Modellierung und Programmierung

Dr. Martin Riplinger

30.1.2013

Einleitung

Matlab

Matlab (***Mat****rix* ***Lab****oratory*) ist ein interaktives Matrix-orientiertes Softwaresystem zur Berechnung und Lösung numerischer Probleme.

Warum Matlab?

- benutzerfreundliche Syntax
- umfangreiche Sammlung mathematischer Algorithmen
- vielfältige und einfach realisierbare Datenausgabe / Visualisierung
- kurze Entwicklungszeiten, in der eingebauten Programmiersprache lassen sich Algorithmen schnell und leicht realisieren (Interpretersprache)
- einfaches Debugging
- hohe Absturzsicherheit

Nachteile:

- etwas höhere Anforderungen an den Rechner (v.a. Speicherbedarf)
- langsamer als kompilierter Code, insbesondere bei `for`-Schleifen
  Aber: Programme anderer Sprachen, z.B. C, lassen sich leicht einbinden.

Literatur

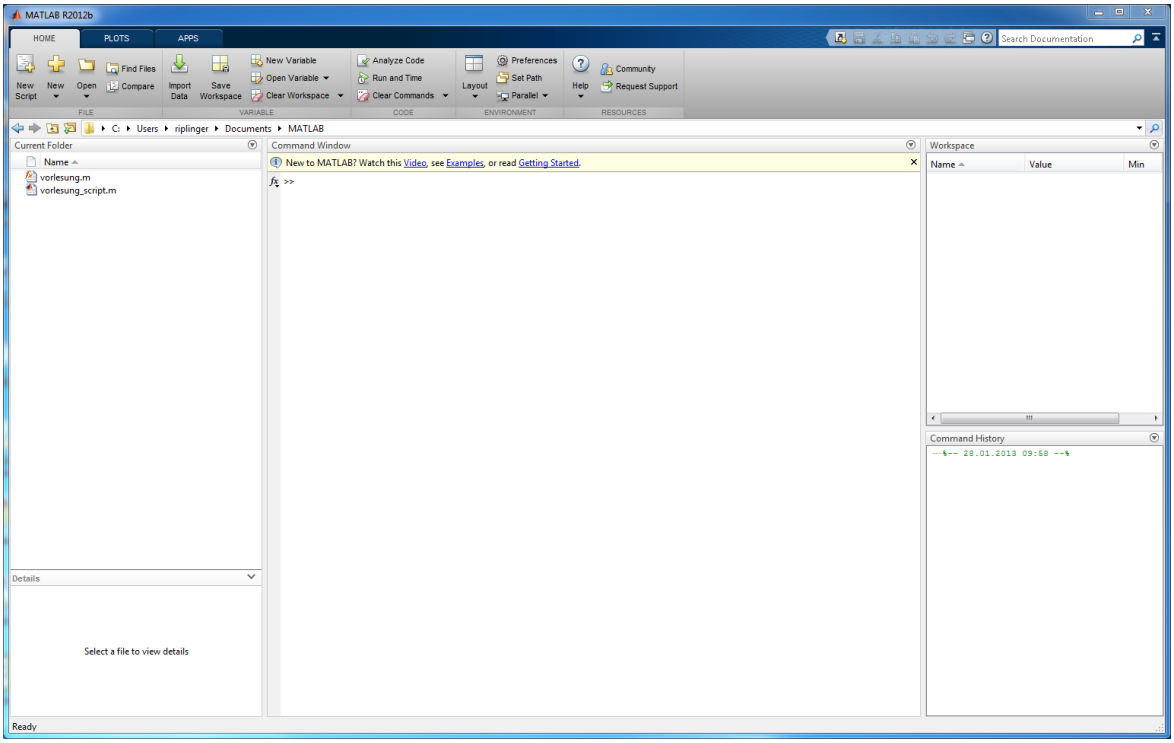
- Anne Angermann et al.: Matlab-Simulink-Stateflow: Grundlagen, Toolboxen, Beispiele, Oldenbourg Verlag, 2011
- Wolfgang Schweizer: Matlab kompakt, Oldenbourg Verlag, 2009
- Cleve Moler: Numerical Computing with Matlab, SIAM, 2010
- Stormy Attaway: MATLAB: A Practical Introduction to Programming and Problem Solving, Butterworth Heinemann, 2011
- Günter Gramlich: Eine Einführung in Matlab aus der Sicht eines Mathematikers
  `www.hs-ulm.de//users/gramlich/EinfMATLAB.pdf`
- Susanne Teschl: MATLAB - Eine Einführung
  `staff.technikum-wien.at/~teschl/MatlabSkriptum.pdf`

weitere Informationen:

- Homepage von Matlab:
  `www.mathworks.de` bzw. `www.mathworks.com`
- Informationen zur Campuslizenz:
  `www.hiz-saarland.de/informationen/arbeitsplatz/sw-lizenzen/mathworks-tah-campuslizenz/`
  `unisb.asknet.de/cgi-bin/program/S1552`

Matlab-Oberfläche

aktuelles Release: R2012b



Matlab-Oberfläche

- Starten mit `$ matlab &`
- Elemente der Matlab-Oberfläche:

Command Window:	direkte Eingabe von Matlab-Befehlen
Command History:	Historie der im Command Window eingegebenen Befehle
Workspace Browser:	Anzeige der Variablen des Base-Speicherbereichs
Current Folder:	Auflistung der Dateien des aktuellen Verzeichnisses
Editor:	Bearbeitung von Matlab-Skripten und Funktionen (mit Syntax-Highlighting, Tab-Vervollständigung, ...)

erste Schritte in Matlab

- Eingabe direkt im Command Window
- Ausgabe erscheint beim Weglassen des Semikolons am Ende der Befehlszeile.
- Es gilt „Punktrechnung vor Strichrechnung“.
- Es wird zwischen Klein- und Großbuchstaben unterschieden.
- Strings werden durch Hochkommata erzeugt.

Beispiele:

```
>> 2.5+1.5
ans = 4

>> a=3;
>> b=4;
>> a/b
ans = 0.75
```

```
>> c=a^2+b^2
c = 25

>> d=a+b*c
d = 103

>> s = 'Hallo'
```

- Unterschied zu C: keine Deklaration der Variablen nötig!
- Variablen werden (intern) automatisch als `double` behandelt und belegen 8 Bytes Speicherplatz.
  $\rightsquigarrow$ kein cast notwendig!
- Ganzzahlige Ergebnisse werden als ganze Zahl ausgegeben.

Script-Files

Skripte / Script-Files

Ein Skript ist eine Folge von Anweisungen, die im Editor eingegeben und mit der Endung „.m“ abgespeichert werden. Es wird durch Eingabe des Dateinamens im Command Window oder durch den Button „Run“ im Editor (alternativ: F5) ausgeführt, d.h. alle Anweisungen werden der Reihe nach abgearbeitet.

Beispiel:
Das Skript `testskript.m`

```
a=3;
b=4;
c=a^2+b^2
```

führt nach Ausführung zu der Ausgabe `c=25` auf dem Bildschirm. Zudem sind die Variablen `a=3`, `b=4` und `c=25` im Workspace enthalten (und können bei Bedarf für weitere Berechnungen verwendet werden).

Operatoren

- Vergleichende und logische Operatoren wie in C

	Notation in Matlab	Notation in C
	`a < b`	`a < b`
	`a >= b`	`a >= b`
	`a == b`	`a == b`
	`A && B`	`A && B`
	`A || B`	`A || B`
Ausnahme:	`a ~= b`	`a != b`
	`~A`	`!A`

Unterschied zu C: In Matlab dürfen `a` und `b` Matrizen gleicher Dimension sein. In diesem Fall wird elementweise verglichen und eine Matrix gleicher Dimension mit Nullen und Einsen zurückgegeben.

- keine Inkrement- und Dekrementoperatoren `a++, a--`
- keine arithmetischen Zuweisungsoperatoren `a+=2`

Ausgewählte Variablen und Konstanten

`ans`	wird dem Ergebnis kein Variablenname zugeordnet, so wird automatisch die Variable `ans` erzeugt
`inf`	liegt das Ergebnis nicht im Intervall [`-realmax; realmax`] erhält dieses automatisch den Wert `inf`
`NaN`	Ergebnis nicht definierter arithmetischer Operationen, z.B. `0/0, inf/inf, 0*inf`, sind vom Typ `NaN` (Not a Number)
`pi`	$= \pi$
`realmax`	größte positive Zahl (PC:1.797710^{308})
`realmin`	kleinste positive Zahl (PC: 2.225110^{-308})

In Matlab ist es möglich mit **komplexen Zahlen** zu rechnen:

`i,j`	imaginäre Einheit ($i^2 = -1$)
`conj(Z)`	konjugiert komplexe Zahl zu Z
`real(Z),imag(Z)`	Real- bzw. Imaginärteil von Z
`angle(Z), abs(Z)`	Polardarstellung von Z

Nützliche Befehle bzw. Funktionen

`clc`	löscht die Oberfläche des Command Window, aber nicht die Variablen selbst
`clear [mod]`	löscht alle Variablen aus dem Workspace `mod` = `var`: nur die Variable `var` wird gelöscht `mod` = `all`: alle Variablen werden gelöscht (Achtung: auch globale!)
`ctrl + c`	„Notbremse“, bricht die aktuelle Berechnung ab
`help [Name]`	listet alle bzw. die zu `Name` gehörigen Hilfe-Themen auf, bequemer: GUI (Grafische Benutzeroberfläche) benutzen (z. B. mit F1)
`isinf(var)`	liefert 1, falls var vom Typ `inf`
`isnan(var)`	liefert 1, falls var vom Typ `NaN`
`who`	listet die im Workspace vorhandenen Variablen auf
`whos`	liefert eine detailierte Liste der Variablen (Typ, Wert)

Erzeugen von Matrizen

- Eingabe in eckigen Klammern:
  - einzelne Komponenten in einer Zeile werden durch Leerzeichen oder Komma getrennt
  - neue Zeilen werden durch ein Semikolon gekennzeichnet

```
x = [1 2 3]
A = [1 2 3; 4 5 6]
```

erzeugt den Zeilenvektor $x = (1, 2, 3)$ sowie die Matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

Merke: Zeilenvektoren werden intern als $1 \times N$-Matrizen behandelt.

- Eingabe durch Steuerung der Schrittweite: `y= a:h:b`
  - erzeugt einen Vektor von `a` bis `b` mit Schrittweite `h`

```
y= 0:2:6
```

erzeugt den Zeilenvektor $y = (0, 2, 4, 6)$

- Eingabe mit der Funktion `linspace(a,b,N)`
  - erzeugt einen Vektor von `a` bis `b` mit `N` Komponenten

```
y=linspace(0,6,4)
```

erzeugt ebenfalls den Zeilenvektor $y = (0, 2, 4, 6)$.

Bemerkung: Bei großen Dimensionen ist die Eingabe mittels `a:h:b` etwas schneller als bei der Verwendung von `linspace`.

Arbeiten mit Matrizen

- Die Indizierung von Matrizen beginnt in Matlab bei **1**!
- Reservierung von Speicherplatz erfolgt automatisch.
- Initialisierung trotzdem sinnvoll, führt in der Regel zu schnelleren Programmen.

Wir betrachten die Matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$.

- einzelnes Matrizenelement : `A(i,j)` liefert das Element $a_{i,j}$

```
>> A(2, 3)
ans = 6
```

- i-te Zeile einer Matrix: `A(i,:)`

```
>> A(1,:)
ans = 1 2 3
```

- j-te Spalte einer Matrix: `A(:,j)`

```
>> A(:,2)
ans = 2
      5
```

- Teilmatrix: `A(p:q,[r,s])` liefert die Zeilen p bis q und die Spalten r und s der Matrix

```
>> A(1:2,[1,3])
ans = 1     3
      4     6
```

Arbeiten mit Matrizen

Wir betrachten zusätzlich die Matrix $B = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$.

```
B=[1,2;0,3]
```

- Addition, Subtraktion und Multiplikation definiert wie in Linearer Algebra

```
>>  B*A
ans = 9   12   15
     12   15   18
```

```
>> B+3 % oder 3+B
ans = 4    5
      3    6
```

- elementweiser Zugriff auf Matrizen mit Punktoperator:

```
>> B.^2 % oder B.*B
ans = 1  4
      0  9
```

```
>> B^2 % oder B*B
ans = 1  8
      0  9
```

- Transponierte Matrix:

```
>> B'
ans = 1 0
      2 3
```

Matrix-Funktionen

`det(B)` Determinante von B

`eig(B)` Eigenwerte und (normierte) Eigenvektoren von B

`lambda=eig(B)` liefert $\lambda = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

`[v,lambda]=eig(B)` liefert $v = \begin{pmatrix} 1 & \sqrt{2}^{-1} \\ 0 & \sqrt{2}^{-1} \end{pmatrix}$ und $\lambda = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$

`rank(B)` Rang der Matrix B

`norm(B,1)` 1-Norm (Spaltensummenorm) der Matrix B, entspricht dem Ausdruck $\max_{1 \le j \le n} \sum_{i=1}^{n} |b_{ij}|$, hier $\|B\|_1 = 5$.

`norm(B,'fro')` Frobeniusnorm der Matrix B, entspricht dem Ausdruck $\|B\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |b_{ij}|^2}$, hier $\|B\|_F = \sqrt{14}$.

`inv(B)` Inverse der Matrix B, Ergebnis: $B^{-1} = \frac{1}{3} \begin{pmatrix} 3 & -2 \\ 0 & -1 \end{pmatrix}$

Hinweis: Die explizite Berechnung einer inversen Matrix, um ein lineares Gleichungssystem zu lösen, ist in der Praxis (fast immer) zu aufwendig!

Ausweg: Vorlesung Praktische Mathematik (QR-Zerlegung, ...)
Matlab: `linsolve`

Matrix-Funktionen

`end`	maximaler Indexwert von Matrizen, z. B. liefert `y(end)` den letzten Eintrag des Vektors y.
`length`	höchste Dimension einer Matrix
`size`	beide Dimensionen einer Matrix

`zeros(M,N)`	$M \times N$-Matrix, deren Einträge 0 sind
`ones(M,N)`	$M \times N$-Matrix, deren Einträge 1 sind
`eyes(M)`	$M \times M$-Einheitsmatrix
`rand(M,N)`	$M \times N$-Zufallsmatrix, wobei jeder Matrixeintrag eine Realisierung einer auf [0,1)-gleichverteilten Zufallsvariablen ist.

`A(:)`	Vektor, der die Spalten der Matrix A hintereinandergereit enthält. Mit der Matrix A der vorigen Folie entspricht A(:) dem Vektor [1; 4; 2; 5; 3; 6].
`A(A>2)`	Vektor, der die Spalten der Matrix A hintereinandergereit enthält, wobei nur Matrixelemente größer zwei berücksichtigt werden. Mit der Matrix A der vorigen Folie entspricht A(A>2) dem Vektor [4; 5; 3; 6].
`max(v)`	kleinster Eintrag des Vektors `v`
`min(v)`	größter Eintrag des Vektors `v`
`sum(v)`	Summe der Vektorelemente

Mathematische Funktionen

- **Trigonometrische Funktionen:** `cos, sin, tan, asin, acos, atan, atan2`

  Soll die Berechnung in Grad durchgeführt werden, so müssen die Funktionen mit 'd' ergänzt werden.

  Beispiel:

```
>>sin(pi/4)
ans = 0.7071
```

```
>> sind(45)
ans = 0.7071
```

- **Exponential- und logarithmische Funktionen:** `exp, log, log10`
- **Wurzeln:** `sqrt, nthroot(x,n)`
- **Rundungsfunktionen:** `ceil, floor, round` (runden zum nächsten integer-Wert)
- **Betragsfunktion:** `abs`

Hinweis: Diese Funktionen akzeptieren als Argumente Matrizen, die dann elementweise ausgewertet werden. Dies führt in der Regel zu schnellerem Code.

if-Anweisung

Syntax in Matlab

```
if Bedingung_1
   Anweisungsblock_1
elseif Bedingung_2
   Anweisungsblock_2

   .
   .
   .

elseif Bedingung_N
   Anweisungsblock_N
else
   Anweisungsblock
end
```

Syntax in C

```
if (Bedingung_1)
  Anweisungsblock_1
else if (Bedingung_2)
  Anweisungsblock_2

     .
     .
     .

else if (Bedingung_N)
   Anweisungsblock_N
else
   Anweisungsblock
```

switch-Anweisung

Syntax in Matlab

```
switch Variable

   case Wert_1
      Anweisungsblock_1

   case Wert_2
      Anweisungsblock_2

      .
      .
      .

   case Wert_N
      Anweisungsblock_N
   otherwise
      Anweisungsblock
end
```

Syntax in C

```
switch (Variable)
{
  case Wert_1:
    Anweisungsblock_1
    break;            // optional
  case Wert_2:
    Anweisungsblock_2

      .
      .
      .

  case Wert_N:
    Anweisungsblock_N
  default:            // optional
    Anweisungsblock   // optional
}
```

Unterschied zu C: In Matlab ist kein `break` notwendig, es wird **immer nur der zum case gehörige Anweisungsblock** ausgeführt.

for- und while-Schleifen

Syntax in Matlab

```
for Variable = Vektor
    Anweisungsblock
end
```

```
while Bedingung
    Anweisungsblock;
end
```

Syntax in C

```
for (Initialisierung; Bedingung; Update)
  Anweisungsblock
```

```
while (Bedingung)
  Anweisungsblock
```

Do-while-Schleifen existieren in Matlab nicht! (Ausweg: `while 1` und `break`)

Beispiel:

```
x=1:1:10;
summe=0;
for k=1:10
summe=summe+x(k);
end
% Ergebnis entspricht sum(x)
```

```
m=0;
Erg=0;
while m<=10
Erg=Erg+2;
m=m+1;
end
```

Ein- und Ausgabe am Bildschirm

Eingabe:

`variable=input(string)`	Ausgabe von `string` auf dem Bildschirm, die Eingabe (Zahl, Variablenname) wird ausgewertet und `variable` zugewiesen.
`variable=input(string,'s')`	Ausgabe von `string` auf dem Bildschirm, die Eingabe wird nicht ausgewertet sondern als String `variable` zugewiesen.

Ausgabe:

`disp(string)`	Ausgabe von `string` auf dem Bildschirm.
`disp(Variable)`	Ausgabe der Werte von `variable` auf dem Bildschirm.
`fprintf(String, Parliste)`	formatierte Ausgabe auf dem Bildschirm. Die Syntax entspricht im Wesentlichen der von `printf` in C.

Matlab als Programmiersprache

- Neben den **Script-Files** gibt es noch **Function-Files**, welche auch in Dateien mit der Endung .m gespeichert werden.

  Sie beginnen mit dem Schlüsselwort `function`.

```
function [ Rueckgabewerte ] = Funktionsname( Parameter )
% Beschreibung als Kommentar

Anweisungsblock

end
```

- Unterschiede zu C:
  - beliebige Anzahl von Rückgabewerten
  - Matrizen können übergeben werden
  - Übergabemethode: shared-data-copy, d.h. intern werden Variablen, die nicht verändert werden, als Zeiger übergeben. Wird eine Variable in der Funktion verändert, so wird eine Kopie im Speicherbereich der Funktion erstellt.
- Funktionenvariablen sind lokale Variablen
- Function-Files unter `Funktionsname.m` abspeichern
- Aufruf anderer m-Files in m-Files möglich, sofern die Verzeichnisse in denen die Dateien liegen unter „Set Path“ eingetragen sind oder mit dem „Current Folder“ übereinstimmen.

Nützliches

`tic-toc`	(primitive) Zeitmessung: Mit `tic` wird die Stoppuhr auf null gesetzt, `toc` gibt die vergangene Zeit aus.
`pause`	wartet bis zu einem Tastendruck auf der Tastatur
`pause(n)`	Pause für *n* Sekunden
`nargin`	Anzahl der Funktionsparameter
`nargout`	Anzahl der Rückgabewerte

Beispiel:

```
function [ erg ] = testfunktion(a, b)
if nargin==1
    erg=a;
else
    erg=a+b;
end %if
end
```

`testfunktion(2)` liefert das Ergebnis 2.
`testfunktion(2,3)` liefert das Ergebnis 5.
`testfunktion(2,3,2)` führt zu der Fehlermeldung: `Too many input arguments`

Beispiel: Fakultät

```
function [ erg ] = fakul(n)
% berechnet n! = n*(n-1)*...*1 rekursiv
if(n==0 || n==1)
    erg = 1;
elseif n<0
    disp('Bitte positive Zahl eingeben!')
else
    erg = n*fak(n-1);
end

end
```

```
function [ erg ] = fakul_iter(n)
% berechnet n! = n*(n-1)*...*1 iterativ
erg=1;
for k=1:n
    erg=erg*k;
end
% Alternative zur for-Schleife: erg=prod(1:n);
end
```

Beispiel: Fakultät

Startscript:

```
% Vergleich Rekursion und Iteration
k=20;

% rekursiv
tic
n1=fakul(k);
zeit1=toc;
fprintf('\n Zeitverbrauch rekursiv: %f',zeit1);

% iterativ
tic
n2=fakul_iter(k);
zeit2=toc;
fprintf('\n Zeitverbrauch iterativ: %f',zeit2);
fprintf('\n Ergebnis rekursiv: %d, iterativ: %d \n',n1,n2);
```

Ergebnis:

Zeitverbrauch rekursiv: 0.000186
Zeitverbrauch iterativ: 0.000023
Ergebnis rekursiv: 2432902008176640000, iterativ: 2432902008176640000