Prof. Dr. Stefan Funken
Kristina Steih
Institut für Numerische Mathematik
Universität Ulm

Numerik 1
WS 2011/12

Matlab-Blatt 6

(Abgabe bis spätestens Dienstag, den 31.01.2012, 10:00 Uhr per Mail (s.u.).)

Aufgabe 1 *(Preconditioned conjugate gradient method)* (13 Punkte)

a) Write a function `[x, its] = cg(A, b, x, tol)` that implements the conjugate gradient method for a s.p.d. matrix $A \in \mathbb{R}^{n\times n}$ (see also P5, Exercise 14), a vector $b \in \mathbb{R}^n$, an initial guess $x_0 \in \mathbb{R}^n$ and a tolerance *tol* that defines the stopping criterion: the *cg*-method has converged if $\|r_k\|_2 \leq tol$.
The output arguments are the solution $x \in \mathbb{R}^n$ of $Ax = b$ and the number *its* of iterations before convergence.

b) Similarly, implement the preconditioned conjugate gradient method (pcg) in a function `[x, its] = pcg(B, A, b, x, tol)`, where $B \in \mathbb{R}^{n\times n}$ is the preconditioner matrix.

c) Write a script `main1.m` in which you compare both methods. In detail, you should

- load the matrix A_1 from the file *A1.txt*,
- define the right hand side $b = A_1 \cdot (1, \ldots, 1)^T$, the initial guess $x_0 = (0, \ldots, 0)^T$ and the tolerance $tol = 10^{-10}$,
- solve $A_1 x = b$ with both the *cg*- and the *pcg*-method (using the diagonal preconditioner $B := \mathrm{diag}(A)^{-1}$) and
- print the number of iterations, the runtime and the resulting error $||x - x_{exact}||_2$ for both methods ($x_{exact} = (1, \ldots, 1)^T$).

d) What can you observe? Comment your results.

Aufgabe 2 *(Dünnbesetzte Matrizen (COO-Format))* (14 + 2 Punkte)

Iterative Verfahren eignen sich besonders gut für dünnbesetzte Matrizen, also für $A \in \mathbb{R}^{n\times n}$ mit

$$\#\{a_{ij} : a_{ij} \neq 0\} = \mathcal{O}(n),$$

weil im Wesentlichen in jeder Iteration nur ein Matrix-Vektor-Produkt berechnet werden muss. Bei dünnbesetzten Matrizen ist das jeweils ein Aufwand von $\mathcal{O}(n)$ – aber natürlich nur, wenn man die Struktur auch ausnutzt.

In dieser Aufgabe geht es um eine (simple) Speicherstruktur für dünnbesetzte Matrizen, das sogenannte Koordinatenformat (*COO*-Format, *Coordinate Storage*, siehe auch Kapitel C.1 im Skript). Dabei kann eine dünnbesetzte Matrix $A \in \mathbb{R}^{n\times n}$ effizient als Struktur mit den drei Feldern `A.aij` für die Einträge, `A.row` für die Zeilenindizes und `A.col` für die Spaltenindizes abgespeichert werden, wobei nur Einträge $a_{ij} \neq 0$ gespeichert werden (dabei ist die Reihenfolge beliebig).

Bsp:

$$A = \begin{pmatrix} 6 & 0 & -1 & 0 \\ 0 & 7 & 0 & 0 \\ -1 & 0 & 8 & -2 \\ 0 & 0 & -2 & 9 \end{pmatrix} \quad \ldots \text{ daraus folgt } \ldots$$

A.row	1	4	3	3	3	2	4	1
A.col	1	4	4	3	1	2	3	3
A.aij	6	9	-2	8	-1	7	-2	-1

Setzen Sie das *cg*-Verfahren für eine solche Speicherstruktur um, indem Sie

a) eine Funktion `Asparse = loadsparse(filename)` schreiben, welche eine dünnbesetzte Matrix $A_{sparse} \in \mathbb{R}^{n\times n}$ aus der Datei *filename* einliest. In der Datei stehen in der ersten Spalte die Zeilen-, in der zweiten Spalte die Spaltenindizes und in der dritten Spalte die dazugehörigen Matrixeinträge;

b) eine Funktion `y = mv(A,x)` implementieren, welche das Matrix-Vektor-Produkt $y = A \cdot x$ für eine Matrix A im obigen Speicherformat realisiert;

c) ihre Funktion `cg` aus Aufgabe 1 a) so abändern, dass Sie mit dünnen Matrizen umgehen kann (*Hinweis:* dazu müssen Sie nur die Matrix-Vektor-Produkte durch Ihre Funktion aus (b) ersetzen). Nennen Sie die abgeänderte Funktion `cg_sparse`.

d) ein Skript `main2.m` schreiben, in dem Sie mit Ihrer Funktion aus (a) die dünnbesetzte Matrix A_2 aus der Datei *A2_sparse.txt* einlesen. Laden Sie zusätzlich die gleiche Matrix im vollen Format aus der Datei *A2_full.txt*. Lösen Sie nun wieder $A_2 x = b$ mit $b = A_2 \cdot (1, \ldots, 1)^T$, $x_0 = (0, \ldots, 0)^T$ und $tol = 10^{-10}$ sowohl mit `cg` als auch mit `cg_sparse` und vergleichen Sie wieder Iterationszahl, Laufzeit und Fehler gegenüber der exakten Lösung.

e)* Mit dem Matlab-Befehl `A_sparseMatlab = sparse(A)` erhalten Sie aus der Matrix im vollen Format eine in der Matlab-internen dünnen Speicherstruktur. In Matlab sind viele Operationen, z.B. Matrix-Vektor-Produkte, sowohl für voll- als auch für dünnbesetzte Matrizen realisiert. Sie können Ihre Funktion `cg` also direkt mit dieser Matrix $A_{sparseMatlab}$ verwenden. Wie verhält sich hierbei die Laufzeit?

Senden Sie alle Dateien (*m*-Files, Plots, Erklärungen) in einer Email mit dem Betreff `Num1-BlattM6` an kristina.steih@uni-ulm.de. Bitte alle Dateien in ein zip-File packen, welches die Namen der Studenten enthält, also z.B. `M6_Student1_Student2.zip`. Aus der Email sollte zusätzlich klar hervorgehen, von welchen beiden Studenten die Lösung ist. Bitte schreiben Sie außerdem dazu, in welcher Übungsgruppe (Wochentag und A bzw. B) Sie jeweils sind.