2. Pflichtübung

Numerische Mathematik

Programm zum erzeugen einer natürlichen kubischen nichtparametrischen Splinefunktion durch einen Satz von Knoten.

von

Jost Griesemann,
Roland Steffen

SFT2

19.11.2002

Quellcode:

```
/***********************************************************
 *                                                         *
 * Das Programm interpoliert mit hilfe der Kubischen       *
 * Spline-Funktion Zwischenpunkte der angegebenen          *
 * Stuetzstellen:                                          *
 *                                                         *
 * Die Inputdatei sollte die folgende Form haben:          *
 *                                                         *
 *      <N>                                                *
 *      n                                                  *
 *                                                         *
 *      <Knoten>                                           *
 *      x0;y0                                              *
 *      x1;y1                                              *
 *      x2;y2  ...                                         *
 *                                                         *
 ***********************************************************/

#include <stdio.h>
#include <stdlib.h> // fuer 'malloc'
#include <string.h> // fuer 'strncmp'
#include <conio.h>  // fuer 'clrscr'

int finde_Schluesselwort(FILE *file_p, char *keyword);
int n_einlesen(FILE *file);
int Knoten_einlesen(FILE *file, double *x, double *y, int n);
void Koeffizienten_berechnen(double *x, double *y, int n, double *b, double *c, double *d);
void triGauss(double *ud, double *hd, double *od, double *lv, double *c, int n);
void Berechung_Plotwerte(FILE *file, double *a, double *b, double *c, double *d, double *x,
int n, int Nplot);

main()
  {
  FILE *file;
  double *x;
  double *y;

  // Koeffizienten:
  double *b;
  double *c;
  double *d;

  char Dateiname[FILENAME_MAX + 1];
  int n;
  int i;
  int Nplot;

  clrscr();  // Bildschirm saubern
  printf("\nDas Programm interpoliert mit hilfe der Kubischen Spline-Funktion\nZwischenpunkte
       der angegebenen Stuetzstellen.\n");
  printf("\nGeben sie den Dateipfad ein ('exit' um zu beenden): ");
  scanf("%s", Dateiname);

  // Pruefen ob Benutzer abbrechen will:
  if(strncmp(Dateiname, "exit", 4) == 0)
    {
    printf("\nAbbruch durch Benutzer. Programm beendet.\n");
    exit(20);
    }

  // Inputdatei oeffnen:
  if((file = fopen(Dateiname, "r")) == NULL) goto Fehler1;

  n = n_einlesen(file);

  // Speicherplatz reservieren:
  if((x = (double *) malloc(n * sizeof(double))) == NULL) goto Fehler2;
  if((y = (double *) malloc(n * sizeof(double))) == NULL) goto Fehler2;
  if((b = (double *) malloc((n) * sizeof(double))) == NULL) goto Fehler2;
  if((c = (double *) malloc((n) * sizeof(double))) == NULL) goto Fehler2;
  if((d = (double *) malloc((n-1) * sizeof(double))) == NULL) goto Fehler2;

  if(Knoten_einlesen(file, x, y, n) != 1) goto Fehler;
  fclose(file);
```

```
  Koeffizienten_berechnen(x, y, n, b, c, d);

  printf("\nDie Koeffizienten wurden berechnet.\nGeben sie die Anzahl der Plotwerte ein: ");
  scanf("%d", &Nplot);

  printf("\nGeben sie den Pfad fuer die Ausgabedatei an: ");
  scanf("%s", Dateiname);

  if((file = fopen(Dateiname, "w")) == NULL) goto Fehler1;
  Berechung_Plotwerte(file, y, b, c, d, x, n, Nplot);
  fclose(file);

  // reservierten Speicherplatz freigeben:
  free(x);
  free(y);
  free(b);
  free(c);
  free(d);

  printf("\nProgramm beendet\n");
  exit(0);

  // Fehlermeldungen:
  Fehler:
    printf("\n\nDas Programm wurde aufgrund eines Fehlers beendet.\n");
    exit(10);

  Fehler1:
    printf("\nDie Datei '%s' konnte nicht geoeffnet werden. Das Programm wurde beendet.\n",
         Dateiname);
    exit(1);

  Fehler2:
    printf("\nBeim Reservieren von Speicherplatz ist ein Fehler aufgetreten. Das Programm
         wurde beendet.\n");
    exit(2);
  }

// Einlesen der Anzahl der Knoten:
int n_einlesen(FILE *file)
  {
  int n;

  // Lesekopf auf Schluesselwort "<N>" setzen und n einlesen:
  if(finde_Schluesselwort(file, "<N>") != 1) return 0;
  if(fscanf(file, "%d", &n) != 1) return 0;
  printf("\nN = %d\n", n);
  return n;
  }

// Einlesen der Knotenpunkte (x,y):
int Knoten_einlesen(FILE *file, double *x, double *y, int n)
  {
  int i;

  // Lesekopf auf Schluesselwort "<Knoten>" setzen und Knoten einlesen:
  if(finde_Schluesselwort(file, "<Knoten>") != 1) return 0;
  printf("\nKnotenpunkte:\n");
  for(i = 0; i < n; i++)
    {
    if(fscanf(file, "%lf;%lf", (x+i), (y+i)) != 2) return 0;
    printf("%lf; %lf\n", *(x+i), *(y+i));
    }

  return 1;
  }

// Bestimmen der Koeffizienten:
void Koeffizienten_berechnen(double *x, double *y, int n, double *b, double *c, double *d)
  {
  int i;
  double *h;
  double *hd;
  double *lv;

  // Speicherplatz reservieren
  if((h = (double *) malloc((n-1) * sizeof(double))) == NULL) goto Fehler;
  if((hd = (double *) malloc((n-1) * sizeof(double))) == NULL) goto Fehler;
```

```
  if((lv = (double *) malloc((n-2) * sizeof(double))) == NULL) goto Fehler;

  // Abstaende zwischen den Stuetzstellen berechnen:
  for(i = 0; i < n-1; i++) *(h+i) = *(x+i+1) - *(x+i);
  // Hauptdiagonale und Loesungsvektor des Gleichungssystems bestimmen:
  for(i = 0; i < n-2; i++)
    {
    *(hd+i) = 2 * (*(h+i) + *(h+i+1));
    *(lv+i) = 3 / *(h+i+1) * (*(y+i+2) - *(y+i+1)) - 3 / *(h+i) * (*(y+i+1) - *(y+i));
    }

  // Gleichungssystem loesen um Koeffizient c zubestimmen:
  triGauss(h, hd, h, lv, c, n-2);

  // c verschieben um vorn und hinten 0 einzufuegen:
  for(i = n-2; i > 0; i--) *(c+i) = *(c+i-1);
  *(c) = 0;
  *(c+n-1) = 0;

  // Koeffizienten d und b berechnen:
  for(i = 0; i < n-1; i++)
    {
    *(d+i) = (*(c+i+1) - *(c+i)) / 3 / *(h+i);
    *(b+i) = (*(y+i+1) - *(y+i)) / *(h+i) - (*(c+i+1) + 2**(c+i)) * *(h+i) / 3;
    }

  // Speicherplatz freigeben:
  free(h);
  free(hd);
  free(lv);

  return;

  // Fehlermeldungen:
  Fehler:
    printf("\nBeim Reservieren von Speicherplatz ist ein Fehler aufgetreten. Das Programm
          wurde beendet.\n");
    exit(2);
  }

// Funktion zum Aufloesen einer tridiagonal Matrix:
void triGauss(double *ud, double *hd, double *od, double *lv, double *c, int n)
  {
  int i;
  double rq;

  // vereinfachter Gaussscher Algorithmus:
  for(i = 1; i < n; i++)
    {
    rq = *(ud+i) / *(hd+i-1);  // Zeile wurde angepasst *(ud+i-1) --> *(ud+i)
    *(hd+i) -= rq * *(od+i);   // Zeile wurde angepasst *(od+i-1) --> *(od+i)
    *(lv+i) -= rq * *(lv+i-1);
    }

  // vereinfachte Rueckwaertsaufloesung:
  *(c+n-1) = *(lv+n-1) / *(hd+n-1);
  for(i = n-2; i >= 0; i--) *(c+i) = (*(lv+i) - *(od+i) * *(c+i+1)) / *(hd+i);
  }

// Positionieren des Lesekopfes in Zeile nach Schluesselwort setzten:
int finde_Schluesselwort(FILE *file_p, char *keyword)
  {
  char line[81];
  int n;
  int nchars;
  rewind(file_p); // Lesekopf auf Dateianfang setzen
  nchars = strlen(keyword); // laenge des Schuesselwortes feststellen
  n = nchars > 80 ? 80 : nchars; // maximal 80 Zeichen zulassen
  do {
    if(fgets(line, 80, file_p) == NULL)
      {
      printf ("\n\nKeyword \"%s\" nicht gefunden\n", keyword);
      return 0;
      }
    } while(strncmp(line, keyword, n) != 0);
  return 1;
  }
```

```
// berechnen der Plotwerte und speichern in der Datei:
void Berechung_Plotwerte(FILE *file, double *a, double *b, double *c, double *d, double *x,
int n, int Nplot)
  {
  int i;
  double Schritt;
  double AktuellePos;
  double x2;
  double y;
  int Abschnitt;

  Schritt = (*(x+n-1) - *(x)) / (Nplot - 1); // Abstaende der neuen x-Werte berechnen
  Abschnitt = 0;
  AktuellePos = *(x); // erster x-Wert

  for(i = 0; i < Nplot; i++)
    {
    // Polynom wechsel wenn Grenze ueberschritten:
    if(AktuellePos >= *(x+Abschnitt+1)) Abschnitt++;

    // x-Wert fuer Aktuelles Polynom berechnen:
    x2 = AktuellePos - *(x+Abschnitt);

    y = *(a+Abschnitt) + *(b+Abschnitt)*x2 + *(c+Abschnitt)*x2*x2 + *(d+Abschnitt)*x2*x2*x2;

    fprintf(file, "%lf;%lf\n", AktuellePos, y);

    AktuellePos += Schritt;  // neue x-Position
    }
  }
```

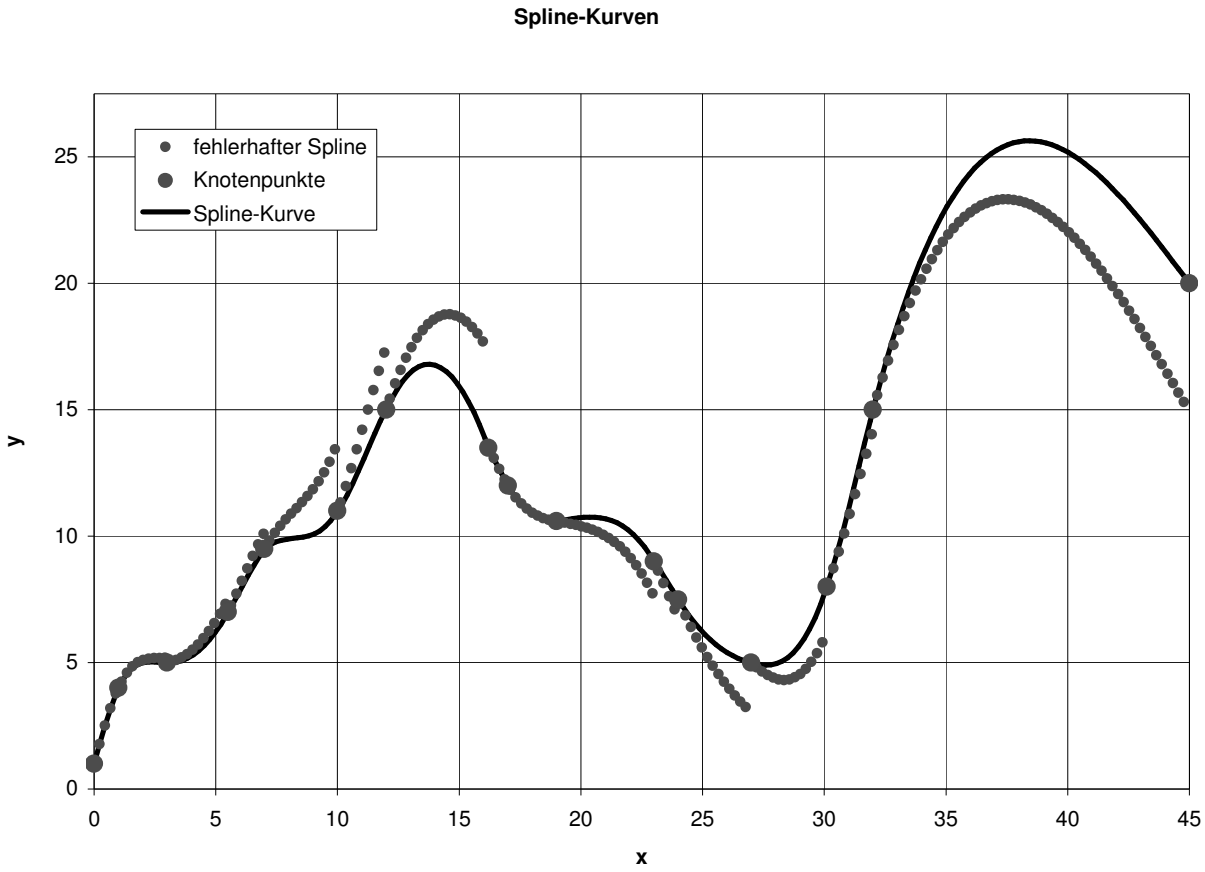
Beispiel:

Vorgabe Knoten:

<N>

16

<Knoten>	
0	1
1	4
3	5
5,5	7
7	9,5
10	11
12	15
16,2	13,5
17	12
19	10,6
23	9
24	7,5
27	5
30,1	8
32	15
45	20



Probleme bei Programmentwicklung:

Die im oben stehenden Diagramm als fehlerhafter Spline bezeichnete Funktion entstand durch eine falsche Berechnung der Koeffizienten b. Die Koeffizienten wurden hier mit der Gleichung 5.15 aus den Script berechnet. Nach ausgiebiger Fehlersuche wurde statt die Gleichung 5.15 die Gleichung 5.12 zur Berechung der Koeffizienten b benutzt. Durch diese Änderung wurde der Fehler behoben und es konnte nun die Spline-Kurve berechnet werden.

Das fehlerhafte Programm funktionierte jedoch bei äquidistanten Stützstellen was darauf schießen lässt, dass der Fehler durch ein falsches einsetzten der Abstände der Knoten in die vorgegeben Formeln entstand.