```
//  Ersatzmodell: Knickbiegeschwingungen -> Singulaerer Masse-
Federschwinger
 import java.awt.*;
 import java.awt.event.*;
 import javax.swing.*;

 public class AxialeVorlast extends JFrame
                              implements ActionListener {
      private static final long serialVersionUID = 0220;
          JTextField  eingabeStartWert, ausgabeEigenWert,
ausgabeMasse, ausgabeFeder,
         eingabeP1;
         double h0, ew = 0., ratio, meff = 0., keff;
         int bc, stuetzwerte = 512;
         Color fbgColor, bgColor;
         private Curve diagram;
         JButton eigenWert;

        // Konstruktor
        public AxialeVorlast() {
          Container fenster = getContentPane();
          fbgColor = new Color(178,214,252);
            fenster.setBackground(fbgColor);
            BorderLayout h1 = new BorderLayout();
            fenster.setLayout(h1);
            JLabel p1Wert = new JLabel("NL\u00B2/2EI\u003E-19.7 :
",
                                       JLabel.RIGHT);
            eingabeP1 = new JTextField(12);

eingabeP1.setBorder(BorderFactory.createLoweredBevelBorder());
            JLabel startWert = new JLabel("Sch\u00E4tzwert [1/L] :
", JLabel.RIGHT);
            eingabeStartWert = new JTextField(12);

eingabeStartWert.setBorder(BorderFactory.createLoweredBevelBorder(
));
            bgColor = new Color(140,181,222);
            eigenWert = new JButton("Eigenwert");
            eigenWert.setBackground(bgColor);
            ausgabeEigenWert = new JTextField(12);
            ausgabeEigenWert.setBackground(fbgColor);

ausgabeEigenWert.setBorder(BorderFactory.createLoweredBevelBorder(
));
            JLabel effektiveMasse = new JLabel("effektive Masse :
", JLabel.RIGHT);
            ausgabeMasse = new JTextField(12);
            ausgabeMasse.setBackground(fbgColor);

ausgabeMasse.setBorder(BorderFactory.createLoweredBevelBorder());
```

```
            JLabel effektiveFeder = new JLabel("effektive Feder :
", JLabel.RIGHT);
            ausgabeFeder = new JTextField(12);
            ausgabeFeder.setBackground(fbgColor);

ausgabeFeder.setBorder(BorderFactory.createLoweredBevelBorder());
            JPanel h2d = new JPanel();

h2d.setBorder(BorderFactory.createLineBorder(Color.darkGray));
            h2d.setBackground(bgColor);
            h2d.setLayout(new GridLayout(5,2,5,5));
            h2d.add(p1Wert);
            h2d.add(eingabeP1);
            h2d.add(startWert);
            h2d.add(eingabeStartWert);
            h2d.add(eigenWert);
            eigenWert.addActionListener(this);
            h2d.add(ausgabeEigenWert);
            h2d.add(effektiveMasse);
            h2d.add(ausgabeMasse);
            h2d.add(effektiveFeder);
            h2d.add(ausgabeFeder);
            JLabel titel = new JLabel
                ("Axiale Vorlast N",
                 JLabel.CENTER);
            fenster.add(titel, BorderLayout.NORTH);
            fenster.add(h2d, BorderLayout.SOUTH );
            // Grafikkomponente
            diagram = new Curve();

diagram.setBorder(BorderFactory.createLoweredBevelBorder());
            fenster.add(diagram, BorderLayout.CENTER);
            // Erscheinungsbild: Nimbus
            try {
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFee
l");
               }
               catch (InstantiationException e) {
               }
               catch (ClassNotFoundException e) {
               }
               catch (UnsupportedLookAndFeelException e) {
               }
               catch (IllegalAccessException e) {
               }
               SwingUtilities.updateComponentTreeUI(fenster);
            fenster.setVisible(true);

        }

    // Initialisierung
    public static void main(String[] args) {
```

```
            int xPos,yPos;
            JFrame frame = new AxialeVorlast();
            ExitWindow abbrechen = new ExitWindow();
            frame.addWindowListener(abbrechen);
            // Abfrage Bildschirmabmessungen
            Dimension dim =
Toolkit.getDefaultToolkit().getScreenSize();
            // Abmessungen des Applikationsfensters
            frame.setSize(640,320);
            // Positionierung des Applikationsfensters auf dem
Bildschirm
            xPos = (dim.width-640)/2;
            yPos = (dim.height-320)/2;
            frame.setLocation(xPos,yPos);
            // Anzeige des Rahmenfensters auf dem Desktop
            frame.setVisible(true);
        }

        public void actionPerformed(ActionEvent event) {
            double x, g1, g2, g3, g4, g5, h1, h2, l1, l2;
            float rew, rmeff, rkeff;
            // Eigenwert
              if (event.getSource() == eigenWert) {
                x =
Double.parseDouble(eingabeStartWert.getText());
                h0 = Double.parseDouble(eingabeP1.getText());
                if(h0==0.)
                h0=0.0000001;
                h1 = h0/Math.pow(x,2.);
                h2 = Math.sqrt(h1*h1+1.);
                l1 = x*Math.sqrt(h2+h1);
                l2 = x*Math.sqrt(h1-h2);
                // Nullstellensuche: Newtonsches
Naeherungsverfahren
                double inkrement = cf(x)/cfs(x);
                while (Math.abs(inkrement/x) > 1.e-7) {
                  x = x - inkrement;
                  inkrement = cf(x)/cfs(x);
                }
                ew = x - inkrement;
                rew = Math.round(ew*10000f);
                rew = rew/10000f;
                ausgabeEigenWert.setText((float) rew + " / " +
"L");
                h1 = h0/Math.pow(ew,2.);
                h2 = Math.sqrt(h1*h1+1.);
                l1 = ew*Math.sqrt(h2+h1);
                l2 = ew*Math.sqrt(h2-h1);
                g1 = Math.exp(l1);
                g2 = Math.sin(l2);
                g3 = Math.cos(l2);
                g4 = 0.5*(g1-1./g1);
```

```
                g5 = 0.5*(g1+1./g1);
                ratio = (g5-g3)/(l2/l1*g4-g2);
                repaint();
                meff = integration(stuetzwerte);
                keff = Math.pow(ew,4.)*meff;
                rmeff = Math.round(meff*10000f);
                meff = rmeff/10000f;
                ausgabeMasse.setText((float) meff + "
\u03C1"+"AL");
                rkeff = Math.round(keff*1000f);
                keff = rkeff/1000f;
                ausgabeFeder.setText((float) keff + " EI/L" +
"\u00B3");
              }
            }

                double cf(double x) {
                  // Charakteristische Eigenwertgleichung
                  double e1, e2, e3, sh, ch, f1, f2, f3, h1, h2,
l1, l2;
                  h1 = h0/Math.pow(x,2.);
                  h2 = Math.sqrt(h1*h1+1.);
                  l1 = x*Math.sqrt(h2+h1);
                  l2 = x*Math.sqrt(h2-h1);
                  e1 = Math.exp(l1);
                  e2 = Math.sin(l2);
                  e3 = Math.cos(l2);
                  // Hyperbelfunktionen
                  sh = 0.5*(e1-1./e1);
                  ch = 0.5*(e1+1./e1);
                  f1 = e2*sh;
                  f2 = e3*ch;
                  f3 = 0.5*(l2/l1-l1/l2);
                  return f2+f3*f1-1.;
                }

                double cfs(double x) {
                  // Ableitung der Eigenwertgleichung
                  double e1, e2, e3, sh, ch, f1, f2, f3, f4, h1,
h2, l1, l2, l1s, l2s;
                  h1 = h0/Math.pow(x,2.);
                  h2 = Math.sqrt(h1*h1+1.);
                  l1 = x*Math.sqrt(h2+h1);
                  l2 = x*Math.sqrt(h2-h1);
                  e1 = Math.exp(l1);
                  e2 = Math.sin(l2);
                  e3 = Math.cos(l2);
                  // Hyperbelfunktionen
                  sh = 0.5*(e1-1./e1);
                  ch = 0.5*(e1+1./e1);
                  l1s = l1/x-0.5*(h1/h0)/l1*(2.*Math.pow(h1,2.)/
(h2*x)+3.*h1/x);
```

```
                    l2s = l2/x-0.5*(h1/h0)/l2*(2.*Math.pow(h1,2.)/
(h2*x)-3.*h1/x);
                    f1 = l1s*sh*e3-l2s*ch*e2;
                    f2 = 0.5*((l2s*l1-l1s*l2)/Math.pow(l1,2.)-
(l1s*l2-l2s*l1)/Math.pow(l2,2.));
                    f3 = f2*sh*e2;
                    f4 = 0.5*(l1s*ch*e2+l2s*sh*e3)*(l2/l1-l1/l2);
                    return (f1+f3+f4);
                  }

                double integration(int stuetzwerte) {
                    // Kinetische Energie des Biegeschwingers:
Simpsonsche Regel
                    double kinE, qesfmax = 0., fw;
                    kinE = qesf(0,ew) + qesf(stuetzwerte,ew);
                    for (int ix = 1 ; ix < stuetzwerte ; ix++) {
                         fw = qesf(ix,ew);
                         if(fw > qesfmax) {
                              qesfmax = fw;
                              }
                         if(ix%2 == 0) {
                         kinE =    kinE + 2.*fw;
                         }
                         else {
                         kinE =    kinE + 4.*fw;
                         }
                    }
                    // effektive Masse
                    return kinE/3./qesfmax/stuetzwerte;
                }

                double qesf(int i, double e) {
                    // Eigenschwingform
                    double g1, g2, g3, g4, g5, h1, h2, l1, l2;
                    double z = (double) i;
                    // Normierung Abszisse
                    z = z/stuetzwerte;
                    h1 = h0/Math.pow(e,2.);
                    h2 = Math.sqrt(h1*h1+1.);
                    l1 = e*Math.sqrt(h2+h1);
                    l2 = e*Math.sqrt(h2-h1);
                    g1 = Math.exp(l1*z);
                    g2 = Math.sin(l2*z);
                    g3 = Math.cos(l2*z);
                    // Hyperbelfunktionen
                    g4 = 0.5*(g1-1./g1);
                    g5 = 0.5*(g1+1./g1);
                    return Math.pow((g5-g3+ratio*(g2-l2/
l1*g4)),2.);
                }

        class Curve extends JComponent {
```

```
    private static final long serialVersionUID = 0220;
    double esfmax, esfmin, dif, tr;
    Color cColor;
           public void paintComponent(Graphics gc) {
              Graphics2D gc2D = (Graphics2D) gc;
              super.paintComponent(gc2D);
               // Abmessungen Grafikobjekt
               int hc = getSize().height -30;
               bc = getSize().width - 20;
               // maximale, minimale Kurvenwerte
               esfmax = esf(0, ew);
               esfmin = esfmax;
               float strichDicke = 2.0f;
               BasicStroke stroke = new BasicStroke(strichDicke);
               gc2D.setStroke(stroke);
               cColor = new Color(255,130,38);
               for (int x = 0 ; x < bc ; x++) {
                   double fr = esf(x + 1, ew);
                   if (esfmax <  fr) {
                       esfmax = fr;
                   }
                   if (esfmin > fr) {
                      esfmin = fr;
                   }
               }
               // Achsenhoehe Ordinate
               dif = esfmax - esfmin;
               // Verschiebung Null-Linie
               tr = hc * Math.abs(esfmin) / dif;
               gc2D.setColor(Color.darkGray);
               // Einspannung
               gc2D.drawLine(10, (int) tr, 10, (int) tr + 20);
               // Null-Linie
               gc2D.drawLine(10, (int) tr + 10, bc + 10, (int) tr
+10);
               gc2D.setColor(cColor);
               if(ew <= 0.) {
                   tr = hc / 2;
                       if(ew < 0.) {

ausgabeEigenWert.setBackground(Color.red);
                       }
               } else {
                   // Zeichnen der normierten Kurve
                   ausgabeEigenWert.setBackground(fbgColor);
                   for (int ix = 0 ; ix < bc ; ix++) {
                   gc2D.drawLine(ix + 10, (int) (hc * esf(ix, ew)
/ dif + tr)
                   + 10, ix + 11, (int) (hc * esf(ix + 1, ew) /
dif + tr)
                   + 10);
                   }
```

```
                }
                gc2D.setColor(Color.darkGray);
                gc2D.drawString("\u00a9" + "pwill", 15, hc + 25);
            }

            double esf(int i, double e) {
                // Eigenschwingform
                double g1, g2, g3, g4, g5, h1, h2, l1, l2;
                double z = (double) i;
                // Normierung Abszisse
                z = z/bc;
                h1 = h0/Math.pow(e,2.);
                h2 = Math.sqrt(h1*h1+1.);
                l1 = e*Math.sqrt(h2+h1);
                l2 = e*Math.sqrt(h2-h1);
                g1 = Math.exp(l1*z);
                g2 = Math.sin(l2*z);
                g3 = Math.cos(l2*z);
                // Hyperbelfunktionen
                g4 = 0.5*(g1-1./g1);
                g5 = 0.5*(g1+1./g1);
                return (g5-g3+ratio*(g2-l2/l1*g4));
            }

        }
    }

 class ExitWindow extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
        // Aufruf leerer WindowListener-Methoden
        public void windowIconified(WindowEvent we) {
        }
        public void windowOpened(WindowEvent we) {
        }
        public void windowClosed(WindowEvent we) {
        }
        public void windowDeiconified(WindowEvent we) {
        }
        public void windowActivated(WindowEvent we) {
        }
}
```