Numerik am Computer

Wiederholung Matlab

Marcus Grote und Helmut Harbrecht

Universität Basel

17.–21. Februar 2020

Übersicht

Grundlegendes

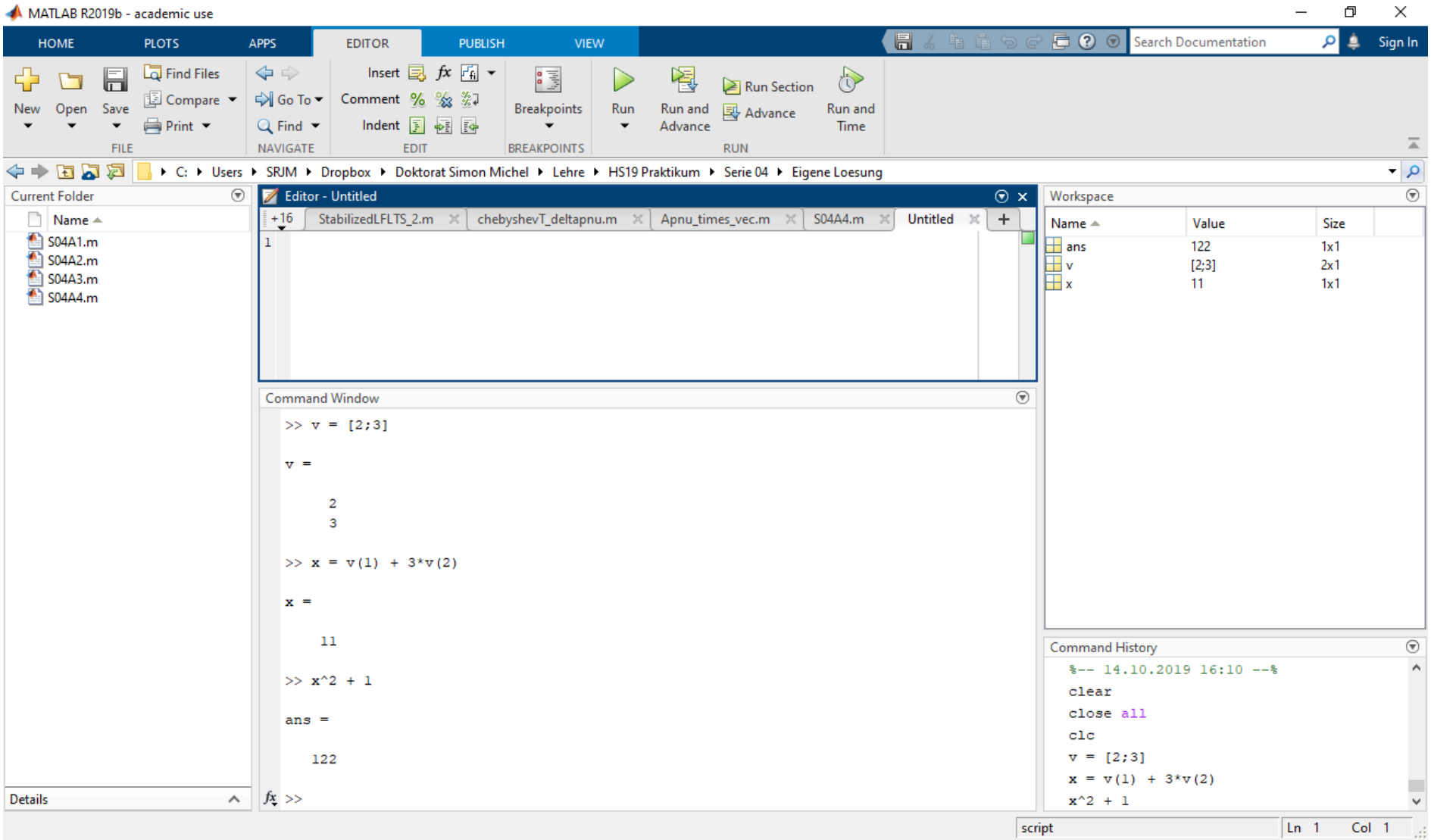
Matlab Features:

1. Command Window zum direkten Ausführen von Matlab-Befehlen
2. Command History Liste ausgeführter Befehle. Erneutes Ausführen durch Anklicken
3. Current Folder Aktuelles Verzeichnis — hier werden selbstprogrammierte Befehle gesucht.
4. Workspace Liste aller aktuell belegten Variablen
5. Editor zum Schreiben von eigenen Befehlen

Befehle stets mit Return beenden.

Abbruch: `strg+C`.

Beispiel


MATLAB R2019b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Sign In
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
C: ▸ Users ▸ SRJM ▸ Dropbox ▸ Doktorat Simon Michel ▸ Lehre ▸ HS19 Praktikum ▸ Serie 04 ▸ Eigene Loesung
Current Folder
Name
S04A1.m
S04A2.m
S04A3.m
S04A4.m
Details
Editor - Untitled
+16
StabilizedLFLTS_2.m
chebyshevT_deltapnu.m
Apnu_times_vec.m
S04A4.m
Untitled
1
Command Window
>> v = [2;3]
v =
2
3
>> x = v(1) + 3*v(2)
x =
11
>> x^2 + 1
ans =
122
>>
Workspace
Name
Value
Size
ans
122
1x1
v
[2;3]
2x1
x
11
1x1
Command History
%-- 14.10.2019 16:10 --%
clear
close all
clc
v = [2;3]
x = v(1) + 3*v(2)
x^2 + 1
script
Ln 1
Col 1

Matlab-Bedienung und Hilfe

Aufgrund der grossen Anzahl von Befehlen und weiterer Optionen ist es unmöglich, sich die volle Funktionalität und Syntax zu merken. Deswegen gibt es Hilfen:

1. `help` + Funktionsname gibt die Hilfe dazu aus
2. Matlab-Hilfsmenü `doc`: Umfangreiche Suchmöglichkeiten
3. Buchstabenfolge + Tabulator: Automatische Befehlsergänzung
4. Pfeil-nach-oben: Befehlsverlauf

Literatur:

- Gute deutschsprachige Einführung:
  `www.hs-ulm.de/users/gramlich/EinfMATLAB.pdf`

Matlab als Taschenrechner

Operationen auf Matrizen

Erzeugen von Matrizen:

```
[1 2 3 ; 4 5 6]      zeros(2,3)      eye(5)
```

Erzeugen von speziellen Zeilenvektoren:

`1:3` ergibt den Vektor

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}.$$

`1:0.2:2` ergibt

$$\begin{pmatrix} 1.0 & 1.2 & 1.4 & 1.6 & 1.8 & 2.0 \end{pmatrix}.$$

Zuweisung (um Werte in einer Variablen abzuspeichern):

```
c1 = 6              A = [1 2 3 ; 4 5 6]      v = 1:10
x = [1;2;3]         y = [1 2 3]
```

Command Window

```
>> c1 = 6

c1 =

     6

>> A = [1 2 3 ; 4 5 6]

A =

     1     2     3
     4     5     6

>> v = 1:10

v =

     1     2     3     4     5     6     7     8     9    10

>> x = [1;2;3]

x =

     1
     2
     3

>> y = [1 2 3]

y =

     1     2     3

fx >>
```

Matlab als Taschenrechner

Operationen auf Matrizen

Verkleben von Matrizen:

`[A, zeros(2,2)]` ergibt

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \end{pmatrix}$$

und `B = [A, zeros(2,2); eye(2), A]` speichert die Matrix

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 1 & 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 5 & 6 \end{pmatrix}.$$

in der Variablen *B* ab.

Matlab als Taschenrechner

Operationen auf Matrizen

Elementauswahl für Zugriff und Zuweisung:

```
A(1,2) A(1:2,2) A(1:2,1:3) A(:,[1 3])
```

Elementweise *arithmetische* Operationen:

```
+   -   .*   ./   .^   .'
```

Elementweise Funktionen:

```
abs   sin   cos   exp   sqrt   min   max ...
```

Elementweise logische Operationen:

Das Ergebnis ist 0 (`false`) bzw. 1 (`true`).

```
 ==   ~=   <   >   <=   >=   &   |   ~
```

Matlab als Taschenrechner

Arithmetische Operationen der Linearen Algebra

Grundrechenoperationen der Linearen Algebra

`+` `-` `*` `^`

Weitere Operationen:

`A\B`	$A^{-1} \cdot B$
`A'`	$A^\top$ (falls A reell)
`det(A)`	$\det(A)$
`inv(A)`	A^{-1}
`rank(A)`	$\operatorname{rang}(A)$

Matlab als Taschenrechner

Das Lösen von linearen Gleichungssystemen

Eingabe:

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

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
b = [ 3 ; 5 ]
```

$$b = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

```
A\b
```

$$A^{-1}b$$

Ausgabe ist die Lösung von $Ax = b$:

```
ans =
    -1
     2
```

Funktionen einer Veränderlichen zeichnen

Matlab zeichnet keine Funktionen, sondern Wertetabellen als Polygonzug!

1. Definiere Spaltenvektor `x` von x-Werten, z.B.
   ```
   x = (0 : pi/100 : pi)';
   ```
2. Definiere Spaltenvektor `y` von y-Werten einer Funktion, z.B.
   ```
   y = sin(x);
   ```
3. Zeichne Wertetabelle
   ```
   plot(x,y);
   ```
4. Alternativ:
   ```
   plot(x,sin(x));
   ```

Beachte: Die Funktion, die gezeichnet werden soll, muss elementweise auf `x` anwendbar sein!

Funktionen einer Veränderlichen zeichnen

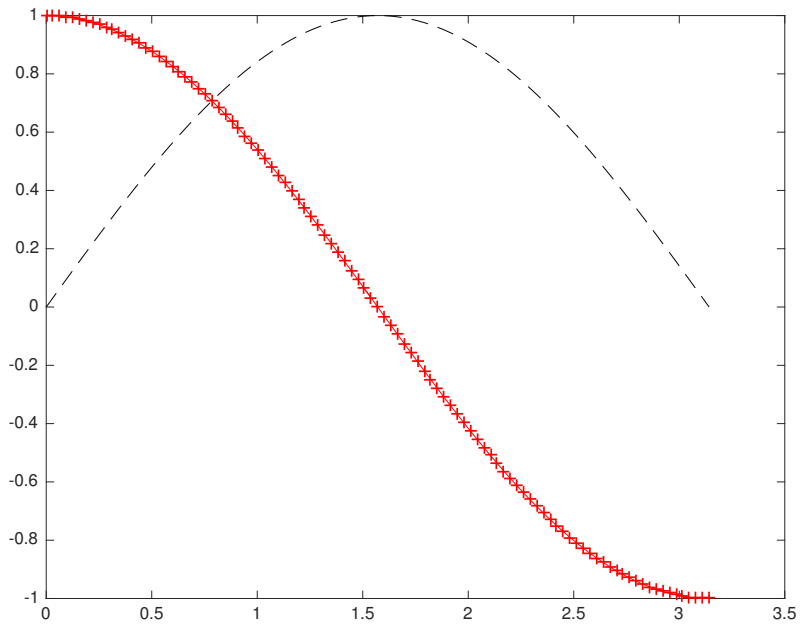
Um mehrere Funktionen zu zeichnen, geht man wie folgt vor:

1. Definiere Spaltenvektor `x` von x-Werten, z.B.
   ```
   x = (0 : pi/100 : pi)';
   ```
2. Definiere Spaltenvektoren `y` und `z` von y-Werten von Funktionen, z.B.
   ```
   y = sin(x); z = cos(x);
   ```
3. Zeichne Wertetabellen
   ```
   plot(x,y,x,z);
   ```
4. Alternativ:
   ```
   plot(x,[y,z]);
   plot(x,sin(x),x,cos(x));
   ```

Plotoptionen und Modifizierung von Graphen

1. Weitere Optionen beim Zeichnen von Kurven, z.B.

```
plot(x,sin(x),'k--',x,cos(x),'r-+')
```



Plotoptionen und Modifizierung von Graphen

2. Modifizierung der Achsen:

```
axis([xmin xmax ymin ymax])
axis equal
xlim([xmin xmax])
```

usw.

3. Beschriftung:

```
title('Überschrift')
xlabel('x-Achse')
legend('Graph 1','Graph 2','Graph 3')
```

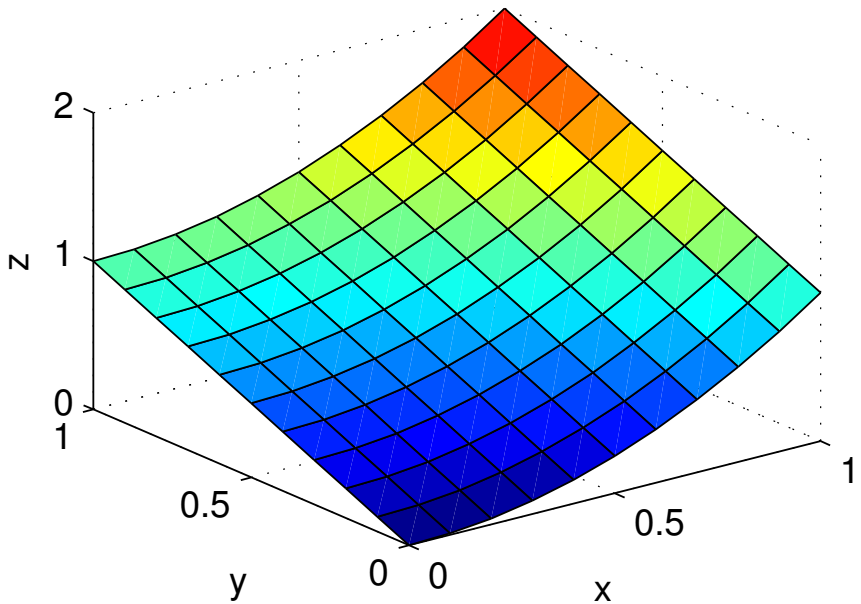
4. Weitere Modifizierungen direkt am Graphen möglich!

Funktionen zweier Veränderlichen zeichnen

Grundlagen

Beispiel: Zeichne

$$z = f(x, y) = x^2 + y, \qquad x \in [0, 1], \quad y \in [0, 1].$$



Funktionen zweier Veränderlichen zeichnen

Grundlagen

Es werden Matrizen `X` und `Y` benötigt, so dass die elementweise Auswertung von `X` und `Y` eine Matrix `Z` mit den Funktionswerten liefert, z.B.

$$X = \begin{pmatrix} 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 \end{pmatrix}$$

ergibt für $z = f(x, y) = x^2 + y$

$$Z = \begin{pmatrix} 0 & 0.25 & 1 \\ 0.5 & 0.75 & 1.5 \\ 1 & 1.25 & 2 \end{pmatrix}$$

Funktionen zweier Veränderlichen zeichnen

Vorgehensweise

1. Erzeuge Matrizen X und Y, z.B.

```
[X,Y] = meshgrid(0:0.1:1, 0:0.1:1);
```

   Genauer: sind `x` und `y` Vektoren der x- bzw. y-Werte bei denen die Funktion f ausgewertet werden soll, so werden die benötigten Matrizen `X` und `Y` erzeugt durch:

```
[X,Y] = meshgrid(x,y);
```

2. Erzeuge Matrizen Z der Funktionswerte, z.B.

```
Z = X.^2+Y;
```

3. Zeichne Funktion

```
surf(X,Y,Z);
```

Verzweigung: if-elseif-else-end

```
if n == 1
      Befehle1
elseif n == 2
      Befehle2
elseif n == 3
      Befehle3
      ⋮
elseif n == N
      BefehleN
else
      BefehleAlt
end
```

elseif und else mit den darauf folgenden Befehlen sind optional.

Die for-Schleife

```
for k = 4:n
      Befehle
end
```

1. Zunächst ist $k = 4$ und es werden alle Befehle zwischen `for` und `end` mit dem Wert $k = 4$ ausgeführt.
2. Es wird $k = 5$ gesetzt und alle Befehle zwischen `for` und `end` mit dem Wert $k = 5$ ausgeführt, usw.
3. Es werden alle Werte von k durchlaufen, bis einschliesslich $k = n$.

Genauer: `for k =` *Vektor*
k durchläuft den *Vektor* von Anfang bis Ende.

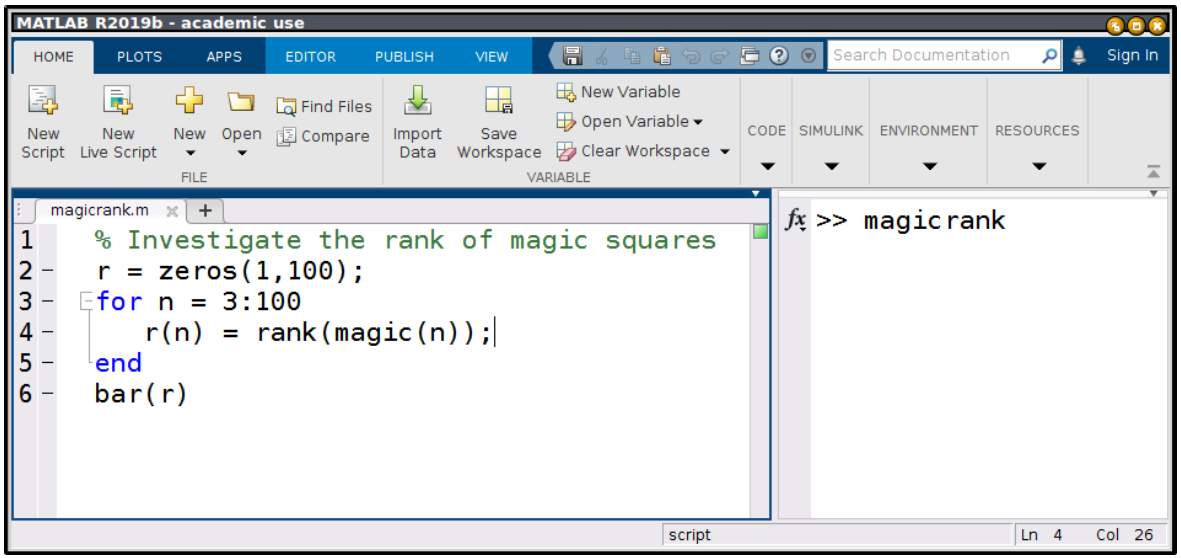
Die while-Schleife

```
while t > 0
      Befehle
end
```

1. Falls $t > 0$ gilt, so werden alle Befehle zwischen `while` und `end` ausgeführt.
2. Es wird wieder geprüft, ob $t > 0$ gilt. Falls dies erfüllt ist, so werden wieder alle Befehle zwischen `while` und `end` ausgeführt.
3. Dies wiederholt sich solange bis $t > 0$ nicht erfüllt ist.

Skripte

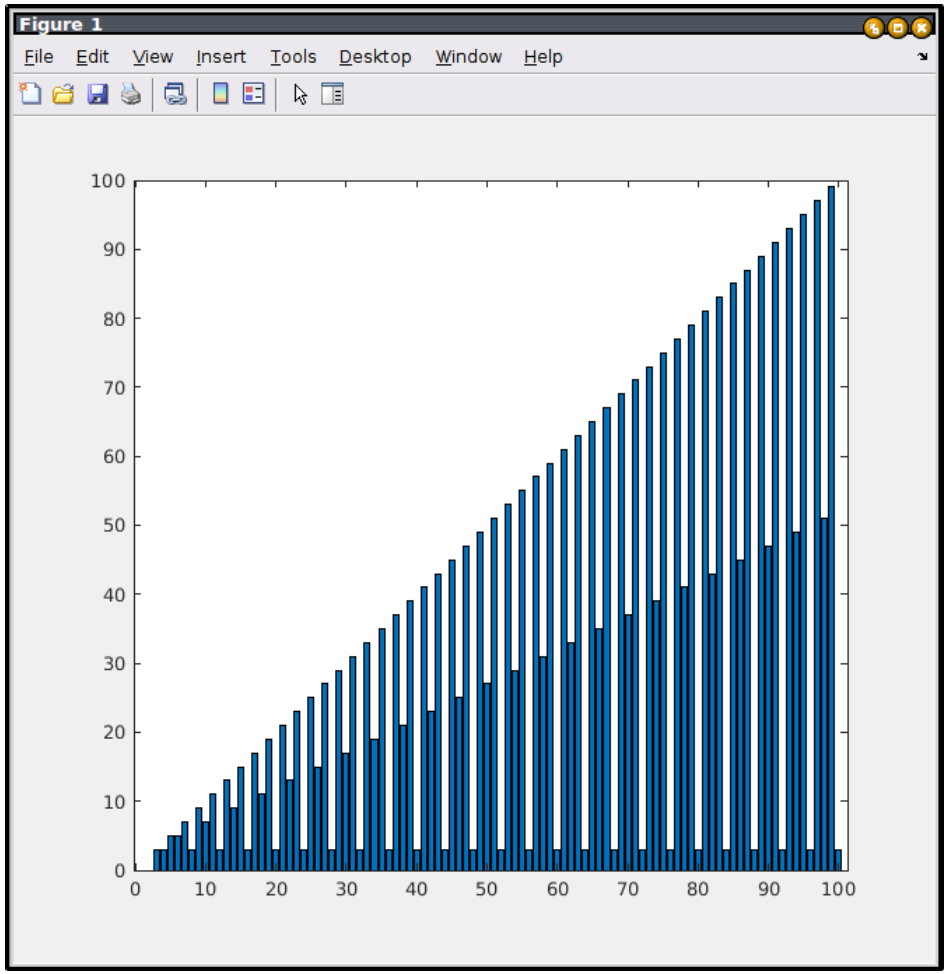
Magische Quadrate



Eingabe von `magicrank` führt `magicrank.m` aus.
Auf aktives Verzeichnis achten!

Skripte

Magische Quadrate: Ausgabe



Skripte

Erläuterungen

- Eine Datei, welche eine Abfolge von Befehlen enthält, heisst Skript oder Programm.
- Name der Datei beliebig, Dateierweiterung muss `.m` sein.
- Aufruf des Skripts (ohne Dateierweiterung) führt Skript sequenziell aus.
- Semicolon `;` unterdrückt die Ausgabe.
- Kommentare beginnen mit einem Prozentzeichen `%`.

Skripte

Einschränkungen

Einschränkungen bei Namen von Variablen und Dateien

- alle Namen müssen sich unterscheiden
- Gross- und Kleinbuchstaben werden unterschieden
- Namen müssen mit einem Buchstaben beginnen
- Namen dürfen keine Sonderzeichen enthalten

Skripte und Funktionen

Einschränkungen

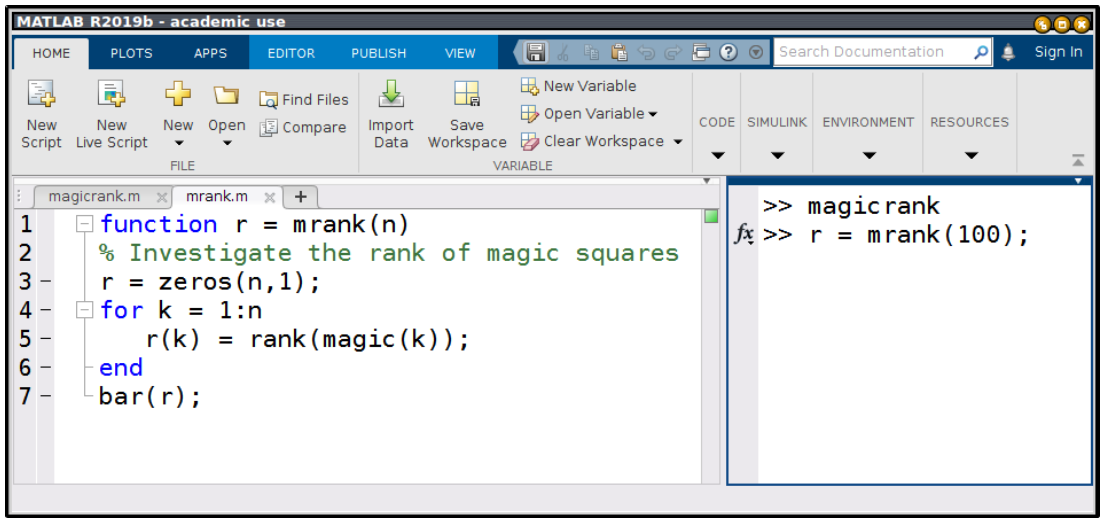
Skripte bieten keine:

- Übergabe von Parametern
- Rückgabe von Werten, Ergebnisse

Lösung: Funktionen.

Weiterer Unterschied: *alle* Parameter *müssen* übergeben werden!

Funktionen



Die Eingabe von `mrank(100)` liefert dasselbe Ergebnis.

Funktionen

- Dateiname und Funktionsname müssen übereinstimmen!
- Die Dateierweiterung muss `.m` sein.
- Alle Variablen sind lokal.
- Die Funktion `function r = mrank(n)` bedarf eines Eingabeparameters und gibt eine Variable als Ergebnis zurück.
- Rückgabe mehrerer Variablen:

```
function [r,k] = mrank(n)
```

Aufruf: `[z,m] = mrank(100);`

Speicherverwaltung

- Wenn man auf einen Matrixeintrag zugreifen will, etwa durch `A(2,3:6)`, so muss die Matrix mindestens die Größe 2×6 besitzen und entsprechend **vorher** definiert werden, etwa als `A = zeros(2,10)`.

  Wenn man auf `A(2,3:6)` zugreift, ohne die Matrix vorher zu definieren oder falls sie zu klein ist, so wird sie automatisch als Nullmatrix der notwendigen Größe angelegt oder entsprechend vergrößert. Trotzdem sollte man die Matrix vorher selbst definieren weil:
  - die Größenanpassung sehr viel Laufzeit kostet
  - Definition der Matrixgröße am Anfang ist guter Programmierstil
- Der Speicher nicht mehr benötigter Variablen wird durch

  `clear` oder `clear`+Variablename

  freigegeben.
- `clc` löscht alle vorherigen Eingaben und Ausgaben im Command Window.

Matrix-Vektor-Schreibweise

Es gibt (mindestens) zwei Möglichkeiten, den Vektor

$$v = \begin{pmatrix} 1 & -2 & 3 & -4 & \cdots & \cdots & 999 & -1000 \end{pmatrix}$$

zu erzeugen:

```
v = 1:1000;
for i = 1:500
    v(2*i) = -v(2*i);
end
```

```
v = 1:1000;
v(2:2:1000) = -v(2:2:1000);
```

Die zweite Möglichkeit verwendet statt Schleifen die vektorielle Rechnung. Das ist viel schneller!

Man sollte daher alle Matrix-Vektor-Operationen möglichst ohne for-Schleifen umsetzen.

Mathematische Funktionen definieren

Beispiel: Die Funktion

$$f(x) = \frac{1}{1+5x^2}, \qquad x \in \mathbb{R}$$

soll als Matlab-Funktion `f` implementiert werden:

`f.m`

Mathematische Funktionen definieren

Beispiel: Die Funktion

$$f(x) = \frac{1}{1+5x^2}, \qquad x \in \mathbb{R}$$

soll als Matlab-Funktion `f` implementiert werden:

`f.m`

```
function y = f(x)

eins = ones(size(x));
y = eins ./ (eins + 5 * x.^2);
```

Mathematische Funktionen definieren

Beispiel: Die Funktion

$$f(x) = \frac{1}{1+5x^2}, \qquad x \in \mathbb{R}$$

soll als Matlab-Funktion `f` implementiert werden:

`f.m`

Alternativ:

```
function y = f(x)
y = 1 ./ (1 + 5 * x.^2);
```