymbol{r}_{ABn} = \Delta y_n \boldsymbol{e}_y$$

und damit

$$\frac{\boldsymbol{L_n}}{q_\infty} = 2\,\Delta y_n \frac{\Gamma_n}{v_\infty}\boldsymbol{e}_z \quad .$$

Damit gilt für den gesamten Auftrieb:

$$\frac{L_z}{q_\infty} = \sum_n \frac{L_{nz}}{q_\infty} = 2\sum_n \Delta y_n \frac{\Gamma_n}{v_\infty}$$

Da die Wirbelstärken Γ_n / v_∞ in einer Spaltenmatrix und die Wirbelabstände Δy_n in einer Zeilenmatrix gespeichert sind, lässt sich die Summe leicht als Skalarprodukt berechnen.

GNU Octave-Code

```
# Übungsblatt 4.2, Aufgabe 3:
#    Rechteckflügel mit aerodynamischer Schränkung
#    Berechnung mit dem einfachen Wirbelgitterverfahren
#
# ------------------------------------------------------------

  warning("off", "Octave:missing-glyph");

  set(0, "defaultlinelinewidth", 2);
  set(0, "defaultaxesfontname", "Arial");
  set(0, "defaultaxesfontsize", 12);

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

# Daten

  N     =     20;     % Anzahl der Hufeisenwirbel
  b     =     15;     % Spannweite in m
  c     =    1.5;     % Flügeltiefe in m
  h     =   0.02;     % Wölbung in m
  alpha = [0, 2];     % Anstellwinkel in Grad

# Diskretisierung

  theta = linspace(pi, 0, N + 1);
  y     = 0.5 * b * cos(theta);

  yA = y(1 : N); yB = y(2 : N + 1); yC = 0.5 * (yA + yB);

  xA = (0.25 * c)(ones(1, N));
  xB = xA;
  xC = (0.75 * c)(ones(1, N));

  z  = zeros(1, N);
  rA = [xA; yA; z]; rB = [xB; yB; z]; rC = [xC; yC; z];

# Steigung der Skelettlinie
```

```
  dw = - (2 * h / c)(ones(1, N));

# Graphische Kontrolle

  figure(1, "position", [100, 700, 1000, 500]);
  subplot(2, 1, 1);
  plot(rA(2, :), rA(1, :), "color", "green", "marker", "o",
       rB(2, :), rB(1, :), "color", "green", "marker", "o",
       rC(2, :), rC(1, :), "color", "red", "marker", "x",
       "linestyle", "none");
  axis([-0.5 * b, 0.5 * b, 0, 1.1 * c], "equal");
  grid;
  ylabel("x");
  subplot(2, 1, 2);
  plot(yC, dw, "color", "red");
  axis([-0.5 * b, 0.5 * b, -2.1 * h / c, 0]);
  grid;
  xlabel("y"); ylabel("dz/dx");

# Wirbelstärken

  a = pi * alpha / 180;
  G = vlms(rA, rB, rC, dw, a);

# Graphische Ausgabe

  text1 = sprintf("\\alpha = %5.2f\\circ",  alpha(1));
  text2 = sprintf("\\alpha = %5.2f\\circ",  alpha(2));

  figure(2, "position", [500, 600, 750, 500],
            "paperposition", [0, 0, 14, 7]);
  plot(yC, G(:, 1), "color", "green",
       yC, G(:, 2), "color", "red");
  legend(text1, text2, "orientation", "horizontal",
                       "location", "south");
  legend("boxoff"); legend("left");
  axis([-0.5 * b, 0.5 * b, 0, 0.3]);
  grid;
  xlabel("y"); ylabel('\Gamma / v_{/Symbol \245}');
  print([file, ".jpg"], "-djpg");

# Auftriebsbeiwert

  A     = b * c;
  dy    = diff(y);
  cL    = 2 * dy * G / A;
  dcLda = diff(cL) ./ diff(a);

# Ausgabe

  fprintf(fid, "alpha :  %5.2f°   %5.2f°\n", alpha);
  fprintf(fid, "cL    : %7.4f  %7.4f\n", cL);
  fprintf(fid, "dcL/da: %7.4f\n", dcLda);

  fclose(fid);
```

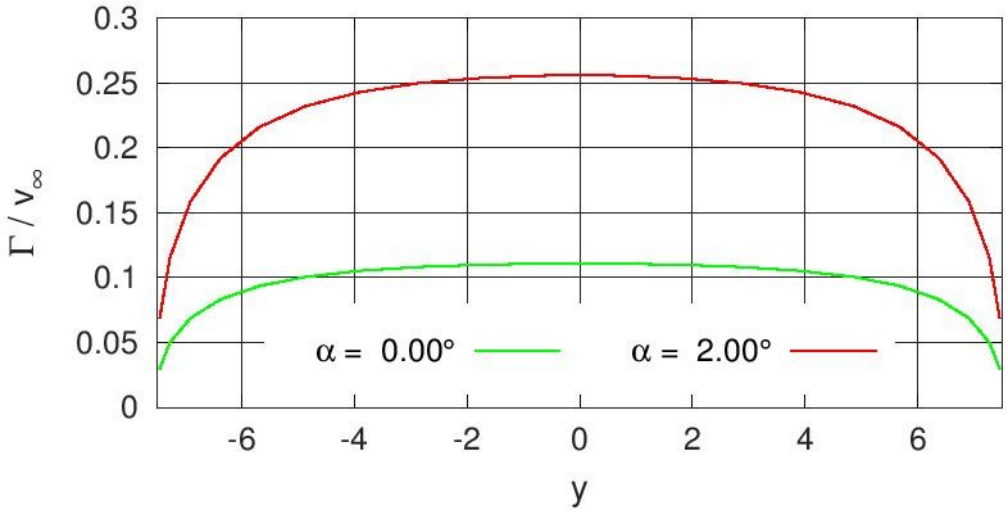

Abbildung 3.1: Zirkulationsverteilung

Ergebnisse

Die Ausgabedatei enthält die folgenden Daten:

```
alpha :   0.00°    2.00°
cL    :  0.1314   0.3034
dcL/da:  4.9283
```

Die Zirkulationsverteilung ist in Abbildung 3.1 dargestellt.

Aufgabe 4

Wegen der einfachen Geometrie kann der Tragflügel durch eine einzige Auftriebsfläche abgebildet werden.

Die Skelettlinie wird in Mefisto durch ein abschnittsweise definiertes Polynom beschrieben, wobei die *x*- und *z*-Werte jeweils auf die Flügeltiefe bezogen werden. Für die vorgegebene Skelettlinie gilt:

$$\frac{z_S}{c} = 4\frac{h}{c}\frac{x}{c}\left(1-\frac{x}{c}\right) = 4\frac{h}{c}\left[-\left(\frac{x}{c}\right)^2+\left(\frac{x}{c}\right)+0\right], \quad 0 \le \frac{x}{c} \le 1$$

Sie wird also durch ein einziges quadratisches Polynom beschrieben, das für das Intervall $[0, 1]$ definiert ist.

GNU Octave-Code

```
# Übungsblatt 4.2, Aufgabe 4:
#    Rechteckflügel mit aerodynamischer Schränkung
#    Berechnung mit dem allgemeinen Wirbelgitterverfahren
#
# ------------------------------------------------------------
```

```
  warning("off", "Octave:missing-glyph");
  colors = [1, 0, 0; 0, 1, 0; 0, 0, 1];   % rot / gruen / blau

  set(0, "defaultlinelinewidth", 2);
  set(0, "defaultaxesfontname", "Arial");
  set(0, "defaultaxesfontsize", 12);
  set(0, "defaultaxescolororder", colors);

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

# Daten

  nx    =     15;   % Anzahl Panel in x-Richtung
  ny    =     60;   % Anzahl Panel in y-Richtung
  b     =     15;   % Spannweite in m
  c     =    1.5;   % Flügeltiefe in m
  h     =   0.02;   % Wölbung in m
  alpha = [0, 2];   % Anstellwinkel in Grad

# Skelettlinie

  breaks = [0, 1]; coeffs = 4 * (h / c)* [- 1, 1, 0];
  zs     = mkpp(breaks, coeffs);

# Modelltyp

  model.type    = "aero";
  model.subtype = "vlm";

# Punkte an der Flügelnase

  points(1).id = 1; points(1).coor = [0, -0.5 * b, 0];
  points(2).id = 2; points(2).coor = [0,  0.5 * b, 0];

  model.points = points;

# Auftriebsfläche

  ls(1).id = 1; ls(1).points = [1, 2]; ls(1).chord = c;
  ls(1).camber = zs; ls(1).nx = nx; ls(1).ny = ny;
  ls(1).typey = "cos";

  model.ls = ls;

# Konfigurationen

  nalpha = length(alpha);
  for n = 1 : nalpha
      config(n).name = ...
          sprintf("Config. %2d: alpha = %4.1f",
                  n, alpha(n));
      config(n).alpha = alpha(n);
  end
```

```
  model.config = config;

# Komponente erzeugen und exportieren

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

# Ergebnisse berechnen und exportieren

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

# Druck in Flügelschnitten

  ncols = 3;
  colno = 1 + [3, 4, 5] * ny / 6;
  p     = zeros(nx, ncols, nalpha);

  for n = 1 : ncols
      [x, p(:, n, :), y] = ...
          mfs_xydata(wing, "statresp", "pressure", 1, colno(n));
      names{n} = sprintf("y = %5.2f m", y);
  end

  figure(1, "position", [100, 500, 750, 500],
            "paperposition", [0, 0, 15, 10]);

  for n = 1 : nalpha
      subplot(1, nalpha, n);
      plot(x / c, squeeze(p(:, :, n)));
      title(sprintf("\\alpha = %1.0f\\circ", alpha(n))
            "fontweight", "normal");
      if (n == nalpha)
         legend(names); legend("boxoff"); legend("left");
      end
      grid;
      hax = gca();
      axpos = get(hax, "position");
      axpos(2) += 0.05; axpos(4) -= 0.1;
      set(hax, "position", axpos);
      xlim([0, 1]);
      xlabel('x/c');
      if (n == 1) ylabel('\Delta c_P'); end
  end

  print([file, ".jpg"], "-djpg");

# Aerodynamische Beiwerte

  [F, M] = mfs_getresp(wing, "statresp", "aeload",
                       [0.25 * c, 0, 0]);
  A      = mfs_getresp(wing, "mesh", "area");
```

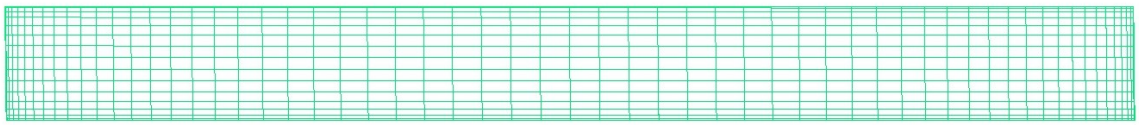

Abbildung 4.1: Vernetzung

```
cL  = F(3, :) / A;
cM  = M(2, :) / (c * A);
cLa = 180 * diff(cL) / (pi * diff(alpha));

fprintf(fid, "Aerodynamische Beiwerte:\n\n");
fprintf(fid, "alpha :  %5.2f°   %5.2f°\n", alpha);
fprintf(fid, "cL    : %7.4f  %7.4f\n", cL);
fprintf(fid, "cM    : %7.4f  %7.4f\n", cM);
fprintf(fid, "dcL/da: %7.4f\n", cLa);

fclose(fid);
```

Ergebnisse

Abbildung 4.1 zeigt die Vernetzung. An der Flügelvorder- und Hinterkante sowie an den beiden Flügelenden wird eine feinere Vernetzung verwendet als im Flügelinneren.

Der Druckbeiwert über der gesamten Flügelfläche ist in den Abbildungen 4.2 und 4.3 dargestellt. Abbildung 4.4 zeigt den Verlauf des Druckbeiwerts in ausgewählten Flügelschnitten. Im inneren Bereich des Flügels ändert sich der Druckbeiwert nur wenig in Spannweitenrichtung. Zu den Flügelenden fällt er stark ab.

Die Ausgabedatei hat folgenden Inhalt:

```
Mefisto 2.1: Building new component from input "model"
```

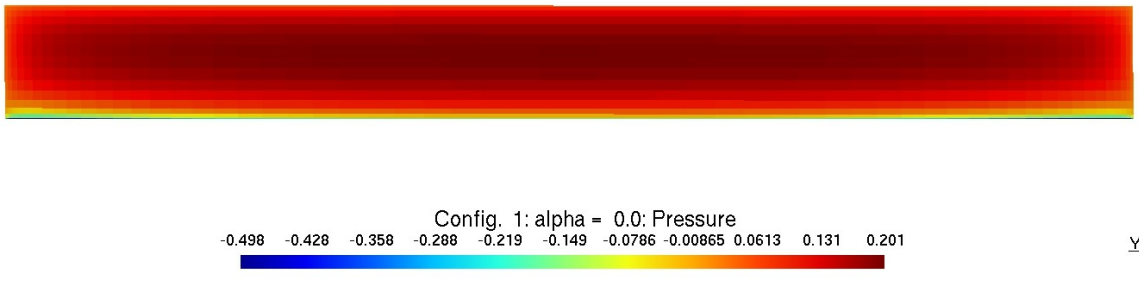

Abbildung 4.2: Druckbeiwert für α = 0°

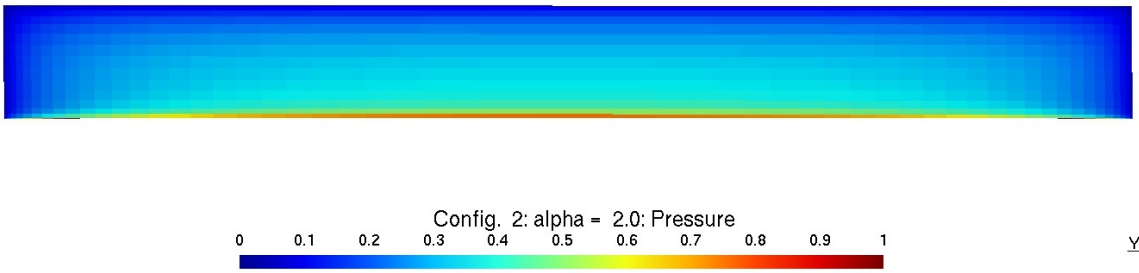

Abbildung 4.3: Druckbeiwert für α = 5°

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

  Number of lifting surfaces =     1
  Number of nodes  =   976,  Number of panels =   900
  Number of configurations   =     2

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

Component "wing"

Panel pressure

  Config.        1              2

      1:  -2.3097e-01    1.2259e-01
      2:  -5.1379e-02    5.1224e-02
      3:  -9.0738e-03    4.0297e-02
      4:   8.4433e-03    3.8475e-02
      5:   1.8082e-02    3.8630e-02
```

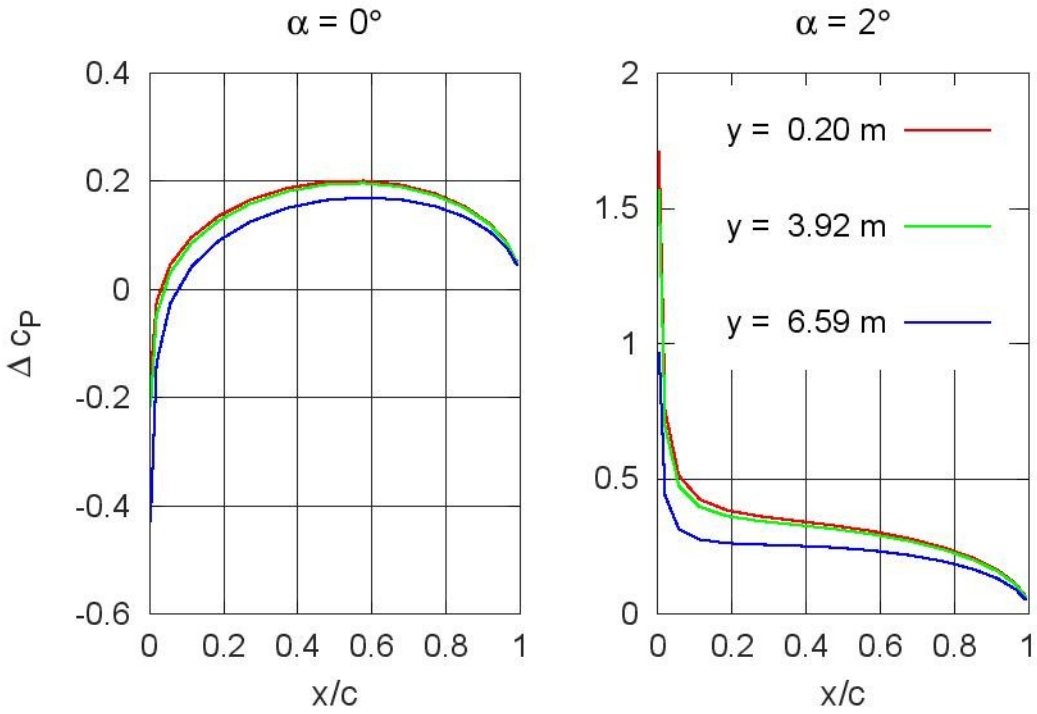

Abbildung 4.4: Druckbeiwert in ausgewählten Flügelschnitten

```
      6:   2.4049e-02    3.9097e-02
      7:   2.7809e-02    3.9309e-02
      8:   2.9984e-02    3.9017e-02

      …         …             …

    894:   3.0870e-02    3.8083e-02
    895:   3.0613e-02    3.6416e-02
    896:   2.9275e-02    3.3938e-02
    897:   2.6865e-02    3.0566e-02
    898:   2.3335e-02    2.6190e-02
    899:   1.8583e-02    2.0653e-02
    900:   1.3017e-02    1.4375e-02

Aerodynamische Beiwerte:

alpha :    0.00°     2.00°
cL    :   0.1331    0.3032
cM    :  -0.0399   -0.0389
dcL/da:   4.8752
```

Der Auftriebsbeiwert und seine Ableitung stimmen gut mit den Ergebnissen aus Aufgabe 3 überein.

Aufgabe 5

Der Flügel besteht aus je einer Auftriebsfläche für den Innenflügel, den Außenflügel und das Querruder, siehe Abbildung 5.1. Für den Innenflügel wird eine gleichmäßige Unterteilung in Spannweitenrichtung und eine symmetrische cos-Verfeinerung in Flügeltiefenrichtung verwendet. Für den Außenflügel wird eine nach außen abnehmende cos-Verfeinerung verwendet. In Flügeltiefenrichtung wird eine gleichmäßige Unterteilung verwendet, die so an-

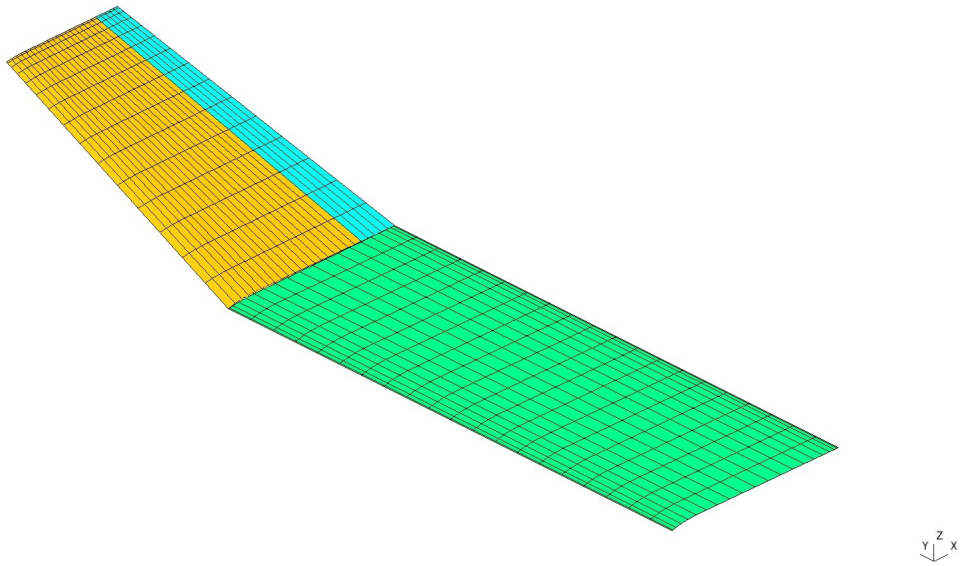

Abbildung 5.1: Auftriebsflächen

gepasst wird, dass die Wirbeltrapeze für das Querruder dieselbe Größe haben wie für den Flügel.

Die Skelettlinien werden mithilfe der Funktion **`mfs_airfoil`** definiert, die die 4- und die 5-ziffrigen NACA-Profile unterstützt.

GNU Octave-Code

```
# Übungsblatt 4.2, Aufgabe 5: Knickflügel
#
# ------------------------------------------------------------

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

# Daten in mm

  ya = 4000;  % y-Position des Querruders
  yt = 6000;  % y-Position der Flügelspitze
  zt = 1000;  % z-Position der Flügelspitze

  cr = 1500;  % Flügeltiefe an der Flügelwurzel
  ct = 1000;  % Flügeltiefe an der Flügelspitze
  k  =  0.2;  % Klappentiefenverhätlnis des Querruders

  nxi =  20;  % Panels in x-Richtung für den Innenflügel
  nxo =  20;  % Panels in x-Richtung für den Außenflügel
  nxa =   5;  % Panels in x-Richtung für das Querruder

  nyi =  20;  % Panels in y-Richtung für den Innenflügel
  nyo =  15;  % Panels in y-Richtung für den Außenflügel

  alpha =   [0, 2];   % Anstellwinkel in Grad
  eta   =   [0, 5];   % Querruderwinkel in Grad
  v     =     40E3;   % Anströmgeschwindigkeit in mm/s
  rho   = 1.21E-12;   % Luftdichte in t/mm^3

# Profildaten

  ppi = mfs_airfoil("NACA", 5, 30);
  ppo = mfs_airfoil("NACA", 5, 30, 0, 1 - k);
  ppa = mfs_airfoil("NACA", 5, 30, 1 - k, 1);

# Definitin des Modells

  model.type    = "aero";
  model.subtype =  "vlm";
  model.symy    =      0;

# Punkte an der Flügelvorderkante

  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];
```

```
# Punkte an der Querrudervorderkante

  points(4).id = 12; points(4).coor = [(1 - k) * cr, ya,  0];
  points(5).id = 13; points(5).coor = [(1 - k) * ct, yt, zt];

# Auftriebsfläche für den Innenflügel

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

# Auftriebsfläche für den Außenflügel

  ls(2).id = 2; ls(2).points = [2, 3];
  ls(2).chord = (1 - k) * [cr, ct]; ls(2).camber = ppo;
  ls(2).nx = nxo; ls(2).typex = "linear";
  ls(2).ny = nyo; ls(2).typey = "cos>";

# Auftriebsfläche für das Querruder

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

# Querruder als Kontrollfläche

  controls(1).name = "aileron";
  controls(1).ls   = 3;

# Konfigurationen

  qdyn = 0.5 * rho * v^2; n = 0;

  for l = 1 : 2
      for m = 1 : 2
          config(++n).name = ...
            sprintf("Conf. %1.0d: alpha = %5.2f, eta = %5.2f",
                    n, alpha(l), eta(m));
          config(n).qdyn    = qdyn;
          config(n).alpha   = alpha(l);
          config(n).aileron = eta(m);
      end
  end

# Definitionen zum Modell hinzufügen

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

# Komponente erzeugen und exportieren

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

```
# Ergebnisse berechnen und exportieren

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

# Aerodynamische Beiwerte

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

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

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

  dcL = 180 * (cL(3) - cL(1)) / (pi * diff(alpha));

  fprintf(fid, "\nWing area = %5.2f m^2, dcL/da = %7.4f,",
               1e-6 * A, dcL);
  fprintf(fid, " qdyn = %10.5e MPa\n", qdyn);

  for n = 1 : 4
      fprintf(fid, "\n%s:\n", config(n).name);
      fprintf(fid, "  cL = %7.4f, cM = %7.4f\n", cL(n), cM(n));
  end

  fclose(fid);
```

Ergebnisse

Die Ausgabedatei hat folgenden Inhalt:

```
Mefisto 2.2: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =   873,  Number of panels =   775
  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


Wing area =  8.50 m^2, dcL/da =  4.7535, qdyn = 9.68000e-04 MPa

Conf. 1: alpha =  0.00, eta =  0.00:
  cL =  0.0943, cM = -0.0342

Conf. 2: alpha =  0.00, eta =  5.00:
  cL =  0.1579, cM = -0.0615

Conf. 3: alpha =  2.00, eta =  0.00:
  cL =  0.2603, cM = -0.0734

Conf. 4: alpha =  2.00, eta =  5.00:
  cL =  0.3239, cM = -0.1008
```

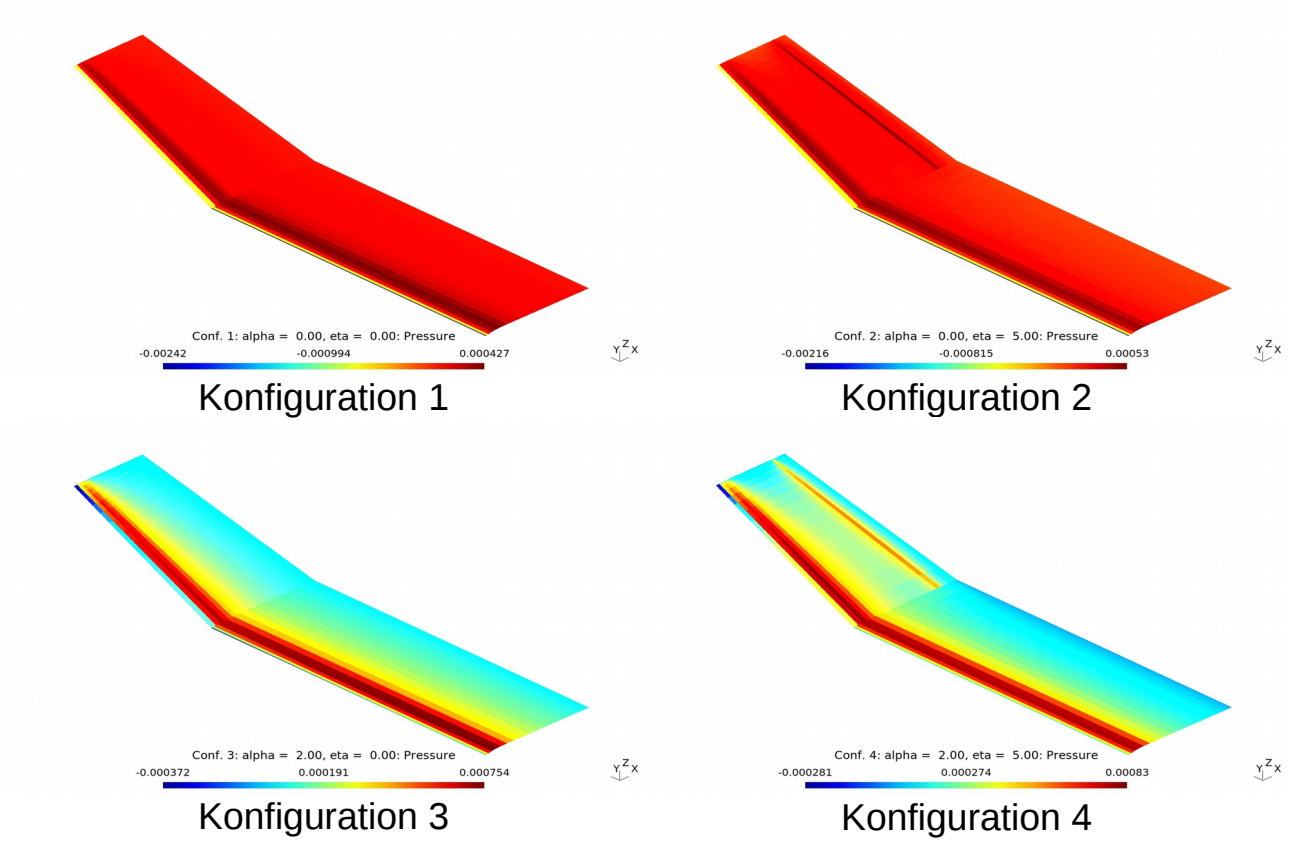

Abbildung 5.2: Druckverteilungen [MPa]

Die Druckverteilungen für die vier Konfigurationen sind in Abbildung 5.2 dargestellt.

Aufgabe 6

Da die Auftriebsflächen gerade Vorderkanten haben, muss der Flügel in *y*-Richtung durch eine Reihe von Auftriebsflächen approximiert werden, die in *y*-Richtung jeweils nur 1 Panel aufweisen.

Für die Flügeltiefe in Abhängigkeit von *y* gilt

$$c(y)=c_0\sqrt{1-\left(2\frac{y}{b}\right)^2},$$

wobei c_0 die Flügeltiefe an der Flügelwurzel ist. Für die Koordinaten der Punkte an der Flügelnase und der Hinterkante folgt:

$$x_{LE}(y)=-c(y)/4,\quad x_{TE}(y)=3c(y)/4$$

Es genügt die Modellierung des halben Flügels.

Die Skelettlinie des Profils wird durch einen kubischen Spline approximiert. Dafür wird die Funktion **`mfs_airfoil`** verwendet.

Der Anstellwinkel bei vorgegebener Geschwindigkeit berechnet sich aus dem Kräftegleichgewicht

$$m\,g = L = \frac{1}{2}\left(c_{L0} + \frac{dc_L}{d\,\alpha}\,\alpha\right)\rho\,v_\infty^2\,A$$

zu

$$\alpha = \frac{1}{dc_L/d\,\alpha}\left(\frac{2\,m\,g}{\rho\,v_\infty^2\,A} - c_{L0}\right).$$

GNU Octave-Skript

```
# Übungsblatt 4.2, Aufgabe 6: Lo 100
#
# ----------------------------------------------------------------

  graphics_toolkit("qt");
  set(0, "defaultlinelinewidth", 2);

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

# Daten (kg, m, s)

  b     =    10.00;  % Spannweite
  c0    =     1.39;  % Flügeltiefe an der Flügelwurzel
  alpha =   [0, 5];  % Anstellwinkel in Grad
  v     =       25;  % Geschwindigkeit
  rho   =     1.21;  % Luftdichte
  m     =      230;  % Masse
  g     =     9.81;  % Erdbeschleunigung

# Diskretisierung

  nx    =       10;  % Anzahl der Panels in x-Richtung
  ny    =       20;  % Anzahl der Panels in y-Richtung
  ns    =       10;  % Anzahl Spline-Segmente für die Skelettlinie

# Geometrie des Flügels

  theta = linspace(0.5 * pi, 0, ny + 1);
  y     = 0.5 * b * cos(theta);
  c     = c0 * sqrt(1 - (2 * y / b).^2);

  xL = -0.25 * c;    % Nase
  xT =  0.75 * c;    % Endkante

  figure(1, "position", [100, 500, 800, 400]);
  plot(y, xL, y, xT);
  grid;
  xlim([0, 0.5 * b]);
  xlabel("y [m]"); ylabel("x [m]");
```

```
# Profil

  data = dlmread("clarky-il.csv", ",", "A134:B194");
  data = data / data(end, 1);
  pp   = mfs_airfoil("fit", "camber", data, ns);

  xp = linspace(0, 1, 20);
  figure(2, "position", [200, 400, 800, 400]);
  plot(data(:, 1), data(:, 2), "marker", "o", "color", "green",
                               "linestyle", "none",
       xp, ppval(pp, xp), "color", "red");
  grid;
  xlim([0, 1]);
  xlabel("x/c"); ylabel("z/c");

# Modell-Definition
# -----------------

  model.type    = "aero";
  model.subtype = "vlm";
  model.symy    = 0;

# Punkte auf der Flügelnase

  for n = 1 : ny + 1
      points(n).id   = n; points(n).coor = [xL(n), y(n), 0];
  end

  model.points = points;

# Auftriebsflächen

  for n = 1 : ny
      ls(n).id     = n;
      ls(n).points = [n, n + 1];
      ls(n).chord  = [c(n), c(n + 1)];
      ls(n).nx     = nx; ls(n).ny = 1;
      ls(n).camber = pp;
  end

  model.ls = ls;

# Konfigurationen

  for k = 1 : length(alpha)
      config(k).name = ...
          sprintf("Config. %2d: alpha = %4.1f°", k, alpha(k));
      config(k).alpha = alpha(k);
  end

  model.config = config;

# Rechnung
# --------
```

```
# Komponente erzeugen und exportieren

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

# Rechnen

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

# Beiwerte

  A      = 2 * mfs_getresp(wing, "mesh", "area");
  [F, M] = mfs_getresp(wing, "statresp", "aeload");

  cL = 2 * F(3, :) / A;
  dcLda = 180 * diff(cL) / (pi * diff(alpha));

  cM = 2 * M(2, :) / (c0 * A);

# Theoretischer Auftriebsanstieg

  S    = b^2 / A;
  dcLt = 2 * pi * S / (S + 2);

# Anstellwinkel bei vorgegebener Geschwindigkeit

  av = 2 * m * g / (rho * v^2 * A) - cL(1);
  av = (av / dcLda) * 180 / pi;

  fprintf(fid, "\nAuswertung:\n\n");
  fprintf(fid, "Flügelfläche: %5.2f\n", A);
  fprintf(fid, "\nalpha :  %5.2f°   %5.2f°\n", alpha);
  fprintf(fid, "cL    : %7.4f  %7.4f\n", cL);
  fprintf(fid, "cM    : %7.4f  %7.4f\n", cM);
  fprintf(fid, "\ndcL/da: %7.4f  (theor. %7.4f)\n", dcLda, dcLt);
  fprintf(fid, "\nalpha bei %5.2f m/s: %5.2f°\n", v, av);

  fclose(fid);
```

Ergebnisse

Die Ausgabedatei hat folgenden Inhalt:

```
Mefisto 2.4: Building new component from input "model"

  Model Type = aero, Model Subtype = vlm

  Number of nodes  =   440,  Number of panels =   200
  Number of lifting surfaces =          20
  Number of configurations   =           2
  Reference chord length     =  0.00000e+00
  Symmetry plane: y          =  0.00000e+00
```

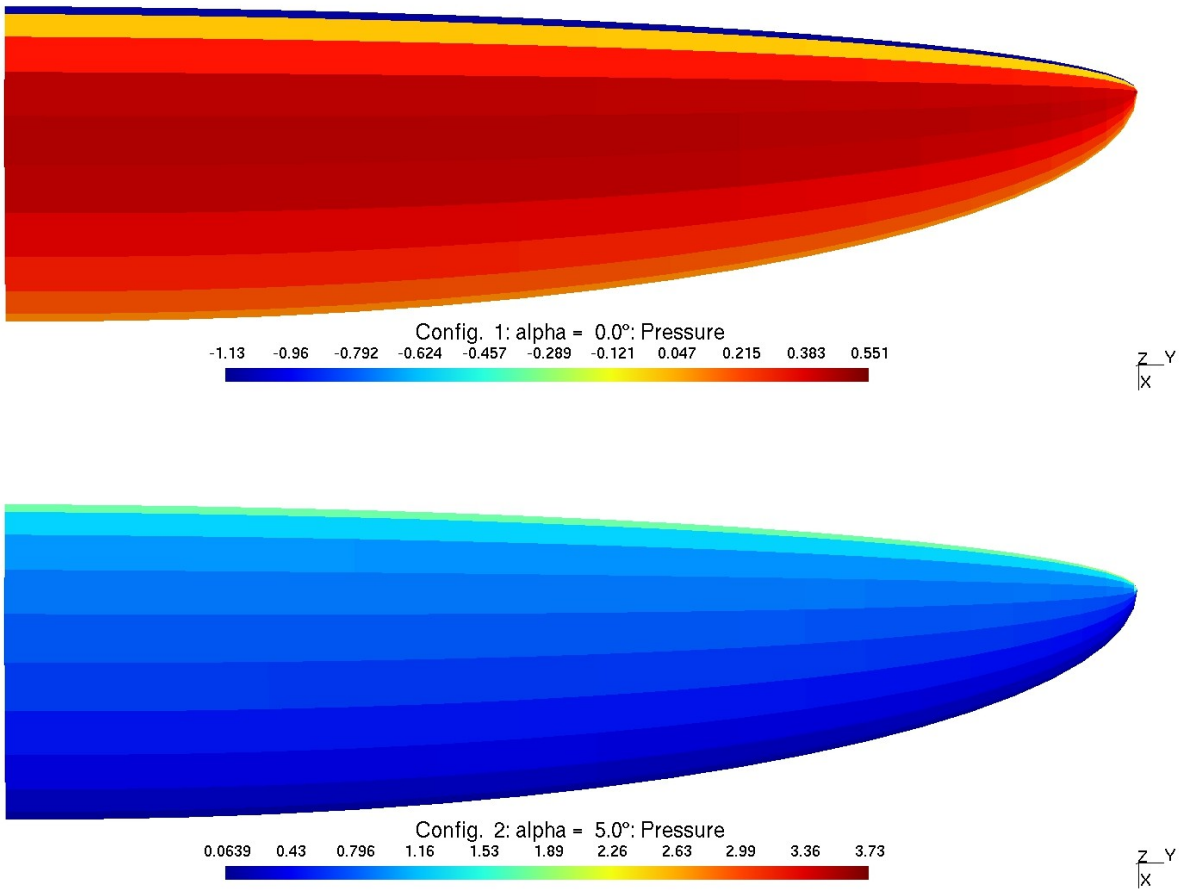

Abbildung 6.1: Druckbeiwert

```
Auswertung:

Flügelfläche: 10.91

alpha :   0.00°    5.00°
cL    :  0.2995   0.7361
cM    : -0.0686  -0.0668

dcL/da:  5.0021  (theor.  5.1581)

alpha bei 25.00 m/s:  2.84°
```

Der Druckbeiwert ist in Abbildung 6.1 dargestellt.