Inhalt

5 Numerische Methoden der praktischen Mathematik (III)

5.7 Zusammenfassung

1. **Eingabefehler** lassen sich in der Regel nur durch die Verbesserung der *Messtechniken verringern*, sind *nie* ganz zu vermeiden. Die Toleranz, in der diese Fehler liegen, entscheidet über die erreichbare Genauigkeit in den Ergebnissen.

2. Mathematisch exakte Verfahren können durch interne **fehlerbehaftete Datendarstellung** und dadurch bedingte **fehlerbehaftete Rechnungen** zu unexakten Ergebnissen führen. Das gilt schon bei der Addition sehr großer Zahlen mit sehr kleinen. Diese Fehler entstehen durch den Widerspruch in der „*Unendlichkeit der Mathematik*“ und der „*Endlichkeit der Rechentechnik*“. Solche **Datendarstellungs- und Rechenfehler** sind auch bei hochentwickelter Computertechnik *nicht* ausschließbar.

   Exp.java.
   Die Reihenentwicklung liefert für betragsmäßig große negative Argumente als Ergebnis negative Werte. Die Ursache liegt in der Fehlerfortpflanzung, die sich bei der Reihenentwicklung aus der immer kleiner werdenden Summe durch wechselnde Addition und Subtraktion mit fehlerbehafteten Summanden ergeben. Der Fehler konnte durch Transformation auf positive Argumente behoben werden, $e^x = \begin{cases} e^x, & x \geq 0 \\ \dfrac{1}{e^{|x|}}, & x < 0 \end{cases}$.

   Sinus.java.
   Die Reihenentwicklung liefert für betragsmäßig große Argumente nicht verwertbare Ergebnisse. *Große* `double`-Werte sind wegen der geringen Dichte ihrer Darstellung als Maschinenzahlen mit Rundungsfehlern behaftet, welche sich in der Reihenentwicklung durch Potenzieren dieser zu „*Monsterfehlern*“ entwickeln. Der Fehler konnte durch Transformation der Argumente in das dichtere Intervall behoben werden, $x \in \left[0, \frac{\pi}{2}\right]$.

3. Durch die Einschränkungen auf vorhandene Datentypen sind **Grenzen in den Rechenoperationen** gesetzt. Ergebnisse sind deshalb stets bzgl. ihrer Korrektheit abzuschätzen.

   Summe.java.

   Die Summenberechnung $s = \sum_{i=1}^{n} i$ liefert bei `long`-Zahlen für $n < 2^{32}$ exakte Ergebnisse, da die Summe *s* die größte darstellbare `long`-Zahl $2^{63} - 1$ ($\hat{=}$ **`Long.MAX_VALUE`**) nicht überschreiten darf. Bei größeren Zahlen wird die Berechnung durch Runden ungenau. Die Wahl eines anderen Elementardatentyps bringt keine Verbesserung. Die Anzahl der exakt dargestellten Ziffern ist bei jedem Zahlentypen begrenzt und liegt bei allen anderen Datentypen unterhalb der maximalen Anzahl der Ziffern einer `long`-Zahl.

4. **Näherungsverfahren** sind mathematisch unexakte Verfahren. Sie ersetzen komplizierte Verfahren durch einfache und liefern Näherungslösungen. **Verfahrensfehler** können durch Verbesserung der Verfahren eingeschränkt, aber *nicht* vollständig aufgehoben werden.

Integral.java.
Bei der bestimmten Integration werden die zu integrierenden Funktionen durch Polynome approximiert, welche sich rechentechnisch besser integrieren lassen. Eine Verbesserung der Ergebnisse konnte durch mehr Stützstellen oder einen höheren Grad des Interpolationspolynoms, also der Verbesserung des verwendeten Verfahrens, erreicht werden.

Iteration.java.
Man kann Nullstellen von Funktionen mit unterschiedlichen Näherungsverfahren berechnen. Das Newtonverfahren konvergiert sehr schnell, kann aber nur auf differenzierbare Funktionen angewandt werden. Für stetige, *nicht* differenzierbare, Funktionen bietet sich das Verfahren Ragula falsi an, welches langsamer zu einem Ergebnis führt.
Wendet man die Verfahren auf Funktionen mit mehreren Nullstellen an, so findet man u. U. *nicht* alle Nullstellen der Funktion. Die Wahl der Anfangsnäherung ist für das Ergebnis ausschlaggebend. Mathematisch Verfahren zur Bestimmung weiterer Nullstellen, wie zum Beispiel die Berechnung des Restpolynoms durch Polynomdivision, werden unbrauchbar, da die gefundenen Nullstellen in der Regel fehlerbehaftet sind und damit das Restpolynom nicht exakt ist.

5. Mathematisch elegante Verfahren sind rechentechnisch *nicht* immer die günstigsten. Rechentechnisch optimale Verfahren verbrauchen wenig Rechenzeit. Eine **Rechenzeitminimierung** erreicht man durch Minimierung der Anzahl der verwendeten Operationen.

   Polynom.java.
   Bei der Wertberechnung eines Polynoms vom Grad r ohne Hornerschema benötigt man $\frac{r^2+3r}{2}$ Additionen und Multiplikationen, das sind zum Beispiel bei $r = 20$ insgesamt 230 Operationen. Durch die Verwendung des ***Hornerschemas*** konnte dieser Rechenaufwand auf $2r$ Additionen und Multiplikationen verringert werden, bei $r = 20$ sind das nur 40 Operationen, also nicht ganz ein Sechstel.
   Die Anzahl der Operationen steigt im ersten Fall quadratisch, im zweiten Fall nur linear zur Polynompotenz.

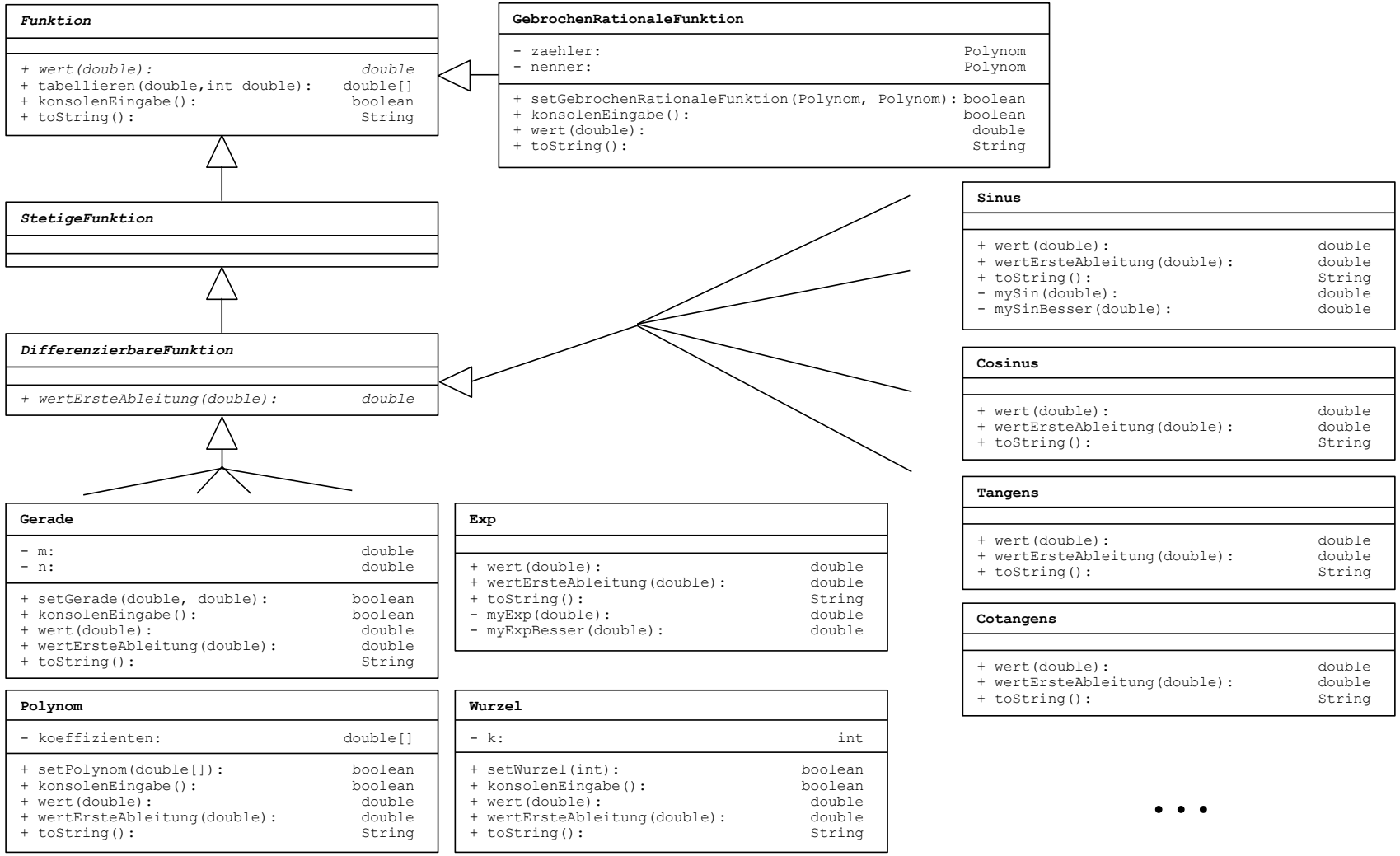
6. Die **modulare Programmierung** gestatten eine mehrfache Verwendung von bereits getesteten Programmbestandteilen, hier Klassen und ihre Methoden. In sich abgeschlossene Programmteile werden in einer Klasse zusammengefasst. Diese können getrennt vom Anwendungsprogramm ausgetestet, jederzeit wieder verwendet und leicht durch andere ausgetauscht oder auch erweitert werden.

   Funktion.java.
   Das Modul bildet die Basis aller Funktionen und legt fest, das jede Funktion ihren Wert berechnen kann. Neue Funktionen können jederzeit als Ableitung dieser Basisklasse aufgenommen werden und damit ein bereits vorhandenes Funktionssystem erweitern.

   FunktionBerechner.java FunktionTabulator.java,
   FunktionsIntegrator.java, ***NullstellenBerechner.java***.
   Anwendungsprogramme können bereits zur Verfügung stehende Funktionen einbinden, ohne diese neu zu entwickeln. Dazu reicht der *Bytecode* dieser aus.

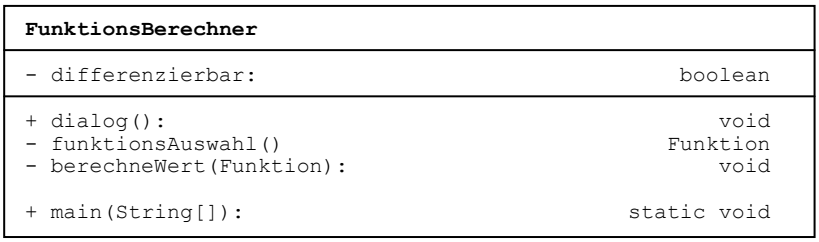
5.8 Einige Anwendungsprogramme für Funktionen

5.8.1 Funktionen im Überblick



5.8.2 Werteberechnung einer Funktion

In diesem Programm kann der Wert einer Funktion an einer Stelle x_0 und deren 1. Ableitung, falls diese differenzierbar ist, unter Verwendung der in der Vorlesung implementierten Funktionsklassen berechnet werden.



FunktionsBerechner.java

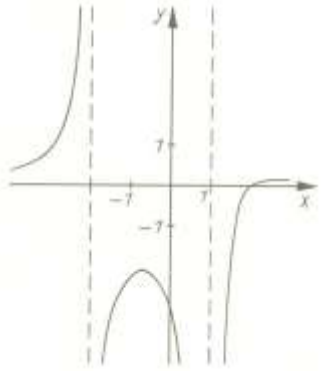
```
// FunktionsBerechner.java                                  MM 2013
import Tools.IO.*;                                       // Eingaben

/**
 * Programm zum Berechnen von Funktionswerten.
 */
public class FunktionsBerechner
{
/* ------------------------------------------------------ */
                                                        // Attribut
/**
 * Speichert Differenzierbarkeit.
 */
  private boolean differenzierbar = true;

/* ------------------------------------------------------ */
                                                          // Dialog
/**
 * Funktionsberechner:
 * Funktionseingabe, Wertberechnung.
 */
  public void dialog()
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Funktionsberechner");

// Dialog
    char weiter = 'j';
    do
    {
// Neue Funktion
      Funktion fkt = funktionsAuswahl();
      if( fkt == null) continue;
      if( fkt.konsolenEingabe())         // Eingabe korrekt
```

```
      {
// Funktionsausgabe
        System.out.println();
        System.out.println( " f( x) = " + fkt);

        do
        {
// Werteberechnug
          System.out.println();
          berechneWert( fkt);

// Weiter
          System.out.println();
          weiter = IOTools.readChar( "Neuer Wert (j/n)? ");
        } while( weiter == 'j');
      }
      weiter = IOTools.readChar( "Neue Funktion (j/n)? ");
    } while( weiter == 'j');

// Programm beendet
    System.out.println();
    System.out.println( "Programm beendet");
  }

/* ------------------------------------------------- */
                                                  // Funktion
/**
 * Funktionsauswahl.
 * @return Funktion
 */
  private Funktion funktionsAuswahl()
  {
// Menue aller Funktionen
    System.out.println( "Funktionsauswahl");
    System.out.println
    ( "  Gerade                    ...  1");
    System.out.println
    ( "  Polynom                   ...  2");
    System.out.println
    ( "  Wurzel-Funktion           ...  3");
    System.out.println
    ( "  e^x - Funktion            ...  4");
    System.out.println
    ( "  a^x-Funktion              ...  5");
    System.out.println
    ( "  sin-Funktion              ...  6");
    System.out.println
    ( "  cos-Funktion              ...  7");
    System.out.println
    ( "  tan-Funktion              ...  8");
    System.out.println
    ( "  cot-Funktion              ...  9");
```

```
      System.out.println
      ( "  arctan-Funktion             ... 10");
      System.out.println
      ( "  ln-Funktion                 ... 11");
      System.out.println
      ( "  log-Funktion                ... 12");
      System.out.print
      ( "  gebrochenrationale Funktion ... 13");

      int eingabe = IOTools.readInteger(": ");

  // Festlegen der Funktion
      Funktion fkt = null;
      differenzierbar = true;

      switch( eingabe)
      {
        case 1: fkt = new Gerade(); break;
        case 2: fkt = new Polynom( ); break;
        case 3: fkt = new Wurzel(); break;
        case 4: fkt = new Exp(); break;
        case 5: fkt = new Power(); break;
        case 6: fkt = new Sinus(); break;
        case 7: fkt = new Cosinus( ); break;
        case 8: fkt = new Tangens(); break;
        case 9: fkt = new Cotangens(); break;
        case 10: fkt = new Arctan(); break;
        case 11: fkt = new Ln(); break;
        case 12: fkt = new LogB(); break;
        case 13: differenzierbar = false;
                 fkt = new RationaleFunktion(); break;
        default: System.out.println();
          System.out.println( "Fehlerhafte Eingabe!");
      }

      return fkt;
    }

  /* ------------------------------------------------------ */
                                                  // Berechnen
  /**
   * Berechnen einer Funktion und deren 1. Ableitung
   * an einer Stlle x0.
   * @param fkt Funktion
   */
    private void berechneWert( Funktion fkt)
    {
  // Argumenteingabe
      System.out.println( "Wertberechnung");
      double x0 = IOTools.readDouble( " x0 = ");
      System.out.println( x0);
```

```
// Wertberechnung
    double y0 = fkt.wert( x0);

// Ausgabe
    System.out.print( " f( " + x0 + ") = " + y0);

// Wertberechnung 1. Ableitung
    if( differenzierbar)
    {
      DifferenzierbareFunktion dfkt
      = (DifferenzierbareFunktion)fkt;
      double y1 = dfkt.wertErsteAbleitung( x0);
      System.out.println( "\t\tf'( " + x0 + ") = " + y1);
    }
  }

/* ------------------------------------------------------ */
                                                  // Programm
/**
 * Hauptprogramm, startet Funktionsberechnung.
 */
  public static void main( String[] args)
  {
// Berechner erzeugen
    FunktionsBerechner berechner = new FunktionsBerechner();

// Berechner starten
    berechner.dialog();
  }
}

/* ------------------------------------------------------ */
                                              // Testbeispiele
/*
f( x) =
-9.0 + 3.0 x^1 + -5.0 x^2 + 0.0 x^3 + -1.0 x^4 + 2.0 x^5
f( 3.0) = 360.0        f'( 3.0) = 675.0

f( x) =
-1.0 + 1.0 x^1 + 1.0 x^2 + 3.0 x^3 + 0.0 x^4 +
-2.0 x^5 + 1.0 x^6
f( -1.0) = -1.0        f'( -1.0) = -8.0
f( 1.0) = 3.0          f'( 1.0) = 8.0

f( x) =
( -6.0 + 3.0 x^1 ) / ( 2.0 + -3.0 x^1 + 0.0 x^2 + 1.0 x^3 )
f( -3.0) = 0.9375
f( -2.0) = -Infinity
f( -1.0) = -2.25
f( 0.0) = -3.0
f( 1.0) = -Infinity
f( 2.0) = 0.0
```

```
f( 3.0) = 0.15
*/
```



Beispiel $f(x)=(3x-6)/(x^3-3x+2),\ x\in[-3,3]$:

5.8.3 Wertetabelle einer Funktion

Tabelliert werden Funktionen unter Verwendung der in der Vorlesung implementierten Funktionsklassen.

FunktionsTabulator	
+ dialog():	void
- funktionsAuswahl()	Funktion
- erzeugeTabelle(Funktion):	void
+ main(String[]):	static void

FunktionsTabulator.java

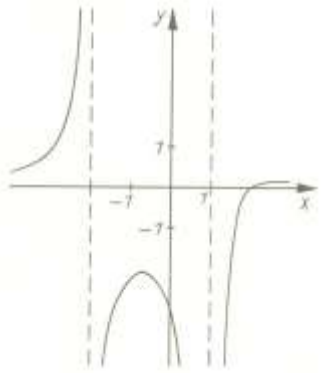
```
// FunktionsTabulator.java                                    MM 2013
import Tools.IO.*;                                          // Eingaben

/**
 * Programm zum Tabellieren von Funktionen.
 */
public class FunktionsTabulator
{
/* ------------------------------------------------------- */
                                                          // Dialog
/**
 * Funktionstabulator:
 * Funktionseingabe, Wertetabelle.
 */
  public void dialog()
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Funktionstabulator");

// Dialog
    char weiter = 'j';
    do
    {
// Neue Funktion
      Funktion fkt = funktionsAuswahl();
      if( fkt == null) continue;
      if( fkt.konsolenEingabe())          // Eingabe korrekt
```

```
      {
// Funktionsausgabe
        System.out.println();
        System.out.println( "f( x) = " + fkt);

        do
        {
// Wertetabelle
          System.out.println();
          erzeugeTabelle( fkt);

// Weiter
          System.out.println();
          weiter = IOTools.readChar( "Neue Tabelle (j/n)? ");
        } while( weiter == 'j');
      }
      weiter = IOTools.readChar( "Neue Funktion (j/n)? ");
    } while( weiter == 'j');

// Programm beendet
    System.out.println();
    System.out.println( "Programm beendet");
  }

/* ------------------------------------------------- */
                                                  // Funktion
/**
 * Funktionsauswahl.
 * @return Funktion
 */
  private Funktion funktionsAuswahl()
  {
// Menue aller Funktionen
    System.out.println( "Funktionsauswahl");
    System.out.println
    ( "  Gerade                    ...  1");
    System.out.println
    ( "  Polynom                   ...  2");
    System.out.println
    ( "  Wurzel-Funktion           ...  3");
    System.out.println
    ( "  e^x - Funktion            ...  4");
    System.out.println
    ( "  a^x-Funktion              ...  5");
    System.out.println
    ( "  sin-Funktion              ...  6");
    System.out.println
    ( "  cos-Funktion              ...  7");
    System.out.println
    ( "  tan-Funktion              ...  8");
    System.out.println
    ( "  cot-Funktion              ...  9");
```

```
      System.out.println
      ( "  arctan-Funktion             ... 10");
      System.out.println
      ( "  ln-Funktion                 ... 11");
      System.out.println
      ( "  log-Funktion                ... 12");
      System.out.print
      ( "  gebrochenrationale Funktion ... 13");

      int eingabe = IOTools.readInteger(": ");

  // Festlegen der Funktion
      Funktion fkt = null;
      switch( eingabe)
      {
        case 1: fkt = new Gerade(); break;
        case 2: fkt = new Polynom( ); break;
        case 3: fkt = new Wurzel(); break;
        case 4: fkt = new Exp(); break;
        case 5: fkt = new Power(); break;
        case 6: fkt = new Sinus(); break;
        case 7: fkt = new Cosinus( ); break;
        case 8: fkt = new Tangens(); break;
        case 9: fkt = new Cotangens(); break;
        case 10: fkt = new Arctan(); break;
        case 11: fkt = new Ln(); break;
        case 12: fkt = new LogB(); break;
        case 13: fkt = new RationaleFunktion(); break;
        default: System.out.println();
          System.out.println( "Fehlerhafte Eingabe!");
      }

      return fkt;
    }

  /* ------------------------------------------------------ */
                                                  // Tabellieren
  /**
   * Erzeugen und Ausgeben einer Wetetabelle.
   * @param fkt Funktion
   */
    private void erzeugeTabelle( Funktion fkt)
    {
  // Eingabe der Tabellenparameter
      System.out.println( "Wertbereich");

      double x0 = IOTools.readDouble( " Startargument x0 = ");
      System.out.println( x0);

      int n = 1;
      do
      {
```

```
      n = IOTools.readInteger( " Anzahl n (n > 0) = ");
    } while( n < 1);
    System.out.println( n);

    double h = IOTools.readDouble( " Schrittweite h = ");
    System.out.println( h);

// Wertetabelle
    double[] tabelle = fkt.tabellieren( x0, n, h);

// Tabellenausgabe
    double x = x0;
    for( int i = 0; i < tabelle.length; i++)
    {
      System.out.println
      ( " f( " + x + ")\t= " + tabelle[ i]);
      x += h;
    }
  }

/* ------------------------------------------------------ */
                                                  // Programm
/**
 * Hauptprogramm, startet Tabellieren von Funktionen.
 */
  public static void main( String[] args)
  {
// Tabulator erzeugen
    FunktionsTabulator tabulator = new FunktionsTabulator();

// Tabulator starten
    tabulator.dialog();
  }
}

/* ------------------------------------------------------ */
                                                // Testbeispiel
/*
f( x) =
-9.0 + 3.0 x^1 + -5.0 x^2 + 0.0 x^3 + -1.0 x^4 + 2.0 x^5
  Startargument x0 = 0.0
  Anzahl n (n > 0) = 11
  Schrittweite h   = 1.0
  f( 0.0)    = -9.0
  f( 1.0)    = -10.0
  f( 2.0)    = 25.0
  f( 3.0)    = 360.0
  f( 4.0)    = 1715.0
  f( 5.0)    = 5506.0
  f( 6.0)    = 14085.0
  f( 7.0)    = 30980.0
  f( 8.0)    = 61135.0
```

```
  f( 9.0)    = 111150.0

f( x) =
-1.0 + 1.0 x^1 + 1.0 x^2 + 3.0 x^3 + 0.0 x^4 +
-2.0 x^5 + 1.0 x^6
  Startargument x0 = -1.0
  Anzahl n (n > 0) = 5
  Schrittweite h   = 0.5
  f( -1.0)   = -1.0
  f( -0.5)   = -1.546875
  f( 0.0)    = -1.0
  f( 0.5)    = 0.078125
  f( 1.0)    = 3.0

f( x) =
( -6.0 + 3.0 x^1 ) / ( 2.0 + -3.0 x^1 + 0.0 x^2 + 1.0 x^3 )
  Startargument x0 = -3.0
  Anzahl n (n > 0) = 7
  Schrittweite h   = 1.0
  f( -3.0)   = 0.9375
  f( -2.0)   = -Infinity
  f( -1.0)   = -2.25
  f( 0.0)    = -3.0
  f( 1.0)    = -Infinity
  f( 2.0)    = 0.0
  f( 3.0)    = 0.15
*/
```



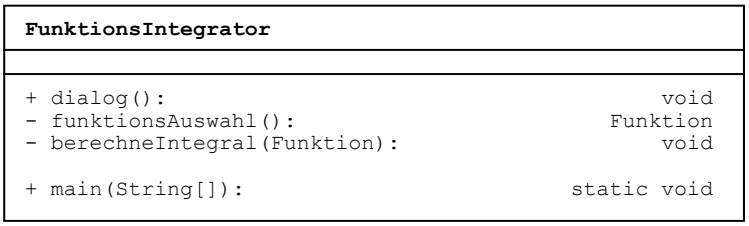
Beispiel $f(x) = (3x-6)/(x^3-3x+2),\ x \in [-3,3]$:

5.8.4 Integration von Funktionen

In der Vorlesung wurden zwei Integrationsverfahren behandelt. Beide Verfahren approximieren die zu integrierende Funktion als Polynom. Im Trapezverfahren wird ein lineares und im Simpsonverfahren ein quadratisches Polynom integriert.

Integral	
+ trapez(double[], double):	double
+ trapez(Funktion, double, double, int):	double
+ simpson(double[], double):	double
+ simpson(Funktion, double, double, int):	double

Das Programm integriert Funktionen unter Verwendung der in der Vorlesung implementierten Funktionsklassen wahlweise mittels der behandelten Integrationsverfahren.



FunktionsIntegrator.java

```
// FunktionsIntegrator.java                                MM 2013
import Tools.IO.*;                                    // Eingaben

/**
 * Programm zur Integrieren von Funktionen.
 */
public class FunktionsIntegrator
{
/* ------------------------------------------------------ */
                                                      // Dialog
/**
 * Funktionsberechner:
 * Funktionseingabe, Wertberechnung.
 */
  public void dialog()
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Funktionsintegrator");

// Dialog
    char weiter = 'j';
    do
    {
// Neue Funktion
      Funktion fkt = funktionsAuswahl();
      if( fkt == null) continue;
      if( fkt.konsolenEingabe())         // Eingabe korrekt
      {
// Funktionsausgabe
        System.out.println();
        System.out.println( "Integral von " +  fkt);

        do
        {
// Integralberechnug
          System.out.println();
          berechneIntegral( fkt);

// Weiter
          System.out.println();
          weiter
```

```
            = IOTools.readChar( "Neues Integral (j/n)? ");
          } while( weiter == 'j');
        }
        weiter = IOTools.readChar( "Neue Funktion (j/n)? ");
      } while( weiter == 'j');

  // Programm beendet
      System.out.println();
      System.out.println( "Programm beendet");
    }

  /* ------------------------------------------------------ */
                                                    // Funktion
  /**
   * Funktionsauswahl.
   * @return Funktion
   */
    private Funktion funktionsAuswahl()
    {
  // Menue aller Funktionen
      System.out.println( "Funktionsauswahl");
      System.out.println
      ( "  Gerade                    ...  1");
      System.out.println
      ( "  Polynom                   ...  2");
      System.out.println
      ( "  Wurzel-Funktion           ...  3");
      System.out.println
      ( "  e^x - Funktion            ...  4");
      System.out.println
      ( "  a^x-Funktion              ...  5");
      System.out.println
      ( "  sin-Funktion              ...  6");
      System.out.println
      ( "  cos-Funktion              ...  7");
      System.out.println
      ( "  tan-Funktion              ...  8");
      System.out.println
      ( "  cot-Funktion              ...  9");
      System.out.println
      ( "  arctan-Funktion           ... 10");
      System.out.println
      ( "  ln-Funktion               ... 11");
      System.out.print
      ( "  log-Funktion              ... 12");

      int eingabe = IOTools.readInteger(": ");

  // Festlegen der Funktion
      Funktion fkt = null;
      switch( eingabe)
      {
```

```
        case 1: fkt = new Gerade(); break;
        case 2: fkt = new Polynom( ); break;
        case 3: fkt = new Wurzel(); break;
        case 4: fkt = new Exp(); break;
        case 5: fkt = new Power(); break;
        case 6: fkt = new Sinus(); break;
        case 7: fkt = new Cosinus( ); break;
        case 8: fkt = new Tangens(); break;
        case 9: fkt = new Cotangens(); break;
        case 10: fkt = new Arctan(); break;
        case 11: fkt = new Ln(); break;
        case 12: fkt = new LogB(); break;
        default: System.out.println();
          System.out.println( "Fehlerhafte Eingabe!");
      }

      return fkt;
    }

  /* ------------------------------------------------- */
                                              // Integrieren
  /**
   * Integralberechnung.
   * @param fkt Funktion
   */
    private void berechneIntegral( Funktion fkt)
    {
  // Eingabe der Integralparameter
      double a, b;
      do
      {
        a = IOTools.readDouble( " Untere Grenze a = ");
        System.out.println( a);
        b = IOTools.readDouble( " Obere  Grenze b = ");
        System.out.println( b);
      } while( a >= b);

  // Eingaben der Stuetzstellen
      int s;
      do
      {
        System.out.print( " Anzahl der Stuetzstellen s > 1, ");
        s = IOTools.readInteger( "s = ");
        System.out.println( s);
      } while( s < 2);

  // Integral
      Integral integral = new Integral();

  // Trapez Integralberechnung
      double ergTrapez = integral.trapez( fkt, a, b, s);
      System.out.println( " Trapez:  F = " + ergTrapez);
```

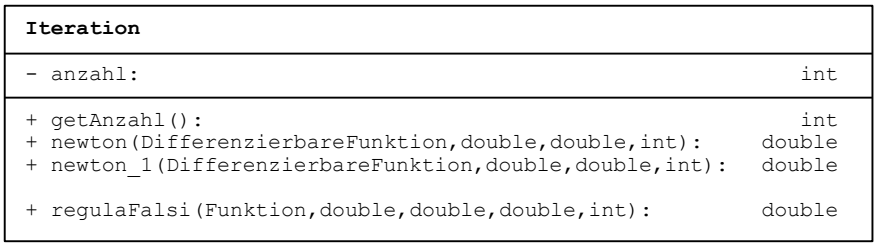
```
  // Simpson Integralberechnung
      if( s % 2 != 0)
      {
        double ergSimpson = integral.simpson( fkt, a, b, s);
        System.out.println( " Simpson: F = " + ergSimpson);
      }
    }

  /* ------------------------------------------------- */
                                                  // Programm
  /**
   * Hauptprogramm, startet Integration von Funktionen.
   */
    public static void main( String[] args)
    {
  // Integrator erzeugen
      FunktionsIntegrator integrator
      = new FunktionsIntegrator();

  // Integrator starten
      integrator.dialog();
    }
  }

  /* ------------------------    ------------------------- */
                                                 // Testwerte
/*
Integral von
-1.0 + 1.0 x^1 + 1.0 x^2 + 3.0 x^3 + 0.0 x^4 +
-2.0 x^5 + 1.0 x^6

 Untere Grenze a = -2.0
 Obere  Grenze b = 2.0
 Anzahl der Stuetzstellen s > 1, s = 3
 Trapez:  F = 132.0
 Simpson: F = 86.66666666666666

 Untere Grenze a = -2.0
 Obere  Grenze b = 2.0
 Anzahl der Stuetzstellen s > 1, s = 9
 Trapez:  F = 45.90625
 Simpson: F = 38.541666666666664

 Untere Grenze a = -2.0
 Obere  Grenze b = 2.0
 Anzahl der Stuetzstellen s > 1, s = 15
 Trapez:  F = 40.55371000664203
 Simpson: F = 37.974807225204586

 Untere Grenze a = -2.0
 Obere  Grenze b = 2.0
```

```
 Anzahl der Stuetzstellen s > 1, s = 21
 Trapez:  F = 39.207167999999996
 Simpson: F = 37.92170666666665

Mathe: F = 37.904761904761904761904761904762
*/
```

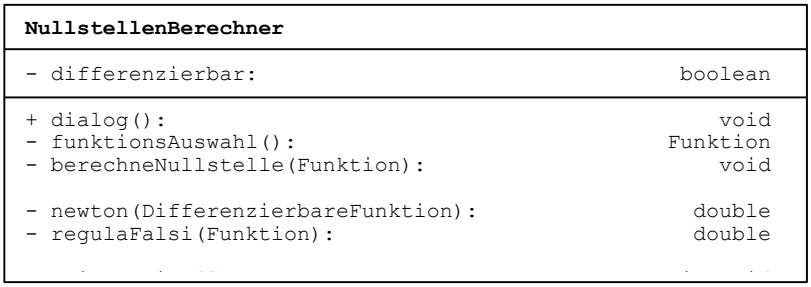
Beispiel: $\int_{-2}^{2}\left(x^6-2x^5+3x^3+x^2+x-1\right)dx = 37.\overline{904761}$

5.8.5 Nullstellenbestimmung von Funktionen

In der Vorlesung wurden zwei Verfahren zur Nullstellenbestimmung von Funktionen behandelt. Beide Verfahren sind Iterationsverfahren, die zu einer bereits vorhandenen Nullstellennäherung eine bessere sucht. Das Newtonverfahren konvertiert schneller, verlangt aber eine differenzierbare Funktion. Das Verfahren Ragula falsi verarbeitet stetige Funktionen.



Das Programm bestimmt Nullstellennäherungen rationaler Funktionen. Polynomen werden unter Verwendung des Newtonverfahrens und gebrochen rationale Funktionen unter Verwendung von Regula falsi berechnet.



NullstellenBerechner.java

```
// NullstellenBerechner.java                          MM 2013
import Tools.IO.*;                                 // Eingaben

/**
 * Programm zur Nullstellenberechnung von Funktionen.
 */
```

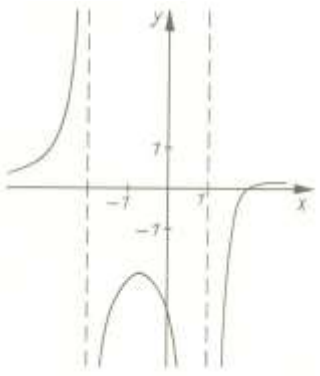
```
public class NullstellenBerechner
{
/* ------------------------------------------------------ */
                                                  // Attribut
/**
 * Speichert Differenzierbarkeit.
 */
  private boolean differenzierbar = true;

/* ------------------------------------------------------ */
                                                    // Dialog
/**
 * Nullstellenberechner:
 * Funktionseingabe, Nullstellenberechnung.
 */
  public void dialog()
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Nullstellenberechner");

// Dialog
    char weiter = 'j';
    do
    {
// Neue Funktion
      Funktion fkt = funktionsAuswahl();
      if( fkt == null) continue;
      if( fkt.konsolenEingabe())         // Eingabe korrekt
      {
// Funktionsausgabe
        System.out.println( "\nNullstelle von " +  fkt);

        do
        {
// Nullstellenberechnung
          System.out.println();
          berechneNullstelle( fkt);

// Weiter
          System.out.println();
          weiter
          = IOTools.readChar( "Neue Nullstelle (j/n)? ");
        } while( weiter == 'j');
      }
      weiter = IOTools.readChar( "Neue Funktion (j/n)? ");
    } while( weiter == 'j');

// Programm beendet
    System.out.println();
    System.out.println( "Programm beendet");
  }
```

```
/* ------------------------------------------------------ */
                                          // Funktionseingabe
/**
 * Funktionsauswahl.
 * @return Funktion
 */
  private Funktion funktionsAuswahl()
  {
// Menue aller Funktionen
    System.out.println( "Funktionsauswahl");
    System.out.println
    ( "  Gerade                        ... 1");
    System.out.println
    ( "  Polynom                       ... 2");
    System.out.print
    ( "  gebrochenrationale Funktion ... 3");

    int eingabe = IOTools.readInteger(": ");

// Festlegen der Funktion
    Funktion fkt = null;
    switch( eingabe)
    {
      case 1: fkt =  new Gerade();
              differenzierbar = true;
              break;
      case 2: fkt =  new Polynom( );
              differenzierbar = true;
              break;
      case 3: fkt = new RationaleFunktion();
              differenzierbar = false;
              break;
      default: System.out.println();
        System.out.println( "Fehlerhafte Eingabe!");
    }
    return fkt;
  }

/* ------------------------------------------------------ */
                                                // Iteration
/**
 * Verfahrensauswahl.
 * @param fkt differenzierbare Funktion
 */
  private void berechneNullstelle( Funktion fkt)
  {
// Berechnen der Nullstelle durch Newton
    if( differenzierbar)
    {
      DifferenzierbareFunktion dfkt
      = (DifferenzierbareFunktion)fkt;
```

```
      System.out.println
      ( " Gefundene Nullstelle: " + newton( dfkt));

    }
// Berechnen des Integrals durch Simpsonregel
    else
    {
      System.out.println
      ( " Gefundene Nullstelle: " + regulaFalsi( fkt));
    }

    System.out.println();
  }

/**
 * Berechnet Nullstelle nach Newton.
 * @param fkt DifferenzierbareFunktion
 * @return Nullstellennaeherung
 */
 private double newton( DifferenzierbareFunktion fkt)
 {
   System.out.println( "Newton:");
   System.out.println( "Eingabe der Iterationswerte");

   double x0 = IOTools.readDouble( " Startwert x0 = ");
   System.out.println( x0);

   double eps = IOTools.readDouble( " Genauigkeit eps = ");
   System.out.println( eps);

   int max;
   do
   {
     max = IOTools.readInteger
           ( " maximale Durchlaeufe > 0, max = ");
   } while( max < 1);
   System.out.println( max);

   Iteration it = new Iteration();
   return it.newton( fkt, x0, eps, max);
  }

/**
 * Berechnet Nullstelle nach Regula Falsi.
 * @param fkt Funktion
 * @return Nullstellennaeherung
 */
 private double regulaFalsi( Funktion fkt)
 {
   System.out.println( "Regula falsi:");
   System.out.println( "Eingabe der Iterationswerte");
```

```
    double x0 = 0, x1 = 0;
    do
    {
      x0 = IOTools.readDouble
           ( " 1. Naeherungswert, f(x0)>0,  x0 = ");
      System.out.println( x0);
      x1 = IOTools.readDouble
           ( " 2. Naeherungswert, f(x1)<0,  x1 = ");
      System.out.println( x1);
     } while( Math.signum( fkt.wert( x0))
              == Math.signum( fkt.wert( x1)));

    double eps = IOTools.readDouble( " Genauigkeit eps = ");
    System.out.println( eps);

    int max;
    do
    {
      max = IOTools.readInteger
           ( " maximale Durchlaeufe > 0, max = ");
    } while( max < 1);
    System.out.println( max);

    Iteration it = new Iteration();
    return  it.regulaFalsi( fkt, x0, x1, eps, max);
  }

/* ------------------------------------------------------ */
                                                  // Programm
/**
 * Hauptprogramm, startet Nullstellenberechnung.
 */
  public static void main( String[] args)
  {
// Berechner erzeugen
    NullstellenBerechner berechner
    = new NullstellenIterator();

// Berechner starten
    berechner.dialog();
  }
}

/* ------------------------------------------------------ */
                                                // Testwerte
/*
Nullstellenberechner
Nullstelle von 24.0 + -2.0 x^1 + -5.0 x^2 + 1.0 x^3

Newton:
 Startwert x0 = -3.0
```

```
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 100
 Gefundene Nullstelle: -2.0                              // -2

Newton:
 Startwert x0 = 0.0
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 100
 Gefundene Nullstelle: 3.9999999999999996                // 4

Newton:
 Startwert x0 = 1.0
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 100
 Gefundene Nullstelle: 3.0                               // 3

Nullstelle von
( -6.0 + 3.0 x^1 ) / ( 2.0 + -3.0 x^1 + 0.0 x^2 + 1.0 x^3 )

Regula falsi:
 1. Naeherungswert, f(x0) > 0, x0 = 1.5
 2. Naeherungswert, f(x1) < 0, x1 = 2.5
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 1000
 Gefundene Nullstelle: 2.0000000000000013                // 2


Neue Nullstelle (j/n)?
Regula falsi:
Eingabe der Iterationswerte
 1. Naeherungswert, f(x0) > 0, x0 = -1.0
 2. Naeherungswert, f(x1) < 0, x1 = 2.5
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 1000
 Gefundene Nullstelle: 2.0                               // 2


Neue Nullstelle (j/n)?
Regula falsi:
Eingabe der Iterationswerte
 1. Naeherungswert, f(x0) > 0, x0 = -2.5
 2. Naeherungswert, f(x1) < 0, x1 = 1.5
 Genauigkeit eps = 1.0E-15
 maximale Durchlaeufe > 0, max = 1000
 Gefundene Nullstelle: NaN
*/
```



Beispiel $f(x) = (3x-6)/(x^3-3x+2),\ \bar{x} \in 2$: