Institut für Informationsverarbeitung
Leibniz Universität Hannover
http://www.tnt.uni-hannover.de

Scientific Computing I - Übung 10

1 Theorie

a) Es sind folgende Datenpunktmengen M_i gegeben:
   - $M_1 = \{(1, 0), (3, 32), (5, 120)\}$
   - $M_2 = \{(1, -1), (2, 9), (3, 11), (5, -33)\}$

   Bestimmen Sie für beide Mengen M_i die Vandermonde-Matrizen (Vorlesung 7, Folie 12), so dass das zugehörige Polynom jeweils eindeutig bestimmt ist.

b) Bestimmen Sie die zugehörigen Polynome zu den Vandermonde-Matrizen aus Aufgabe 1. Geben Sie die beiden Polynome im Horner-Schema (Vorlesung 7, Folie 17) an.

c) Geben Sie die Vandermonde-Matrizen zu den Mengen M_i an, die sich ergeben wenn die Monom-Basis verschoben und skaliert wird (Vorlesung 7, Folie 16).

d) Nutzen Sie die Lagrange-Interpolation (Vorlesung 7, Folie 18) um für die beiden Mengen M_i jeweils den interpolierten Wert an der Stelle 0 zu bestimmen.

e) Bestimmen Sie für jede der beiden Mengen M_i ein Polynom zur Interpolation mit Hilfe der Newtonschen Basis-Funktionen (Vorlesung 7, Folie 21). Stellen Sie für M_1 das zugehörige lineare Gleichungssystem auf und lösen Sie es. Nutzen Sie für M_2 das iterative Verfahren, welches auf Folie 26 beschrieben wird.

2 Dividierte Differenzen

Naive Implementierungen von rekursiv definierten Werten, wie z.B. den dividierten Differenzen (Kapitel 7, Folie 27), berechnen manchmal unnötigerweise manche Zwischenergebnisse mehrfach. Das kann im schlimmsten Fall dazu führen, dass die benötigte Rechenzeit exponentiell mit Größe des Problems (hier: Anzahl der Datenpunkte, die interpoliert werden sollen) steigt! Mit sogenannter dynamischer Programmierung, die Zwischenergebnisse geschickt speichert und wiederverwendet, kann dieses Problem umgangen werden.

a) Angenommen Sie haben eine naive Implementierung der dividierten Differenzen, die direkt den rekursiven Ausdruck aus der Vorlesung implementiert, um $x_j = f[t_1, \ldots, t_j]$ zu berechnen. Wie viele Funktionsaufrufe sind jeweils nötig um x_1, x_2, x_3 und x_4 zu bestimmen? Zählen Sie auch alle rekursiven Aufrufe mit, insbesondere ob manche Aufrufe mit gleichen Parametern mehrfach auftreten.

b) Durch eine geschickte Darstellung und Berechnungsreihenfolge lässt sich der unnötige Mehraufwand, der Teil einer naiven Implementierung ist, vermeiden. Seien y_s und t_s Vektoren der Länge n, welche zusammen die Datenpunkte $(t_1, y_1), \ldots, (t_n, y_n)$ beschreiben. Mit einer $n \times n$-Matrix A, deren Einträge $A_{ij} = f[t_i, \ldots, t_j]$ entsprechen, lassen sich die Koeffizienten x_j für die Newton-Interpolation effizient bestimmen.
Wo können Sie in dieser Matrix A die Koeffizienten x_j ablesen, sobald A vollständig berechnet wurde? Welche Einträge von A sind undefiniert? Welche Einträge der Matrix können Sie ohne Anwendung der rekursiven Formel in einer ersten Iteration bestimmen? Welche Einträge können Sie in einer zweiten Iteration bestimmen, die zwar eine Anwendung der rekursiven Formel erfordern, aber keine rekursiven Funktionsaufrufe, da die erforderlichen Ergebnisse dieser potentiellen rekursiven Aufrufe bereits in der Matrix A stehen? Wie sieht es in der dritten Iteration usw. aus?
Implementieren Sie eine MATLAB-Funktion `x = divDiff(ys, ts)`, welche die Matrix A aus den Vektoren y_s und t_s effizient berechnet und anschließend den Ergebnisvektor x aus A bestimmt. Sie können ihre Implementierung mit den Datenpunktmengen M_1 und M_2 aus dem Theorieteil testen. Ist ihre Implementierung korrekt, berechnet Sie die gleichen Koeffizienten x_j wie die, die Sie in der fünften Teilaufgabe berechnet haben.

c) Es ist gar nicht nötig die komplette Matrix A im Speicher zu behalten. Implementieren Sie eine Funktion `x = divDiff2(ys, ts)`, welche außer Zählvariablen für Schleifen und dem Ergebnisvektor x keinen weiteren Speicher anlegt. Insbesondere keinen Speicher für die Matrix A.
Tipp: Sie dürfen den Inhalt des Speichers, der durch die Eingabeparameter belegt ist, überschreiben.

3 Splines

In der Vorlesung wurden Splines zur stückweisen polynomialen Interpolation einer Menge von Stützstellen vorgestellt. Im Gegensatz zur Polynominterpolation liefert die Spline-Interpolation auch bei ungünstig gesetzten Stützstellen brauchbare Kurvenverläufe, d.h. es gibt keine unrealistischen Oszillationen.

a) Schreiben Sie eine MATLAB-Funktion, `B = bsplines(t,ts,maxDeg)` zur Berechnung von B-Splines bis zum maximalen Grad *maxDeg*. *t* soll dabei Zeitpunkte zur Evaluation der Splines enthalten und *ts* die Stützpunkte für die stückweise Definition der Splines. Die ausgegebene Matrix soll die Basisfunktionen $B_i^k(t)$ enthalten.
Stellen Sie die B-Splines bis zum Grad 3 mit geeigneten *t* und *ts* dar.
Hinweis: Die nötigen Formeln finden Sie in Kapitel 7 Folie 44-45.

b) Implementieren Sie die kubische Spline-Interpolation mit *n* Stützstellen. Nutzen Sie hierfür den Funktionskopf `coeff = cubicSplineInter(ys,ts)`. Dabei enthalten *(ys,ts)* die zu interpolierenden Datenpunkte. Die Funktion soll ein Array mit Polynomkoeffizienten $(\alpha_1, \alpha_2, \ldots) \in \mathbb{R}^{4(n-1)}$ ausgeben.

c) Wenden Sie Ihre Interpolation auf die Datenpunkte

$$\begin{aligned} y_s &= (0.2, -0.1, 1.5, 0.5) \\ t_s &= (-1.5, -0.2, 0.4, 1.5) \end{aligned}$$

an und stellen Sie die resultierende Kurve zusammen mit den Datenpunkten dar. Hierzu benötigen Sie eine Funktion `f = cubicSplineEval(t,ts,coeff)`, die Ihre Spline-Interpolation an den Punkten im Array *t* auswertet.