M1 Matlab Einführung

Matrizen, Dimensionen, Typen,
Erzeugen von Matrizen, Untermatrizen,
einfache Matrix-Operationen, Matlab-Hilfe

05.11.2014

Über Matlab

Geschichte

- Software-Paket für nummerisches Rechnen (Taschenrechner)
- In den 70er Jahren an Universität von New Mexico gestartet
- Interaktiven Zugriff auf Fortran-Bibliotheken (LinPack und EisPack)
- Heute kommerziell (*The MathWorks*)

Hilfe zu Matlab

- Mathworks: `http://www.mathworks.com/help/matlab/`
- Ulrich Stein, Programmieren mit Matlab (z.B. ExLibris, CHF 32.–)
  `http://www.hanser-elibrary.com/isbn/978-3-446-43243-7`
- MAVT-Tutorial: `http://www.imrtweb.ethz.ch/matlab/`
- Peter Arbenz: `http://people.inf.ethz.ch/arbenz/MatlabKurs/`

Das Command Window

```
>> (77 + 22) / 33;
ans = 3

>> q11 = 34 / 22
q11 = 1.5455

>> a = 17, b = 5, c = 7
a = 17
b =  5
c =  7

>> x = 42;

>> x
x = 42

>> sin(pi)
ans = 1.2246e-16

>> asin(2)
ans =  1.5708 - 1.3170i
```

```
>> A = [1 2 3]
A =  1   2   3

>> M = [2 6 7; 9 2 4; 6 9 6]
M =  2   6   7
     9   2   4
     6   9   6

>> M * M
ans = 100   87   80
       60   94   95
      129  108  114

>> inv(M)
ans = -0.0941   0.1059   0.0392
      -0.1176  -0.1176   0.2157
       0.2706   0.0706  -0.1961
```


Eingabe von Vektoren und Matrizen

Literale

- `[1 2 3]` oder `[1,2,3]` – Zeilenvektor (1×3-Matrix)
- `[1; 2; 3]` – Spaltenvektor (3×1-Matrix)
- `[1 2 3; 4 5 6]` – 2×3-Matrix
- 42 oder `[42]` – Skalar (1×1-Matrix)

⚠ **Achtung**: `[1, 2+3]` vs. `[1, 2 +3]` vs. `[1, 2 + 3]`

Matrizen zusammenbasteln

- `[A B C]` – Matrizen A, B, C nebeneinander schreiben (gleiche Höhe!)
- `[A;B;C]` – Matrizen A, B, C untereinander schreiben (gleiche Breite!)

:-Operator

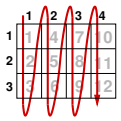
- `a:b` – Zeilenvektor `[a,a+1,...,b]`
- `a:s:b` – Zeilenvektor `[a,a+s,...,b]`

Zugriff auf Untermatrizen

Zugriff auf Elemente

eigentlich: 1×1 Untermatrizen

- `M(i,j)` – Zeile i, Spalte j (Indizes fangen bei 1 an)
- `M(k)` – k-tes Element, spaltenweise durchgezählt



Zugriff auf Untermatrizen

- `M(R,C)` – Zeilen mit Index in Vektor R, Spalten mit Index in Vektor C
- `M(K)` – Elemente mit (linearem) Index in K, Struktur wie K

In Indexvektoren: `end` = letzter gültiger Index

Beispiele

Spaltenvektor `1:end`

- `M(end-2:end,:)` – letzte drei Zeilen
- `M(:,1:2:end)` – ungerade Spalten
- `M(:)` – Spaltenvektor mit allen Elementen von M
- `M(:,end:-1:1)` – horizontal gespiegelt

Aufgaben 1

Berechne die folgenden Ausdrücke

0. `M = [1 2 3 4; 2 4 6 8; 3 5 7 9]`

```
1 2 3 4
2 4 6 8
3 5 7 9
```

1. `M(end,end)`
2. `M(end)`
3. `M([5 3 5])`
4. `M(end:-1:1,end:-1:1)`
5. `M(1:3:end)`
6. `M(M(2:3,3))`
7. `M(M)`

Matrix-Operationen

Grundrechenarten

- `A + B` – Matrixaddition
- `A - B` – Matrixsubtraktion

} A und B haben gleiche Dimension*

- `A * B` – Matrixmultiplikation, Breite von A = Höhe von B *
- `A ^ k` – Exponentiation (Matrix hoch Skalar)
- `inv(A)` – Inverse Matrix (gleich wie `M ^ -1`)
- `A \ y` – Division (finde x so dass `A * x = y`)

Elementweise Operationen (A und B haben gleiche Dimension)

- `.*`, `./`, `.^` – elementweise Multiplikation / Division / Exponentiation

Transponieren

- `A.'` – transponieren (spiegeln an Hauptdiagonalen)
- `A'` – adjungieren (hermitesch konjugieren)

* Falls A oder B Skalar: elementweise rechnen

Präzedenzen in Matlab

1. Klammern: $(\ldots)$
2. Transponieren und Potenzieren; `.'`, `.^`, `'`, `^`
3. Unäre Operatoren: $+$, $-$, $\sim$
4. Punkt-Operatoren: $.*$, $./$, $.\backslash$, $*$, $/$, $\backslash$
5. Strich-Operatoren: $+$, $-$
6. :-Operator `:`
7. Vergleichs-Operatoren: $==$, $\sim=$, $<$, $>$, $<=$, $>=$
8. Elementweises AND: &
9. Elementweises OR: |
10. Short-circuit AND: &&
11. Short-circuit OR: ||

Matlab-Befehle

Zuweisung

- ⟨*Variable*⟩ = ⟨*Ausdruck*⟩

Befehlssequenzen, Ausgabe

- Semikolon nach Befehl: keine Ausgabe
- mehrere Befehle mit Semikolon oder Komma trennen
- Kommentare: %-Zeichen bis Zeilenende

Hilfe in Matlab

- `help` Befehl, `doc` Befehl – kurze / Lange Hilfe zum Befehl
- `lookfor` name – Befehle ähnlich name
- `which` Befehl, `type` Befehl – Pfad / Code vom Befehl

Matrizen Erzeugen 1/2

Spezielle Matrizen

- `zeros(m,n)` – $m \times n$ Nullmatrix
- `ones(m,n)` – $m \times n$ Einsmatrix
- `eye(m,n)` – $m \times n$ Einheitsmatrix

Zufällige Matrizen

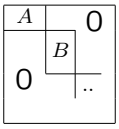
- `rand(m,n)` – $m \times n$ Matrix mit zufälligen Werten aus $(0, 1)$
- `randi(k,m,n)` – $m \times n$ Matrix mit zufälligen Werten aus $\{1, \ldots, k\}$

Quadratische Matrizen und Skalare

- `zeros(m)` ≡ `zeros(m,m)`
- `rand` ≡ `rand(1,1)`
- `ones([m,n])` ≡ `ones(m,n)`

Matrizen Erzeugen 2/2

- `diag(V)` – Matrix mit Vektor V auf der Diagonalen
- `diag(M)` – Diagonalelemente von M (als Spaltenvektor)
- `blkdiag(A,B,...)` – Block-Diagonal-Matrix:



- `fliplr(M)` – links-rechts spiegeln
- `flipud(M)` – oben-unten spiegeln
- `triu(M)` – obere Dreiecksmatrix
- `tril(M)` – untere Dreiecksmatrix
- `rot90(M)` – rotiert Matrix gegen Uhrzeigersinn
- `reshape(M,m,n)` – ordnet Elemente von M als $m \times n$ Matrix an
- `repmat(M,m,n)` – schreibt M m-mal unter- und n-mal nebeneinander

Aufgaben 2

Erzeuge die folgenden Matrizen

1. ……

$$\begin{vmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{vmatrix}$$

2. ……

$$\begin{vmatrix} 0 & 0 & 0 & 3 \\ 0 & 0 & 3 & 0 \\ 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{vmatrix}$$

3. ……

$$\begin{vmatrix} 7 & 0 & 0 & 7 \\ 0 & 7 & 7 & 0 \\ 0 & 7 & 7 & 0 \\ 7 & 0 & 0 & 7 \end{vmatrix}$$

4. ……

$$\begin{vmatrix} 1 & 3 & 5 & 7 \\ 9 & 11 & 13 & 15 \\ 17 & 19 & 21 & 23 \end{vmatrix}$$

5. ……

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

6. ……

$$\begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix}$$

Summen und Produkte

Summen

- `sum(V)` – Summe der Elemente
- `sum(A)` – Zeilenvektor mit Summen der Spalten von A
- `cumsum(A)` – kumulierte Summen

Produkte

- `prod(V)` – Produkt der Elemente
- `prod(A)` – Zeilenvektor mit Produkte der Spalten von A
- `cumprod(A)` – kumulierte Produkte

Was ist mit *zeilenweiser* Summe?

Informative Funktionen

Grösseninformationen

- `size(A)` – Vektor mit allen Dimensionen
- `size(A,1)`, `size(A,2)` – Anzahl Zeilen / Anzahl Spalten
- `numel(A)` – Anzahl Elemente (= `prod(size(A))`)
- `length(A)` – grösste Dimension (= `max(size(A))`)

Strukturinformationen

- `isrow(A)` – `true` falls A ein Zeilenvektor ist
- `iscolumn(A)` – `true` falls A ein Spaltenvektor ist
- `isempty(A)` – `true` falls A eine 0×0 Matrix ist
- `isscalar(A)` – `true` falls A eine 1×1 Matrix ist

Typinformationen

- `whos A` – zeigt den Typen von A an

Typisierung in Matlab

Statische vs. dynamische Typisierung

- Pascal: Variable hat fixen Typ und flexiblen Wert
- Matlab: Variable hat flexiblen Typ und flexiblen Wert

Vor- und Nachteile von dynamischer Typisierung

+ Flexibilität
+ einfachere Programme
– anfällig für Programmierfehler
– Effizienz (Typprüfung zur Laufzeit)

„Korrekter“ Matlab-Code:

```
if 3 * 'false'
  a = 17
else
  a = 20
end
```


Typen in Matlab

Typ	Bedeutung	Wertebereich
`single`	Matrix von reellen[1] Zahlen	$-3.4\text{E}38 \ldots 3.4\text{E}38$
`double`	Matrix von reellen[1] Zahlen	$-1.8\text{E}308 \ldots 1.8\text{E}308$
`int8`	Matrix von ganzen Zahlen	$-2^7 \ldots 2^7\text{-}1$
`uint8`	Matrix von pos. ganzen Zahlen	$0 \ldots 2^8 - 1$
...	...	...
`int64`	Matrix von ganzen Zahlen	$-2^{63} \ldots 2^{63} - 1$
`uint64`	Matrix von ganzen Zahlen	$0 \ldots 2^{64} - 1$
`char`	Matrix von Buchstaben	Unicode
`logical`	Matrix von Wahrheitswerten	0/1
`function`	Funktionen	

[1] oder komplexen

Variablen in Matlab

`double`

- `42` – ein 1×1 double ← „Skalare“ sind 1×1 Matrizen
- `[1 2 3]` – ein 1×3 double

`char`

- `'Hello'` – ein 1×5 `char`
- `['Hello', 'World']` – ein 1×10 `char`
- `['Hello'; 'World']` – ein 2×5 `char`
- `['Hallo'; 'Welt']` – ein Fehler ...

`logical`

- `true` – ein 1×5 `logical`
- `false(2,4)` – ein 2×4 `logical`
- `true(4)` – ein 4×4 `logical`

`true(m,n)` – $m \times n$ True-Matrix
`false(m,n)` – $m \times n$ False-Matrix

Typ-Konversion

Syntax: ⟨*Typname*⟩ (⟨*Wert*⟩)

Beispiele

- `int64(7)`
- `int64([1 2 3; 4 5 6])`
- `logical(M)`

⚠ Achtung

```
>>  i = int32(3), r = 3.5, s = i + r
i =   3
r =   3.5
s =   7
>> whos s
  Name  Size  Bytes  Class  Attributes
  s     1x1       4  int32
```