Universität Bielefeld	Computerphysik	Prof. Dr. Jürgen Schnack
Fakultät für Physik	SS 2012	jschnack@uni-bielefeld.de

Aufgabenblatt 5

5.1 Polynominterpolation: kubisches Lagrangeverfahren

a. **Machen Sie sich zu Hause mit dem Lagrange-Verfahren zur Polynominterpolation vertraut. Überlegen Sie sich insbesondere, wie man eine Polynominterpolation vorgegebener Ordnung für einen beliebig langen Datensatz durchführt (d.h. wenn mehr Punkte zur Verfügung stehen, als für die Interpolation benötigt werden).**

b. Für eine kubische Lagrange-Interpolation steht das Hauptprogramm `lagra.c` zur Verfügung. Dieses liest zunächst mit Hilfe der Funktion `readin.h` aus einer wählbaren Datei die Datenpaare (x_i, f_i), die interpoliert werden sollen, in die Felder `xin` und `yin` ein. Schauen Sie sich an, wie diese Funktion funktioniert.

   Anschließend durchläuft das Hauptprogramm das vorgegebene Intervall `[xmin, xmax]` und berechnet an Punkten `xs` in diesem Intervall mit Hilfe der in c. zu schreibenden Unterfunktion `lagra.h` eine kubische Interpolation `ys`. Dazu müssen für jedes `xs` vier Punkte aus dem Feld `xin` bestimmt werden, die für die Interpolation verwendet werden. Ausgangspunkt hierfür ist die Funktion `locate.h`, die den Index i bestimmt, so dass `xin[i] <= xs < xin[i+1]`. Das Feld `xin` wird dann so an die Unterfunktion `lagra.h` übergeben, dass *von dieser aus gesehen* der erste für die Interpolation zu verwendende Datenpunkt den Index 0 hat. Vollziehen Sie nach, wie genau dies funktioniert.

c. Schreiben Sie die benötigte Unterfunktion `lagra.h` für die kubische Lagrange-Interpolation. Verwenden Sie hierfür folgenden Prototyp:

```
void lagra(double xin[], double yin[], double xs, double *ys);
```

   Wiederholen Sie zu Hause den Unterschied beim Aufruf einer Funktion über *call by value* und *call by reference*.

   Gehen Sie beim Schreiben der Unterfunktion davon aus, dass als Datenpaare (x_i, f_i) für die Interpolation `(xin[0], yin[0])`, `(xin[1], yin[1])`, `(xin[2], yin[2])` und `(xin[3], yin[3])` zu benutzen sind. Die Interpolation soll an der Stelle `xs` berechnet und das Ergebnis über den Pointer `ys` in `*ys` geschrieben werden.

d. Testen Sie das Programm anhand der Daten in der Datei `inter.dat`.

5.2 Zusatzaufgabe: kubischer Spline

a. **Machen Sie sich zu Hause mit der (kubischen) Spline-Interpolation vertraut. In welchem Sinn soll die Spline-Interpolation zu einer „besseren“ Interpolation führen? Welche Informationen werden für die Berechnung der Spline-Interpolation benötigt?**

b. Mit `spline.c` steht hier ein Hauptprogramm ähnlich zu `lagra.c` zur Verfügung. Beachten Sie, dass für die Spline-Interpolation in jedem Schritt nur zwei statt wie zuvor vier Datenpunkte benötigt werden. Die für die Spline-Interpolation erforderlichen zweiten Ableitungen werden in der Funktion `derivatives.h` über die Lösung eines tridiagonalen Gleichungssystems bestimmt. Die Parameter sind dabei so gewählt, dass sich ein *natürlicher Spline* ergibt, dessen zweite Ableitungen am Rand verschwinden. Der verwendete Algorithmus stammt aus den *Numerical Recipes*.

   Schreiben Sie die noch fehlende Unterfunktion `spline.h` für die kubische Spline-Interpolation. Verwenden Sie dazu den folgenden Prototyp:

   ```
   void spline(double xin[], double yin[], double y2in[], double xs, double *ys);
   ```

   Das Feld `y2in` enthält die zuvor berechneten zweiten Ableitungen. Nehmen Sie ähnlich wie in Aufgabe 5.1 c. an, dass die zu verwendenden Daten in `(xin[0], yin[0], y2in[0])` und `(xin[1], yin[1], y2in[1])` liegen.

c. Testen Sie das Programm wieder anhand der Daten in der Datei `inter.dat` und vergleichen Sie die Ergebnisse mit der kubischen Lagrange-Interpolation.