OTH Regensburg

Studiengang
IN, IT B. Sc.
Prof. Dr. Martin Weiß
Sommersemester 2019

Computerarithmetik und Rechenverfahren

4. Praktikum: Kondition und Gleichungssysteme

4.1. Beispiel aus der Vorlesung

a) Berechnen Sie von Hand die LR-Zerlegung (ohne Pivotisierung) von

$$A = \begin{bmatrix} 2 & 4 & 2 & 6 \\ 1 & 3 & 1 & 2 \\ 1 & 7 & 3 & 8 \\ 3 & 1 & 0 & -3 \end{bmatrix}$$

und lösen Sie damit $Ax = b$ mit der rechten Seite $b = [4,2,4,3]^t$.

b) Prüfen Sie z.B. durch Multiplikation mit MATLAB nach, ob tatsächlich $A = LR$ gilt.

c) Berechnen Sie die *LR*-Zerlegung mit dem MATLAB-Kommando `lu` (englisch: *LU*-Zerlegung, in lower und upper triangular matrix). Dabei wird eine Zerlegung *bei Bedarf, und das schließt Bedarf für günstiges numerisches Verhalten mit ein,* **mit** Pivot-Strategie und Permutationsmatrix bestimmt. Können Sie die Matrix A daraus rekonstruieren?

d) Berechnen Sie mit der *LR*-Zerlegung $\det(A)$. Bestimmen Sie die Inverse A^{-1}, indem Sie über die *LR*-Zerlegung $Ax = e_i$ für $i = 1,\ldots,n$ lösen, dabei e_i der i-te Einheitsvektor in $\mathbb{R}^n$. Vergleichen Sie mit dem Ergebnis des MATLAB-Befehls `inv`.

4.2. Implementieren Sie Routinen zur Vorwärtselimination und Rückwärtssubstitution. Arbeiten Sie vektoriell, d.h. verwenden Sie so wenige explizite MATLAB-Schleifen wie möglich! Vorlagen mit vielen Schleifen finden Sie im Skript.

Verwenden Sie zum Testen die Matrizen und Vektoren vom Arbeitsblatt:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 3 & 1 \end{bmatrix}, \qquad R = \begin{bmatrix} 2 & 1 & 3 \\ 0 & -1 & 2 \\ 0 & 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 9 \\ -10 \\ 21 \end{bmatrix}$$

$$Lz = b \Leftrightarrow z = \begin{bmatrix} 9 \\ 8 \\ 6 \end{bmatrix} \qquad Rx = z \Leftrightarrow x = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

4.3. Einführung in automatisierte Tests mit MATLAB

Zu jedem Programm sollten Sie automatisierte Tests schreiben - zahlt sich langfristig aus. Wir wollen solche Tests für die *LR*-Zerlegung samt Vorwärtselimination und Rücksubstitution schreiben.

In der Datei `testDreieck` finden Sie eine Sammlung von 4 Testfunktionen, die dann mit

```
runtests('testDreieck')
```

über die Hauptfunktion `testDreieck` aufgerufen werden:

```
function tests = testDreieck(testCase)
tests = functiontests(localfunctions);
end
```

Die Funktion `functiontests` ermittelt (recht kompliziert, Implementierung mit `edit functiontests` mal anschauen...) alle lokalen Funktionen, ruft diese auf und sammelt die Ergebnisse über eine Variable `testCase`, die passend angelegt wird. In unserem Fall werden die Ergebnisse im command window angezeigt.

Die einzelnen Testfunktionen verwenden "verifiables", die verschiedene "qualifications" wie Gleichheit, Ungleichheit, größer-als usw. testen.

Die Funktion `testUpperFloat` liefert einen Fehler, da Fließkommazahlen auf Gleichheit getestet wurden (was Sie natürlich aufgrund Ihres Wissens nie machen würden).

Die Funktion `testLR` ist nicht fertig implementiert, und ruft gar kein verifiable auf. Damit gilt diese Funktion als bestanden.

a) Modifizieren Sie die Funktion `testUpperFloat` mit einem passenden verifiable, das Sie aus den vorhandenen auswählen. Suchen Sie mit `doc verifyEqual` als Start.

b) Ergänzen Sie Testfälle für Ihre vektorisierten Varianten von Vorwärtselimination und Rückwärtssubstitution.

c) Füllen Sie die Funktion `testLR` mit Leben.

Bemerkung: Die hier verwendete Teststrategie verwendet Funktionentests. Man kann auch einen objektorientierten Ansatz wählen, aber zu Klassen in MATLAB kommen wir aus Zeitgründen nicht. Objektorientierung und Klassen sind als nachträglich eingebautes MATLAB-Feature auch nicht so komfortabel und natürlich implementiert, wie Sie es von C++/Java/C#/... kennen.

4.4. Hilbertmatrix

a) Erzeugen Sie in Matlab mit dem Befehl `hilb(m)` eine sogenannte Hilbert-Matrix H_m und geben Sie für $m = 1,2,3,4,5$ jeweils die Matrix mit Zahlenformat für Brüche `format rat` aus.

b) Geben Sie eine Formel für den Eintrag (i, j) von H_m an.

c) Bestimmen Sie für H_5 mit Matlab-Befehlen die Matrix $H_5^{-1} * H_5$ und vergleichen Sie mit dem exakten Ergebnis. Wie groß ist die Kondition der Matrix H_5? Lesen Sie die Dokumentation des Befehls `cond` und wenden Sie diesen auf die Matrix an.

d) Behandeln Sie die Fragen aus dem vorigen Punkt für die Matrix H_{20}. Interpretieren Sie die Ergebnisse mit der "Faustregel" für die Stellenverluste beim Lösen von Gleichungssystemen in Abhängigkeit der Kondition.
Faustregel: Eine Konditionszahl $\kappa(A) = 10^q$ kostet q Dezimalstellen Genauigkeit bei der Lösung von $Ax = b$.

e) Erzeugen Sie die Hilbert-Matrix über eine eigene Funktion, die *keine* Schleifen verwendet. Vergleichen Sie Ihre Funktion mit der MATLAB-Implementierung. Mit `which hilb` erfahren Sie, wo der MATLAB-Code abgelegt ist.

f) Wir setzen $x_m = [1, 1, \ldots, 1]^t \in \mathbb{R}^m$, und $b_m := H_m x_m$. Dann gilt $b_{m,i} = \sum_{j=1}^{m} \frac{1}{i+j-1}$. Berechnen Sie für $m = 3, \ldots, 20$ numerisch die Lösung aus x_m aus b_m. Vergleichen Sie mit der bekannten Lösung, und erklären Sie durch die Kondition!

4.5. Logische Operatoren - Vorbereitung Pivot-Strategie Gauß-Algorithmus

a) Verwenden Sie die Funktionen `all`, `any`, um festzustellen, ob mindestens ein Eintrag bzw. alle Einträge einer Matrix mit ganzzahligen Einträgen durch 3 teilbar sind!

b) Schreiben Sie eine Funktion `nonneg(v)`, die die nichtnegativen Einträge des Arguments `v` zurückliefert, z.B. `nonneg([-1 0.4 -2 3 0])` liefert

```
0.4000    3.0000         0
```

c) Schreiben Sie eine Funktion `[value, index] = pivot1(v)`, die den betragsgrößten Eintrag eines Vektors und den zugehörigen Index liefert bzw. 0 und -1, wenn alle Einträge bis auf Maschinengenauigkeit 0 sind.

d) Schreiben Sie eine Funktion `[value, index] = pivot2(v, start)`, die den betragsgrößten Eintrag eines Vektors in den Komponenten ab Index `start` und den zugehörigen Index liefert bzw. 0 und -1, wenn alle Einträge bis auf Maschinengenauigkeit 0 sind.

4.6. ** Gauß-Algorithmus, (*** mit Pivot). Vollziehen Sie zumindest die Lösungen der Gleichungssysteme nach!

a) Schreiben Sie ein MATLAB-Programm zur Lösung eines linearen Gleichungssystems

$$Ax = b, \qquad A \in \mathrm{Mat}_{n,n}(\mathbb{R}), b \in \mathbb{R}^n$$

mit dem Gauß-Algorithmus zuerst ohne, dann mit Spaltenpivotsuche. Verwenden Sie die bereitgestellten Routinen zur Lösung von Systemen mit oberer bzw. unterer Dreiecksmatrix (Vorwärtselimination, Rückwärtssubstitution)

Dann sollen Lösungen für verschiedene rechte Seiten b berechnet werden. Verwenden Sie folgende Funktionsköpfe:

```
function [L, R, P, err] = GaussZerlegung(A)
function x = GLSmitGaussZerlegung(L, R, P, b)
```

Das Fehlerflag `err` soll gesetzt werden, wenn die Lösung wegen Singularität von A nicht berechnet werden kann, oder wenn Ihr Verfahren keine Pivotstrategie implementiert hat, diese aber wegen eines 0-Eintrags erforderlich wäre.

b) Testen Sie Ihre Implementierung mit

i) A, b wie in Aufgabe 3.4; vergleichen Sie auch die Matrizen L und R. Vergleichen Sie mit den Ergebnissen der MATLAB-Funktion `lu`

ii) eigenen Testfällen, bei denen L, R, P die Lösung x einfach erkennbar sind, bzw. Sonderfälle des Algorithmus auftreten

iii) durch Vergleich mit der MATLAB-Methode `A\b` zur Lösung von linearen Gleichungssystemen

c) Hilbertmatrix unterschiedlicher Größe (bauen Sie diese *ohne* Schleifen auf!)

$$\begin{aligned} a_{ij} &= \frac{1}{i+j-1}, \quad 1 \le i,j \le n \\ b_i &= \sum_{j=1}^{n} j \cdot a_{ij}, \quad 1 \le i \le n \end{aligned}$$

für $n = 3,\ldots,20$, mit exakter Lösung

$$x = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n \end{bmatrix}$$

Bestätigen Sie, dass $Ax = b$ gilt. Die Lösung mit Ihrer Gauß-Variante liegt vermutlich weit daneben. Erklären Sie die schlechte Genauigkeit durch die Kondition!

d) pathologisches Beispiel von Wilkinson (bauen Sie diese Matrix *ohne* Schleifen auf! MATLAB-Befehle `triu`, `tril`)

$$A \in \mathrm{Mat}_{n,n}(\mathbb{R}) : A = \begin{bmatrix} 1 & 0 & \ldots & 0 & 1 \\ -1 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ -1 & \ldots & \ldots & -1 & 1 \end{bmatrix} \qquad b \in \mathbb{R}^n : b = \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix}$$

Vergleichen Sie die Norm von A mit der Norm der Dreiecksmatrix R am Ende des Gauß-Verfahrens (für $\|\cdot\|_1$ und / oder $\|\cdot\|_\infty$).

4.7. Import und Export von Daten

Variablen aus dem Workspace können Sie mit `save` und `load` speichern und wieder laden. Es gibt neben der Kommando-Variante auch eine funktionale Variante, bei der Variablen z.B. für den Dateinamen verwendet werden können:

```
save <filename>
save <filename> Var1 Var2 -ascii
fileName = 'myName'
save(fileName, Var1, Var2, '-mat');
save(fileName, Var1, Var2, '-ascii');

load <filename>
```

4.8. Wenn Ihnen manche mathematischen oder MATLAB-Konstrukte kryptisch vorkommen: Lesen Sie "C und Unix - alles Quatsch", z.B. `https://debianforum.de/forum/viewtopic.php?f=15&t=5901`.

Zu den fehlenden Operatoren ++, - und Zuweisungen +=, -=, ... siehe die länglichen Diskussionen auf den MATLAB-Seiten. Nach meinem Verständnis der MATLAB-Interna könnte man MATLAB um +=, -=, ... erweitern, ++, - passt wirklich nicht ins MATLAB-Konzept.