Winfried Auzinger
Gregor Gantner
Alexander Haberl
Dirk Praetorius

Sommersemester 2016
25. März 2015

Übungen zur Vorlesung Computermathematik

Serie 4

Aufgabe 4.1. Erweitern Sie den Code von Folie 110, indem Sie das Konvergenzverhalten des einseitigen und zentralen Differenzenquotientens mit dem Aitkenschen Δ^2-Verfahren aus Aufgabe 1 Übung 3 verbessern. Welche Konvergenzraten beobachten Sie für verschiedene Beispiele? Visualisieren Sie diese geeignet!

Aufgabe 4.2. Ziel ist die Implementierung des MergeSort-Algorithmus. Schreiben Sie dazu eine Funktion `mergesort` und eine Unterfunktion `merge` (im File `mergesort.m`) mit folgender Funktionalität:

- Die Funktion `merge` übernimmt zwei aufsteigend sortierte Zeilenvektoren $a \in \mathbb{R}^m$ und $b \in \mathbb{R}^n$ und vereinigt diese (mittels geeigneter Schleifen) so, dass der resultierende Zeilenvektor $c \in \mathbb{R}^{m+n}$ ebenfalls sortiert ist, z.B. soll $a = (1, 3, 3, 4, 7)$ und $b = (1, 2, 3, 8)$ als Ergebnis $c = (1, 1, 2, 3, 3, 3, 4, 7, 8)$ liefern. Dabei soll ausgenutzt werden, dass die Vektoren a und b bereits sortiert sind. Es soll insbesondere *kein* Sortierverfahren auf den vereinigten Vektor `[a,b]` angewendet werden, d.h. `sort` darf nicht verwendet werden!
- Die rekursive Funktion `mergesort` übernimmt einen unsortierten Zeilenvektor $c \in \mathbb{R}^N$ und soll c aufsteigend sortiert zurückgeben. Das Vorgehen ist wie folgt: Für $N \leq 2$ sortiert man c direkt. Für $N > 2$ zerlegt man c in zwei Hälften a und b, ruft `mergesort` rekursiv für a und b auf und verwendet `merge`, um aus den sortierten Hälften a und b den sortierten Vektor c zu erhalten.

Aufgabe 4.3. Schreiben Sie ein Skript, das die Laufzeiten von Mergesort für zufällige Vektoren $x \in \mathbb{R}^N$ und $N = 100 \cdot 2^n$ mit $n = 0, 1, 2, \ldots$ geeignet visualisiert. Zeichnen Sie geeignete Vergleichsgeraden in den Plot, um visuell den Aufwand Ihrer Implementierung zu bestimmen. Welchen Aufwand erwarten Sie theoretisch?

Aufgabe 4.4. Der folgende Code berechnet die sparse Matrix $A \in \mathbb{R}^{N \times N}$ (Der Code kann auf der COMPMATH Webseite heruntergeladen werden).

```
function A = matrix(N)
x = rand(1,N);
y = rand(1,N);
triangles = delaunay(x,y);
n = size(triangles,1);
A = sparse(N,N);

for i = 1:n
   nodes = triangles(i,:);
   B = [1 1 1 ; x(nodes) ; y(nodes)];
   grad = B \ [0 0 ; 1 0 ; 0 1];
   A(nodes,nodes) = A(nodes,nodes) + det(B)*grad*grad'/2;
end
```

Plotten Sie die Rechenzeit $t(N) = \mathcal{O}(N^\alpha)$ über N für $N = 100 \cdot 2^k$ und $k = 0, 1, 2, \ldots$. Welchen Aufwand beobachten Sie? Woran liegt das? Was kann getan werden, um das Laufzeitverhalten zu verbessern? Schreiben Sie einen verbesserten Code, der zu einer besseren Rechenzeit führt. Visualisieren Sie die Laufzeit in dem gleichen Plot, um zu zeigen, dass der neue Code tatsächlich schneller ist. Welchen Aufwand erwarten und beobachten Sie für Ihren verbesserten Code? **Hinweis:** Verwenden Sie `help sparse`.

Aufgabe 4.5. Seien $m, n, N \in \mathbb{N}$. Seien $I, J, a \in \mathbb{R}^N$ die Repräsentanten des Koordinatenformates einer sparse Matrix $A \in \mathbb{R}^{m \times n}$, d.h., für alle $k = 1, \ldots, N$ gilt $A_{ij} = a_k$ mit $i = I_k$, $j = J_k$. Schreiben Sie eine MATLAB Funktion

```
[II,JJ,AA] = naive2ccs(I,J,a,m,n),
```

welche die zugehörigen Vektoren des CCS Formats zurückgibt.

Aufgabe 4.6. Gegeben sei das CCS Format einer sparse Matrix $A \in \mathbb{R}^{m \times n}$ der letzten Aufgabe. Schreiben Sie eine MATLAB Funktion

```
Ax = mvm(II,JJ,AA,m,n,x)
```

welche die Matrix-Vektor Multiplikation $b = Ax \in \mathbb{R}^m$ für einen gegebenen Vektor $x \in \mathbb{R}^n$ berechnet. Die Komplexität des Codes soll $\mathcal{O}(N)$ sein. **Hinweis:** Sie können Ihren Code wie folgt verifizieren: Angenommen A ist eine sparse Matrix (z.B. die Tridiagonalmatrix von Folie 124 der Vorlesung). Dann erhält man in MATLAB das Koordinatenformat von A durch `[I,J,a] = find(A)`. Verwenden Sie den Code aus Aufgabe 4.5 zur Berechnung der Vektoren des CCS Formats und zum Vergleich des Ergebnisses Ihrer Funktion `mvm` mit der Matrix-Vektor Multiplikation `A*x` in MATLAB.

Aufgabe 4.7. Schreiben Sie eine Funktion `plotPotential`, welche die Funktion $f : [a, b]^2 \to \mathbb{R}$, das Intervall $[a, b]$ und eine Schrittweite $\tau > 0$ nimmt und die Projektion $f(x, y)$ auf die Ebene (d.h. `view(2)`) plottet. Fügen Sie dem Plot eine `colorbar` hinzu. Für die Visualisierung, verwenden Sie ein Tensorgitter der Schrittweite τ. Sie dürfen annehmen, dass f so implementiert ist, dass es Matrizen $x, y \in \mathbb{R}^{M \times N}$ als Eingabe nimmt und eine Matrix $z \in \mathbb{R}^{M \times N}$ der zugehörigen Funktionswerte zurückgibt, also $z_{jk} = f(x_{jk}, y_{jk})$. Optional, habe die Funktion den Eingabeparameter $n \in \mathbb{N}$. Zu gegebenem n, fügen Sie der Figure n (schwarze oder weiße) Kontourlinien hinzu. Zur Verifizierung Ihres Codes, Schreiben Sie ein MATLAB Skript, welches das Potential $f(x, y) = x \cdot \exp(-x^2 - y^2)$ aus den Vorlesungsunterlagen visualisiert.

Aufgabe 4.8. Schreiben Sie eine MATLAB Funktion `saveMatrix`, welche die Matrix $A \in \mathbb{R}^{M \times N}$ nimmt und via `fprintf` (siehe auch `help fopen`) in eine ASCII Datei `matrix.dat` speichert. Verwenden Sie `%1.16e` für die Matrixkoeffizienten! (Warum macht das Sinn?) Optional, habe die Funktion den String `name` als Eingabeparameter, wobei die Matrix in der ASCII Datei `name.dat` gespeichert werde. Um Ihren Code zu verifizieren, schreiben Sie ein MATLAB Skript, welches eine Zufallsmatrix $A \in \mathbb{R}^{M \times N}$ generiert und sie in einer ASCII Datei `A.dat` speichert. Laden Sie die Matrix via `B = load('A.dat')` und checken Sie, ob A und B übereinstimmen.