Kurzeinführung Matlab

Sven Beuchler

24. März 2015

Inhaltsverzeichnis

1 Was ist MATLAB ? 1

2 Matrizen und Vektoren 2
2.1 Eingabe von Matrizen . . . 2
2.2 Operationen mit Matrizen . . . 2
2.3 Blockmatrizen . . . 4
2.4 Eingabe großer Matrizen . . . 5

3 Programmierelemente in MATLAB 5
3.1 Verzweigung . . . 6
3.2 Zählschleife . . . 6
3.3 Wiederholschleife . . . 6
3.4 Beispiele . . . 6

4 M-Files 8
4.1 Skript-Files . . . 8
4.2 Funktions-Files . . . 9

5 Zeichnen von Daten 10

1 Was ist MATLAB ?

MATLAB ist eine Umgebung für die Entwicklung von Algorithmen, Darstellung und Analyse von Daten und numerische Berechnungen. Es basiert auf FORTRAN/C++ Programmen und kann zur Simulation und Tests numerischer Verfahren verwendet werden.

Für praktische Rechnungen sind richtige Programmiersprachen wie C++ und FORTRAN geeigneter.

MATLAB ist ein kommerzieller Code. Für den Preis von 80 Euro kann man eine Studentenversion käuflich erwerben. Es gibt zusätzlich den kostenfreien Klon OCTAVE, der mit den meisten Befehlen von MATLAB kompatibel ist.

2 Matrizen und Vektoren

2.1 Eingabe von Matrizen

Matrizen können in MATLAB auf verschiedene Art und Weisen eingegeben werden:

- explizit:
  Wir demonstrieren dies am Beispiel der Matrix

$$A = \begin{bmatrix} 1 & 2 & -2 \\ 0.5 & 4 & 3 \end{bmatrix}.$$

  Dann lautet der Eingabebefehl

```
A=[ 1 2 -2;0.5 4 3];
```

  Damit ist die Variable A mit dem entsprechenden Wert belegt. Das Semikolon am Ende verhindert, daß der Wert von A ausgegeben wird.

- mittels vordefinierter Funktionen und Programmierelemente, siehe Abschnitt 2.4.

- extern aus einem Datenfile:
  Hier verweisen wir auf die Literatur und Hilfe von MATLAB.

2.2 Operationen mit Matrizen

Dazu stehen die Operatoren $=, +, -, *, \backslash, /, \hat{}$ zur Verfügung. Wir demonstrieren dies an folgendem Beispiel:

Beispiel 2.1. *Es seien*

$$B = \begin{bmatrix} 3 & 1 & -1 \\ 0 & 2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ und } c = \begin{bmatrix} 1 \\ -1 \\ 3 \end{bmatrix}.$$

Wir berechnen nun die Matrizen $C = 2A + B$, $E = C\,C^\top$, $F = C^\top C$, $G = E^2$ lösen die linearen Gleichungssyteme $Ex = b$, $y^\top E = b^\top$ und $Fz = c$ und berechnen $b^\top b$.

Dies erfolgt nun wiefolgt

```
B=[3 1 -1;0 2 3];
C=2*A+B
C =

     5     5    -5
     1    10     9
```

```
E=C*C'

E =

    75    10
    10   182

F=C'*C

F =

    26    35   -16
    35   125    65
   -16    65   106

G=E^2

G =

        5725        2570
        2570       33224

b=[1;2];
c=[1,-1,1];
x=E\b

x =

    0.0120
    0.0103

y=b'/E

y =

    0.0120    0.0103

b'*b

ans =

     5
```

```
z=F\c

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND =
7.085745e-18.
> In matrix1 at 12

z =

   1.0e+14 *

   -6.2724
    3.3012
   -2.9711
```

Generell nutzt MATLAB numerische Verfahren, d.h. es rechnet mit Gleitkommazahlen. Da die Matrix E regulär ist, sind x und y eindeutig bestimmt. Die Matrix F ist jedoch singulär, wie uns

```
>> eig(F)

ans =

    0.0000
   74.0734
  182.9266
```

zeigt, was die Eigenwerte berechnet. Dies erklärt auch die Warnung bei der Berechnung von z.

2.3 Blockmatrizen

Man kann auch Matrizen der Form

$$H = \begin{bmatrix} C & A \\ A & B \\ C & A \end{bmatrix}$$

eingeben. Dies erfolgt analog

```
H=[C A;A B;C A];
```

und erzeugt eine 6×6 Matrix H. Wichtig ist dabei, daß die Dimensionen passen.

Man kann aus H auch wieder einen Teil der Matrix generieren. So erzeugt

```
J=H(2:5,3:4);
```

eine Matrix J mit 4 Zeilen und 2 Spalten, die die Einträge von H aus den Zeilen 2 bis 5 und Spalten 3 bis 4 enthält.

2.4 Eingabe großer Matrizen

Die Eingabe großer Matrizen, z.B.

$$R_n = \left[i^2\right]_{i=1}^{n} \tag{2.1}$$

mit, z.B. $n = 100$, oder

$$S_{100} = \begin{bmatrix} 2 & 1 & 1/2 & 0 & 0 & \dots & 0 \\ 0 & 2 & 1 & 1/2 & 0 & & \vdots \\ \vdots & 0 & 2 & 1 & 1/2 & & \\ \vdots & & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & 2 & 1 & 1/2 \\ 0 & \dots & 0 & 0 & 0 & 2 & 1 \\ 0 & \dots & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \in \mathbb{R}^{100\times 100} \tag{2.2}$$

kann nicht mehr explizit erfolgen. Dazu benötigen wir Funktionen zum Bau von Matrizen und Programmierelemente.

Die wesentlichsten Funktionen lauten

- `eye(n)` generiert die $n \times n$ Einheitsmatrix,
- `zeros(m,n)` generiert eine $m \times n$ Rechtecksmatrix mit Nullen,
- `ones(m,n)` generiert eine $m \times n$ Rechtecksmatrix mit Einsen,
- `diag(v)` generiert eine Diagonalmatrix mit dem Vektor v auf der Hauptdiagonalen,
- `rand(m,n)` generiert eine $m \times n$ Rechtecksmatrix mit Zufallseinträgen,

Weitere Befehle befinden sich in der Hilfe und Literatur.

Damit allein können wir aber R_n (2.1) und S_{100} (2.2) nicht allein generieren. Deshalb benötigen wir programmiersprachliche Elemente, die wir im nächsten Abschnitt vorstellen werden.

3 Programmierelemente in MATLAB

Die wesentlichen Elemente sind

- Verzweigung,
- Zählschleife,
- Wiederholschleife.

3.1 Verzweigung

Die Syntax lautet

```
if Bedingung
   anweisung1;
else
   anweisung2;
end
```

Falls die *Bedingung* erfüllt ist, wird die *anweisung1* ausgeführt, sonst *anweisung2*.

3.2 Zählschleife

Hier ist die Syntax

```
for i=min:max
   anweisung(i);
end
```

Hier wird für alle i von min bis max die vom Parameter i abhängige *anweisung(i)* ausgeführt.

3.3 Wiederholschleife

Diese lautet

```
while Bedingung
  anweisung;
end
```

Solange die *Bedingung* erfüllt ist, wird die *anweisung* ausgeführt.

3.4 Beispiele

Wir demonstrieren nun die Programmierelemente bei der Generierung der Matrizen R_n (2.1) und S_{100} (2.2). Die Diagonalmatrix R_n kann über zwei Varianten generiert werden:

- einerseits mit dem `diag` Befehl,
- durch Setzen seiner Haupdiagonalelemente,

Im letzteren Fall erzeugt man z.B. für $n = 100$ die folgende Befehlskette

```
% Skriptfile zur Generierung von R_{100} (2.1) (1. Variante)

n=100; % Festsetzen der Groesse
R=zeros(n,n); % alle Eintraege auf Null
for i=1:n
    R(i,i)=i^2; % Hauptdiagonale
end
```

Im ersten Fall lautet die Befehlskette

```
% Skriptfile zur Generierung von R_{100}(2.1) (2. Variante)

n=100; % Festsetzen der Groesse
R=zeros(n,n); % alle Eintraege auf Null
for i=1:n
    v(i)=i^2; % Vektor mit der Hauptdiagonale
end
R=diag(v);
```

Die Matrix S_{100} ist eine obere Dreiecksmatrix mit Bandbreite 2. Desweiteren sind die Einträge auf den Diagonalen gleich. In den ersten zwei Varianten generieren wir S_{100} zunächst als Zweifaches der Einheitsmatrix. Anschließend arbeiten wir die Nichtnulleinträge auf den Nebendiagonalen ein, entweder

- zeilenweise oder
- spaltenweise.

Dies erfolgt erneut mit Wiederhol- und Zählschleife. In beiden Fällen müssen wir beachten, daß die Nebendiagonalen die Länge $n-1$ und $n-2$ haben. Die Befehlsketten lauten dann

```
% Skriptfile zur Generierung der Matrix S_n (2.2) (1. Variante)
% vorher n eingeben

S=2*eye(n); % zweimal Einheitsmatrix
for i=1:n-1
    S(i,i+1)=1; % 1. Nebendiag
    if i<(n-1)
        S(i,i+2)=1/2;  % 2. Nebendiag (um 1 kuerzer)
    end
end
```

oder alternativ

```
% Skriptfile zur Generierung der Matrix S_n (2.2) (2. Variante)
% vorher n eingeben

S=2*eye(n); % Hauptdiagonale
S(1,2)=1; % 2. Spalte
i=3; % gehe nun alle Spalten (ab 3) durch
while i<=n
    S(i-2,i)=1/2;
    S(i-1,i)=1;
    i=i+1; % gehe in die naechste Spalte
end
```

Noch kürzer ist die Befehlskette mit dem `toeplitz` Befehl. Dabei wird ausgenutzt, daß S_{100} (2.2) eine unsymmetrische Toeplitzmatrix ist.

```
% Skriptfile zur Generierung der Matrix S_n (2.2) (3. Variante)
% vorher n eingeben, n>=3!!!!


v=zeros(n,1); % n ist mindestens 3
w=zeros(n,1);
v(1)=2;  % 1. Zeile [2 0 ... 0]
w(1)=2;  % 1. Spalte [2 1 1/2 0 ... 0]
w(2)=1;
w(3)=0.5;
S=toeplitz(v,w);
```

4 M-Files

Generell sollte man nicht alle Befehle in der Kommandozeile eingeben. Als Ausweg gibt es die M-Files. Dazu gibt es zwei Typen

- Skript-Files,
- Funktions-Files.

4.1 Skript-Files

Hier wird einfach eine Reihenfolge von Befehlen abgearbeitet. Dies haben wir schon bei der Generierung der Matrizen R_n (2.1) und S_{100} (2.2) verwendet. Geben wir z.B. $n = 340$ ein und lassen danach das File `genS_v1.m` mit den Befehlen

```
% Skriptfile zur Generierung der Matrix S_n (2.2) (1. Variante)
% vorher n eingeben

S=2*eye(n); % zweimal Einheitsmatrix
for i=1:n-1
    S(i,i+1)=1; % 1. Nebendiag
    if i<(n-1)
        S(i,i+2)=1/2;  % 2. Nebendiag (um 1 kuerzer)
    end
end
```

laufen, erzeugt dies die analoge Matrix S_{340} der Dimension 340.

4.2 Funktions-Files

Damit können selbstdefinierte Funktionen erzeugt werden. Wir demonstrieren dies für die Berechnung der Polarkoordinaten (r, ϕ) eines Vektors $\begin{bmatrix} x \\ y \end{bmatrix}$, d.h.

- Input: $x, y \in \mathbb{R}$,
- Ouput: $r \geq 0, \phi \in [0, 2\pi)$ mit
  $x = r\cos\phi$, $y = r\sin\phi$.

Dann lautet die Befehlslinie

```
function [r,phi]=polar(x,y);
```

wobei `polar.m` der Name und das File der Funktion ist.

```
function [r,phi] = polar(x,y)
% Funktionsfile zum Berechnen der Polarkoordinaten
% eines Punktes (x,y)\in \R^2

% Input: x,y
% Output: r,phi

r=sqrt(x^2+y^2);
if (r>0)
    phi=acos(x/r); % acos ist der Arccos
else
    phi=0; % Im Nullpunkt
end
end
```

Ruft man nun

```
[r,phi]=polar(3,-4)
```

so lautet der Output

```
r =

     5


phi =

    0.9273
```

5 Zeichnen von Daten

Zur Visualiserung von Daten gibt es den `plot(x,y)`-Befehl mit $x, y \in \mathbb{R}^n$. Dabei werden Punktepaare $(x_i, y_i)_{i=1}^n$ im $\mathbb{R}^2$ gezeichnet und evtl. miteinander verbunden.

Mit `help plot` erhalten wir eine ausführliche Beschreibung dieses Befehls.

Wir demonstrieren dies an

Beispiel 5.1. *Wir möchten die Funktion*

$$f : [0,2] \to \mathbb{R} \text{ mit } f(x) = (x+1)|x-1| \tag{5.1}$$

zeichnen.

Da `plot` Punktepaare miteinander verbinden kann, ist die Idee nun das Intervall fein genug zu unterteilen.

```
% Skriptfile zur Zeichnen von f(x)=(x+1)|x-1| in [0,2]
% Vorher Anzahl der Gitterpunkte n eingeben.

h=2/n;    % Schrittweite, n+1 Punkte
x=zeros(n+1,1);
y=zeros(n+1,1);
for i=0:n
    xw=h*i;
    x(i+1)=xw;
    y(i+1)=abs(xw-1)*(xw+1);
end
plot(x,y);
```

Gibt man nun $n = 4$ ein, so ist die Qualität sehr bescheiden, besser wird es bei $n = 10$ und $n = 100$, siehe Abbildung 1.

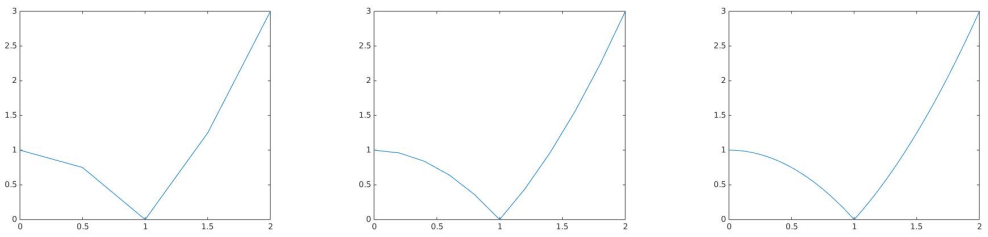

Abbildung 1: Darstellung der Funktion f (5.1) mit $n = 4$ (links), $n = 10$ (mitte) und $n = 100$ (rechts) Punkten.

Abschließend noch

Beispiel 5.2. *Man berechne das Integral*

$$\int_0^1 \tan x \, \mathrm{d}x = -\ln(\cos 1)$$

approximativ mittels einer Riemann-Summe mit äquidistanter Unterteilung in $n = 2^k$, $k = 1, 2, \ldots, 15$ *Intervalle und stelle den Fehler graphisch in Abhängigkeit von* n *dar.*

Zunächst schreiben wir ein M-File, daß für gegebenes n die Riemann-Summe berechnet

```
function [app] = riemanntan(k)
% Funktion berechnet die Riemannsumme von int_0^1 tan(x) dx
% mit 2^k Intervallen,

% Input: k
% Output: app.. Die Riemannsumme



n=2^k; % Anz Intervalle
h=1/n; % Schrittweite
app=0; % Integral auf 0 setzen
for i=1:n
    x=((i-1)+rand(1))*h; % zufaellige Stuetzstelle
    f=tan(x);
    app=app+h*f; % Aufsummieren
end



end
```

Anschließend können wir den Fehler wiefolgt visualisieren. Hierbei sollten aber die Achsen logarithmisch skaliert werden.

```
exakt=-log(cos(1));
for k=1:15
    appint=riemanntan(k);
    error(k)=abs(appint-exakt);
    n=2^k;
    x(k)=n;
end
```

```
loglog(x,error,'linewidth',2);
xlabel('Anzahl Intervalle'); % x-Achse bezeichnen
ylabel('Fehler'); % y-Achse bezeichnen

% Programm zeichnet den Fehler bei der Integration
% einer Riemannsumme mit
% n=2^k, k=1..15 Stuetzstellen

% Programm ruft riemanntan.m
```