Universität zu Köln

Prof. Dr. Angela Kunoth
Max Glembock, Laslo Hunhold

Matlab Präsenzaufgaben — WS 2017/2018

Übungsblatt 7

Ausgabe: 29.11.2017

Dünnbesetzte Matrizen

Dünnbesetzte Matrizen (englisch: „sparse matrices“) sind Matrizen, bei denen ein großer Teil der Einträge aus Nullen besteht. Solche Matrizen spielen in der Numerischen Mathematik (bspw. bei der Diskretisierung von partiellen Differentialgleichungen) eine wichtige Rolle. Ein typisches Beispiel sind Bandmatrizen, speziell Tridiagonalmatrizen, die nur in bestimmten Nebendiagonalen Einträge ungleich Null enthalten. Bei quadratischen Matrizen mit n^2 möglichen Einträgen haben dünnbesetzte Matrizen oft nur $O(n)$ oder $O(n \cdot \log n)$ viele Einträge ungleich Null. Diese Eigenschaft gilt es zu nutzen.
Für gewöhnlich werden (in Matlab) die Einträge einer Matrix intern spaltenweise hintereinander abgelegt. Dies hat den Vorteil, dass bei vollbesetzten Matrizen auf einzelne Einträge schnell zugegriffen werden kann und Matrixoperationen schnell ausgeführt werden können. Bei dünnbesetzten Matrizen hat dies aber Nachteile:

- Alle Nulleinträge werden genauso gespeichert wie Einträge ungleich Null. Dies belegt unnötig viel Speicherplatz.
- Operationen werden mit Nulleinträgen genauso ausgeführt wie Operationen mit Einträgen ungleich Null. Dies kostet unnötige CPU-Zeit.

Beide genannten Punkte kann man mithilfe einer anderen Speicherung der Matrizen optimieren. Matlab bietet hierfür die Speichermethode `sparse` an, die speziell für dünnbesetzte Matrizen geeignet ist und nur die Einträge ungleich Null speichert. Mathematisch sind voll- und dünnbesetzte Matrizen äquivalent, jedoch gilt es, diese numerisch anders zu interpretieren. Bei einer `sparse`-Speicherung speichert Matlab nur die Position des Eintrags innerhalb der Matrix sowie den Eintrag selbst (also insgesamt 2 bzw. 3 Werte pro Eintrag). Die Vor- und Nachteile dieser Speicherung werden wir im Folgenden untersuchen.
Zu einer bereits initialisierten Matrix `A` kann das `sparse`-Pendant in Matlab durch den Befehl `sparse(A)` erzeugt werden. Möchte man dagegen eine Matrix `B` aus dem `sparse`-Format in das standardmäßige, vollbesetzte Format transformierten, so geht dies mit `full(B)`. Operationen mit oder auf `sparse`-Matrizen haben grundsätzlich auch `sparse`-Matrizen als Ausgabe.

Aufgabe 27 (Transformation und Speicherung von `sparse`-Matrizen)

a) Gegeben sei die Matrix

$$A = \begin{pmatrix} 1 & 9 & 0 & 0 \\ 0 & 4 & 3 & 0 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & 3 & 1 \end{pmatrix}.$$

Initialisieren Sie diese Matrix in Matlab und wandeln Sie sie in das `sparse`-Format um. Lassen Sie sich sich die Matrix ausgeben und greifen Sie auf verschiedene Einträge zu. Wandeln Sie anschließend die Matrix wieder in das `full`-Format um.

b) Die bekannteste dünnbesetzte Matrix ist die Einheitsmatrix. Zur Erstellung einer Einheitsmatrix im `sparse`-Format bietet Matlab den direkten Befehl `speye(m,n)`. Erstellen Sie eine 1000×1000 Einheitsmatrix `I_full` im `full`-Format und eine 1000×1000 Einheitsmatrix `I_sparse` im `sparse`-Format. Lassen Sie sich durch den Befehl `whos I_full` bzw. `whos I_sparse` Informationen über die Speicherung der Variablen ausgeben und vergleichen Sie den Platzverbrauch beider Speichervarianten.

c) Erzeugen Sie eine 1000×1000 Zufallsmatrix mit dem Befehl `rand(m,n)` und speichern Sie dieselbe Matrix ebenfalls im `sparse`-Format. Lassen Sie sich hier ebenfalls von beiden Matrizen die Informationen ausgeben. Was fällt auf?

Wichtig: Die letzte Aufgabe hat gezeigt, dass die Speicherung im `sparse`-Format wirklich nur für dünnbesetzte Matrizen geeignet ist. Eine unangebrachte Verwendung des `sparse`-Formats kann den Speicherverbrauch einer Variablen unnötig erhöhen.

Man kann `sparse`-Matrizen in Matlab auf unterschiedlichen Wegen erzeugen. Die direkte Transformation einer Matrix `A` über `sparse(A)` sowie die Erzeugung der Einheitsmatrix über `speye` kennen wir bereits. Weitere Möglichkeiten der Erzeugung sind:

- `sparse(m,n)`: Erzeugt eine $m \times n$-Matrix mit Nulleinträgen.
- `sparse(i,j,v)`: Für n-dimensionale Vektoren i, j und v wird eine Matrix S mit Einträgen $S(i(k), j(k)) = v(k)$ für $k \in \{1, \ldots, n\}$ erzeugt. Die Dimension der Matrix ist $\max_k\{i(k)\} \times \max_k\{j(k)\}$ und Einträge an den gleichen Positionen werden aufsummiert.
- `sparse(i,j,v,m,n)`: Analog zu `sparse(i,j,v)`, zusätzliche Festlegung der Dimension auf $m \times n$.
- `sprand(m,n,dens)`: Erzeugt eine $m \times n$-Matrix mit (ungefähr) $dens \cdot m \cdot n$ gleichverteilten Einträgen zwischen 0 und 1 (`sparse`-Pendant zu `rand`). *dens* beschreibt die Dichte der Matrix und es gilt $0 \leq dens \leq 1$.
- `spdiags(B,d,m,n)`: Erzeugt eine $m \times n$-Bandmatrix aus den Zeilen der Matrix `B`, deren Einträge gemäß der vorgegebenen Positionen im Vektor `d` angeordnet werden (`sparse`-Pendant zu `diag`).

Wir möchten den Umgang mit diesen Funktionen mit folgender Aufgabe festigen.

Aufgabe 28 (Erstellung von `sparse`-Matrizen)

a) Gegeben seien die Vektoren

$$i = \begin{pmatrix} 6, & 6, & 6, & 5, & 10, & 10, & 9, & 9 \end{pmatrix}^T,$$
$$j = \begin{pmatrix} 1, & 1, & 1, & 2, & 3, & 3, & 10, & 10 \end{pmatrix}^T,$$
$$v = \begin{pmatrix} 7, & -2, & 3, & 0, & -4, & -7, & 3, & 8 \end{pmatrix}^T.$$

Initialisieren Sie diese Vektoren in Matlab und lassen Sie sich die Matrix `sparse(i,j,v)` ausgeben. Was fällt Ihnen auf?

b) Erzeugen Sie eine quadratische Zufallsmatrix A mit Dimension $n = 100$ und einer Dichte von 25%. Wie viele nichttriviale Einträge erwarten Sie in der Matrix? Lassen Sie sich mit `nnz(A)` die Anzahl der Einträge ungleich Null ausgeben. Wie erklärt sich eine mögliche Abweichung von der erwarteten Anzahl an Einträgen?

c) Gegeben Sei die Matrix

$$B = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

sowie der Verteilungsvektor $d = \begin{pmatrix} -2, & 0, & 2 \end{pmatrix}^T$. Seien $m = 5$ und $n = 4$. Lassen Sie sich die `sparse`-Matrix `spdiags(B,d,m,n)` im `full`-Format ausgeben und werden Sie sich der Erstellung dieser Bandmatrix bewusst.

Der Umgang mit `sparse`-Matrizen ist jedoch nur im Falle dünnbesetzter Matrizen empfehlenswert. Viele integrierte Funktionen von Matlab können erkennen, ob eine übergebene Matrix im `sparse`-Format oder im `full`-Format vorliegt. Entsprechend verwendet Matlab auch für beide Formate unterschiedliche Algorithmen.

Wir möchten uns im Folgenden mit den Auswirkungen von `sparse`-Matrizen auf die Rechenzeit beschäftigen. In Matlab kann man die Rechenzeit, in der ein Algorithmus bzw. ein Code-Fragment durchlaufen wird, mit den Befehlen `tic` und `toc` messen. Der Startbefehl `tic` wird dabei vor das Code-Fragment gesetzt und der Stoppbefehl `toc` dahinter. Die gemessene Zeit wird nach einem Durchlauf im Command Window ausgegeben und kann durch Initialisieren von `toc` gespeichert werden. Eine Implementierung für eine zu messende Funktion `myfunction(A)` sieht also wie folgt aus:

```
tic
myfunction(A)
toc
```

Bemerkung: Man mache sich bewusst, dass es in Matlab zwei Möglichkeiten des Zugriffs auf Matrixeinträge gibt. Zum einen gibt es für eine Matrix `A` die bekannte zweidimensionale Indizierung `A(i,j)`, zum anderen gibt es die **lineare Indizierung** `A(k)`. Dabei gilt für alle $m \times n$-Matrizen:

$$A(k) = A(m \cdot (j-1) + i) = A(i, j)$$

mit $k \in \{1, \ldots, m \cdot n\}$ und $i \in \{1, \ldots, m\}$ sowie $j \in \{1, \ldots, n\}$. Diese lineare Indizierung resultiert aus der spaltenfortlaufenden internen Speicherung von Matlab.

Aufgabe 29 (Rechenzeiten von `sparse`-Matrizen)

a) Schreiben Sie eine Funktion `max_for(n,dens)`, in der Sie für eine n-dimensionale Zufallsmatrix `A` im `sparse`-Format mit Dichte *dens* über **eine** `for`-Schleife mittels **linearer Indizierung** eine Bestimmung des Maximums durchführen. Fügen Sie eine weitere Schleife ein, um für ein transformiertes Pendant von `A` im `full`-Format das Maximum zu bestimmen. Messen Sie für beide Bestimmungen die Zeit und lassen Sie sich diese für $n = 1000$ und $dens = 0.005$ ausgeben.

b) Schreiben Sie eine Funktion `max_matlab(n,dens)`, in der Sie abermals für eine n-dimensionale Zufallsmatrix `B` im `sparse`-Format sowie auch im `full`-Format das Maximum bestimmen. Nutzen Sie diesmal jedoch den Befehl `max(B(:))`. Messen Sie für beide Datenformate die Zeit für $n = 1000$ und $dens = 0.005$.

Wie sind die Ergebnisse von a) im Vergleich zu den Ergebnissen von b) zu interpretieren?

Wichtig: Durch die vorherige Aufgabe sollte noch einmal deutlich geworden sein, dass der Gebrauch von `for`-Schleifen oftmals äußerst ineffizient ist. Bei `sparse`-Matrizen ist die Auswirkung dieser Ineffizienz aufgrund der internen Speicherung nochmal gravierender. Es empfiehlt sich daher (gerade für `sparse`-Matrizen) auf die bereits integrierten Funktionen von Matlab zurückzugreifen, sofern dies möglich ist.

Ein häufig verwendeter Befehl im Umgang mit `sparse`-Matrizen ist der Befehl `nonzeros`. Dieser erstellt einen Spaltenvektor aus einer übergebenen Matrix `A`, der alle Einträge ungleich Null fortlaufend gemäß den Spalten von `A` speichert.
Ein weiterer wichtiger Befehl ist `find`. Zu einer übergebenen $m \times n$-Matrix `A` gibt `z=find(A)` einen $(m+n)$-dimensionalen Vektor `z` mit allen Positionen von Einträgen ungleich Null in `A` gemäß linearer Indizierung wieder. Für `[x,y]=find(A)` erhält man einen m-dimensionalen Vektor `x` und einen n-dimensionalen Vektor `y` gemäß der gebräuchlicheren zweidimensionalen Indizierung.

Bei einem häufigen Umgang mit `sparse`-Matrizen ist es manchmal wünschenswert, sich das Besetzungsmuster seiner `sparse`-Matrix ausgeben zu lassen. Dies kann für eine Matrix `A` direkt über den Befehl `spy(A)` erfolgen. `spy` ist eine Plot-Funktion, die speziell für die Darstellung von Besetzungsmustern entwickelt ist. Insofern bietet sie auch nur geringe Formatierungsmöglichkeiten. So lässt sich beispielsweise über `spy(A,s)` durch einen Skalar `s` direkt die Punktgröße des Plots einstellen.

Aufgabe 30 (Besetzungsstruktur von `sparse`-Matrizen)
Initialisieren Sie die Matrix `B=bucky`, wobei `bucky` die in Matlab bereits integrierte 60×60-Adjazenzmatrix des Graphen eines Carbon-60-Moleküls sowie auch eines Fußballs beschreibt.

a) Lassen Sie sich mittels `[x,y]=find(B)` die Koordination der Einträge ungleich Null von `B` wiedergeben. Stellen Sie nun die Besetzungsstruktur von `B` über den Plotbefehl `plot(x,y)` dar.

b) Lassen Sie sich ferner die Besetzungsstruktur direkt mittels `spy(B)` ausgeben.

Wie erklärt sich der Unterschied?