Interdisziplinäres Zentrum
für Wissenschaftliches Rechnen (IWR)
Ruprecht–Karls–Universität Heidelberg

Dr. Peter Zaspel

Übungsblatt 11

04.07.2016 – 11.07.2016

Einführung in die Numerik SS 2016

Zu den Aufgaben auf diesem Zettel: Auf diesem Übungzettel soll zur Vorbereitung der Klausur die Berbeitung von „praktischen Aufgaben auf Papier“ geübt werden. Die Aufgaben 4, 5 und 6 sind derartige Aufgaben.
Zur Wertung: Aufgabe 4 zählt als Theorie-Aufgabe, d.h. die Aufgaben 1 bis 4 zählen wie immer für die Theorie-Punkte. Dahingegen werden die Aufgaben 5 und 6 als praktische Aufgaben gewertet.
Zur Abgabe: Die Aufgaben 4, 5 und 6 geben Sie bitte in der Vorlesung gesondert ab. Es wird eine entsprechende Mappe bereitgestellt. Notieren Sie auf diesen gesonderten Abgaben bitte, in welcher Theorie-Übungsgruppe Sie sind. Die Besprechung erfolgt **in den Programmierübungen**, d.h. statt ihr Programm vorzuzeigen, bekommen Sie von dem Programmiertutor die korrigierte Lösung, über die Sie kurz sprechen können.

Aufgabe 1. Irreduzible Matrizen (6 Punkte)
Man zeige die Equivalenz der beiden nachfolgenden Definitionen für die „Irreduzibilität“ einer Matrix:
a) Eine Matrix $A \in \mathbb{R}^{n\times n}$ m heißt “irred uzibel”, wenn es keine Partitionierung J, K von $\{1,\ldots,n\} =: \mathbb{N}_n$ mit $J \cup K = \mathbb{N}_n$ und $J \cap K = \emptyset$ gibt, so daß $a_{jk} = 0$ ist für alle $j \in J$ und alle $k \in K$.

b) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt “irreduzibel”, wenn es zu jedem Indexpaar $j, k \in \mathbb{N}_n$ eine Teilmenge $\{i_1,\ldots,i_m\} \in \mathbb{N}_n$ gibt, so daß $a_{j,i_1} \neq 0, a_{i_1,i_2} \neq 0, \ldots, a_{i_{m-1},i_m} \neq 0, a_{i_m,k} \neq 0$.

Aufgabe 2. Jacobi- und Gauß-Seidel-Verfahren (5 Punkte)
Man möchte das lineare Gleichungssystem

$$\begin{aligned} 3x - y &= 1 \\ -x + 3y &= -1\,, \end{aligned}$$

mit dem Jacobi- und dem Gauß-Seidel-Verfahren lösen.

a. Man führe jeweils zwei Iterationen mit dem Startwert $(0,0)^T$ aus.

b. Man schätze die Anzahl Iterationen k, um den Anfangsfehler in der l^2-Norm um den Faktor 10^{-6} zu reduzieren, nach oben ab. Wir nehmen hier als Näherung

$$k \approx \frac{\ln(TOL)}{\ln(\rho)}\,.$$

Aufgabe 3. (4 Punkte)

Für eine Matrix $A \in \mathbb{R}^{n \times n}$ zeige man die Äquivalenz

$$\rho(A) < 1 \iff \lim_{k \to \infty} A^k = 0\,.$$

Hinweis zu den Programmieraufgaben:
Im Rahmen der *Programmieraufgaben* sollen jeweils einzelne C/C++ Funktionen geschrieben werden. Hierbei soll klassische C/C++–Syntax verwendet werden. Der Einsatz der `pow`/`exp`/`sqrt`-Funktion sowie die Verwendung von anderen in C/C++ definierten oder über Header-Dateien einbindbaren Mathematik-Funktionen ist, sofern nicht anders angegeben, **nicht** gestattet. Das Schreiben eines Hauptprogrammes ist nicht nötig. **Kleinere syntaktische Fehler im C/C++ - Quelltext führen hier nicht zum Punktabzug, solange der eigentliche Algorithmus verständlich ist.**
Zur Verwendung von Arrays: Auf den $i$ten Eintrag eines Array `double* x` wird mittels `x[i]` zugegriffen. Analog wird auf den Eintrag (i, j) eines zwei-dimensionalen Arrays `double** A` mittels `A[i][j]` zugegriffen. Die Indizierung beginnt bei 0. Es kann stets davon ausgegangen werden, dass die entsprechenden Arrays vor dem Funktionsaufruf mit ausreichend Speicherplatz angelegt wurden.

Aufgabe 4. Theoretische Programmieraufgabe: SOR-Verfahren (5 Punkte)
Es sei eine Matrix $A \in \mathbb{R}^{n \times n}$ und eine rechte Seite $b \in \mathbb{R}^n$ gegeben. Implementieren Sie die Funktion `void sor(double** A, double* b, double* x1, double* x0, double omega, int n)`, die aus einer alten Iterierten `x0` $\in \mathbb{R}^n$ eine neue Iterierte `x1` $\in \mathbb{R}^n$ mit Hilfe des SOR-Verfahrens berechnet. Die Arrays `A`, `b` und `x0` sowie die Variablen `n` und `omega` seien dabei mit geeigneten Werten gefüllt. Weiter können Sie davon ausgehen, dass `x1` als Nullvektor initialisiert wurde.

Aufgabe 5. Theoretische Programmieraufgabe: Cholesky-Zerlegung (5 Punkte)
Die Cholesky-Zerlegung $A = LL^\top$ für symmetrische und positiv definite Matrizen $A \in \mathbb{R}^{n \times n}$, wobei $L \in \mathbb{R}^{n \times n}$ eine linke untere Dreiecksmatrix ist, lässt sich mit Hilfe des Algorithmus

$$\begin{aligned}
&\text{Für } j = 1, ..., n: \\
&\quad \text{falls } j = 1: \\
&\qquad l_{jj} = \sqrt{a_{jj}} \\
&\quad \text{falls } j > 1: \\
&\qquad l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} (l_{jk})^2} \\
&\quad \text{für } i = j+1, ..., n: \\
&\qquad l_{ij} = l_{jj}^{-1} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right)
\end{aligned}$$

berechnen, wobei $A = (a_{ij})_{i,j=1}^n$, $L = (l_{ij})_{i,j=1}^n$ gilt.

Schreiben Sie eine Funktion `void cholesky(double** A, double** L, int n)`, die die Cholesky-Faktorisierung durchführt, d.h. für eine gegebene Matrix A den Faktor L berechnet. Sie dürfen hierbei die Quadratwurzelfunktion `double sqrt(double x)` aus der `math.h`-Datei verwenden. Sie können davon ausgehen, dass die Matrix L auf eine Nullmatrix initialisiert wurde.

Aufgabe 6. Code-Verständnis (1+2+2 Punkte)
Gegeben sei das nachfolgende C/C++–Programm:

```
#include <stdio.h>
#include <math.h>

double s(double* a, double* b, int n)
{
        double r = 0;

        for (int i=0; i<n; i++)
                r = r + a[i]*b[i];

        return r;
}


int main(void)
{
        double psi[3][3];
        double phi[3][3];

        double n_phi;

        psi[0][0] = 1.0;  psi[1][0] = 1.0;  psi[2][0] = 1.0;
        psi[0][1] = 0.0;  psi[1][1] = 1.0;  psi[2][1] = 1.0;
        psi[0][2] = 0.0;  psi[1][2] = 0.0;  psi[2][2] = 1.0;


        for (int i=0; i<3; i++)
                phi[0][i] = psi[0][i];

        n_phi = s(phi[0], phi[0], 3);
        n_phi = sqrt(n_phi);

        for (int i=0; i<3; i++)
                phi[0][i] = phi[0][i] / n_phi;

        for (int k=1; k<3; k++)
        {
                for (int i=0; i<3; i++)
                        phi[k][i] = psi[k][i];

                for (int j=0; j<k; j++)
                {
                        double s_psi_phi = s(psi[k], phi[j], 3);
```

```
                        for (int i=0; i<3; i++)
                                phi[k][i] = phi[k][i] - s_psi_phi * phi[j][i];
                }

                n_phi = s(phi[k],phi[k], 3);
                n_phi = sqrt(n_phi);

                for (int i=0; i<3; i++)
                        phi[k][i] = phi[k][i] / n_phi;
        }


        for (int i=0; i<3; i++)
        {
                printf("phi_%d:\n",i);
                for (int j=0; j<3; j++)
                        printf("%lf\n", phi[i][j]);
                printf("\n");
        }

        for (int i=0; i<3; i++)
        {
                for (int j=0; j<3; j++)
                        printf("%lf ", s(phi[i], phi[j], 3));
                printf("\n");
        }



        return 0;
}
```

a. Was macht die Funktion `double s(double* a, double* b, int n)`?

b. Welcher Algorithmus wird im Hauptprogramm umgesetzt ?

c. Wie lautet die Bildschirmausgabe des Programms ?
Hinweise: Sie brauchen hier nicht die exakte Bildschirmausgabe mit der genau richtigen Nachkommastellenzahl etc. wiederzugeben. Es reicht, wenn Sie selbst das Ergebnis errechnen und ggf. analytisch analog zur echten Bildschirmausgabe anzugeben.

Abgabe: 11.07.2016, 14:00-14:15 Uhr. (Mappen in der Vorlesung)