5.2 Modalanalyse

Lösungen

Aufgabe 1

Das folgende GNU Octave-Skript berechnet die Eigenschwingungen unter Verwendung einer konsistenten Massenmatrix. Für eine Berechnung mit einer konzentrierten Massenmatrix muss der Wert der Variablen **`mtype`** geändert werden.

```
# Übungsblatt 5.2, Aufgabe 1: Balkenschwingungen
#
# ------------------------------------------------------------------

  set(0, "defaultaxesfontsize", 12);

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

% mtype = "lumped";       % konzentrierte Massenmatrix
  mtype = "consistent";   % konsistente Massenmatrix

# Daten (N, mm):
# --------------

  L =      1000;          % Länge
  A =       500;          % Querschnittsfläche
  I =     10400;          % Flächenträgheitsmoment
  E =    210000;          % Elastizitätsmodul
  ny =      0.3;          % Querkontraktionszahl
  rho = 7.85E-9;          % Massendichte

  nofmod = 8;             % Anzahl der Eigenschwingungen

  nel = [ 10, 15, 20];    % Anzahl der Elemente

# Analytische Lösung
# ------------------

  f1 = pi * sqrt(E * I / (rho * A)) /(2 * L^2);

  fprintf(fid, "Analytical frequencies of ");
  fprintf(fid, "bending modes:\n\n");
  for n = 1 : nofmod
      fprintf(fid, "  Mode %2d: f = %8.3f Hz\n", n, n^2 * f1);
  end
  fprintf(fid, "\n");

# Numerische Berechnung mit Mefisto
```

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

  fprintf(fid, "Numerical solution with ");
  fprintf(fid, "%s mass matrix\n\n", mtype);

# Geometrieparameter

  geom.A = A;
  geom.I = I;

# Materialkennwerte

  mat.type = "iso";
  mat.E    = E;
  mat.ny   = ny;
  mat.rho  = rho;

# Schleife über die Vernetzungen

  for nofel = nel

      clear model; clear cmp;
      clear nodes; clear elem; clear hinge;

      model.type    = "solid";     % Modelltyp
      model.subtype = "2d";        % 2-dim. Modell

  % Knoten und Elemente

      idl = 1 + nofel;
      nodes(1).id = 1;   nodes(1).coor = [0, 0];
      nodes(2).id = idl; nodes(2).coor = [L, 0];
      [nodes, elem] = mfs_line(nodes, 1, idl, 2 : nofel,
                               1 : nofel, "b2", geom, mat);

      model.nodes    = nodes;
      model.elements = elem;

  % Einspannung

      hinge(1).id = 1;   hinge(1).dofs = [1, 2];
      hinge(2).id = idl; hinge(2).dofs = [1, 2];

      model.constraints.prescribed = hinge;

  % Analyse

      cmp = mfs_new(fid, model);        % Komponente erzeugen
      cmp = mfs_stiff(cmp);             % Steifigkeitsmatrix
      cmp = mfs_mass(cmp, mtype);       % Massenmatrix
      cmp = mfs_freevib(cmp, nofmod);   % Eigenschwingungen

      mfs_print(fid, cmp, "modes", "freq");

  end
```

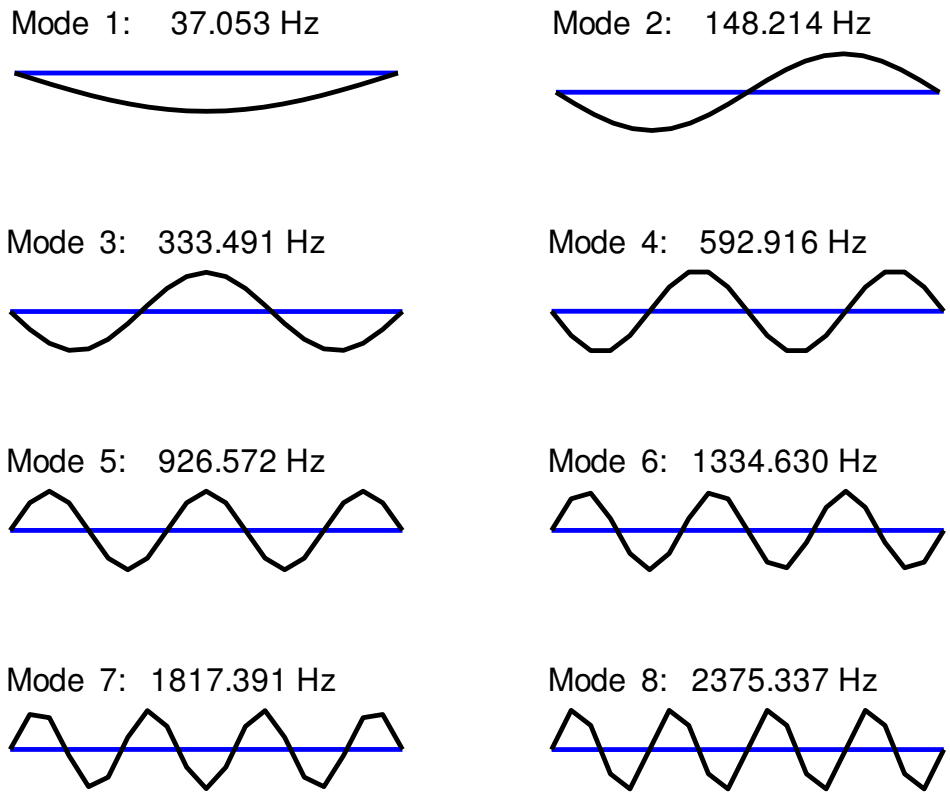


Abbildung 1.1: Eigenschwingungen

```
# Graphische Darstellung der Eigenschwingungen
# (feinste Vernetzung)

  fgh = figure(1, "position", [100, 100, 800, 600],
               "paperposition", [0, 0, 16, 13]);
  nrow = ceil(nofmod / 2);
  for m = 1 : nofmod
      subplot(nrow, 2, m);
      mfs_plot(cmp, "deform", 2, "modes", m,
               "figure", fgh);
  end
  print([file, ".svg"], "-dsvg");

  fclose(fid);
```

Abbildung 1.1 zeigt die Eigenschwingungen bei einer Unterteilung des Balkens in 20 Elemente.

Tabelle 1.1 vergleicht die Eigenfrequenzen. Bei Verwendung einer konsistenten Massenmatrix konvergieren die Eigenfrequenzen von oben, wie es aus

Nr.	analytisch	konzentrierte Massenmatrix			konsistente Massenmatrix		
		10	15	20	10	15	20
1	37,05	37,05	37,05	37,05	37,05	37,05	37,05
2	148,2	148,2	148,2	148,2	148,2	148,2	148,2
3	333,5	333,3	333,4	333,5	333,7	333,5	333,5
4	592,9	591,4	592,6	592,8	593,8	593,1	592,9
5	926,3	919,6	925,3	926,0	930,0	927,1	926,6
6	1334	1309	1331	1333	1345	1336	1335
7	1816	1738	1806	1813	1841	1821	1817
8	2371	2156	2348	2366	2426	2383	2375
	Hz	Hz	Hz	Hz	Hz	Hz	Hz

Tabelle 1.1: Vergleich der Eigenfrequenzen

der Eigenschaft des Rayleigh-Quotienten folgt. Bei Verwendung einer konzentrierten Massenmatrix erfolgt die Konvergenz von unten.

Aufgabe 2

Das GNU Octave-Skript beginnt mit der Definition der Daten.

```
# Übungsblatt 5.2, Aufgabe 2: Ebenes Fachwerk
#
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 10);

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

# Daten (N, mm):
# --------------

  a   =    3000;   % Länge eines Segments
  A   =     500;   % Stabquerschnitt
  E   =   2.1E5;   % E-Modul
  rho = 7.85E-9;   % Massendichte

  nmodes =   10;   % Anzahl der Eigenschwingungen
```

Anschließend wird das in Abbildung 2.1 dargestellte Finite-Elemente-Modell definiert.

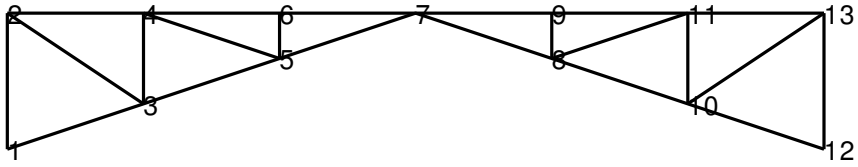


Abbildung 2.1: Finite-Elemente-Modell

```
# Modell
# ------

  model = struct("type", "solid", "subtype", "2d");

# Material

  mat = struct("type", "iso", "E", E, "rho", rho);

# Knoten

  nodes( 1).id =  1; nodes( 1).coor = [    0,         0];
  nodes( 2).id =  2; nodes( 2).coor = [    0,         a];
  nodes( 3).id =  3; nodes( 3).coor = [    a,     a / 3];
  nodes( 4).id =  4; nodes( 4).coor = [    a,         a];
  nodes( 5).id =  5; nodes( 5).coor = [2 * a, 2 * a / 3];
  nodes( 6).id =  6; nodes( 6).coor = [2 * a,         a];
  nodes( 7).id =  7; nodes( 7).coor = [3 * a,         a];
  nodes( 8).id =  8; nodes( 8).coor = [4 * a, 2 * a / 3];
  nodes( 9).id =  9; nodes( 9).coor = [4 * a,         a];
  nodes(10).id = 10; nodes(10).coor = [5 * a,     a / 3];
  nodes(11).id = 11; nodes(11).coor = [5 * a,         a];
  nodes(12).id = 12; nodes(12).coor = [6 * a,         0];
  nodes(13).id = 13; nodes(13).coor = [6 * a,         a];

  model.nodes = nodes;

# Elemente

  elem( 1).id =  1; elem( 1).nodes = [  1,  2];
  elem( 2).id =  2; elem( 2).nodes = [  3,  4];
  elem( 3).id =  3; elem( 3).nodes = [  5,  6];
  elem( 4).id =  4; elem( 4).nodes = [  8,  9];
  elem( 5).id =  5; elem( 5).nodes = [ 10, 11];
  elem( 6).id =  6; elem( 6).nodes = [ 12, 13];

  elem( 7).id =  7; elem( 7).nodes = [  1,  3];
  elem( 8).id =  8; elem( 8).nodes = [  3,  5];
  elem( 9).id =  9; elem( 9).nodes = [  5,  7];
  elem(10).id = 10; elem(10).nodes = [  7,  8];
  elem(11).id = 11; elem(11).nodes = [  8, 10];
  elem(12).id = 12; elem(12).nodes = [ 10, 12];
```

```
  elem(13).id = 13; elem(13).nodes = [  2,  4];
  elem(14).id = 14; elem(14).nodes = [  4,  6];
  elem(15).id = 15; elem(15).nodes = [  6,  7];
  elem(16).id = 16; elem(16).nodes = [  7,  9];
  elem(17).id = 17; elem(17).nodes = [  9, 11];
  elem(18).id = 18; elem(18).nodes = [ 11, 13];

  elem(19).id = 19; elem(19).nodes = [  2,  3];
  elem(20).id = 20; elem(20).nodes = [  4,  5];
  elem(21).id = 21; elem(21).nodes = [ 11,  8];
  elem(22).id = 22; elem(22).nodes = [ 13, 10];

  geom.A = A;

  for k = 1 : length(elem)
      elem(k).type = "r2";
      elem(k).geom = geom;
      elem(k).mat  = mat;
  end

  model.elements = elem;

# Einspannung

  gelenk = struct("id", {1, 12}, "dofs", 1 : 2);
  model.constraints.prescribed = gelenk;
```

Die Rechnung beginnt mit der Erzeugung einer neuen Komponente. Anschließend wird das Finite-Elemente-Modell graphisch dargestellt und nach Gmsh exportiert. Abbildung 2.1 zeigt das mit Mefisto erstellte Bild des Finite-Elemente-Modells.

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

  fachwerk = mfs_new(fid, model);
  mfs_plot(fachwerk, "nodid", 1, "paperposition", [0, 0, 16, 12]);
  print([file, ".svg"], "-dsvg");
  mfs_export([file, ".msh"], "msh", fachwerk, "mesh");
```

Dann werden die Matrizen aufgestellt, die Masseneigenschaften berechnet und ausgegeben sowie die Eigenschwingungen berechnet.

```
  fachwerk = mfs_stiff(fachwerk);
  fachwerk = mfs_mass(fachwerk);
  mfs_massproperties(fid, fachwerk);
  fachwerk = mfs_freevib(fachwerk, nmodes);
```

Zum Schluss werden die Eigenfrequenzen in die Ausgabedatei geschrieben und die Eigenschwingungen nach Gmsh exportiert.

```
  mfs_print(fid, fachwerk, "modes", "freq");
  mfs_export([file, ".pos"], "msh", fachwerk,
             "modes", "disp");
```

```
  fclose(fid);
```

Die Ausgabedatei enthält die folgenden Ergebnisse:

```
Mass properties of component "fachwerk"

  Coordinates of reference point:      0.0000,     0.0000

  Rigid body mass matrix:

     2.4535e-01   0.0000e+00  -5.2867e+02
     0.0000e+00   2.4535e-01   2.2081e+03
    -5.2867e+02   2.2081e+03   3.0177e+07

  Mass =  2.4535e-01, Moment of inertia =  3.0177e+07

  Coordinates of center of mass:  9000.0000,  2154.7840

  Moment of inertia with respect to center of mass:  9.1645e+06


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

Component "fachwerk"

Natural frequencies:

  Mode    Circ. Frequency     Frequency
  ------------------------------------
     1        177.83197       28.30284 Hz
     2        187.50291       29.84202 Hz
     3        321.17442       51.11650 Hz
     4        469.46452       74.71760 Hz
     5        590.39912       93.96494 Hz
     6        597.66996       95.12213 Hz
     7        929.17033      147.88205 Hz
     8       1334.21297      212.34659 Hz
```

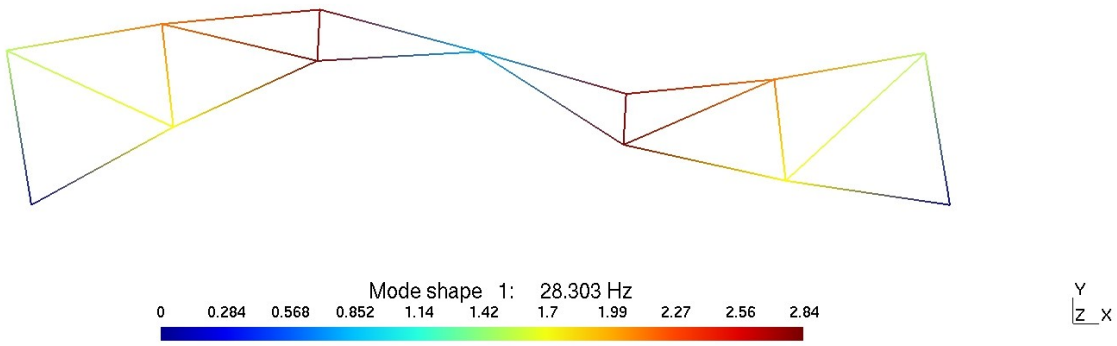


Abbildung 2.2: 1. Eigenschwingung

```
     9       1378.19022      219.34579 Hz
    10       1532.19291      243.85608 Hz
```

Die Abbildungen 2.2 bis 2.5 zeigen einige der Eigenschwingungen. Diese Bilder wurden mit Gmsh erstellt.

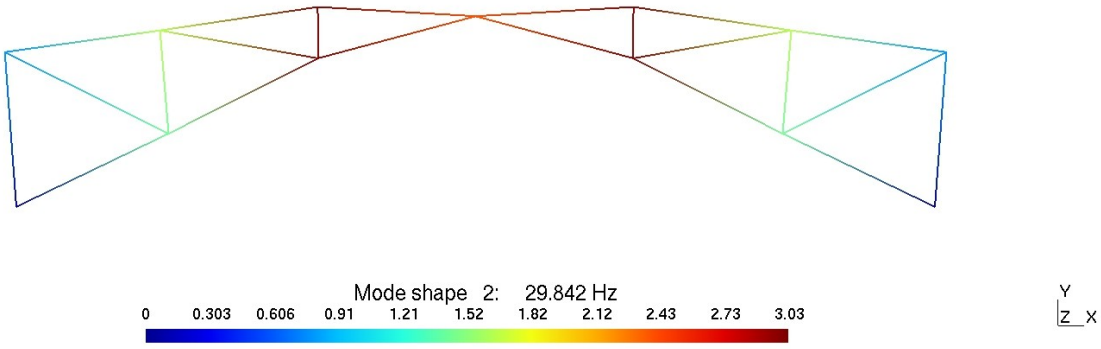


Abbildung 2.3: 2.Eigenschwingung

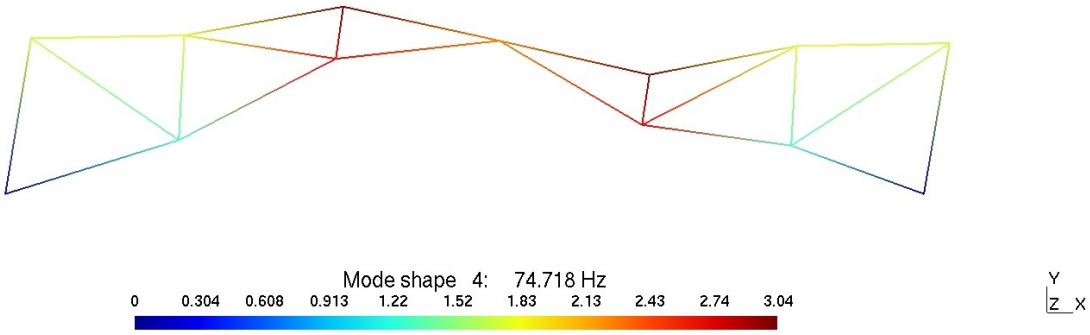


Abbildung 2.4: 4. Eigenschwingung

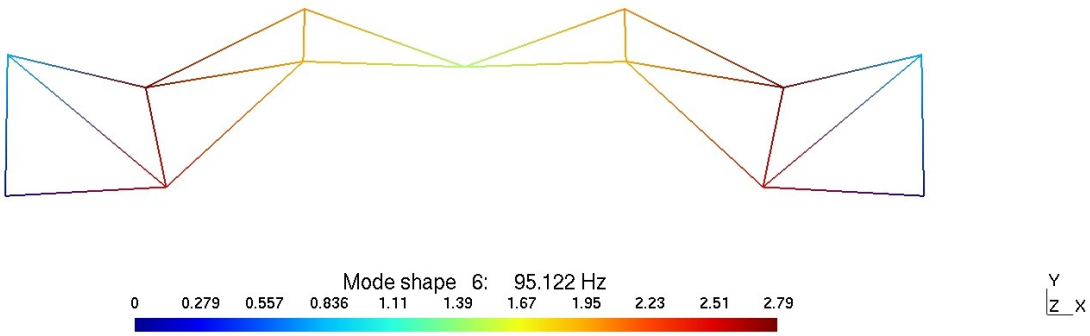


Abbildung 2.5: 6. Eigenschwingung

Aufgabe 3

Berechnung

Das GNU Octave-Skript beginnt mit der Definition der Daten.

```
# Übungsblatt 5.2, Aufgabe 3: Platte
#
# ------------------------------------------------------------

  colormap("jet");
  set(0, "defaultaxesfontsize", 12);

# Daten (N, mm)
# -------------

  a    =    2000;   % Abmessung in x-Richtung
  b    =    1500;   % Abmessung in y-Richtung
  h    =      10;   % Plattendicke
  E    =  208000;   % Elastizitätsmodul
  ny   =     0.3;   % Querkontraktionszahl
  rho  = 7.85E-9;   % Massendichte

  fmax =     200;   % maximale Frequenz

  mtype = "consistent";  % Typ der Massenmatrix

  NE = [10, 20, 30, 40]; % Elemente in x-Richtung

# Ausgabedatei

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

# Materialdaten

  mat = struct("type", "iso", "E", E, "ny", ny, "rho", rho);
```

Zuerst wird die analytische Lösung berechnet und ausgegeben. Dafür wird die Funktion **platevib** aus Aufgabe 2 von Übungsblatt 4.2 verwendet. Das Ausgabeargument **mn** enthält die Anzahl von Halbwellen in *x*- und *y*-Richtung.

```
# Analytische Lösung
# ------------------

# Plattengeometrie

  platten_geom = struct("a", a, "b", b, "h", h);

# Rechnung

  [f0, mn] = platevib(platten_geom, mat, fmax);
  nmodes  = length(f0);
```

```
# Eigenfunktionen

  function w = Wmn(xi, eta, m, n)
     w = sin(pi * m * xi) .* sin(pi * n * eta);
  end

# Ausgabe

  fprintf(fid, "Analytische Lösung\n");
  fprintf(fid, "------------------\n\n");
  fprintf(fid, "  Nr.    m    n   Frequenz \n");
  fprintf(fid, "---------------------------\n");

  for n = 1 : nmodes
      fprintf(fid, "  %3d  %3d  %3d   %5.1f Hz\n",
              n,  mn(n, :), f0(n));
  end

# Darstellung

  mnmax = max(mn);
  mmax  = 12 * mnmax(1);
  nmax  = 12 * mnmax(2);

  x = linspace(0, a, mmax);
  y = linspace(0, b, nmax);

  [x, y] = meshgrid(x, y);

  xi  = x / a; eta = y / b;

  for n = 1 : nmodes
      w = Wmn(xi, eta, mn(n, 1), mn(n, 2));
      figure(1, "position", [20, 300, 1000, 1000 * b / a],
            "paperposition", [0, 0, 15, 15 * b / a]);
      contourf(x, y, w);
      txt = sprintf("Mode %2d: f = %5.1f Hz", n, f0(n));
      title(txt, "fontsize", 12);
      set(gca(), "visible", "off");
      print(sprintf("mode_%d.svg", n), "-dsvg");;
  end
```

Anschließend erfolgt die Berechnung nach der Methode der finiten Elemente. Dabei werden in einer Schleife verschiedene Diskretisierungen verwendet. Die Ergebnisse werden in den Arrays **df** (relativer Fehler in der Frequenz) und **epw** (Elemente pro Wellenlänge) abgelegt. Dabei entsprechen die Zeilen der Arrays den Eigenschwingungen und die Spalten den Diskretisierungen.

```
# Lösung mit Finiten Elementen
# ----------------------------

  fprintf(fid, "\nLösung mit Finiten Elementen\n");
  fprintf(fid, "-----------------------------\n\n");
```

```
  nd = length(NE);

  df  = zeros(nmodes, nd);  % Frequenzfehler
  epw = zeros(nmodes, nd);  % Anzahl Elemente pro Wellenlänge

  for nn = 1 : nd

      nex = NE(nn); ney = ceil(nex * b / a);
      prefix = sprintf("%s_%d", fname, nn);

      fprintf(fid, "Diskretisierung %d: nex = %d, ney = %d ",
              nn, nex, ney);
      fprintf(fid, "mtype = %s\n\n", mtype);

      legnd{nn} = sprintf("ne = %d", nex);

#   Modelldefinition

      model = struct("type", "solid", "subtype", "3d");

      % Knoten

      npx = nex + 1; npy = ney + 1; k = 1;
      x = linspace(0, a, npx); y = linspace(0, b, npy);

      for n = 1 : npx
          for m = 1 : npy
              nodes(k) = struct("id", k++,
                                "coor", [x(n), y(m), 0]);
          end
      end

      model.nodes = nodes;

      % Elemente

      k = 1;
      enode1 = [1, 1 + npy, 2 + npy, 2];
      geom.t = h;

      for n = 1 : nex
          enodes = enode1;
          for m = 1 : ney
              elem(k) = struct("id", k++, "type", "s4",
                               "nodes", enodes++,
                               "geom", geom, "mat", mat);
          end
          enode1 += npy;
      end

      model.elements = elem;
```

Die Platte ist an den Rändern gelenkig gelagert. Zusätzlich muss bei einer ebenen Platte an mindestens einem Knotenpunkt der 6. Freiheitsgrad, der ei-

ner Drehung um die Achse senkrecht zur Plattenebene entspricht, festgehalten werden.

```
      % Lagerung

      p1 = 1; p2 = p1 + nex * npy; p3 = p2 + ney; p4 = p1 + ney;
      ps = [1 : npy : p2, p2 : p3, p4 : npy : p3, p1 : p4];
      ps = unique(ps);

      fixed = struct("id", num2cell(ps), "dofs", 1 : 3);
      fixed = [fixed, struct("id", 1, "dofs", 6)];

      model.constraints.prescribed = fixed;

      model.constraints.prescribed = fix;

#   Rechnung

      platte = mfs_new(fid, model);
      mfs_export([prefix, ".msh"], "msh", platte, "mesh");

      platte = mfs_stiff(platte);
      platte = mfs_mass(platte, mtype);
      platte = mfs_freevib(platte, nmodes);

#   Ausgabe

      mfs_print(fid, platte, "modes", "freq");
      mfs_export([prefix, ".dsp"], "msh", platte,
                 "modes", "disp");

#   Auswertung

     f = mfs_getresp(platte, "modes", "freq");
     df(:, nn)  = (f - f0') ./ f0';
     epw(:, nn) = 2 * min(nex ./ mn(:, 1), ney ./ mn(:, 2));

  end
```

Die Anzahl der Elemente pro Wellenlänge kann aus der Anzahl der Elemente in *x*- bzw. *y*-Richtung und der Anzahl der Halbwellen ermittelt werden. Gespeichert wird jeweils der kleinere Wert.

```
# Ausgabe
# -------
```

Zunächst werden die analytisch berechneten Frequenzen und die relativen Fehler für die verschiedenen Diskretisierungen in die Ausgabedatei geschrieben.

```
# Frequenzfehler

  fprintf(fid, "Fehler\n");
  fprintf(fid, "------\n\n");
```

```
  fprintf(fid, " nr     f  ");
  for m = 1 : nd
      fprintf(fid, "     %d    ", m);
  end
  fprintf(fid, "\n");
  for n = 1 : nmodes
      fprintf(fid, " %2.0d  %6.2f", n, f0(n));
      for m = 1 : nd
          fprintf(fid, "  %6.2f %%", 100 * df(n, m));
      end
      fprintf(fid, "\n");
  end
```

Anschließend wird der Fehler durch eine Ausgleichskurve der Form

$$\delta = A\, n_\lambda^s$$

approximiert. Zur Bestimmung der Koeffizienten A und s wird zunächst der Logarithmus der Gleichung gebildet:

$$\ln(\delta) = \ln(A) + s \ln(n_\lambda)$$

Zwischen den Variablen $x = \ln(n_\lambda)$ und $y = \ln(\delta)$ besteht ein linearer Zusammenhang. Wie in Kapitel 1.2.1 gezeigt gilt

$$y = \mu_y - s\,\mu_x + s\,x$$

mit

$$s = \frac{C_{xy}}{\sigma_x^2} \quad .$$

Zusätzlich wird der Korrelationskoeffizient ρ_{xy} berechnet. Sein Wert gibt an, wie gut die Ausgleichskurve zu den gegebenen Werten passt.

```
# Ausgleichskurve

  x = log(reshape(epw, nd * nmodes, 1));
  y = log(reshape(df, nd * nmodes, 1));

  Cxy = cov(x, y);      % Kovarianz
  rxy = corr(x, y);     % Korrelationskoeffizient
  vx  = var(x);         % Varianz

  s = Cxy / vx;
  C = mean(y) - s * mean(x);
  A = exp(C);

  fprintf(fid, "\nDaten der Ausgleichskurve:\n\n");
  fprintf(fid, "  s = %7.4f, C = %7.4f, A = %7.4f\n", s, C, A);
  fprintf(fid, "  Korrelationskoeffizient = %7.4f\n", rxy);

  xmin = min(x); xmax = max(x);
  xe   = exp(linspace(xmin, xmax, 10));
  ye   = A * xe.^s;
```

Zum Schluss wird der Frequenzfehler in Abhängigkeit von der Anzahl der Elemente pro Wellenlänge zusammen mit der Ausgleichskurve graphisch dargestellt. Dafür wird eine doppelt logarithmische Darstellung gewählt, da der Zusammenhang in dieser Darstellung linear sein sollte.

```
# Graphische Ausgabe

  figure(1, "position", [100, 100, 700, 700],
            "paperposition", [0, 0, 14, 10]);
  loglog(epw, df, "marker", "*", "linestyle", "none",
         xe, ye, "color", "red");
  legnd{nd + 1} = "Ausgleich";
  legend(legnd, "location", "southwest");
  grid;
  xlabel('Elemente pro Wellenlänge');
  ylabel('Relativer Frequenzfehler');
  print([fname, ".svg"], "-dsvg");

  fclose(fid);
```

Auswertung

Die Ausgabedatei enthält zunächst die Ergebnisse der analytischen Lösung.

```
Analytische Lösung
------------------

  Nr.    m    n   Frequenz
--------------------------
    1    1    1    17.0 Hz
    2    2    1    35.3 Hz
    3    1    2    49.6 Hz
    4    3    1    65.9 Hz
    5    2    2    68.0 Hz
    6    3    2    98.6 Hz
    7    1    3   104.0 Hz
    8    4    1   108.7 Hz
    9    2    3   122.3 Hz
   10    4    2   141.4 Hz
   11    3    3   152.9 Hz
   12    5    1   163.8 Hz
   13    1    4   180.1 Hz
   14    4    3   195.7 Hz
   15    5    2   196.4 Hz
   16    2    4   198.5 Hz
```

m und n geben die Anzahl der Halbwellen in x- bzw. y-Richtung an. Diese Ergebnisse stimmen mit denen aus Aufgabe 2 von Übungsblatt 4.2 überein. Die ersten vier Schwingungsformen sind in Abbildung 3.1 dargestellt.

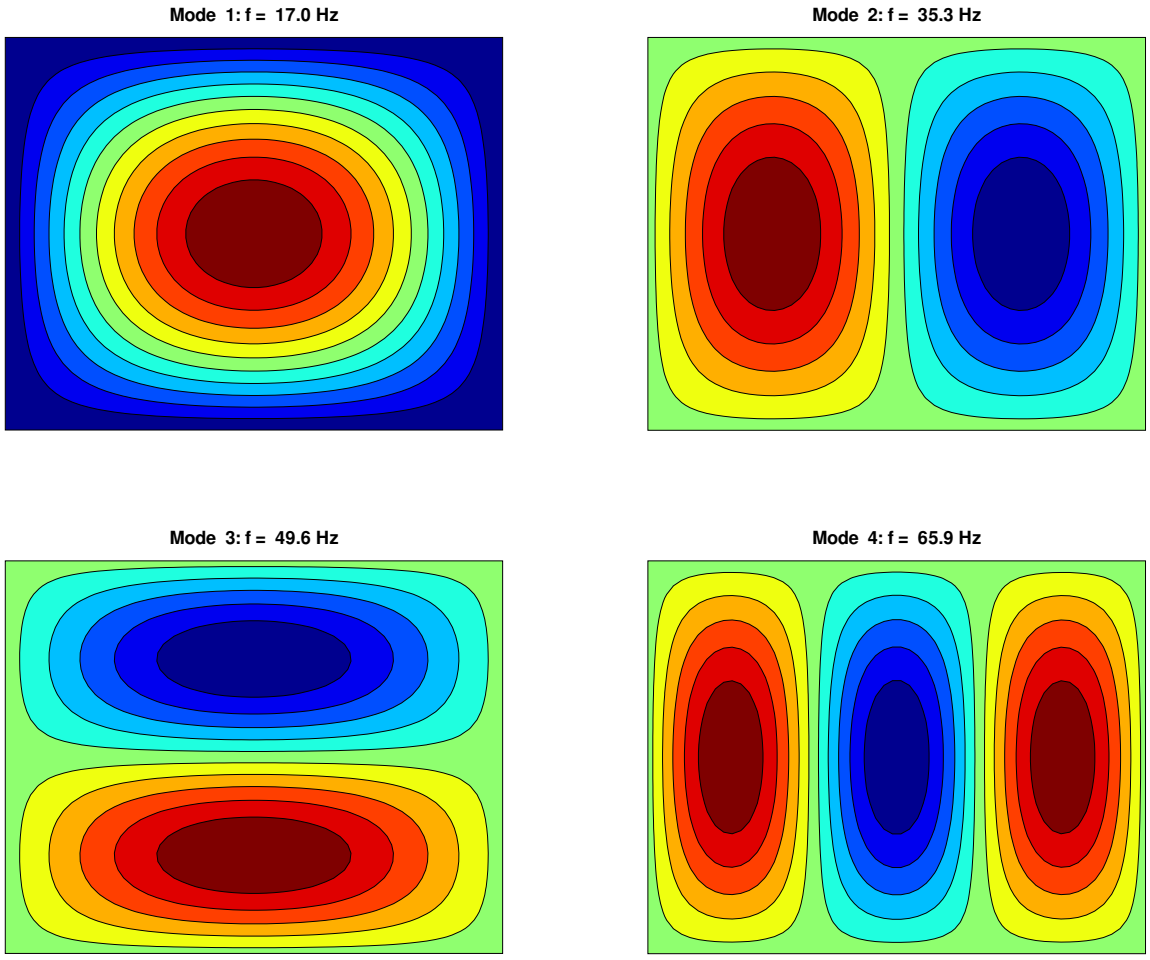


Abbildung 3.1: Schwingungen 1 bis 4

Anschließend folgen die Ausgaben für die einzelnen Analysen, die hier nicht wiedergegeben werden. Am Ende werden die relativen Fehler sowie die Koeffizienten der Ausgleichskurve ausgegeben.

Fehler
------

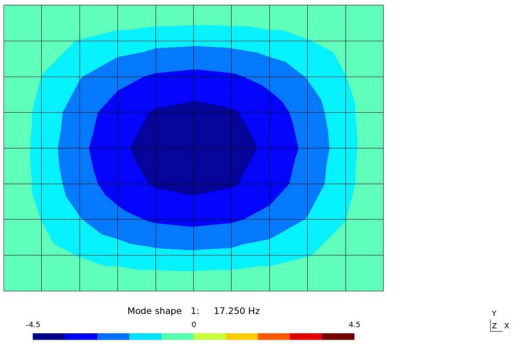
nr	f	1	2	3	4
1	16.99	1.52 %	0.40 %	0.16 %	0.09 %
2	35.34	3.79 %	0.92 %	0.39 %	0.22 %
3	49.62	7.09 %	1.91 %	0.80 %	0.46 %
4	65.93	9.63 %	2.32 %	1.01 %	0.56 %
5	67.97	6.69 %	1.66 %	0.69 %	0.40 %
6	98.55	9.15 %	2.23 %	0.95 %	0.54 %
7	103.99	18.60 %	4.74 %	1.96 %	1.14 %
8	108.75	20.57 %	4.51 %	1.95 %	1.09 %
9	122.34	16.04 %	4.07 %	1.68 %	0.98 %
10	141.37	16.93 %	3.82 %	1.64 %	0.92 %
11	152.93	15.26 %	3.81 %	1.59 %	0.92 %
12	163.80	37.08 %	7.50 %	3.21 %	1.78 %

```
 13   180.11    29.01 %     8.95 %     3.64 %     2.11 %
 14   195.75    27.49 %     4.45 %     1.88 %     1.07 %
 15   196.43    31.07 %     6.38 %     2.72 %     1.51 %
 16   198.47    34.71 %     8.06 %     3.28 %     1.90 %

Daten der Ausgleichskurve:

  s = -2.1555, C =  1.8372, A =  6.2791
  Korrelationskoeffizient = -0.9977
```

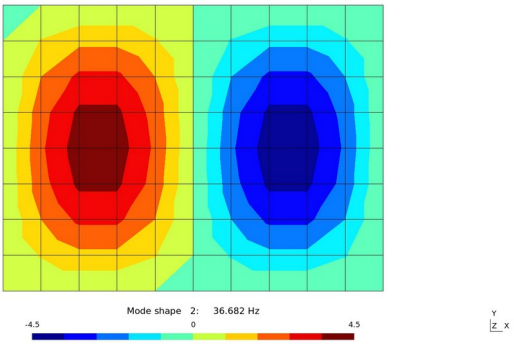
Die Tabelle zeigt, dass der Fehler bei der gröbsten Diskretisierung bereits ab der dritten Eigenschwingung recht groß ist.

Abbildung 3.2 zeigt die ersten vier mit der gröbsten Diskretisierung berechneten Schwingungen. Es ist zu erkennen, dass die Auflösung bereits für die dritte Schwingung sehr grob ist. Ein Vergleich mit Abbildung 3.1 zeigt, dass bereits bei der vierten Schwingung eine Vertauschung der Reihenfolge auftritt.

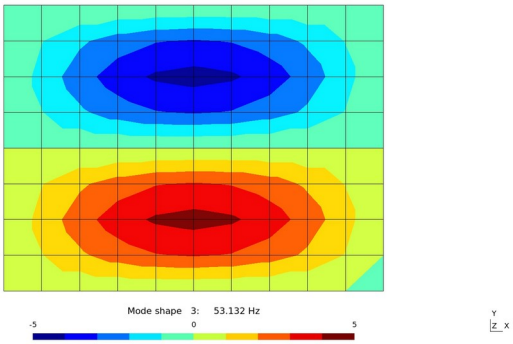
Bei den feineren Diskretisierungen 2 bis 4 treten keine Vertauschungen der Reihenfolge mehr auf. Abbildung 3.3 zeigt einen Vergleich der mit den Diskretisierungen 2 bis 4 berechneten 16. Schwingform mit der analytischen Schwingform. Die Schwingform kann bereits mit der 2. Diskretisierung wiedergegeben werden.



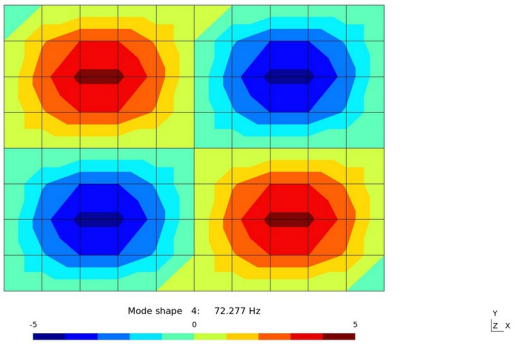
Schwingung 1: 17,25 Hz



Schwingung 2: 36,68 Hz

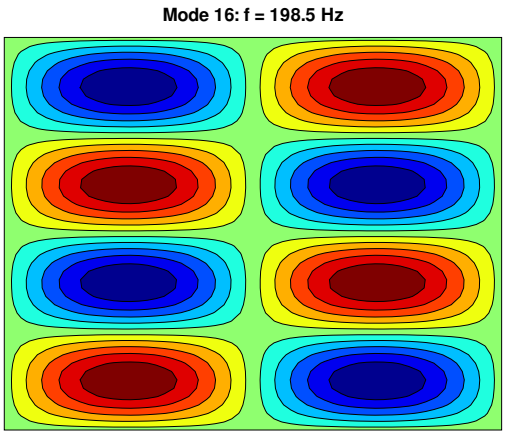


Schwingung 3: 53,13 Hz

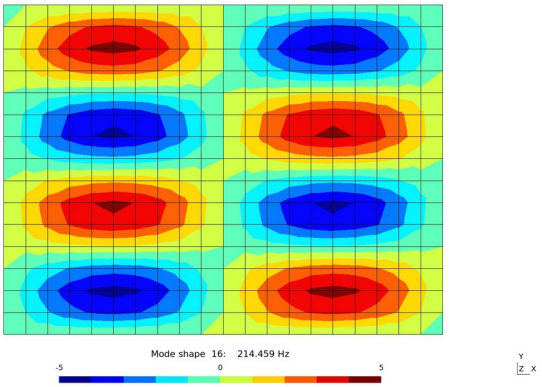


Schwingung 4: 72,28 Hz

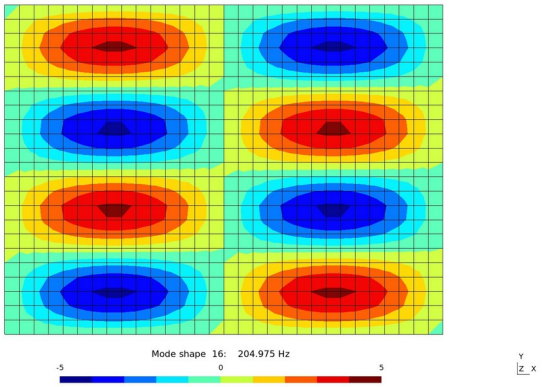
Abbildung 3.2: Diskretisierung 1: Schwingungen 1 bis 4



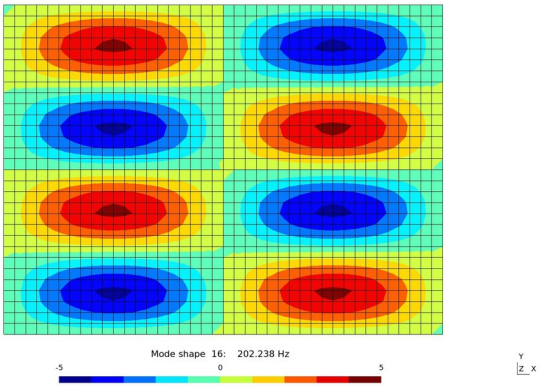
Analytisch: 198,5 Hz



Diskretisierung 2: 214,5 Hz



Diskretisierung 3: 205,0 Hz



Diskretisierung 4: 202,2 Hz

Abbildung 3.3: Vergleich der 16. Schwingform

Die Abhängigkeit des relativen Fehlers in der Frequenz von der Anzahl der Elemente pro Wellenlänge ist in Abbildung 3.4 dargestellt. Die berechneten Werte liegen recht gut auf der Ausgleichskurve, was durch den Korrelationskoeffizienten von -0,9977 bestätigt wird. Größere Abweichungen zeigen sich vor allem bei den Fehlern der 1. Diskretisierung, was auch darauf zurückgeführt werden kann, dass es bei dieser Diskretisierung zu Vertauschungen in der Reihenfolge der Schwingungen kommt.

Abbildung 3.4 zeigt, dass mindestens etwa 20 Elemente pro Wellenlänge benötigt werden, wenn der Fehler in der Frequenz kleiner als 1 % sein soll. Aus der Formel für die Ausgleichskurve ergibt sich ein Fehler von 13,2 %, wenn 6 Elemente pro Wellenlänge verwendet werden, und von 3 %, wenn 12 Elemente pro Wellenlänge verwendet werden.

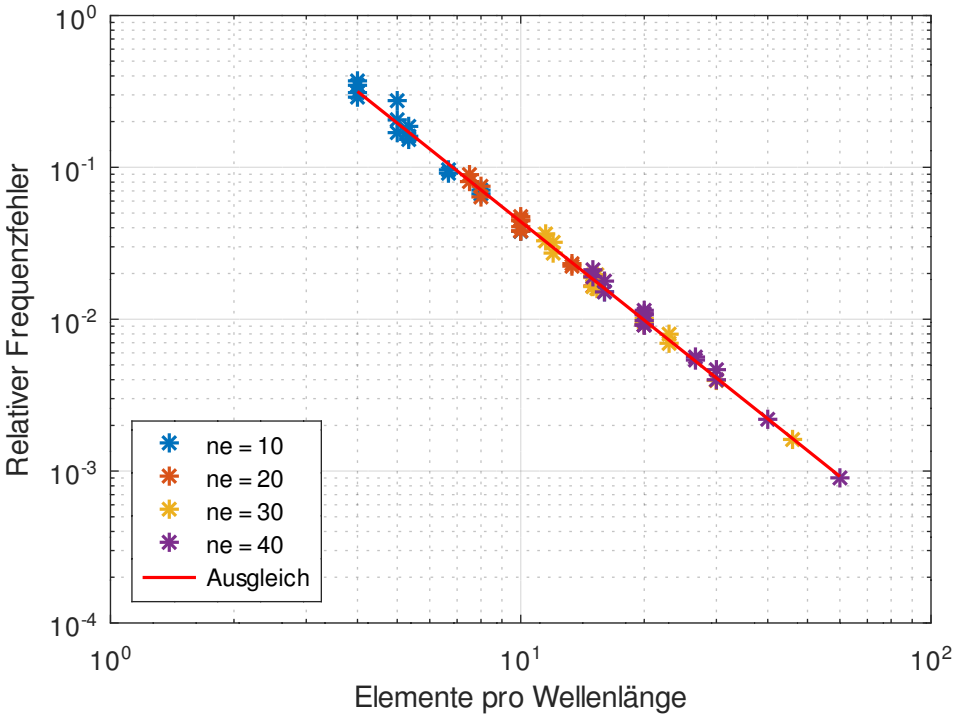


Abbildung 3.4: Frequenzfehler in Abhängigkeit von der Diskretisierung

Aufgabe 4

a) Geometrie und Vernetzung

Das folgende Gmsh-Skript definiert die Geometrie und die Netzfeinheit.

Die Elementknoten liegen auf der Oberkante des unteren Längsträgers und auf der Unterkante des oberen Längsträgers. Auf diese Weise lässt sich ein Durchdringen der Elemente vermeiden.

```
/* ------------------------------------------------------
   Übungsblatt 5.2, Aufgabe 4: Gitter
------------------------------------------------------ */

/* Parameter */

   a   = 1000;        // Segmentlänge in mm
   h   =  200;        // Querschnittshöhe in mm
   nea =   20;        // Elemente pro Segment

   lne = a / nea;     // Elementlänge

/* Punkte */

   z_oben  = 0.5 * (a - h);
```

```
   z_unten = -z_oben;

   Point( 1) = {     0, 0, z_unten, lne};
   Point( 2) = {     a, 0, z_unten, lne};
   Point( 3) = { 2 * a, 0, z_unten, lne};
   Point( 4) = { 3 * a, 0, z_unten, lne};
   Point( 5) = { 4 * a, 0, z_unten, lne};

   Point( 6) = {     0, 0, z_oben, lne};
   Point( 7) = {     a, 0, z_oben, lne};
   Point( 8) = { 2 * a, 0, z_oben, lne};
   Point( 9) = { 3 * a, 0, z_oben, lne};
   Point(10) = { 4 * a, 0, z_oben, lne};

/* Einspannung */

   Physical Point("Einspannung") = {1, 5, 6, 10};

/* Längsträger */

   Line( 1) = { 1,  2};
   Line( 2) = { 2,  3};
   Line( 3) = { 3,  4};
   Line( 4) = { 4,  5};

   Line( 5) = { 6,  7};
   Line( 6) = { 7,  8};
   Line( 7) = { 8,  9};
   Line( 8) = { 9, 10};

   Physical Line("Laengstraeger_unten") = {1 : 4};
   Physical Line("Laengstraeger_oben")  = {5 : 8};

/* Streben */

   Line( 9) = { 2,  7};
   Line(10) = { 3,  8};
   Line(11) = { 4,  9};

   Physical Line("Streben") = {9 : 11};
```

Abbildung 4.1 zeigt die damit erzeugte Vernetzung sowie die Richtung der *z*-Achsen des Balkenkoordinatensystems.

b) Modalanalyse

Das GNU Octave-Skript beginnt mit der Definition der Daten und des Materials. Anschließend wir die Ausgabedatei geöffnet.

```
# Übungsblatt 5.2, Aufgabe 4: Gitter
#
# ------------------------------------------------------------

# Daten (N, mm):
```

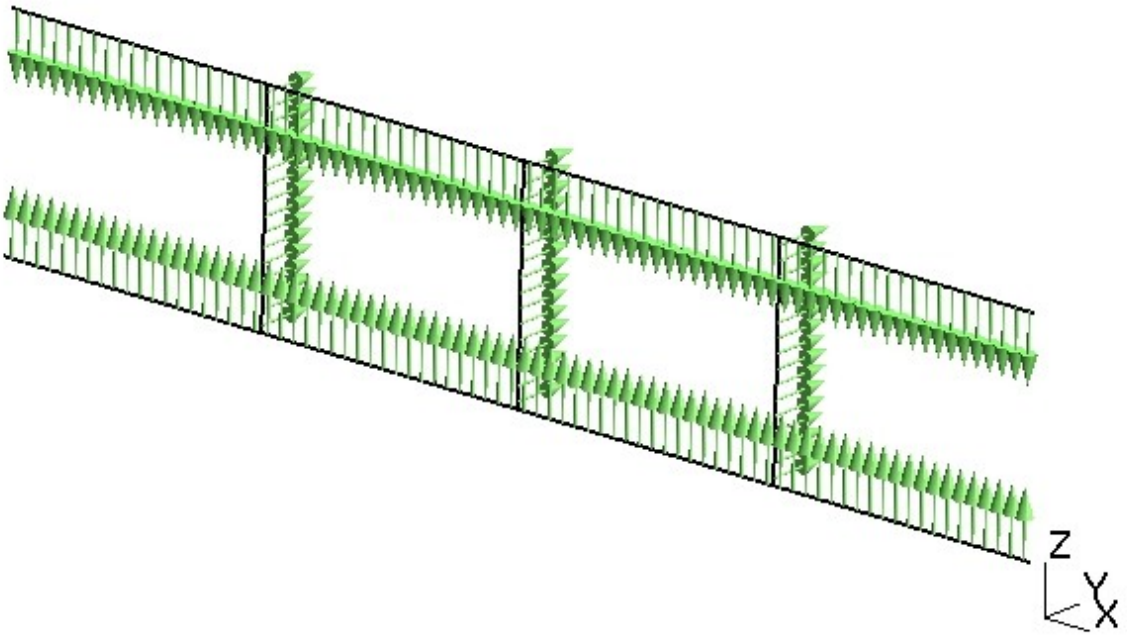


Abbildung 4.1: Finite-Elemente-Modell

```
  b = 150;    % Breite des Rechteckprofils
  h = 200;    % Höhe des Rechteckprofils
  r =   8;    % Radius des Kreisringprofils
  t1 =  2;    % Wandstärke des Rechteckprofils
  t2 =  1;    % Wandstärke des Kreisringprofils

  mat = struct("type", "iso", "E", 210000, "ny", 0.3,
               "rho", 7.85E-9);

# Anzahl der Eigenschwingungen

  nmodes = 20;

# Augabedatei

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

Als nächstes werden die Übersetzungsdaten definiert, die benötigt werden, um aus der mit Gmsh erzeugten Vernetzung ein Finite-Elemente-Modell zu erstellen. Zunächst werden der Modelltyp und der Untertyp definiert.

```
# Modelltyp

  td = struct("type", "solid", "subtype", "3d");
```

Anschließend werden die Daten für die beiden Längsträger definiert. In der mit Gmsh erzeugten msh-Datei sind die Elemente der beiden Längsträger in

den physikalischen Gruppen **Laengstraeger_unten** bzw. **Laengstraeger_oben**. enthalten. Zunächst wird festgelegt, dass es sich um Balkenelemente (**"b2"**) handelt. Die Querschnittskennwerte für das dünnwandige Kastenprofil werden mit der Funktion **mfs_beamsection** berechnet. Feld **geom.v** enthält einen Vektor in der $x_E z_E$-Ebene des Balkenelements. Dieser Vektor definiert die Lage des Elementkoordinatensystems. Da die Elementknoten nicht im Flächenschwerpunkt liegen, müssen ihre Koordinaten angegeben werden. Feld **geom.P** enthält die y_E- und z_E-Koordinaten der Elementknoten im Elementkoordinatensystem.

```
# Längsträger

  geom   = mfs_beamsection("box", "thin", b, h, t1);
  geom.v = [0, 0, 1];
  geom.P = [0, 0.5 * h];
  ltu = struct("type", "elements", "name", "b2",
               "geom", geom, "mat", mat);

  lto = ltu;
  lto.geom.v = [0, 0, -1];

  td.Laengstraeger_unten = ltu;
  td.Laengstraeger_oben  = lto;
```

Die Daten für die Streben werden auf die gleiche Weise wie bei den Längsträgern definiert. Der Querschnitt ist ein dünnwandiger Kreisring. Da hier die Elementknoten im Flächenschwerpunkt liegen, müssen ihre Koordinaten nicht definiert werden.

```
# Streben

  geom   = mfs_beamsection("ring", "thin", r, t2);
  geom.v = [0, 1, 0];
  st = struct("type", "elements", "name", "b2",
              "geom", geom, "mat", mat);

  td.Streben = st;
```

Zuletzt wird die Einspannung definiert. Die Knoten, an denen die Struktur eingespannt ist, sind in der physikalischen Gruppe **Einspannung** enthalten. Der Typ dieser Gruppe ist **"constraints"**. Festgehalten werden alle sechs Freiheitsgrade, d. h. alle drei Translationen und alle drei Rotationen.

```
# Einspannung

  td.Einspannung = struct("type", "constraints",
                          "name", "prescribed", "dofs", 1 : 6);
```

Nun wird die msh-Datei eingelesen, aus der mithilfe der Übersetzungsdaten eine Modellbeschreibung für Mefisto erzeugt wird. Anschließend wird eine

neue Komponente erzeugt. Die Koordinatensysteme der Balkenelemente werden zur Kontrolle nach Gmsh exportiert.

```
# msh-Datei einlesen

  model = mfs_import(fid, [file, ".msh"], "msh", td);

# Komponente initialisieren und Achsen exportieren

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

# Matrizen

  gitter = mfs_stiff(gitter);
  gitter = mfs_mass(gitter);
  mfs_massproperties(fid, gitter);

# Eigenschwingungen

  gitter = mfs_freevib(gitter, nmodes);
  mfs_print(fid, gitter, "modes", "freq");
  mfs_export([file, ".pos"], "msh", gitter, "modes", "disp");

  fclose(fid);
```

Einige der Eigenschwingungen sind in Abbildung 4.2 dargestellt.

Die Ausgabedatei enthält die folgende Liste der Eigenfrequenzen:

```
Component "gitter"

Natural frequencies:

  Mode   Circ. Frequency    Frequency
  ------------------------------------
     1       454.60343      72.35238 Hz
     2       458.42914      72.96126 Hz
     3       565.17265      89.95002 Hz
     4      1010.45987     160.81968 Hz
     5      1017.33670     161.91416 Hz
     6      1020.23893     162.37607 Hz
     7      1022.07352     162.66805 Hz
     8      1022.20796     162.68945 Hz
     9      1022.24491     162.69533 Hz
    10      1259.14614     200.39933 Hz
    11      1272.57810     202.53710 Hz
    12      1556.67343     247.75227 Hz
    13      1631.54533     259.66850 Hz
    14      2104.44998     334.93362 Hz
    15      2150.15227     342.20736 Hz
    16      2200.63824     350.24245 Hz
    17      2445.32584     389.18570 Hz
    18      2472.21714     393.46558 Hz
    19      2806.66280     446.69426 Hz
    20      2813.42288     447.77016 Hz
```

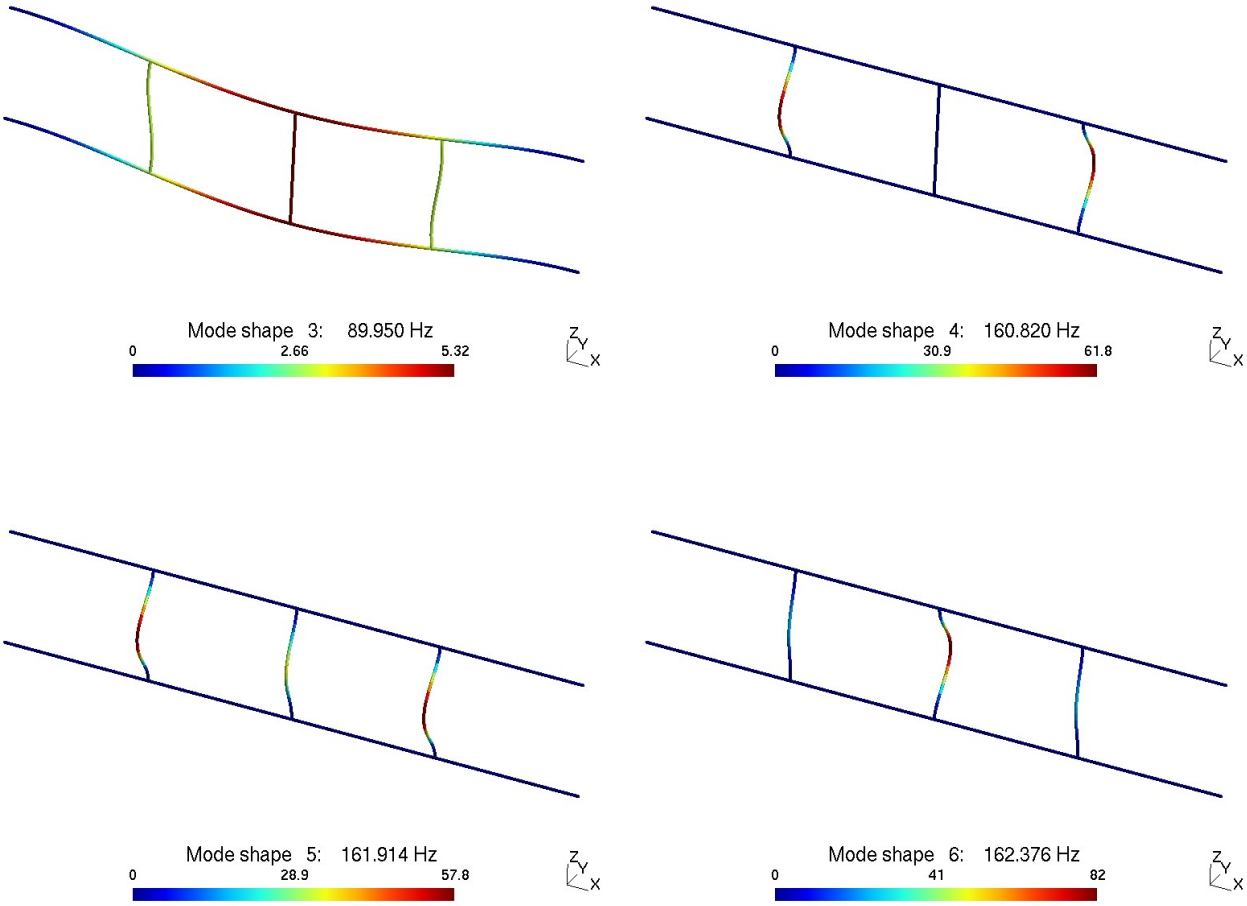


Abbildung 4.2: Einige Eigenschwingungen

Die Liste zeigt, dass Cluster auftreten, bei denen mehrere Frequenzen nahe beieinander liegen. Abbildung 4.2 zeigt, dass es sich dabei um lokale Schwingungen der vertikalen Streben handelt.

Aufgabe 5

a) Geometrie und Vernetzung

Das folgende Gmsh-Skript definiert die Geometrie und die Netzfeinheit.

Bei diesem Beispiel sind die Querschnittsabmessungen klein gegenüber den Längen der Balken. Daher wird hier ein Durchdringen der Balken an den Knoten akzeptiert, das auftritt, wenn die Knoten der Balken im Schwerpunkt des Querschnitts liegen.

```
/* ------------------------------------------------
```

```
   Übungsblatt 5.2, Aufgabe 5: Balkenstruktur
------------------------------------------------------ */

/* Daten */

   a    =    1000;   // Länge in x-Richtung
   b    =     750;   // Länge in y-Richtung
   h    =    2000;   // Höhe
   hm   =    2200;   // Höhe von Punkt M
   nel  =       5;   // Elemente entlang Länge a
   lne  = a / nel;   // Elementlänge

/* Punkte */

  A = 1; B = 2; C = 3; D = 4; E = 5;
  F = 6; G = 7; H = 8; M = 9;

  Point(A) = { -0.5 * a, -0.5 * b, 0, lne};
  Point(B) = {  0.5 * a, -0.5 * b, 0, lne};
  Point(C) = {  0.5 * a,  0.5 * b, 0, lne};
  Point(D) = { -0.5 * a,  0.5 * b, 0, lne};

  Point(E) = { -0.5 * a, -0.5 * b, h, lne};
  Point(F) = {  0.5 * a, -0.5 * b, h, lne};
  Point(G) = {  0.5 * a,  0.5 * b, h, lne};
  Point(H) = { -0.5 * a,  0.5 * b, h, lne};

  Point(M) = {0, 0, hm, lne};

  Physical Point("Punktmasse") = {M};

/* Senkrechte Balken */

  AE = 1; BF = 2; CG = 3; DH = 4;

  Line(AE) = {A, E};
  Line(BF) = {B, F};
  Line(CG) = {C, G};
  Line(DH) = {D, H};

  Physical Line("Pfeiler") = {AE, BF, CG, DH};

/* Waagerechte Balken */

  EF = 5; FG = 6; GH = 7; HE = 8;

  Line(EF) = {E, F};
  Line(FG) = {F, G};
  Line(GH) = {G, H};
  Line(HE) = {H, E};

  Physical Line("Querstreben") = {EF, FG, GH, HE};

/* Spitze */
```

```
  EM =  9; FM = 10; GM = 11; HM = 12;

  Line(EM) = {E, M};
  Line(FM) = {F, M};
  Line(GM) = {G, M};
  Line(HM) = {H, M};

  Physical Line("Spitze") = {EM, FM, GM, HM};

/* Lagerung */

  Physical Point("Einspannung") = {A, B, C, D};
```

b) Modalanalyse

Das GNU Octave-Skript zur Berechnung der Eigenschwingungen stimmt im Wesentlichen mit dem GNU Octave-Skript zur Lösung von Aufgabe 4 überein.

```
# Kapitel 5.2, Aufgabe 5: Balkenstruktur
#
# ------------------------------------------------------------

# Daten (N, mm):

  c       =  20;      % Kantenlänge des Hohlquadrats
  t       =   2;      % Wandstärke des Hohlquadrats
  mtop    = 0.1;      % Punktmasse (in t)
  nmodes  =  10;      % Anzahl der Eigenschwingungen

  mat = struct("type", "iso", "E",   210000, "ny", 0.3,
               "rho", 7.85E-9);

# Augabedatei

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

# Modelltyp

  td = struct("type", "solid", "subtype", "3d");

# Elementdaten allgemein

  geom = mfs_beamsection("box", "thin", c, c, t);

# Elementdaten für Pfeiler

  geom.v = [1, 0, 0];
  td.Pfeiler = struct("type", "elements", "name", "b2",
                      "geom", geom, "mat", mat);

# Elementdaten für Querstreben

  geom.v = [0, 0, 1];
```

```
  td.Querstreben = struct("type", "elements", "name", "b2",
                          "geom", geom, "mat", mat);

# Elementdaten für Spitze

  geom.v = [0, 0, 1];
  td.Spitze = struct("type", "elements", "name", "b2",
                     "geom", geom, "mat", mat);

# Punktmasse an der Spitze

  geom = struct("m", mtop);
  td.Punktmasse = struct("type", "elements", "name", "m1",
                         "geom", geom);

# Einspannung

  td.Einspannung = struct("type", "constraints",
                          "name", "prescribed", "dofs", 1 : 6);

# msh-Datei einlesen

  model = mfs_import(fid, [file, ".msh"], "msh", td);

# Komponente initialisieren

  turm = mfs_new(fid, model);
  mfs_export("axes.msh", "msh", turm, "mesh", "axes");

# Steifigkeits- und Massenmatrix

  turm = mfs_stiff(turm);
  turm = mfs_mass(turm, "consistent");
  mfs_massproperties(fid, turm);
```

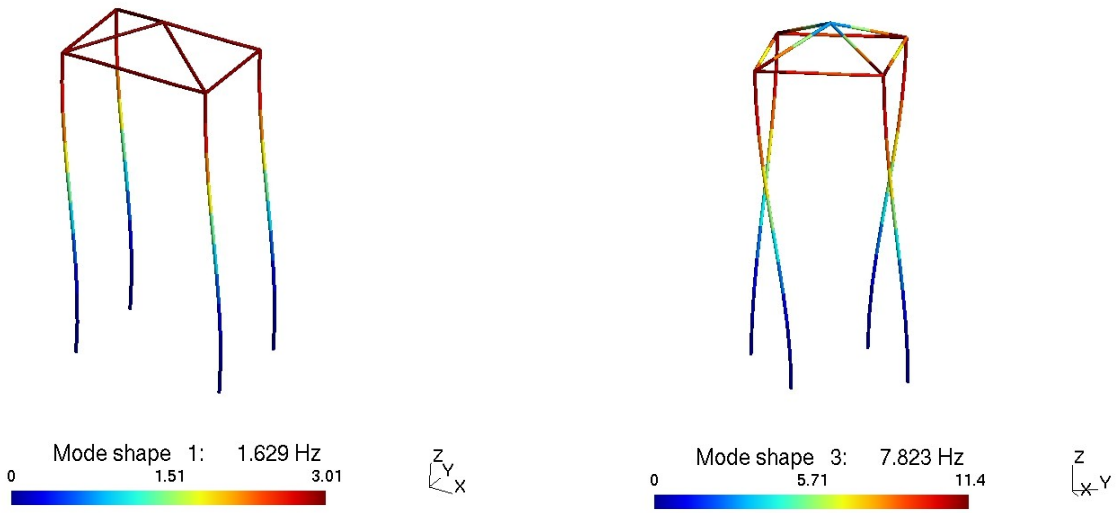


Abbildung 5.1: 1. und 3. Eigenschwingung

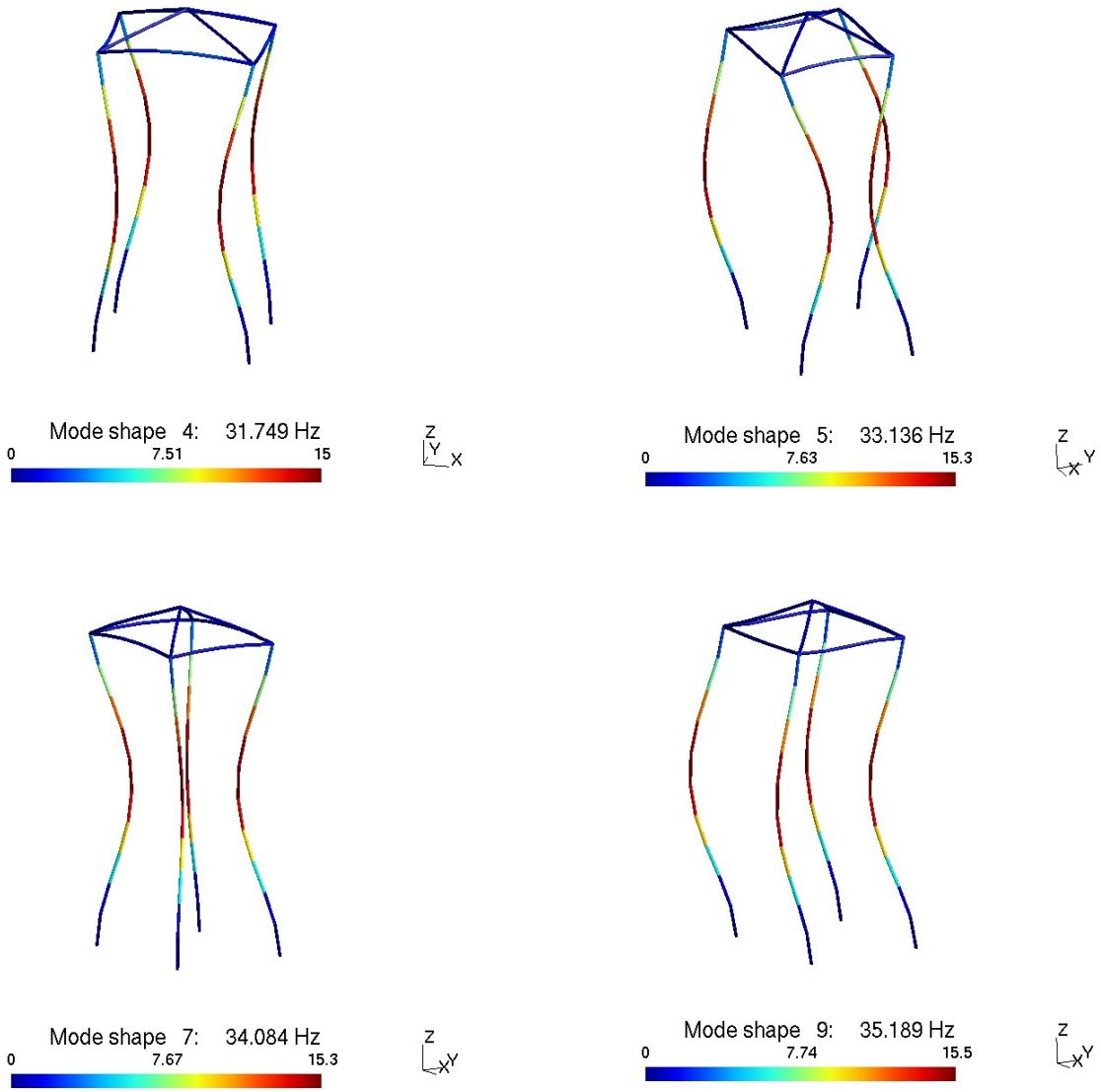


Abbildung 5.2: Weitere Eigenschwingungen

```
# Eigenschwingungen

  turm = mfs_freevib(turm, nmodes);

# Ausgabe

  mfs_print(fid, turm, "modes", "freq");
  mfs_export([file, ".pos"], "msh", turm, "modes", "disp");

  fclose(fid);
```

Die Ausgabedatei enthält die folgende Liste der Eigenfrequenzen:

```
Component "turm"
```

```
Natural frequencies:

  Mode    Circ. Frequency     Frequency
  -------------------------------------
     1         10.23546        1.62902 Hz
     2         10.29770        1.63893 Hz
     3         49.15475        7.82322 Hz
     4        199.48631       31.74923 Hz
     5        208.20138       33.13628 Hz
     6        209.38080       33.32399 Hz
     7        214.15651       34.08407 Hz
     8        219.64106       34.95696 Hz
     9        221.09980       35.18913 Hz
    10        222.11889       35.35132 Hz
```

Einige der Eigenschwingungen sind in den Abbildungen 5.1 und 5.2 dargestellt.

Aufgabe 6

a) Geometrie und Vernetzung

Damit eine Vernetzung mit viereckigen Elementen möglich ist, werden die beiden Kugelkalotten in jeweils fünf Vierecke unterteilt. Dabei befindet sich ein Viereck mittig zur Rotationsachse des Behälters. Durch seine vier Ecken verlaufen vier Kreisbögen, die dieses Viereck mit dem Umfang des Zylinders verbinden und die übrige Fläche der Kugelkalotte in vier weitere Vierecke unterteilen. Daraus ergibt sich auch eine Unterteilung des Zylindermantels in vier Vierecke. Die gesamte Geometrie ist in Abbildung 6.1 dargestellt.

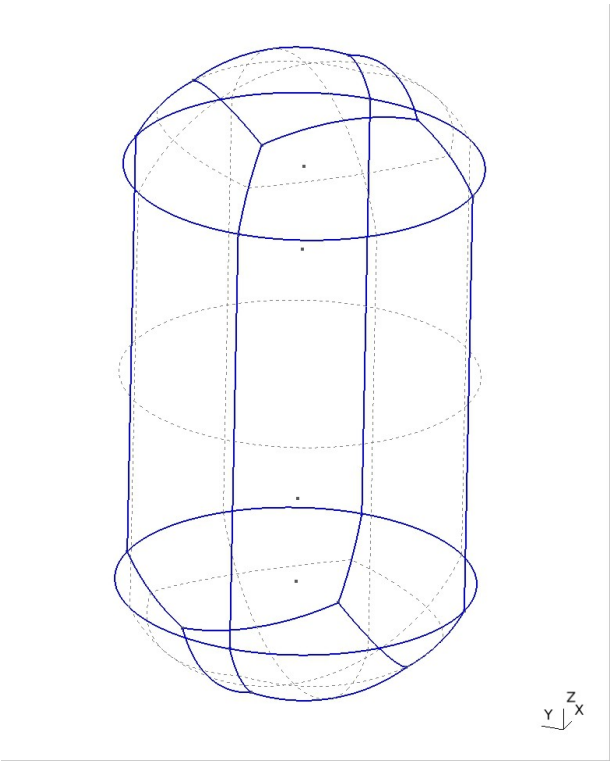


Abbildung 6.1: Geometrie des Behälters

Das Gmsh-Skript beginnt mit der Definition der Abmessungen. Die Netzfeinheit wird durch die Anzahl der Elemente entlang des Umfangs (Variable **neu**) sowie die Anzahl der Elemente über die Zylinderhöhe (Variable **neh**) gesteuert. Die Werte dieser Variablen werden interaktiv abgefragt, wenn die Abfrage aktiviert ist.

```
/* ------------------------------------------------------------
```

```
   Übungsblatt 5.2, Aufgabe 6: Zylindrischer Behälter
------------------------------------------------------------- */

// Abmessungen (in mm)

   DefineConstant [ r = 2000,     // Radius
                    h = 5000,     // Höhe des zylindrischen Teils
                    e = 1000 ];   // Abstand des Kugelmittelpunkts

   R  = Sqrt(r^2 + e^2);  // Kugelradius
   hh = 0.5 * h;          // halbe Höhe

// Netzfeinheit

   DefineConstant [ neu = 40,     // Elemente entlang Umfang
                    neh = 20,     // Elemente über die Höhe
                    ask =  1 ];   // Abfrage aktiviert

   If (ask == 1)
      neu = GetValue("Anzahl Elemente entlang Umfang?", neu);
      neh = GetValue("Anzahl Elemente über die Höhe?", neh);
   EndIf

   neq = Ceil(0.25 * neu);
```

Die Definition der Geometrie beginnt mit der oberen Kugelkalotte. Als erstes wird der Mittelpunkt der Kugel definiert.

```
/* -------------------------------------------------------------
   Obere Kugelkalotte
------------------------------------------------------------- */

   zmo = hh - e;
   Point(1) = {0, 0, zmo};  // Kugelmittelpunkt
```

Dann werden die vier Kreisbögen des Kreises definiert, in dem die Kugelkalotte an den Zylinder angeschlossen ist.

```
// Anschlusskreis

   Point(2) = { 0, -r, hh};
   Point(3) = { r,  0, hh};
   Point(4) = { 0,  r, hh};
   Point(5) = {-r,  0, hh};

   Point(6) = { 0,  0, hh};   // Mittelpunkt

   Circle(1) = {2, 6, 3};
   Circle(2) = {3, 6, 4};
   Circle(3) = {4, 6, 5};
   Circle(4) = {5, 6, 2};
```

Anschließend wird die Kappe definiert. Das ist die Fläche, die mittig zur Rotationsachse liegt. Die vier Eckpunkte der Kappe entstehen aus den Eckpunk-

ten der Kreisbögen auf dem Anschlusskreis durch Rotation um 27° um Achsen durch den Kugelmittelpunkt, die parallel zur *x*- bzw. *y*-Achse sind. Die Kanten der Kappe sind Kreisbögen durch die Eckpunkte mit dem Kugelmittelpunkt als Mittelpunkt. Die Kappe ist eine Fläche auf dieser Kugel.

```
// Kappe

   phi = 0.15 * Pi;   // 0.15 = 27 / 180

   Rotate {{-1,  0, 0}, {0, 0, zmo}, phi}
          { Duplicata { Point{2}; } }
   Rotate {{ 0, -1, 0}, {0, 0, zmo}, phi}
          { Duplicata { Point{3}; } }
   Rotate {{ 1,  0, 0}, {0, 0, zmo}, phi}
          { Duplicata { Point{4}; } }
   Rotate {{ 0,  1, 0}, {0, 0, zmo}, phi}
          { Duplicata { Point{5}; } }

   Circle(5) = { 7, 1,  8};
   Circle(6) = { 8, 1,  9};
   Circle(7) = { 9, 1, 10};
   Circle(8) = {10, 1,  7};

   Curve Loop(1) = {5, 6, 7, 8};
   Surface(1) = {1} In Sphere {1};
```

Die Eckpunkte der Kappe werden durch vier Kreisbögen mit den Kreisbögen des Anschlusskreises verbunden. Damit können die restlichen vier Flächen der Kugelkalotte definiert werden. Alle fünf Flächen sind so orientiert, dass der Normalenvektor nach außen zeigt.

```
// Verbindungen zum Anschlusskreis

   Circle( 9) = { 2, 1,  7};
   Circle(10) = { 3, 1,  8};
   Circle(11) = { 4, 1,  9};
   Circle(12) = { 5, 1, 10};

   Curve Loop(2) = {1, 10, -5,  -9};
   Curve Loop(3) = {2, 11, -6, -10};
   Curve Loop(4) = {3, 12, -7, -11};
   Curve Loop(5) = {4,  9, -8, -12};

   Surface(2) = {2} In Sphere {1};
   Surface(3) = {3} In Sphere {1};
   Surface(4) = {4} In Sphere {1};
   Surface(5) = {5} In Sphere {1};
```

Die Kurven der unteren Kugelkalotte entstehen durch Spiegelung der Kurven der oberen Kugelkalotte an der *xy*-Ebene. Damit werden dann die fünf Flächen der unteren Kugelkalotte definiert. Die Orientierung der Elemente auf diesen Flächen wird umgedreht, damit die Normalenvektoren nach außen zeigen.

```
/* ------------------------------------------------------------
   Untere Kugelkalotte
------------------------------------------------------------ */

   Symmetry {0, 0, 1, 0} { Duplicata { Curve{1 : 12}; } }

   Curve Loop( 6) = {20, 17, 18, 19};
   Curve Loop( 7) = {14, 23, -18, -22};
   Curve Loop( 8) = {15, 24, -19, -23};
   Curve Loop( 9) = {16, 21, -20, -24};
   Curve Loop(10) = {13, 22, -17, -21};

   Surface( 6) = { 6} In Sphere {32};
   Surface( 7) = { 7} In Sphere {32};
   Surface( 8) = { 8} In Sphere {32};
   Surface( 9) = { 9} In Sphere {32};
   Surface(10) = {10} In Sphere {32};

   ReverseMesh Surface {6 : 10};
```

Nun müssen nur noch die Verbindungslinien zwischen dem oberen und dem unteren Anschlusskreis definiert werden, damit die Flächen auf dem Zylindermantel definiert werden können.

```
/* ------------------------------------------------------------
   Zylindermantel
------------------------------------------------------------ */

   Line(25) = {11, 2};
   Line(26) = {13, 3};
   Line(27) = {18, 4};
   Line(28) = {23, 5};

   Curve Loop(11) = {13, 26, -1, -25};
   Curve Loop(12) = {14, 27, -2, -26};
   Curve Loop(13) = {15, 28, -3, -27};
   Curve Loop(14) = {16, 25, -4, -28};

   Surface(11) = {11};
   Surface(12) = {12};
   Surface(13) = {13};
   Surface(14) = {14};
```

Die physikalische Gruppe **Wand** enthält alle Flächen. Sie wird für die Definition der Schalenelemente benötigt. Die physikalische Gruppe **Einspannung** enthält die vier Kreisbögen des unteren Anschlusskreises. Mit ihr wird die Lagerung definiert.

```
/* ------------------------------------------------------------
   Physikalische Gruppen
------------------------------------------------------------ */

// Wand
```

```
   Physical Surface("Wand") = {1 : 14};

// Einspannung

   Physical Curve("Einspannung") = {13 : 16};
```

Zum Schluss wird die Vernetzung definiert. Die Anzahl der Elemente für die Kreisbögen, die die Kappe mit dem Anschlusskreis verbinden, wird halb so groß gewählt wie die Anzahl der Elemente auf einem Viertelkreis in Umfangsrichtung. Die Elementgröße wird über eine Progression auf die Anschlusskreise hin verkleinert.

```
/* ------------------------------------------------------------
   Vernetzung
------------------------------------------------------------ */

   Mesh.RecombineAll = 1; // Dreiecke zu Vierecken zusammenfassen

// Unterteilung in Umfangsrichtung

   npu = neq + 1;
   Transfinite Curve {1 : 8, 13 : 20} = npu;

// Unterteilung Kappe

   nek = Ceil(0.5 * neq);
   npk = nek + 1;
```

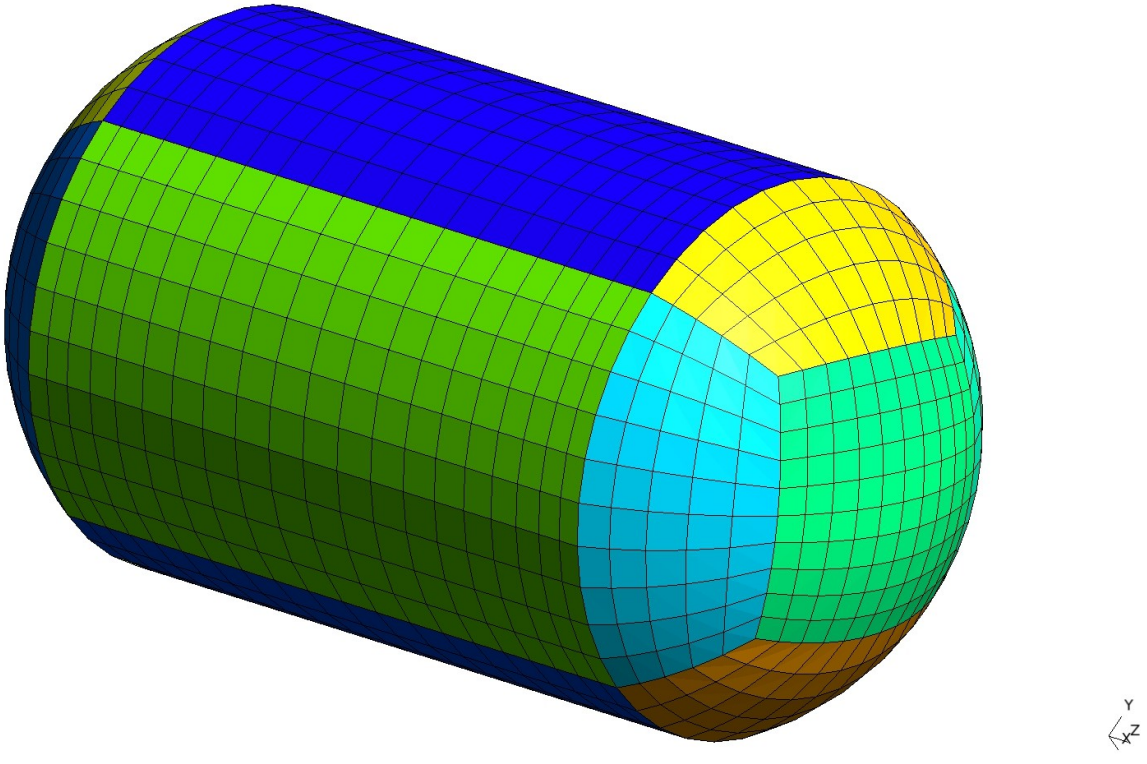


Abbildung 6.2: Vernetzung des Behälters

```
   Transfinite Curve {9 : 12, 21 : 24} = npk
      Using Progression 1.1;

// Unterteilung der Zylinderhöhe

   nph = neh + 1;
   Transfinite Curve {25 : 28} = nph Using Bump 0.5;

// Flächen

   Transfinite Surface{1 : 14};
```

Abbildung 6.2 zeigt eine mit den Voreinstellungen für die Vernetzungsparameter erzeugte Vernetzung des Behälters.

b) Modalanalyse

Das GNU Octave-Skript für die Modalanalyse stimmt im Wesentlichen mit den Skripten zur Lösung der vorhergehenden Aufgaben überein. Bei Schalenelementen muss in der Regel nur die Schalendicke als Geometrieparameter angegeben werden.

```
# Übungsblatt 5.2, Aufgabe 6: Zylindrischer Behälter
#                             Einzelne Analyse
#
# ------------------------------------------------------------

  nmodes =  20;  % Anzahl der Eigenschwingungen

# Daten (N, mm):

  geom = struct("t", 2);                          % Wandstärke
  mat  = struct("type", "iso", "E", 210000,       % Materialdaten
                "ny", 0.3, "rho", 7.85E-9);

  mtype    = "lumped"; % Typ der Massenmatrix

# Modelltyp

  td = struct("type", "solid", "subtype", "3d");

# Elementdaten

  td.Wand = struct("type", "elements", "name", "s4",
                   "geom", geom, "mat", mat);

# Einspannung

  td.Einspannung = struct("type", "constraints",
                          "name", "prescribed", "dofs", 1 : 3);

# Augabedatei
```

Diskretisierung	1	2	3	4	5	6
Elemente in Umfangsrichtung	40	60	80	100	120	240
Elemente in Längsrichtung	20	20	25	25	30	60
Anzahl Elemente	1400	2610	4400	6350	9000	3600
Anzahl Freiheitsgrade	8292	15492	26172	37812	53652	215292

Tabelle 6.1: Diskretisierungen

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

# Datei einlesen, Komponente erzeugen und Achsen ausgeben

  model = mfs_import(fid, [file, ".msh"], "msh", td);
  tank  = mfs_new(fid, model);
  mfs_export([file, ".axes"], "msh", tank, "mesh", "axes");

# Steifigkeits- und Massenmatrix

  tank = mfs_stiff(tank);
  tank = mfs_mass(tank, mtype);
  mfs_massproperties(fid, tank);
```

Nr.	n_z	n_ϕ	f [Hz]					
			1	2	3	4	5	6
1	0,5	9	16,32	15,76	15,59	15,52	15,48	15,42
3	0,5	10	17,32	16,30	16,02	15,91	15,85	15,75
5	0,5	8	17,00	16,70	16,59	16,55	16,52	16,47
7	0,5	11	19,78	17,99	17,55	17,37	17,28	17,13
9	0,5	7	19,63	19,43	19,35	19,32	19,29	19,24
11	0,5	12	23,55	20,58	19,90	19,63	19,49	19,27
13	0,5	13	28,72	23,88	22,87	22,47	22,28	21,97
15	0,5	6	24,56	24,39	24,30	24,27	24,24	24,19
17	0,5	14		27,82	26,36	25,79	25,52	25,10
19	1	12	31,61	29,08	28,51	28,28	28,17	27,98
21	1	11	31,37					

Tabelle 6.2: Ergebnisse

```
# Eigenschwingungen

  tank = mfs_freevib(tank, nmodes);

# Ausgabe

  mfs_print(fid, tank, "modes", "freq");
  mfs_export([file, ".pos"], "msh", tank, "modes", "disp");

  fclose(fid);
```

Die Berechnung wurde für sechs verschiedene Diskretisierungen durchgeführt (siehe Tabelle 6.1). Die Ergebnisse sind in Tabelle 6.2 zusammengestellt. Dabei bedeutet n_z die Anzahl der Wellen in Längsrichtung und n_φ die Anzahl der Wellen in Umfangsrichtung. Aufgrund der Symmetrie treten alle Eigenschwingungen doppelt auf. Bei der gröbsten Diskretisierung ändern einige Eigenschwingungen ihre Reihenfolge.

Tabelle 6.2 zeigt, dass die Wellenlänge im Gegensatz zur ebenen Platte mit höheren Frequenzen nicht monoton abnimmt. Ursache dafür ist die versteifende Wirkung der Krümmung. Dieser Effekt ist für Schwingungen mit kleiner Wellenlänge kleiner als bei Schwingungen mit großer Wellenlänge. Daher treten die Schwingungen mit großer Wellenlänge erst bei höheren Frequenzen auf.

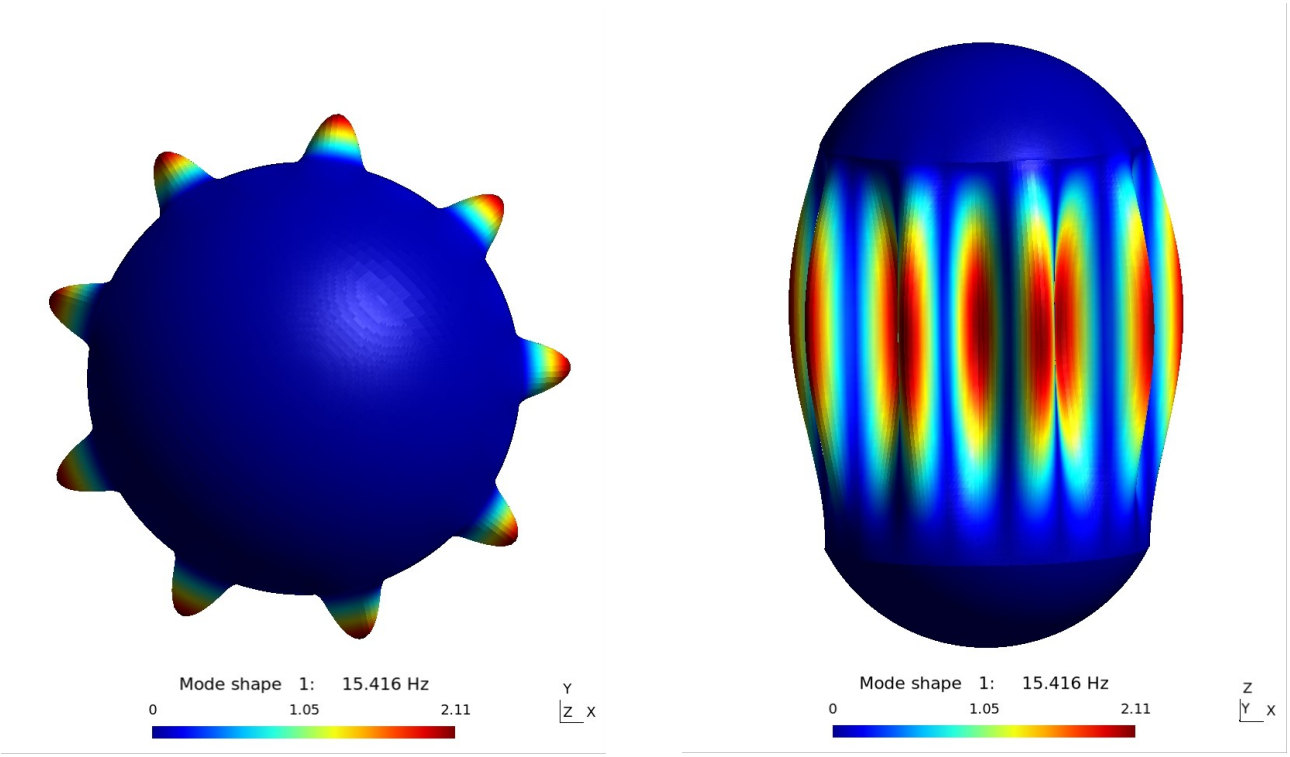


Abbildung 6.3: 1. Eigenschwingung

Einige der Schwingungen sind in den Abbildungen 6.3 bis 6.5 dargestellt. Die Anzahl der Wellen in Umfangsrichtung lässt sich am besten in der Draufsicht erkennen.

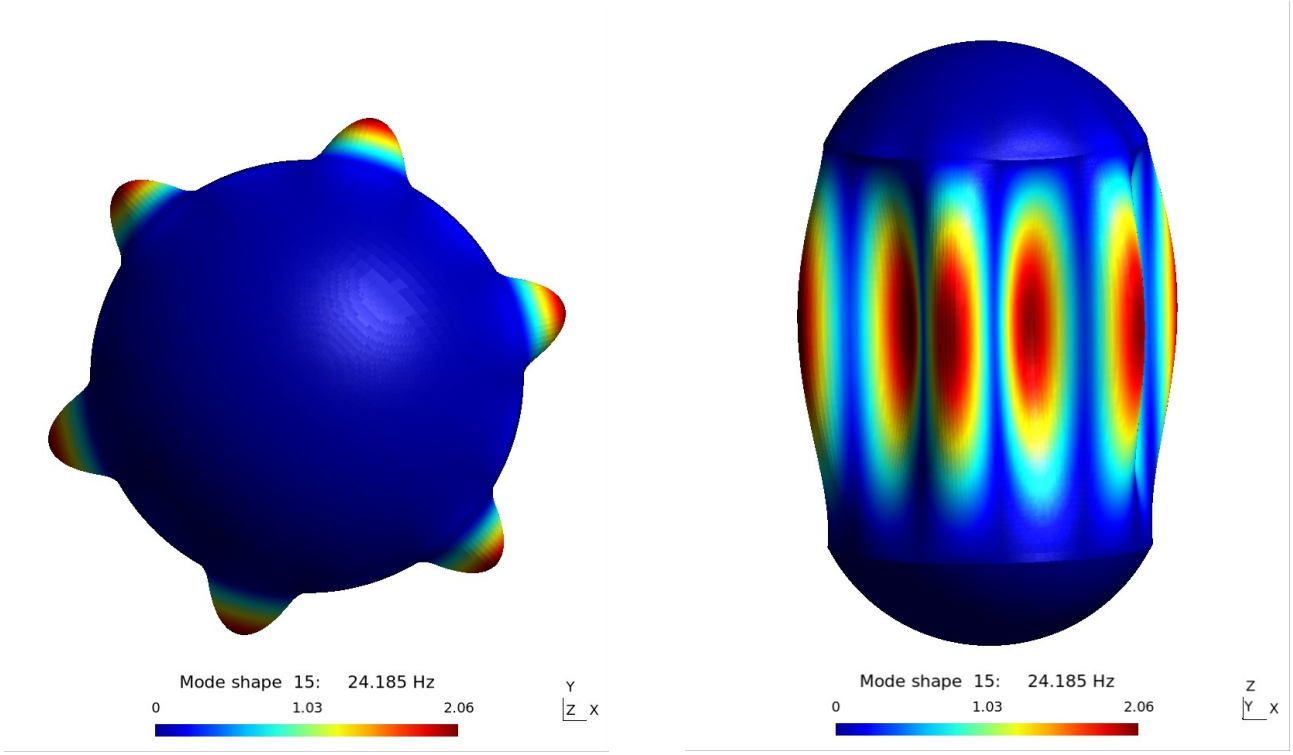


Abbildung 6.4: 15. Eigenschwingung

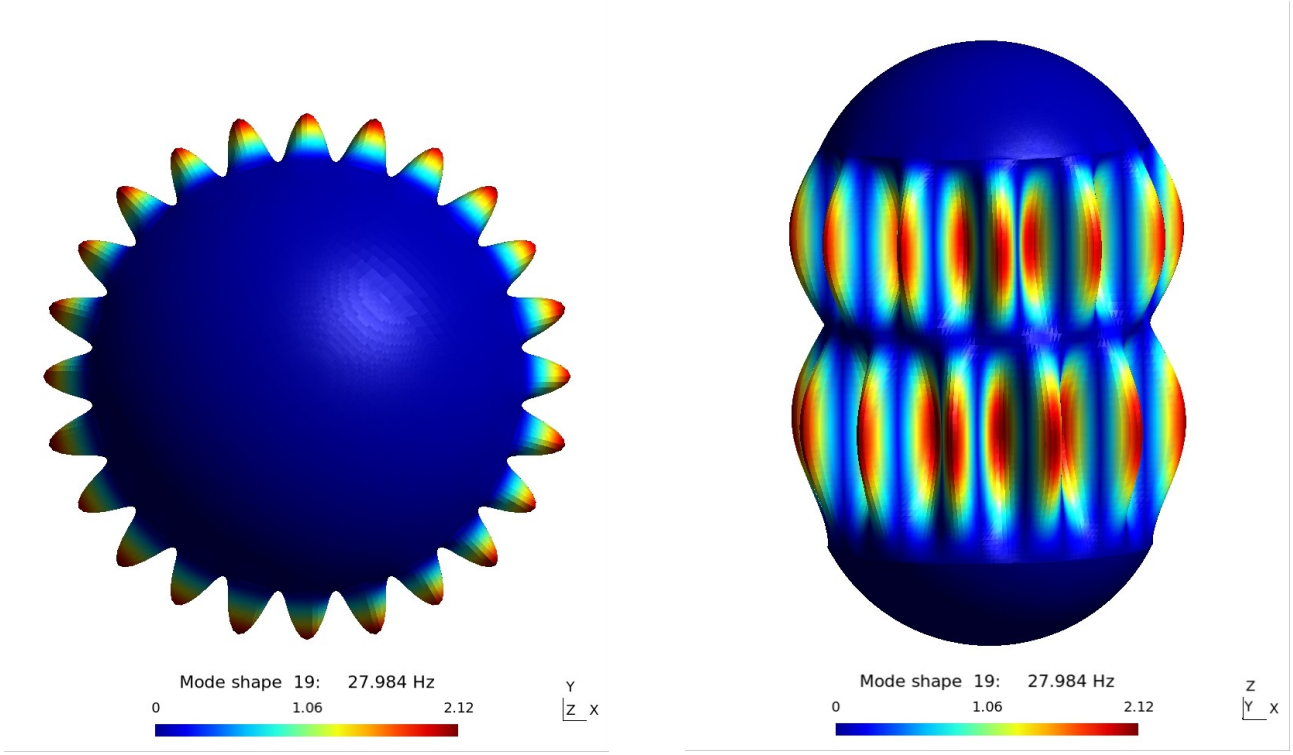


Abbildung 6.5: 19. Eigenschwingung