Interpolationspolynome

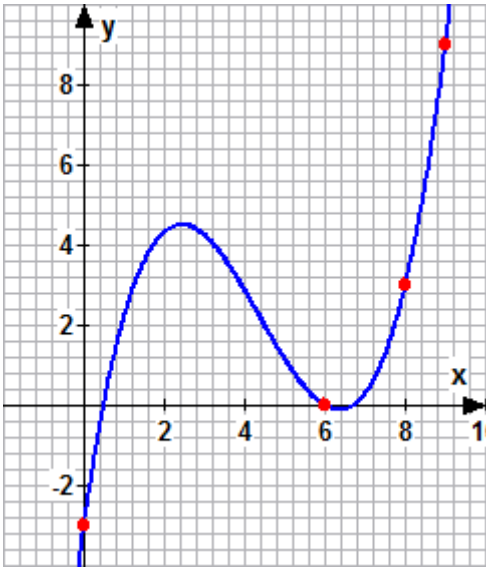
Ac 2016-2019

Problemstellung:

Gegeben sind n+1 Stützstellen x_0 bis x_n zusammen mit ihren Stützwerten y_0 bis y_n , wobei $x_i \neq x_j$ für $i \neq j$.

Durch die Punkte (x_i / y_i) , die in beliebiger Reihenfolge vorgegeben werden können, soll ein Polynom p(x) n-ten Grades gelegt werden :

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i \, x^i$$



Das Bild zeigt ein Beispiel für n = 3 !

$x_0 = 0 \quad x_1 = 6 \quad x_2 = 8 \quad x_3 = 9$
$y_0 = -3 \quad y_1 = 0 \quad y_2 = 3 \quad y_3 = 9$

Da alle Stützpunkte auf dem Polynomgraphen liegen, folgt : **$p(x_i) = y_i$ für i = 0 bis n**

Es ergeben sich somit n+1 Gleichungen mit n+1 Unbekannten **a_i** , mit eindeutiger Lösung.

Entwicklung der Formeln:

$$\left|\begin{array}{lcl} p(x_0)=y_0 & \Leftrightarrow & a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n = y_0 \\ p(x_1)=y_1 & \Leftrightarrow & a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n = y_1 \\ p(x_2)=y_2 & \Leftrightarrow & a_0 + a_1 x_2 + a_2 x_2^2 + \ldots + a_n x_2^n = y_2 \\ \vdots & \vdots & \vdots \\ p(x_n)=y_n & \Leftrightarrow & a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n = y_n \end{array}\right| \Leftrightarrow \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \Leftrightarrow V \cdot \vec{a} = \vec{y}$$

Die Matrix V heißt VANDERMONDE – Matrix !

Das LGS (Lineares Gleichungssystem) kann z.B. mit dem Gaußschen Eliminationsverfahren gelöst werden.

In den hier gezeigten Beispielen wird die
rref = reduced row echelon form (Zeilen-Stufen-Normalform) verwendet.

Anwendung auf das Beispiel von oben:

Stützpunkte (0 ; -3) (6 ; 0) (8 ; 3) (9 ; 9)
Wir haben hier 4 Stützpunkte (i = 0 bis 3) , was ein Polynom 3.Ordnung nahelegt.

Das 4 mal 5 – LGS (Lineares Gleichungssystem) lautet dann folgendermaßen:

$a_0 + 0a_1 + 0^2a_2 + 0^3a_3 = -3$
$a_0 + 6a_1 + 6^2a_2 + 6^3a_3 = 0$
$a_0 + 8a_1 + 8^2a_2 + 8^3a_3 = 3$
$a_0 + 9a_1 + 9^2a_2 + 9^3a_3 = 9$

Vereinfachtes LGS als Schema:

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
1	6	36	216	0
1	8	64	512	3
1	9	81	729	9

Einige Umformungsschritte:

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
0	1	6	36	1/2
0	4	32	256	3
0	3	27	243	4

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
0	1	6	36	1/2
0	0	8	112	1
0	0	9	135	5/2

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
0	1	6	36	1/2
0	0	0	72	11
0	0	9	135	5/2

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
0	1	6	36	1/2
0	0	0	72	11
0	0	648	0	-1305

a_0	**a_1**	**a_2**	**a_3**	**·1**
1	0	0	0	-3
0	648	0	0	4590
0	0	0	72	11
0	0	648	0	-1305

Es folgt: $a_0 = -3$ $a_1 = 85/12$ $a_2 = -145/72$ $a_3 = 11/72$

Das gesuchte Polynom ist dann: **$p(x) = 11/72x^3 – 145/72x^2 + 85/12x – 3$**

Weiteres Beispiel (8 Punkte):

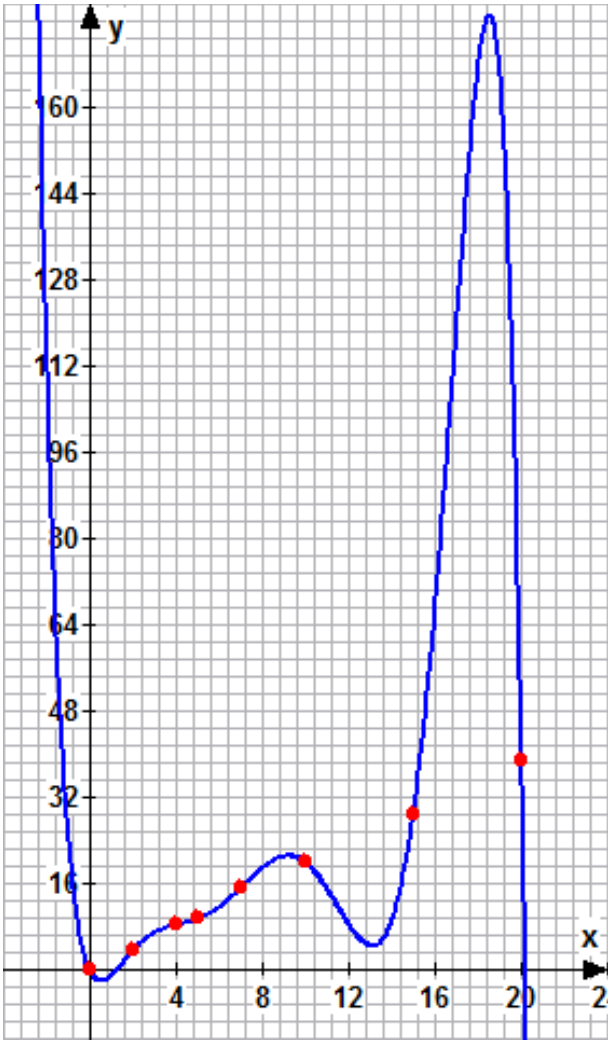
x	0	2	4	5	7	10	15	20
y	0	3,8	8,5	9,6	15,4	20	29,1	39

LGS (Schema):

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	*1
1	0	0	0	0	0	0	0	0
1	2	4	8	16	32	64	128	3,8
1	4	16	64	256	1024	4096	16384	8,5
1	5	25	125	625	3125	15625	78125	9,6
1	7	49	343	2401	16807	117649	823543	15,4
1	10	100	1000	10000	100000	1000000	10000000	20
1	15	225	3375	50625	759375	11390625	170859375	29,1
1	20	400	8000	160000	3200000	64000000	1280000000	39

Lösung für das Polynom 7. Grades :

$a_0 = 0$
$a_1 = -8680171 / 1029600 \approx -8{,}43062451$
$a_2 = 10726769 / 950400 \approx 11{,}28658354$
$a_3 = -1365559331 / 308880000 \approx -4{,}421002755$
$a_4 = 5069474353 / 6177600000 \approx 0{,}8206219815$
$a_5 = -95084807 / 1235520000 \approx -0{,}0769593426$
$a_6 = 21565939 / 6177600000 \approx 0{,}0034909899$
$a_7 = -373513 / 6177600000 \approx -0{,}00006046247734$



Der Graph des Polynoms ist auffallend wellig !

Java-Methode zur Berechnung der a_i (i von 0 bis n-1) :

```
// double[] datX, double[] datY  sind gegeben

int n = datX.length;
double[][] matrix = new double[n][n + 1];
// Matrix(n,n+1) zur Bestimmung des Interpolationspolynoms aufbauen
for (int ze = 0; ze < n; ze++) {
   matrix[ze][n] = datY[ze]; // rechte Seite des LGSs
   matrix[ze][0] = 1.0; // linke Spalte für die Faktoren von a0
   for (int sp = 1; sp < n; sp++) {
      double xPotenz = datX[ze];
      // bilde datX[ze]^sp
      for (int k = 1; k < sp; k++)
         xPotenz = xPotenz * datX[ze];
      matrix[ze][sp] = xPotenz;
   }
}
// matrix(n,n+1) auf rref-Form (reduced row echelon) bringen
MatrixAc matrref = new MatrixAc(matrix).rref();
// der Lösungsvektor (0..n-1) steht dann in der rechten Spalte von matrref
double[] loesVektor = new double[n];
for (int ze = 0; ze < n; ze++)
   loesVektor[ze] = matrref.getElem(ze, n);
```

Java-Methode zur Berechnung von Funktionswerten $f(x_0)$ mit dem HORNER-Schema :

```
// double[] x, double[] a   sind gegeben
fx = a[n];
for (int i = n-1; i >= 0; i--)
   fx = fx * x0 + a[i];
```

Ein alternativer Ansatz für das Interpolationspolynom p(x) geht auf NEWTON zurück :

$$p(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) + \ldots + a_n(x-x_0)(x-x_1)\cdot \ldots \cdot(x-x_{n-1})$$

kurz: $$p(x) = \sum_{i=0}^{n} a_i \cdot \prod_{k=0}^{i-1}(x - x_k)$$ Newtonsches Interpolationspolynom

Dieser Ansatz führt auf ein gestaffeltes LGS für die a_i , welches sich leicht lösen lässt :

$p(x_0) = y_0 \rightarrow\ a_0 = y_0$
$p(x_1) = y_1 \rightarrow\ a_0 + a_1(x_1-x_0) = y_1$
$p(x_2) = y_2 \rightarrow\ a_0 + a_1(x_2-x_0) + a_2(x_2-x_0)(x_2-x_1) = y_2$
$p(x_3) = y_3 \rightarrow\ a_0 + a_1(x_3-x_0) + a_2(x_3-x_0)(x_3-x_1) + a_3(x_3-x_0)(x_3-x_1)(x_3-x_2) = y_3$
…

$p(x_n) = y_n \rightarrow\ a_0 + a_1(x_n-x_0) + a_2(x_n-x_0)(x_n-x_1) + \ldots + a_n(x_n-x_0)\cdot \ldots \cdot(x_n-x_{n-1}) = y_n$

Für das obige Beispiel 1 mit 4 Stützstellen (0 ; -3) (6 ; 0) (8 ; 3) (9 ; 9) ergibt sich dann:

$a_0 = -3$
$a_0 + 6a_1 = 0$, also $a_1 = 1/2$
$a_0 + 8a_1 + 16a_2 = 3$, also $a_2 = 1/8$
$a_0 + 9a_1 + 27a_2 + 27a_3 = 9$, also $a_3 = 11/72$

Das Newtonsche Interpolationspolynom ist folgerichtig

$p(x) = -3 + 1/2(x-0) + 1/8(x-0)(x-6) + 11/72(x-0)(x-6)(x-8)$
vereinfacht: **$p(x) = -3 + 1/2x + 1/8x(x-6) + 11/72x(x-6)(x-8)$**

Umformung in die Normalform:
$p(x) = -3 + 1/2x + 1/8x^2 - 3/4x + 11/72x(x^2-14x+48)$
$= -3 + -1/4x + 1/8x^2 + 11/72x^3 - 77/36x^2 + 22/3x$
Ergebnis Normalform: **$p(x) = -3 + 85/12x - 145/72x^2 + 11/72x^3$**

Dies entspricht dem im **obigen** Verfahren ermittelten Polynom !

Beispiel 2 mit 8 Stützstellen (0|0) (2|3,8) (4|8,5) (5|9,6) (7|15,4) (10|20) (15|29,1) (20|39) :

$p(x_0) = y_0 \rightarrow a_0 = 0$
$p(x_1) = y_1 \rightarrow 0 + a_1 \cdot 2 = 3{,}8$, also $a_1 = 19 / 10$
$p(x_2) = y_2 \rightarrow 0 + 1{,}9 \cdot 4 + a_2 \cdot 4 \cdot 2 = 8{,}5$, also $8a_2 = 0{,}9$ und somit $a_2 = 0{,}9/8 = 9 / 80$
$p(x_3) = y_3 \rightarrow 0 + 1{,}9 \cdot 5 + 0{,}1125 \cdot 5 \cdot 3 + a_3 \cdot 5 \cdot 3 \cdot 1 = 9{,}6$, also $15a_3 = -1{,}5875 \rightarrow a_3 = -127 / 1200$
$p(x_4) = y_4 \rightarrow 0 + 1{,}9 \cdot 7 + 0{,}1125 \cdot 7 \cdot 5 - 127/1200 \cdot 7 \cdot 5 \cdot 3 + a_4 \cdot 7 \cdot 5 \cdot 3 \cdot 2 = 15{,}4$, also $210a_4 = 371/40 \rightarrow a_4 = 53 / 1200$
$p(x_5) = y_5 \rightarrow 0 + 1{,}9 \cdot 10 + 9/80 \cdot 10 \cdot 8 - 127/1200 \cdot 10 \cdot 8 \cdot 6 + 53/1200 \cdot 10 \cdot 8 \cdot 6 \cdot 5 + a_5 \cdot 10 \cdot 8 \cdot 6 \cdot 5 \cdot 3 = 20$,
also $83{,}2 + 7200a_5 = 20 \rightarrow 7200a_5 = -632/10 \rightarrow a_5 = -632/72000 = -79 / 9000$

Analog errechnet man:

$a_6 = 22937 / 25740000$
$a_7 = -373513 / 6177600000$

Der Vorteil der Newton-Methode liegt in der recht einfachen Berechnung (ohne rref) sowie in der Tatsache, dass ohne komplette Neuberechnung neue Stützstellen hinzugenommen werden können. Die vorherige Rechnung kann dann einfach ohne Änderung übernommen werden.

Bei der Berechnung der Koeffizienten a_i des Newtonschen Interpolationspolynoms treten Divisionen und Differenzen auf, deren Auswertung rekursiv erfolgen kann.

Die dividierten Differenzen für die a_i seien mit $f[x_0, x_1, \dots, x_i]$ bezeichnet.
Sie werden folgendermaßen rekursiv berechnet:

Für i von 0 bis n: $f[x_i] = y_i$
Für k von 1 bis n:
  Für i von 0 bis n-k:
    $f[x_i, x_{i+1}, \dots, x_{i+k}] = (f[x_{i+1}, x_{i+2}, \dots, x_{i+k}] - f[x_i, x_{i+1}, \dots, x_{i+k-1}]) / (x_{i+k} - x_i)$

Das folgende Schema zeigt die Anwendung dieser Rekursionsformeln:

xi	yi	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
0	-3 = **$f[x_0]$**			
		$\frac{0-(-3)}{6-0} = 0{,}5 = f[x_0, x_1]$		
6	0		$\frac{1{,}5-0{,}5}{8-0} = 0{,}125 = f[x_0, x_1, x_2]$	
		$\frac{3-0}{8-6} = 1{,}5 = f[x_1, x_2]$		$\frac{1{,}5-0{,}125}{9-0} = \frac{11}{72} = f[x_0,x_1,x_2,x_3]$
8	3		$\frac{6-1{,}5}{9-6} = 1{,}5 = f[x_1, x_2, x_3]$	
		$\frac{9-3}{9-8} = 6 = f[x_2, x_3]$		
9	9			

Rot eingezeichnet sind die errechneten Koeffizienten ai . Wie man sieht, lässt sich das Schema nach unten durch Hinzunahme neuer Stützstellen erweitern.

Java-Methode zur Berechnung der a_i (i von 0 bis n-1) :

```
// double[] x, double[] y  sind gegeben
      double[] a = new double[n];
      for (int i = 0; i < n; i++)
         a[i] = y[i];
      for (int ze = 1; ze < n; ze++)
         for (int sp = n-1; sp >= ze; sp--)
            a[sp] = (a[sp] - a[sp - 1]) / ( x[sp] - y[sp - ze]);
```

Java-Methode zur Berechnung von Funktionswerten f(x0) mit dem HORNER-Schema :

```
      // double[] x, double[] a   sind gegeben
      fx = a[n];
      for (int i = n-1; i >= 0; i--)
         fx = fx * (x0 - x[i]) + a[i];
```