Lösung der Übungsaufgabenserie 11
Grundlagen der Informatik und Numerik

---

1. ***PolynomBerechner.java***

PolynomBerechner.java

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

/**
 * Berechnet Polynome.
 */
public class PolynomBerechner
{
/**
 * Polynomberechner:
 * Polynomeingabe, Wertberechnung, Ergebnisausgabe.
 */
  public static void main( String[] args)
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Polynomberechner");

// Dialog
    char weiter = 'j';
    do
    {
// Neues Polynom
      System.out.println();
      Polynom p = new Polynom();
      if( p.konsolenEingabe())           // Eingabe korrekt
      {
// Polynomausgabe
        System.out.println();
        System.out.println( "p( x) = " + p);

        do
        {
// Argumenteingabe
          System.out.println();
          System.out.println( "Wertberechnung");
          double x0 = IOTools.readDouble( " x0 = ");
          System.out.println( " " + x0);

// Wertberechnung
          double y0 = p.wert( x0);
          double y1 = p.wertErsteAbleitung( x0);

// Ausgabe
          System.out.print( " p( " + x0 + ")  = " + y0);
          System.out.println
          ( "\t\tp'( " + x0 + ") = " + y1);

          System.out.println();

// Weiter
          weiter = IOTools.readChar( "Neues Argument (j/n)? ");
        } while( weiter == 'j');
      }

      weiter = IOTools.readChar( "Neues Polynom (j/n)? ");
```

```
      } while( weiter == 'j');

  // Programm beendet
      System.out.println();
      System.out.println( "Programm beendet");
    }
  }
  /* ------------------------------------------------------ */
                                              // Testbeispiele
  /*
  p( x) =
  -9.0 + 3.0 x^1 + -5.0 x^2 + 0.0 x^3 + -1.0 x^4 + 2.0 x^5
  Wertberechnung
   x0 =  3.0
   p( 3.0)  = 360.0          p'( 3.0) = 675.0

  p( 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
  Wertberechnung
   x0 =  -1.0
   p( -1.0)  = -1.0          p'( -1.0) = -8.0

   x0 =  1.0
   p( 1.0)  = 3.0                 p'( 1.0) = 8.0

  */
```

2. ***RationaleFunktionenTabulator.java***

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

/**
 * Tabelliert rationale Funktionen.
 */
public class RationaleFunktionenTabulator
{
/**
 * Funktionstabulator:
 * Funktionseingabe, Werttabelle, Ergebnisausgabe.
 */
  public static void main( String[] args)
  {
// Ueberschrift
    System.out.println();
    System.out.println( "Funktionstabulator");

// Dialog
    char weiter = 'j';
    do
    {
// Neue Funktion
      System.out.println();
      RationaleFunktion f = new RationaleFunktion();
      if( f.konsolenEingabe())
      {
// Funktionsausgabe
        System.out.println();
        System.out.println( "f( x) = " + f);

        do
        {
// Eingabe der Tabellenparameter
```

```
          System.out.println();
          System.out.println( "Wertbereich");

          double x0
          = IOTools.readDouble( " Startargument x0 = ");
          int n
          = IOTools.readInteger( " Anzahl n (n > 0) = ");
          double h = 0;
          if( n > 1)
          h = IOTools.readDouble( " Schrittweite h   = ");

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

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

// Weiter
          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");
  }
}

/* ------------------------------------------------------ */
                                          // Testbeispiele
/*
f( x) = ( 1.0 ) / ( 1.0 + 1.0 x^1 )
Wertbereich
 Startargument x0 = -2
 Anzahl n (n > 0) = 9
 Schrittweite h   = 0.25
 f( -2.0)              = -1.0
 f( -1.75)             = -1.3333333333333333
 f( -1.5)              = -2.0
 f( -1.25)             = -4.0
 f( -1.0)              = Infinity
 f( -0.75)             = 4.0
 f( -0.5)              = 2.0
 f( -0.25)             = 1.3333333333333333
 f( 0.0)               = 1.0
```

Funktion (i)

```
f( x) =
( 8.0 + 4.0 x^1 + -10.0 x^2 + -5.0 x^3 + 2.0 x^4 + 1.0 x^5 )
/ ( 1.0 )
Wertbereich
 Startargument x0 = -2.5
 Anzahl n (n > 0) = 11
```

```
 Schrittweite h   = 0.5
 f( -2.5)               = -5.90625
 f( -2.0)               = 0.0
 f( -1.5)               = -1.09375
 f( -1.0)               = 0.0
 f( -0.5)               = 4.21875
 f( 0.0)                = 8.0
 f( 0.5)                = 7.03125
 f( 1.0)                = 0.0
 f( 1.5)                = -7.65625
 f( 2.0)                = 0.0
 f( 2.5)                = 53.15625
```

Funktion (ii)

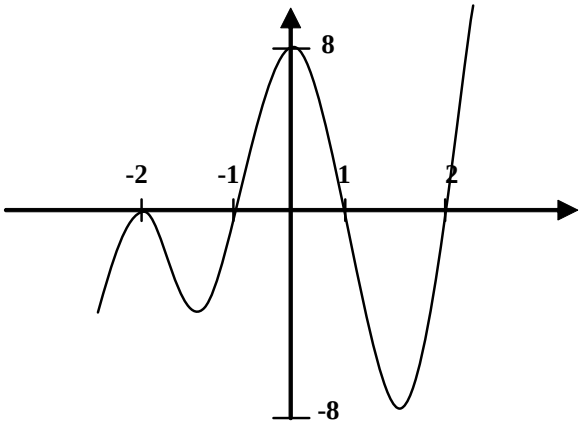
```
f( x) = ( -1.0 + 0.0 x^1 + 1.0 x^2 ) / ( 1.0 + 0.0 x^1 + 1.0 x^2 )
Wertbereich
 Startargument x0 = -4
 Anzahl n (n > 0) = 17
 Schrittweite h   = 0.5
 f( -4.0)               = 0.8823529411764706
 f( -3.5)               = 0.8490566037735849
 f( -3.0)               = 0.8
 f( -2.5)               = 0.7241379310344828
 f( -2.0)               = 0.6
 f( -1.5)               = 0.38461538461538464
 f( -1.0)               = 0.0
 f( -0.5)               = -0.6
 f( 0.0)                = -1.0
 f( 0.5)                = -0.6
 f( 1.0)                = 0.0
 f( 1.5)                = 0.38461538461538464
 f( 2.0)                = 0.6
 f( 2.5)                = 0.7241379310344828
 f( 3.0)                = 0.8
 f( 3.5)                = 0.8490566037735849
 f( 4.0)                = 0.8823529411764706
```

Funktion (iii)

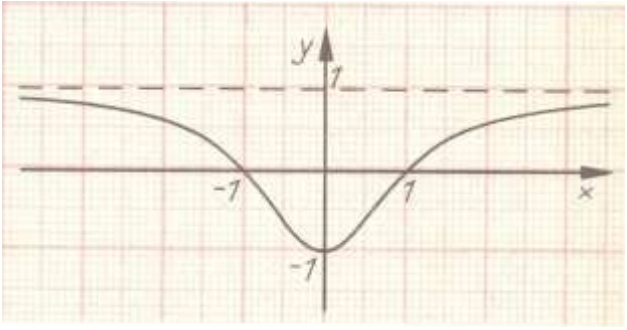
```
f( x) = ( -6.0 + 3.0 x^1 ) / ( 2.0 + -3.0 x^1 + 0.0 x^2 + 1.0 x^3 )
Wertbereich
 Startargument x0 = -2.5
 Anzahl n (n > 0) = 51
 Schrittweite h   = 0.1
 f( -2.5)               = 2.204081632653061
 f( -2.4)               = 2.8546712802768166
 f( -2.3)               = 3.9485766758494054
 f( -2.1999999999999997)      = 6.152343750000008
 f( -2.0999999999999996)      = 12.799167533818993
 f( -1.9999999999999996)      = -3.00239975158033E15
 f( -1.8999999999999995)      = -13.91200951248507
 f( -1.7999999999999994)      = -7.270408163265285
 f( -1.6999999999999993)      = -5.075445816186546
 f( -1.5999999999999992)      = -3.99408284023668
 f( -1.4999999999999991)      = -3.359999999999995
 f( -1.399999999999999) = -2.9513888888888857
 f( -1.299999999999999) = -2.673507966513635
 f( -1.1999999999999988)      = -2.4793388429752046
 f( -1.0999999999999988)      = -2.343159486016628
 f( -0.9999999999999988)      = -2.249999999999999
 f( -0.8999999999999988)      = -2.1908839083354312
 f( -0.7999999999999988)      = -2.1604938271604937
 f( -0.6999999999999988)      = -2.1559755123768967
 f( -0.5999999999999989)      = -2.176339285714286
```

```
 f( -0.4999999999999989)      = -2.2222222222222228
 f( -0.3999999999999989)      = -2.2959183673469394
 f( -0.2999999999999893)      = -2.401670727462584
 f( -0.1999999999999893)      = -2.546296296296298
 f( -0.0999999999999892)      = -2.7403218790778623
 f( 1.0824674490095276E-15)   = -3.000000000000003
 f( 0.10000000000000109)      = -3.350970017636689
 f( 0.2000000000000011) = -3.835227272727279
 f( 0.3000000000000011) = -4.525288376220062
 f( 0.40000000000000113)      = -5.5555555555555705
 f( 0.5000000000000011) = -7.200000000000025
 f( 0.6000000000000011) = -10.096153846153888
 f( 0.7000000000000011) = -16.04938271604948
 f( 0.800000000000001)  = -32.14285714285747
 f( 0.900000000000001)  = -113.79310344827785
 f( 1.000000000000001)  = -Infinity
 f( 1.1000000000000012) = -87.09677419354601
 f( 1.2000000000000013) = -18.749999999999723
 f( 1.3000000000000014) = -7.07070707070699
 f( 1.4000000000000015) = -3.3088235294117303
 f( 1.5000000000000016) = -1.714285714285697
 f( 1.6000000000000016) = -0.9259259259259164
 f( 1.7000000000000017) = -0.49641478212906237
 f( 1.8000000000000018) = -0.24671052631578594
 f( 1.900000000000002)  = -0.0949667616334261
 f( 2.0000000000000018) = 1.3322676295501825E-15
 f( 2.100000000000002)  = 0.060471679096957234
 f( 2.200000000000002)  = 0.0992063492063498
 f( 2.300000000000002)  = 0.12384752992982005
 f( 2.400000000000002)  = 0.13914656771799652
 f( 2.500000000000002)  = 0.14814814814814833
*/
```

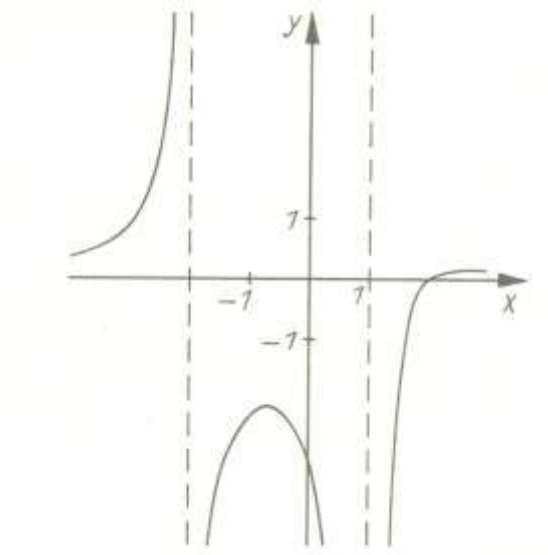
Kurvenskizze (i):



Kurvenskizze (ii):



Kurvenskizze (iii):



3. ***Summe.java***

zu (a) **Beweis (induktiv)**

IA $s = \sum_{i=0}^{0} i = 0 = \frac{0 \cdot 1}{2}$

IS $s = \sum_{i=0}^{n+1} i = \sum_{i=0}^{n} i + (n+1)$, laut IV

$$= \frac{n \cdot (n+1)}{2} + (n+1) = \frac{n \cdot (n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

zu (b) **Programm**

Summe.java

```
...

  public long gauss( long n)
  {
    long summe = 0;

    if( n % 2 == 0) summe = n / 2 * ( n + 1);
    else summe = ( n + 1) / 2 * n;

/*
    summe = (long)(n / 2. * ( n + 1));
*/

    return summe;
  }
...

/* ------------------------------------------------ */

// (c) Testergebnisse
// Summe von 0 bis 4294967295 (2^32 - 1)
// Schleife : 9223372034707292160
// Gauss    : 9223372034707292160
//            ->  korrekt

// Summe von 0 bis 4294967296 (2^32)
// Schleife : -9223372034707292160
// Gauss    : 9223372036854775807
//            ->  FEHLER: Long.MAX_VALUE ueberschritten
```

```
// (n = 4294967296, s = 9223372039002259456)
```

zu (c) **Berechnung**

Setze $s = 2^{63} - 1$ $\hat{=}$ **`Long.MAX_VALUE`**.

$$\frac{n}{2}(n+1) = \frac{n^2}{2} + \frac{n}{2} = s \Rightarrow n^2 + n - 2s = 0$$

Abschätzung:

$$n = -\frac{1}{2} + \sqrt{\frac{1}{4} + 2s} = -\frac{1}{2} + \sqrt{\frac{1}{4} + 2^{64} - 2} = -\frac{1}{2} + \sqrt{2^{64} - \frac{7}{4}}$$

$$< -\frac{1}{2} + \sqrt{2^{64}} = -\frac{1}{2} + 2^{32}$$

Setze $n = 2^{32} - 1$ ⇒ $s = \frac{2^{32} - 1}{2} \cdot 2^{32} = (2^{32} - 1)2^{31} = 2^{63} - 2^{31} < 2^{63} - 1.$

Setze $n = 2^{32}$ ⇒ $s = \frac{2^{32}}{2}(2^{32} + 1) = 2^{31}(2^{32} + 1) = 2^{63} + 2^{31} > 2^{63} - 1.$

⇒ Eine korrekte Lösung ist für jede natürliche Zahl $n \in [0, 2^{32} - 1]$ zu erwarten.

zu (d) **`double` – Zahlen**

Die Exaktheit der Darstellung ergibt sich aus der Anzahl der darstellbaren Dualzahlen. Allgemein gilt: Sei t_b die Anzahl der darstellbaren Ziffern zur Basis b, dann gilt $t_{10} = \log_{10} 2^{t_2} = t_2 * \log_{10} 2$ (Abrunden!).

s als `long` - Zahl ⇒ $t_2 = 64$ (⇒ $t_{10} = 19$).

s als `double` - Zahl ⇒ $t_2 = 53$ (⇒ $t_{10} = 15$).

Folglich kann man unter Verwendung des Datentyps `double` keine bessere Darstellung erreichen.

Test mit dem Datentyp* `double` *liefert für* $n = 2^{32} - 1$ *ein fehlerhaftes Ergebnis:

```
4294967295     :  9223372034707292160   ->  n = 2^32 -1: long
4.294967295E9  :  9.223372034707292E18  ->  n = 2^32 -1: double
```