Kapitel 7

Kurven

Problem: Spezifiziere und zeichne eine beliebige "glatte" Kurve.

7.1 Algebraischer Ansatz

Gegeben n Stützpunkte.
Bestimme Polynom $(n-1)$-ten Grades

$$y = a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \ldots + a_1 \cdot x + a_0$$

welches durch alle Stützpunkte läuft.
Problem: Bei der Auswertung des Polynoms treten wegen der hohen Potenzen hohe Rechenzeiten und große Rundungsfehler auf. Außerdem oszilliert die Kurve.

7.2 Splines

Gegeben n Stützpunkte.
Wähle $n-1$ Kurven kleiner Ordnung für den Verlauf innerhalb der $n-1$ Intervalle.
Kurven erster Ordnung (Geraden) ergeben nur den Linienzug.
Kurven zweiter Ordnung (Parabeln) sind an den Intervallgrenzen nur einmal differenzierbar.
Kurven dritter Ordnung mit stetigen ersten und zweiten Ableitungen an den Intervallgrenzen (kubische Splines) können wie folgt bestimmt werden:

Gesucht: Approximation einer Kurve $f(t)$, die durch n vorgegebene Punkte laufen soll.

Bewertung: f ist in parametrischer Form gegeben (d.h. $x = g(t), y = h(t)$), da die explizite Form $y = f(x)$ pro x-Wert nur einen y-Wert erlaubt.

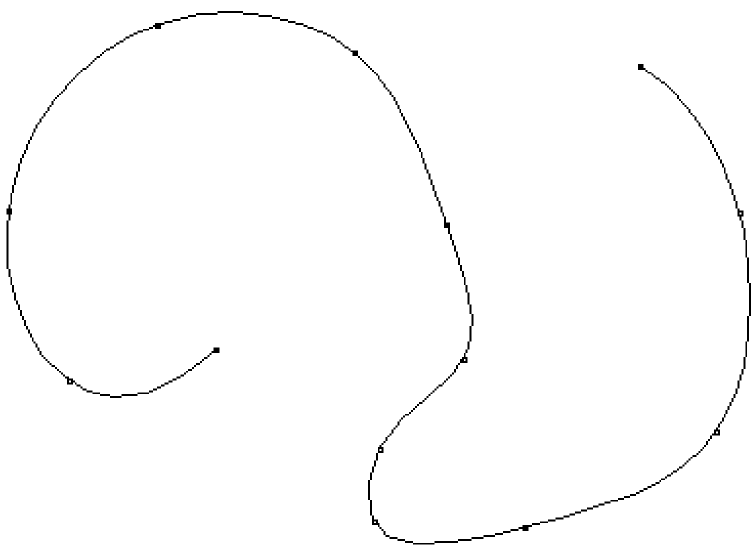
Die n Stützpunkte seien $f(t_i)$, die resultierenden Intervalle seien $[t_i, t_{i+1}]$, die für das Intervall i zuständige Funktion sei $f_i(t)$, $1 \leq i \leq n-1$. Jedes f_i hat die Form

$$\begin{aligned} f_i(t) &= f_i(t_i) + a_i \cdot (t - t_i) + b_i \cdot (t - t_i)^2 + c_i \cdot (t - t_i)^3 \\ & \quad t_i \leq t \leq t_{i+1}, \ i = 1, \ldots, n-1 \end{aligned}$$

Es muß gelten

$f_i(t_{i+1})$	$=$	$f_{i+1}(t_{i+1})$	für	$i = 1, \ldots, n-2$	stetig
$f_i'(t_{i+1})$	$=$	$f_{i+1}'(t_{i+1})$	für	$i = 1, \ldots, n-2$	differenzierbar
$f_i''(t_{i+1})$	$=$	$f_{i+1}''(t_{i+1})$	für	$i = 1, \ldots, n-2$	keine Steigungsänderung

Nach Vorgabe von a_1 und a_{n-1} entsteht ein Gleichungssystem mit $n-3$ Gleichungen und $n-3$ Unbekannten $a_2, a_3, \ldots, a_{n-2}$.
Die Werte von a_1 und a_{n-1} können z.B. ermittelt werden durch Festlegung $f''(t_1) = f''(t_n) = 0$ (natürliche Spline-Interpolation).
Für die Folge t_i reicht jede monoton wachsende Folge. Geeignet ist z.B. die Folge der euklidischen Abstände zwischen den Stützpunkten. Statt den Abstand $d = \sqrt{dx^2 + dy^2}$ zu berechnen, verwendet man die Näherungsformel $\frac{1}{3} \cdot (dx + dy + 2 \cdot \max(dx, dy))$.



Vom Spline-Algorithmus erzeugte Kurveninterpolation mit 13 Stützpunkten und 65 Interpolationspunkten

```
/****************************************************************************************/
/*              Spline-Interpolation durch Polynome 3. Grades                           */
/****************************************************************************************/

private static double[] calcfx = new double[MAX_POINTS];        // x-Funktionswerte
private static double[] calcfy = new double[MAX_POINTS];        // y-Funktionswerte
private static double[] x = new double[MAX_POINTS];     // x-Koordinaten der Stuetzpunkte
private static double[] y = new double[MAX_POINTS];     // y-Koordinaten der Stuetzpunkte
private static double[] t = new double[MAX_POINTS];     // t-Werte der Stuetzpunkte

void besetze_arrays(                            // berechnet Intervallgrenzen
              int point_cnt,                    // Anzahl der Polygonpunkte
              Point punkt[])                    // Punktliste
{
      int i;
      double ax,ay,dd;

      for (i=0; i<point_cnt; i++) {             // besetze x- und y-Koordinaten
              x[i] = (double) punkt[i+1].x;
              y[i] = (double) punkt[i+1].y;}
      t[0] = 0.0;                               // besetze t-Werte mit dem
      for (i=1; i<point_cnt; i++) {             // Euklidischen Abstand der Stuetzpunke
              ax=Math.abs(x[i]-x[i-1]);
              ay=Math.abs(y[i]-y[i-1]);
              dd = ax + ay;
              if (ax > ay) dd = dd+2*ax;        // Naeherungsformel fuer Euklid:
                      else dd = dd+2*ay;        // Abstand = 1/3*(dx+dy+2*max{dx,dy})
              t[i] = t[i-1]+dd/3;
      }
}

void curve_fitting(                             // berechnet die Approximation der Kurve
      int point_cnt,                            // fuer point_cnt Stuetzpunkte
      Point punkt[],                            // uebergeben im Array punkt
      int spline_size)                          // fuer spline_size Interpolationswerte
{
      int i,n=0;
      besetze_arrays(point_cnt, punkt);                      // erzeuge Intervallgrenzen und
                                                             // Gleitkommaversion von punkt
      cubic_spline(point_cnt, t, x, spline_size, calcfx);   // bestimme die Interpol.werte
      cubic_spline(point_cnt, t, y, spline_size, calcfy);   // bestimme die Interpol.werte
      for (i=0; i<spline_size; i++)                          // und uebertrage sie
        spline_punkt[i+1] = new Point((int)calcfx[i],(int)calcfy[i]);
}
```

```
void cubic_spline(                    // loest Gleichungssystem zur Bestimmung
                                      // der Koeffizienten a,b,c
              int n,                  // Anzahl der Stuetzpunkte
              double t[], double f[], // Stuetzpunkte in der Form (t[i],f[t[i]])
              int spline_size,        // Anzahl der Interpolationswerte
              double calcf[])         // Ausgabearray: Interpolationswerte
{
      int i,j;
      double a[] = new double[MAX_POINTS];
      double b[] = new double[MAX_POINTS];
      double c[] = new double[MAX_POINTS];
      double delta_t[] = new double[MAX_POINTS];
      double D[] = new double[MAX_POINTS];
      double m[] = new double[MAX_POINTS];
      double k[] = new double[MAX_POINTS];
      double bh,dh,e,h,wt,dt;
      for (i=1; i<n; i++) {
              delta_t[i] = t[i]-t[i-1];
              D[i] = (f[i]-f[i-1])/delta_t[i];
      }
      m[0] = delta_t[2];
      delta_t[0] = t[2]-t[0];
      h = delta_t[1];
      k[0] = ((h + 2*delta_t[0])*D[1]*delta_t[2]+h*h*D[2])/delta_t[0];
      for (i=1; i<(n-1); i++) {
              h = -delta_t[i+1]/m[i-1];
              k[i] = h*k[i-1]+3*(delta_t[i]*D[i+1]+delta_t[i+1]*D[i]);
              m[i] = h *delta_t[i-1] + 2* (delta_t[i] + delta_t[i+1]);
      }
      h = t[n-1]-t[n-3];
      dh = delta_t[n-1];
      k[n-1] = ((dh+h+h)*D[n-1]*delta_t[n-2] +  dh*dh*D[n-2])/h;
      h = -h/m[n-2];
      m[n-1] = delta_t[n-2];
      m[n-1] = h*delta_t[n-2] + m[n-1];
      a[n-1] = (h*k[n-2]+k[n-1])/m[n-1];
      for (i=n-2; i>=0; i--) a[i] = (k[i]-delta_t[i]*a[i+1])/m[i];
      for (i=1; i<n; i++) {
              dh = D[i]; bh = delta_t[i];
              e = a[i-1]+a[i]-dh-dh;
              b[i-1] = 2*(dh-a[i-1]-e)/bh;
              c[i-1] = 6*e/(bh*bh);
      }
      wt = 0; j=0;
      dt = t[n-1]/(double)(spline_size-1);
      for (i=0; i<spline_size; i++) {
              while ((t[j+1]<wt)&&(j<spline_size)) j++;
              h = wt - t[j];
              calcf[i] = f[j]+h*(a[j]+h*(b[j]+h*c[j]/3)/2);
              wt = wt+dt;
      }
      calcf[spline_size-1]= f[n-1];
}
```

7.3 Bézier-Kurven

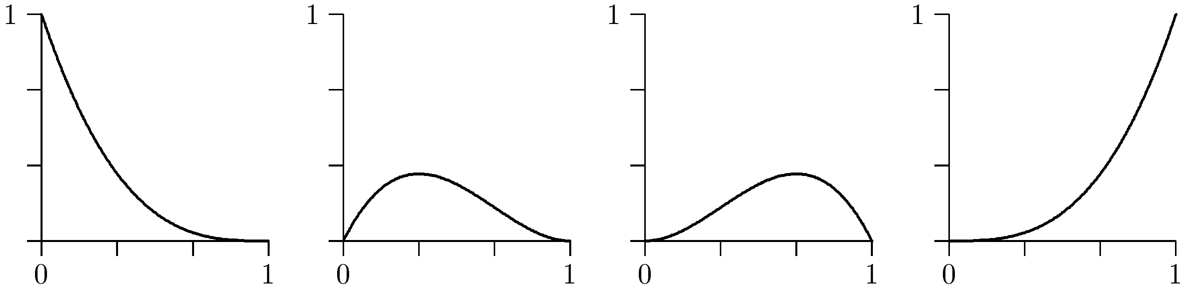
Spezifiziere die Kurve durch $n+1$ Stützpunkte $P_0, P_1, \ldots, P_n$. Die Kurve läuft nicht durch alle Stützpunkte, sondern wird von ihnen beeinflußt.

$$P(t) = \sum_{i=0}^{n} B_{i,n}(t) \cdot P_i, 0 \le t \le 1$$

Der Punkt P_i wird gewichtet mit Hilfe eines Bernstein-Polynoms.

$$B_{i,n}(t) = \frac{n!}{i! \cdot (n-i)!} \cdot t^i \cdot (1-t)^{n-i},\ i = 0, ..., n$$

Beispiel: $n = 3$



$B_{0,3} = (1-t)^3$ $B_{1,3} = 3 \cdot t \cdot (1-t)^2$ $B_{2,3} = 3 \cdot t^2 \cdot (1-t)$ $B_{3,3} = t^3$

Die wichtigsten Eigenschaften der Bernsteinpolynome:

- alle Bernsteinpolynome sind positiv auf $[0,1]$,
- für jedes feste t gilt $\sum_{i=0}^{n} B_{i,n}(t) = 1$,
- $B_{i,n}(t) = B_{n-i,n}(1-t)$,
- $\frac{dB_{i,n}(t)}{dt} = n \cdot (B_{i-1,n-1}(t) - B_{i,n-1}(t)), i > 0$,
- Maxima in $[0,1]$ an den Stellen $t = \frac{i}{n}, i = 0, \ldots, n$.

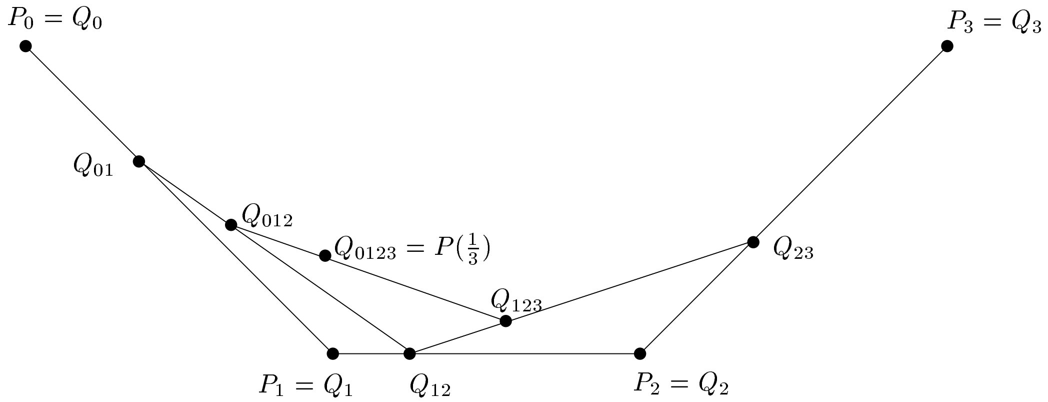
Da alle Bernsteinpolynome für $0 < t < 1$ ungleich Null sind, beeinflussen alle Stützpunkte in diesem Intervall den Kurvenverlauf.
Die Kurve beginnt im Stützpunkt P_0 tangential zur Geraden $\overline{P_0 P_1}$, endet im Stützpunkt P_n tangential zur Geraden $\overline{P_{n-1} P_n}$ und verläuft innerhalb der konvexen Hülle der Stützpunkte. Somit können mehrere Bézier-Kurven aneinandergesetzt werden durch Identifikation von Endpunkt und Anfangspunkt aufeinanderfolgender Bézierkurven. Einen stetig differenzierbaren Übergang erreicht man bei Kollinearität der Punkte $P_{n-1}, P_n = Q_0, Q_1$.

Berechnung der Bézier-Kurve nach de Casteljau

Es gilt:

$$P(t)_{0,n} = (1-t) \cdot P(t)_{0,n-1} + t \cdot P(t)_{1,n}$$

wobei die tiefgestellten Indizes angeben, welche Stützpunkte in die Berechnung einfließen. Somit läßt sich eine Bézier-Kurve vom Grad n durch zwei Bézier-Kurven vom Grad $n-1$ definieren, indem für ein festes t die Punkte beider Kurven berechnet und die Verbindungsstrecke im Verhältnis t geteilt wird.



Berechnung eines Kurvenpunktes für $t = \frac{1}{3}$. Die Indizes geben an, welche Stützpunkte beteiligt sind.

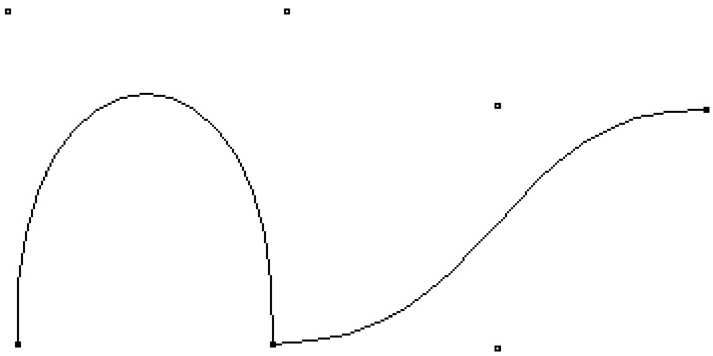
```
/****************************************************************************************/
/*                                                                                      */
/*                         Zeichnen einer Bezierkurve 3. Grades nach De Casteljau       */
/*                                                                                      */
/****************************************************************************************/

private void bezier(                              // Zeichne eine Bezierkurve 3. Grades
                     double px0, double py0,      // anhand des Stuetzpunktes p0
                     double px1, double py1,      // anhand des Stuetzpunktes p1
                     double px2, double py2,      // anhand des Stuetzpunktes p2
                     double px3, double py3,      // anhand des Stuetzpunktes p3
                     int depth)                   // aktuelle Rekursionstiefe
{
        double qx01,qy01,qx12,qy12,qx23,qy23,qx012,qy012,qx123,qy123,qx0123,qy0123;
                                                  // Hilfspunkte

        if (depth > iter)                         // Iterationstiefe erreicht
                set_line( new Point( (int)px0, (int)py0), new Point( (int)px3, (int)py3));
        else {
                depth++;

                qx01   = (px0+px1)/2;     qy01   = (py0+py1)/2;
                qx12   = (px1+px2)/2;     qy12   = (py1+py2)/2;
                qx23   = (px2+px3)/2;     qy23   = (py2+py3)/2;
                qx012  = (qx01+qx12)/2;   qy012  = (qy01+qy12)/2;
                qx123  = (qx12+qx23)/2;   qy123  = (qy12+qy23)/2;
                qx0123 = (qx012+qx123)/2; qy0123 = (qy012+qy123)/2;

                bezier (px0,py0,qx01,qy01,qx012,qy012,qx0123,qy0123,depth);
                bezier (qx0123,qy0123,qx123,qy123,qx23,qy23,px3,py3,depth);
        }
}
```



Vom De Casteljau-Algorithmus mit Rekursionstiefe 3 gezeichnete Bézierkurven

7.4 B-Splines

Spezifiziere die Kurve durch $n+1$ Stützpunkte $P_0, \ldots, P_n$ und einen Knotenvektor $T = (t_0, t_1, \ldots, t_{n+k}), t_j \leq t_{j+1}$.

$$P(t) = \sum_{i=0}^{n} N_{i,k}(t) \cdot P_i$$

Die Punkte P_i werden gewichtet durch die Basis des B-Splines, Polynome $N_{i,k}(t)$ vom Grad $k-1$ $(0 \leq i \leq n)$:

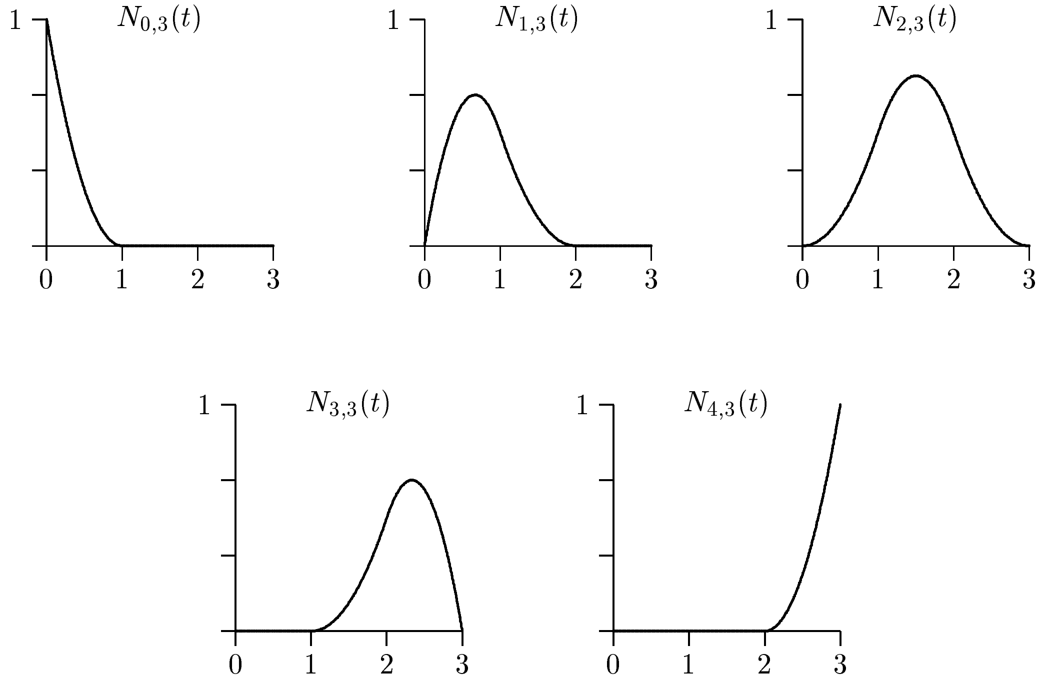
$$\begin{aligned} N_{0,k}(0) &= 1, \text{ für alle } k \\ N_{i,1}(t) &= \begin{cases} 1 & \text{falls } t_i < t \leq t_{i+1} \\ 0 & \text{sonst} \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} \cdot N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \cdot N_{i+1,k-1}(t), \ k > 1 \end{aligned}$$

Bei Division durch Null wird der Quotient gleich 0 gesetzt.
Der Knotenvektor wird häufig wie folgt gewählt für $0 \leq j \leq n+k$:

$$t_j = \begin{cases} 0 & \text{falls } j < k \\ j - k + 1 & \text{falls } k \leq j \leq n \\ n - k + 2 & \text{falls } j > n \end{cases}$$

Hierdurch wird t im Intervall $[0, n-k+2]$ definiert.
Beispiel: $k = 3$, $n = 4$ ergibt einen uniformen quadratischen B-Spline mit dem Knotenvektor $T = (0,0,0,1,2,3,3,3)$. Die Stützpunkte P_0, P_1, P_2, P_3, P_4 haben nur lokal Einfluß auf den

Kurvenverlauf, und zwar in den Intervallen $t \in [0,1], [0,2], [0,3], [1,3], [2,3]$.



Sonderfall

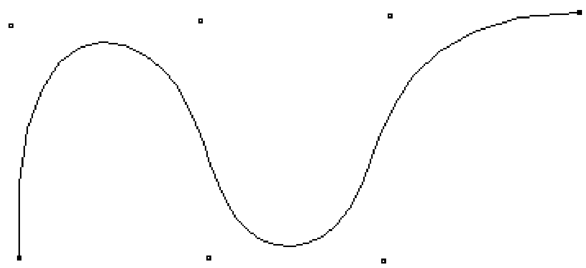
Für $k = n+1$ ergibt sich für den Knotenvektor der Sonderfall $T = (\underbrace{0,...,0}_{k \text{ mal}}, \underbrace{1,...,1}_{k \text{ mal}})$.

Die zugehörigen B-Splinefunktionen haben die schon bekannte Form

$$N_{i,k}(t) = \frac{(k-1)!}{i! \cdot (k-1-i)!} \cdot t^i \cdot (1-t)^{k-1-i}, \; i = 0, ..., k-1$$

Somit lassen sich also die B-Splinefunktionen als Verallgemeinerung der Bernsteinpolynome auffassen.

Wie bei den Bernsteinpolynomen gilt auch für die B-Splinefunktionen $N_{i,k}$, daß sie positiv sind, und $\sum_{i=0}^{n} N_{i,k} = 1$.



B-Spline mit $n = 6$, $k = 4$ und 35 Interpolationspunkten