Tipps und Tricks für Matlab

J. Schweitzer

Sommersemester 2012

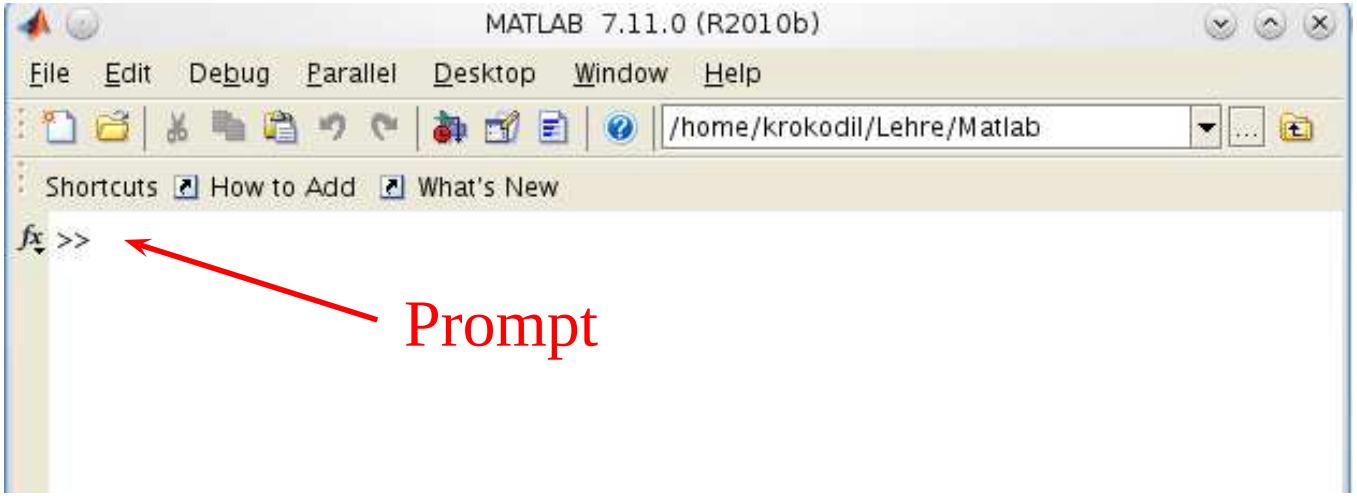
Inhalt

Matlab als Taschenrechner

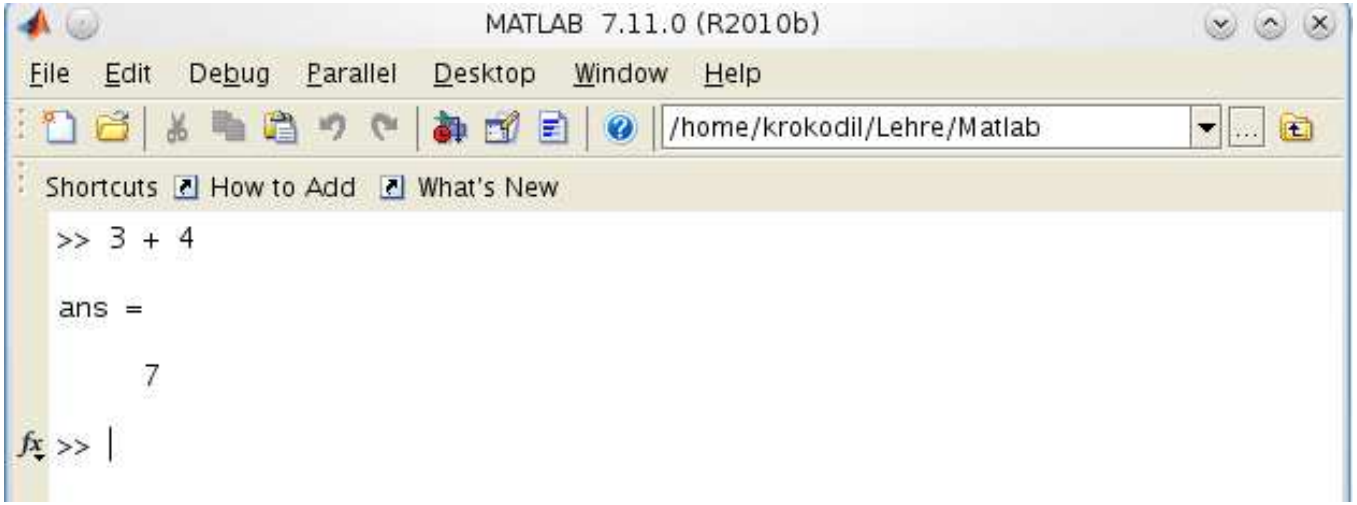
- Prompt
- Elementare Rechnungen
- Wichtige Funktionen
- Variablenzuweisung
- Matlabhilfe

Taschenrechner?

- Prompt



- 3 + 4 ¶



Zahlen, Komandos und Funktionen

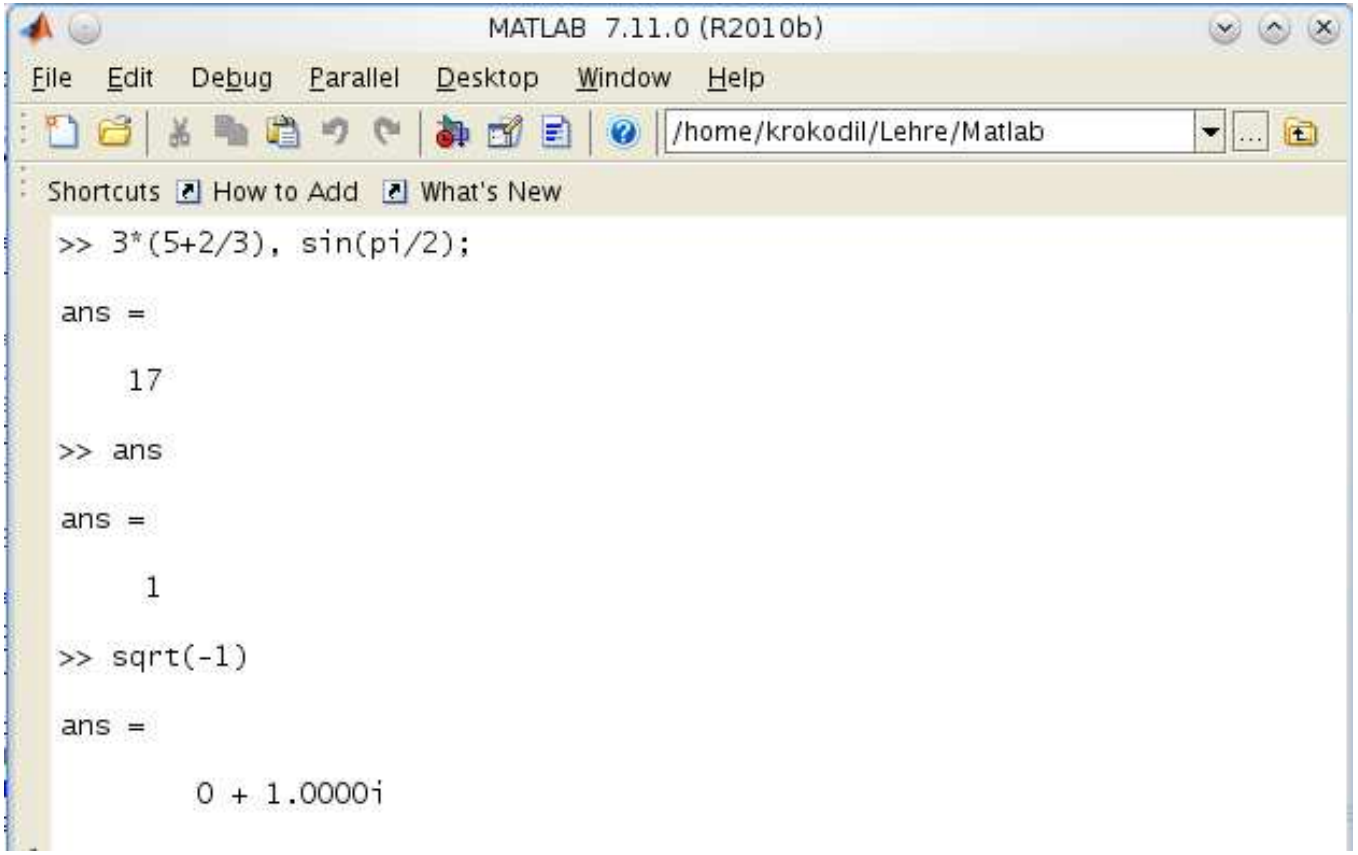
- Dezimalzahlen mit Punkt: `3.5`
- Grundrechenarten: `+ - * /^`
- Punkt vor Strich und Klammergesetze gelten: `()`

- Komandos können durch Kommata bzw. Semicolons getrennt oder abgeschlossen werden.
- Beim Semicolon wird zusätzlich die Ausgabe unterdrückt.

- Bekannte Konstanten: `pi, i, j`
- Bekannte Funktionen: `sqrt(), sin(), cos(), tan(), asin(), sinh(), exp(), log(), nchoosek(n,k), factorial()`

Zwischenergebnisse werden in `ans` gespeichert.

Beispiel


MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
/home/krokodil/Lehre/Matlab
Shortcuts How to Add What's New
>> 3*(5+2/3), sin(pi/2);
ans =
17
>> ans
ans =
1
>> sqrt(-1)
ans =
0 + 1.0000i

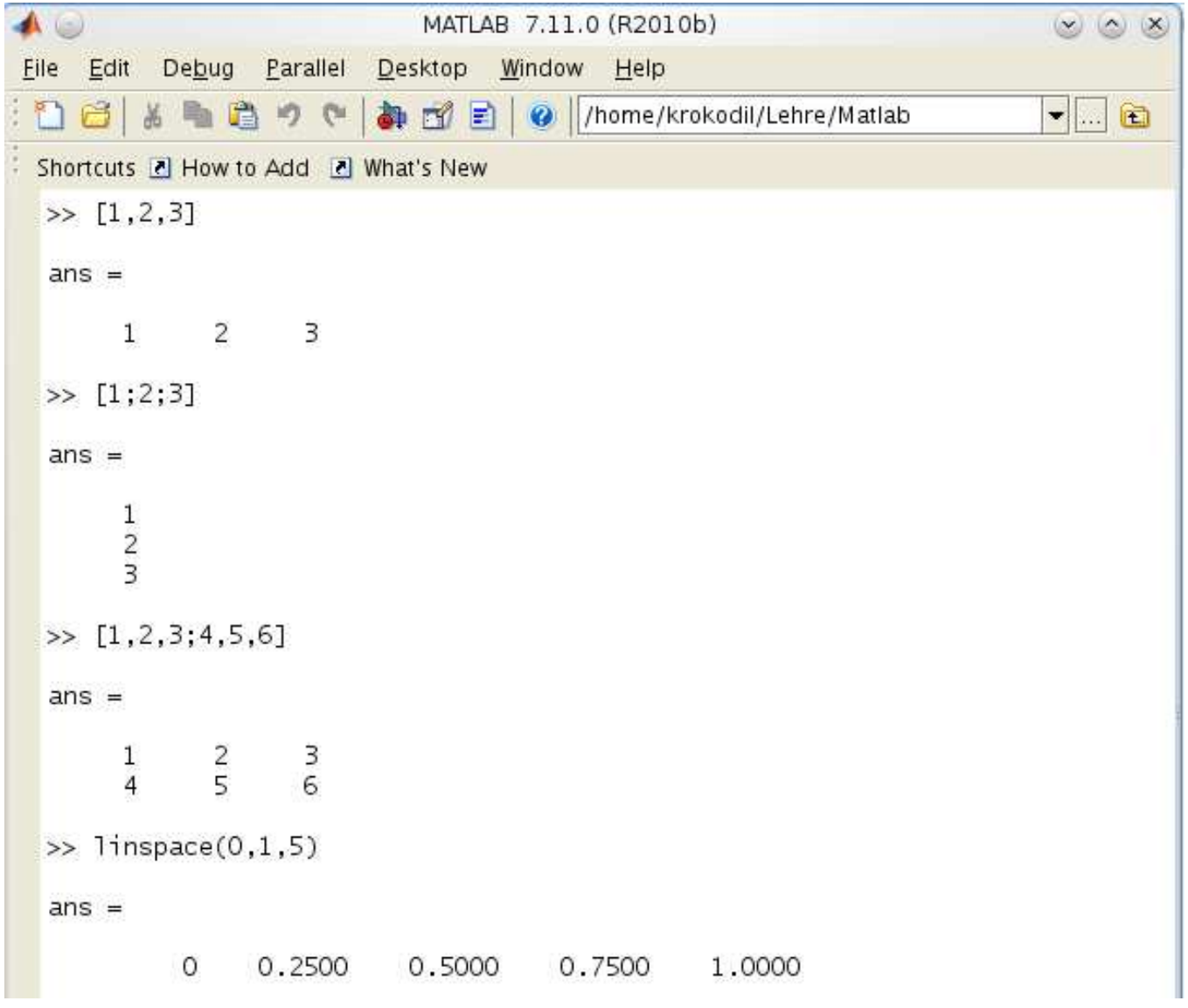
Ein paar Vektoren und Matrizen

- Vektoren und Matrizen werden manuell in eckigen Klammern definiert. `[]`
- Zeilen werden durch Semicolons getrennt, Spalten durch Kommata.

Befehle zur Erstellung und Manipulation von Matrizen und Vektoren.

- Matrizenrechenregeln gelten.
- `linspace(l1,l2,n)` erstellt einen Vektor mit n äquidistanten Einträgen von l_1 bis l_2.
  `logspace` erstellt einen Vektor mit logarithmischen Abständen.
- `l1:h:l2` beginnt mit l_1 und addiert solange h bis das Ergebnis l_2 passiert.
- `reshape` verändert das Verhältnis von Zeilen zu Spalten.
- `kron` berechnet das Kronnecker-Produkt von zwei Matrizen.
- Weitere Befehle: `flipud, fliplr, repmat, rot90, diag, tril, triu`

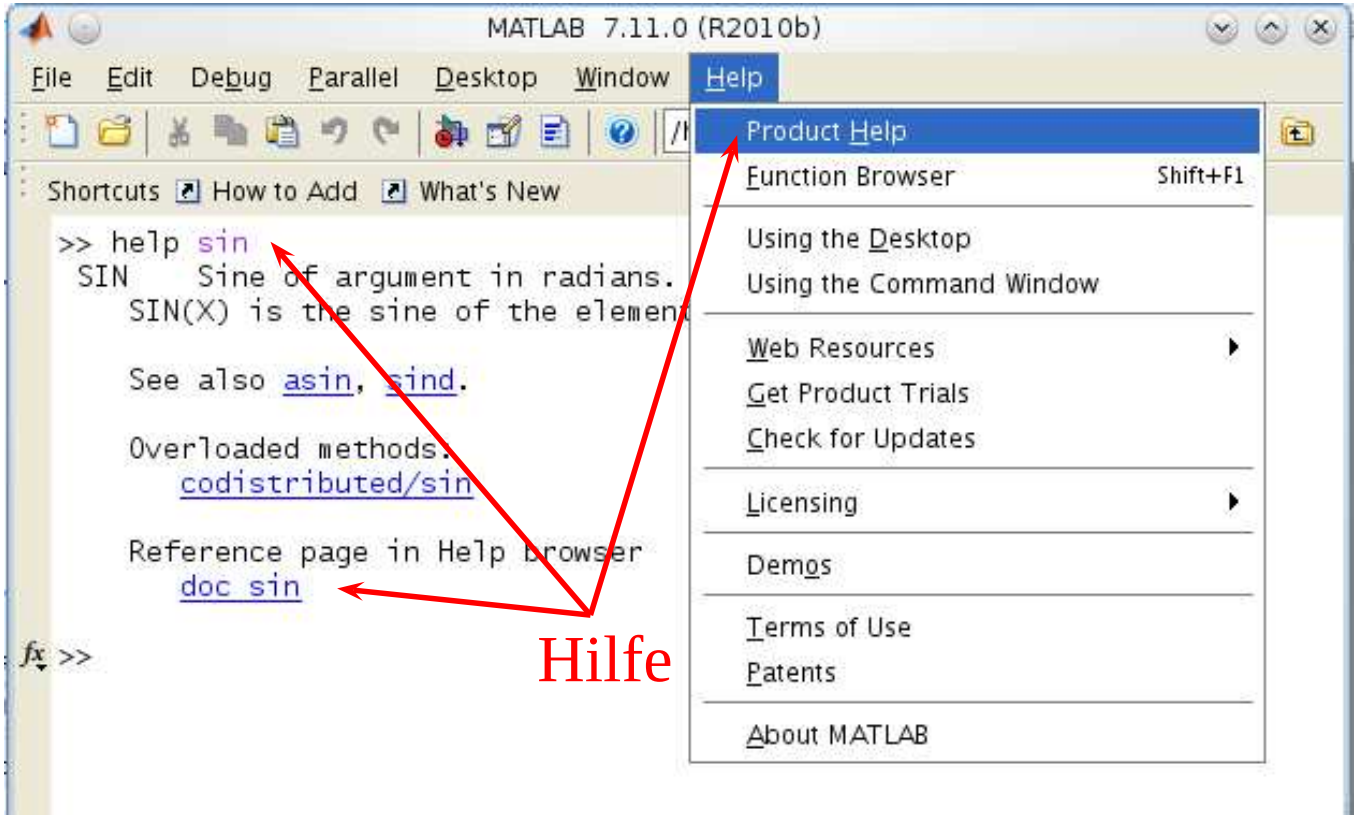
Beispiel


MATLAB 7.11.0 (R2010b)
File Edit Debug Parallel Desktop Window Help
/home/krokodil/Lehre/Matlab
Shortcuts How to Add What's New
>> [1,2,3]
ans =
1 2 3
>> [1;2;3]
ans =
1
2
3
>> [1,2,3;4,5,6]
ans =
1 2 3
4 5 6
>> linspace(0,1,5)
ans =
0 0.2500 0.5000 0.7500 1.0000

Variablen

- Zuweisung von Werten auf Variablen.
- Werte werden gespeichert und können später wieder abgerufen werden.
- Variablen können jederzeit überschrieben oder gelöscht werden.
- `whos` gibt Übersicht über aktuelle Variablen.
- `clear var` löscht die Variable `var`.
- `clear all` löscht alle Variablen.

Matlabhilfe



Aufgaben

- Bilden Sie in Matlab die Telefonmatrix auf mindestens drei verschiedene Arten.
- Informieren Sie sich mit der Matlabhilfe über die Befehle aus diesem Abschnitt.

Datenformate

- Arrays
- Strings
- Function handles
- Cell Arrays
- Structure Arrays

Zahlen, Vektoren, Matrizen. Alles Arrays?

- Array: Anordnung von Zahlen (double/complex/single)
- Leeres Array:
  mindestens eine der initialisierten Dimensionen hat die Länge 0.
- Zahl $\widehat{=}$ 1×1-Array
- Vektor $\widehat{=}$ $1 \times d$ oder $d \times 1$-Array
- Matrix $\widehat{=}$ $m \times n$-Array
- Multidimensionale Arrays:
  beliebig viele Indizes möglich

Rechenregeln

- Matrizen-Rechenregeln:
  - `A*B` falls $A \in \mathbb{C}^{m,k}$, $B \in \mathbb{C}^{k,n}$
  - `A+B` falls $A, B \in \mathbb{C}^{m,n}$
  - `A^2` falls $A \in \mathbb{C}^{n,n}$
- Ausnahme: 1×1-Matrizen
  - `s*A` $\Rightarrow$ $(sa_{ij})_{ij}$
  - `s+A` $\Rightarrow$ $(s + a_{ij})_{ij}$
- Punktweise Operationen für Arrays identischer Dimensionen
  - `A.*B` $\Rightarrow$ $(a_{ij}b_{ij})_{ij}$
  - `A.^2` $\Rightarrow$ $(a_{ij}^2)_{ij}$

Indizes von Arrays

- Zeilen- und Spaltenvektoren:
  `v(k)` $\Rightarrow$ k-tes Element des Vektors für $v \in \mathbb{C}^n$ und $k = 1, \ldots, n$
- Matrizen mit Paaren von Zeilen- und Spaltenindizes:
  `M(k,l)` $\Rightarrow$ Element in der k-ten Zeile und l-ten Spalte für $M \in \mathbb{C}^{m,n}$, $k = 1, \ldots, m$ und $l = 1, \ldots, n$
- Matrizen mit nur einem Index:
  `M(i)` $\Rightarrow$ Element in der k-ten Zeile und l-ten Spalte für $M \in \mathbb{C}^{m,n}$, $k = 1, \ldots, m$ und $l = 1, \ldots, n$, falls $i = (l-1)n + k$
  (Elemente Spaltenweise durchnummeriert)

Man kann größere Anteile von Arrays ansteueren, wenn man Vektoren von Indizes benutzt.

Andere Datenformate

- Strings: “Array von Characters”
  `str = 'Das ist ein String.'` $\Rightarrow$ `str(3) = 's'`
- Function Handle: erlaubt das Speichern von Funktionen, z.B. als Eingabeelement anderer Funktionen.
  `f = @sin` speichert die Sinusfunktion
  Spätere Auswertung: Benutze `f` genauso wie die zugewiesene Funktion
  `y = f(x)` liefert das gleiche wie `y = sin(x)`
- Cell-Arrays: “Arrays von beliebigen Datenformaten”

```
A = {'String', @sin, [5,3]}   ← Cell-Array
A{3} = [5,3]                  ← normales Array
A{3}(1) = 5                   ← double
A{1} = 'String'               ← Character
A(1:2) = {'String', @sin}     ← Cell-Array
```

Andere Datenformate

- Structure-Array: “Benannte Datensätze”
```
Student.Name = 'Julia Schweitzer'
Student.Semester = 3
Student.Lieblingsfunktion = @sin
Student.Noten = [2 1 3 2 2]
```
- Jedes Feld kann beliebige Daten aufnehmen (Zahlen, Arrays, Cells, Funktionen, Sturctures)
- Arrays von Structures:
```
Student(2).Name = 'Test Student'
```

Nützliche Befehle dazu erhält man mit `help struct` in Matlab.

Aufgaben

- Erstellen Sie die Matrix mit den Elementen

$$M_{ij} = 3j \sin(5i) \ , \quad i = 1, \ldots 35 \ , \quad j = 1, \ldots 24$$

  in Matlab. Erstellen Sie eine weitere Matrix, die alle ungeraden Zeilen und alle geraden Spalten der ersten Matrix enthalten, wobei die Spalten in umgekehrter Reihenfolge angeordnet sind.
- Erstellen Sie eine 35×24-Matrix, deren Elemente zeilenweise von 1 bis $35 \cdot 24$ heraufzählen. Erstellen Sie eine weitere Matrix, die jede 3. Spalte und jede 5. Spalte von hinten angefangen der ersten Matrix enthält.

M-files

- Workspace
- Scripte
- Funktionen

Workspace und Scripte

- Aktueller Arbeitsplatz
- hier initialisierte Variablen sind hier verfügbar
- Funktionen aus aktuellem Searchpath sind verfügbar

Scripte:

- M-file (`dateiname.m`) ohne “Syntax-Vorschrift”
- Ansammlung von Befehlen, die im aktuellen Workspace ausgeführt werden

Funktionen

- i.A. eigener Workspace
- kennt nur Eingabevariablen
- löscht alles nach beenden der Funktion

Sorten von Funktionen:

- Kommando-orientierte “anonyme” Funktionen
- permanent gespeicherte Funktionen (M-file)
  - Primärfunktion
  - Subfunktionen
  - geschachtelte (“nested”) Funktionen
  - private Funktionen
  - “overloaded” Funktionen

Anonyme Funktionen

Listing 1: Beispiele für anonyme Funktionen.

```
1 f = inline('x.^2 + y.^2', 'x', 'y');
2 g = @(r, phi) (r.*exp(i*phi));
3 h = @sin;
```

- `f = inline('expr', 'in1', ...)`
- `f = @(in1, ...)  (expr)`
- `f = @sin`

Kurzer Inhalt, man brauch keine Datei zu verwalten.

Funktionsdateien

Listing 2: Matlab Funktionstemplate.

```
1  function [out1, out2] = fct_template(in1, in2, in3)
2  % [out1, out2] = fct_template(in1, in2, in3)
3  % Hilfreicher Kommentar fuer Matlab-Hilfe
4
5
6  % Programm-statements, die aus den Variablen in1, in2, in3
7  % die Ausgabe ou1, out2 erzeugt.
```

- komplizierte Funktionen, nicht in einer Zeile zu definieren
- Funktionen, die oft und an unterschiedlichen Stellen wiederverwendet werden sollen.
- Nur eine Ausgabe:
  `function out = fct_name(in1, in2, ...)`
- Keine Ausgabe:
  `function fct_name(in1, in2, ...)`
- Keine Eingabe:
  `function [out ...]  = fct_name`

Aufgaben

- Finden Sie Beispiele, wo es angebracht wäre, ein Script, eine anonyme oder eine echte Funktion zu verwenden.
- Untersuchen Sie mithilfe des Kommandos `whos`, in welchen Workspaces welche Variablen bekannt sind. Schreiben Sie dazu kleine Testfunktionen.

Schleifen und Abfragen

- `if`-Abfragen
- `switch case`-Abfragen
- `for`-Schleifen
- `while`-Schleifen

if-Abfrage

Listing 3: Beispiel einer if-Abfrage.

```
1  if logische Aussage 1
2    % Die folgenden Statements werden ausgefuehrt, wenn die
3    % logische Aussage wahr ist.
4    Statements 1
5  elseif logische Aussage 2
6    % Diese Abfrage wird ueberprueft, falls die erste
7    % nicht zutrifft.
8    Statements 2
9  else
10   % Diese Statements werden nur ausgefuehrt, wenn keine der
11   % oberen logischen Aussagen wahr ist.
12   Statements 3
13 end
```

Logische Abfragen:

- `var == 1`
- `var > 0`
- `norm(var - ref) <= tol`
- `var1~= var2`

switch case-Abfrage

Listing 4: Beispiel einer `switch case`-Abfrage.

```
1 switch flag
2  case 'case1'
3   Statements
4  case 'case2'
5   Statements
6  otherwise
7   Statements
8 end
```

- Die Stringvariable `flag` wird mit den Strings der cases verglichen und entsprechende Statements werden ausgeführt.
- Die Statements im Block `otherwise` werden ausgeführt, wenn `flag` keinem der Strings entspricht.

`for`-Schleife

Listing 5: Beispiel einer `for`-Schleife.

```
1  for k = irgendein Vektor
2    % In der Schleife werden der Reihe nach alle Eintraege des
3    % Vektors durchlaufen und die Laufvariable k nimmt in jedem
4    % Durchlauf den naechsten Wert an.
5
6    Statements(k)
7  end
```

Mögliche Laufvariablen:

- Einfache Indizes: `k = 1:N` (zählt in Einerschritten hoch)
- Rückwärtslaufende Indizes: `k = N:(-1):0`
- Parameter: `k = [p1 p2 p3 ...]`

`while`-Schleife

Listing 6: Beispiel einer `while`-Schleife.

```
1 while logische Abfrage
2   % Die Schleife wird solange durchlaufen, wie die
3   % Abfrage eine wahre Aussage ergibt
4   Statements
5 end
```

- Vor der Schleife müssen die Daten für die Abfrage definiert werden.
- In der Schleife sollten diese angepasst werden. (Gefahr einer Endlosschleife)

Listing 7: `do until`-Variatnte.

```
1 fertig = 0;
2
3 while ~fertig
4   % Die Schleife wird solange durchlaufen, wie die
5   % Abfrage eine wahre Aussage ergibt
6   Statements
7
8   fertig = neue logische Aussage
9 end
```

Warnung!

Schleifen sind SEHR langsam!
Am besten vermeiden!

- Schleifen sind unumgänglich, wenn die Rechnungen von vorangegangenen Durchläufen abhängen.
- Wenn die Daten und Rechnungen unabhängig von einander sind, braucht man keine Schleife.
- Wenn es für bestimmte Schleifen vorgefertigte Routinen gibt (z.B. Matrixmultiplikationen), sind diese viel effizienter.

Aufgaben

- Finden Sie ein Beispiel, bei dem Standard-Matlab-Routinen schneller sind, als entsprechende Schleifen.
- Informieren Sie sich in die Matlabhilfe über den Befehl `input`. Schreiben Sie ein Script, wo Sie in Abhängigkeit der Eingaben eines Benutzers unterschiedliche Ausgaben produzieren.

2D Plots

- `plot`, `semilogx`, `semilogy`, `loglog`
- Mehrere Plots in einem Bild
- Line styles
- Titel, Legenden und Achsenskalierung und -beschriftungen
- `figure` und `subplot`

Argumente von `plot`

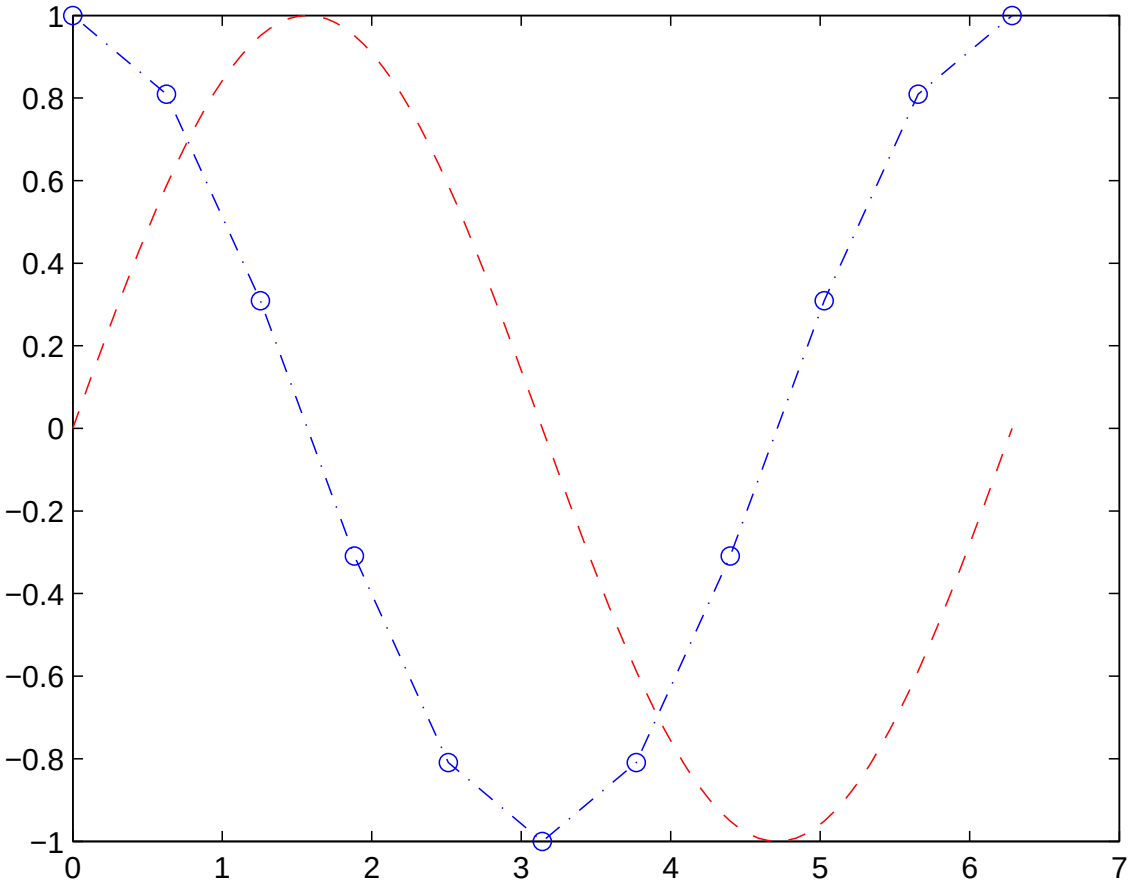
- `plot(y)` trägt die Werte des Vektors `y` über ihren Indizes auf.
- `plot(x,y)` trägt für zwei gleichlange Vektoren die Werte von `y` über den zugehörigen Werten von `x` auf.
- `plot(x,y,'c-*')` produziert den selben Plot wie vorher nur mit benutzerdefinierten Farben, Linestyle und Markern.
- Bei den beiden letzten Versionen kann man beliebig viele Gruppen von Plots hintereinander als Argument geben.
- `x` und `y` können Matrizen der gleichen Dimension sein. Es werden zeilenweise die Daten gegeneinander geplottet. Falls `x` nur aus einer Zeile besteht, werden alle Zeilen von `y` gegen die selben Daten `x` geplottet.

Beispiel

Listing 8: Plotbeispiel 1.

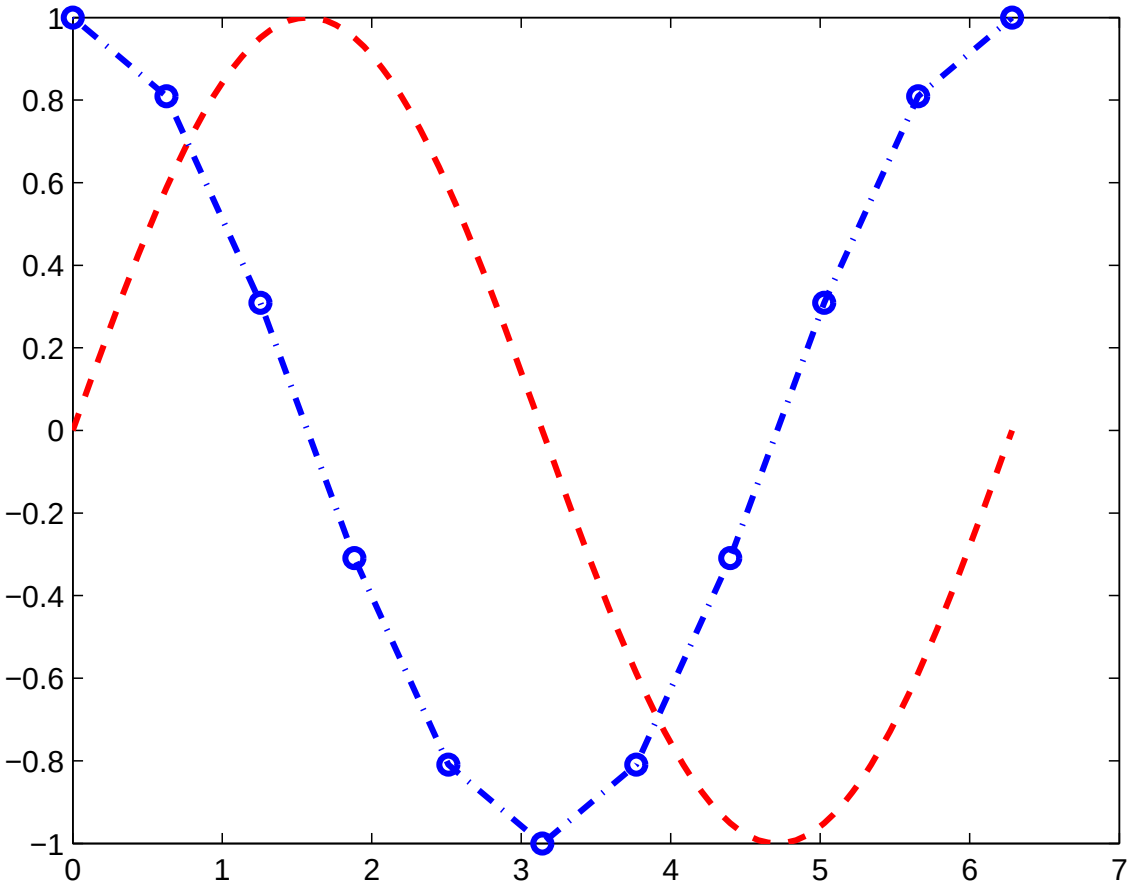
```
1  clear all
2  close all
3
4  x1 = linspace(0,2*pi,101);
5  y1 = sin(x1);
6  x2 = x1(1:10:101);
7  y2 = cos(x2);
8
9  plot(x1, y1, 'r--', x2, y2, 'b-.o')
```

Beispiel



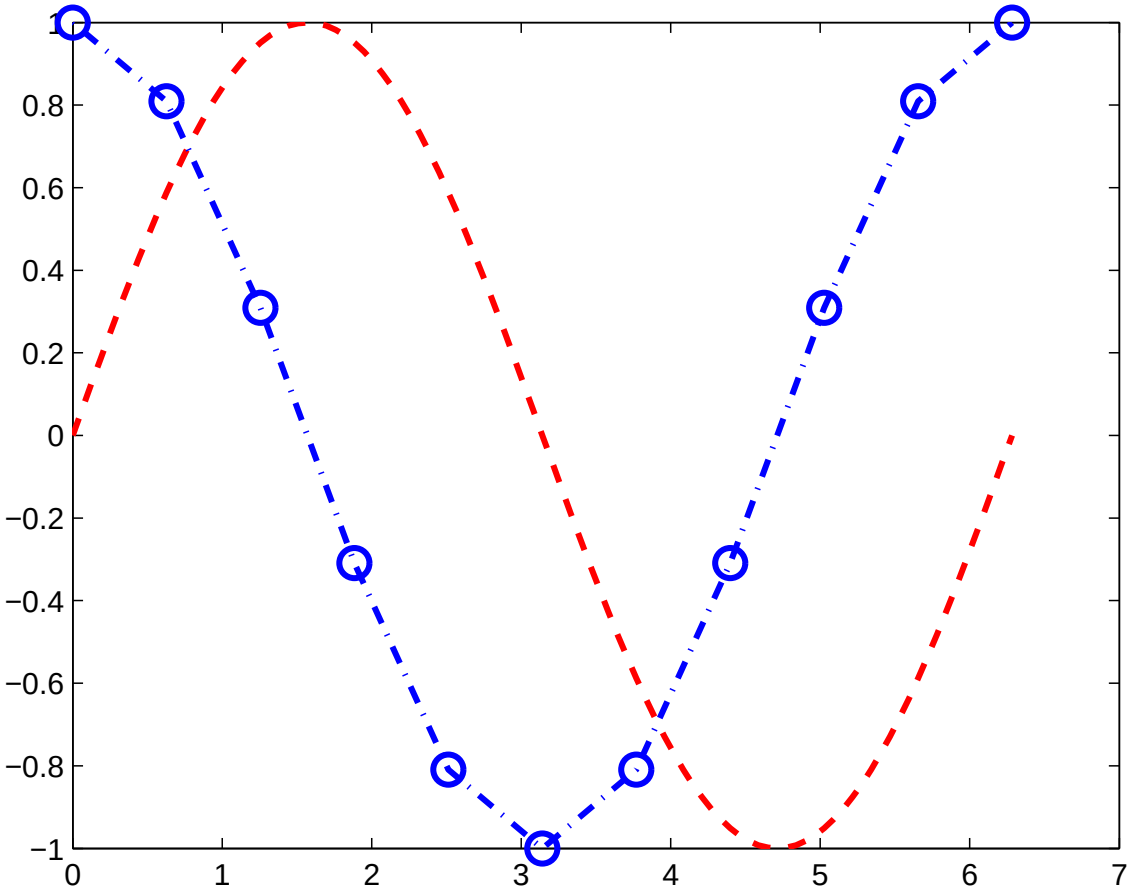
Beispiel

```
1 plot(x1, y1, 'r--', x2, y2, 'b-.o', ...
2      'Linewidth',2)
```



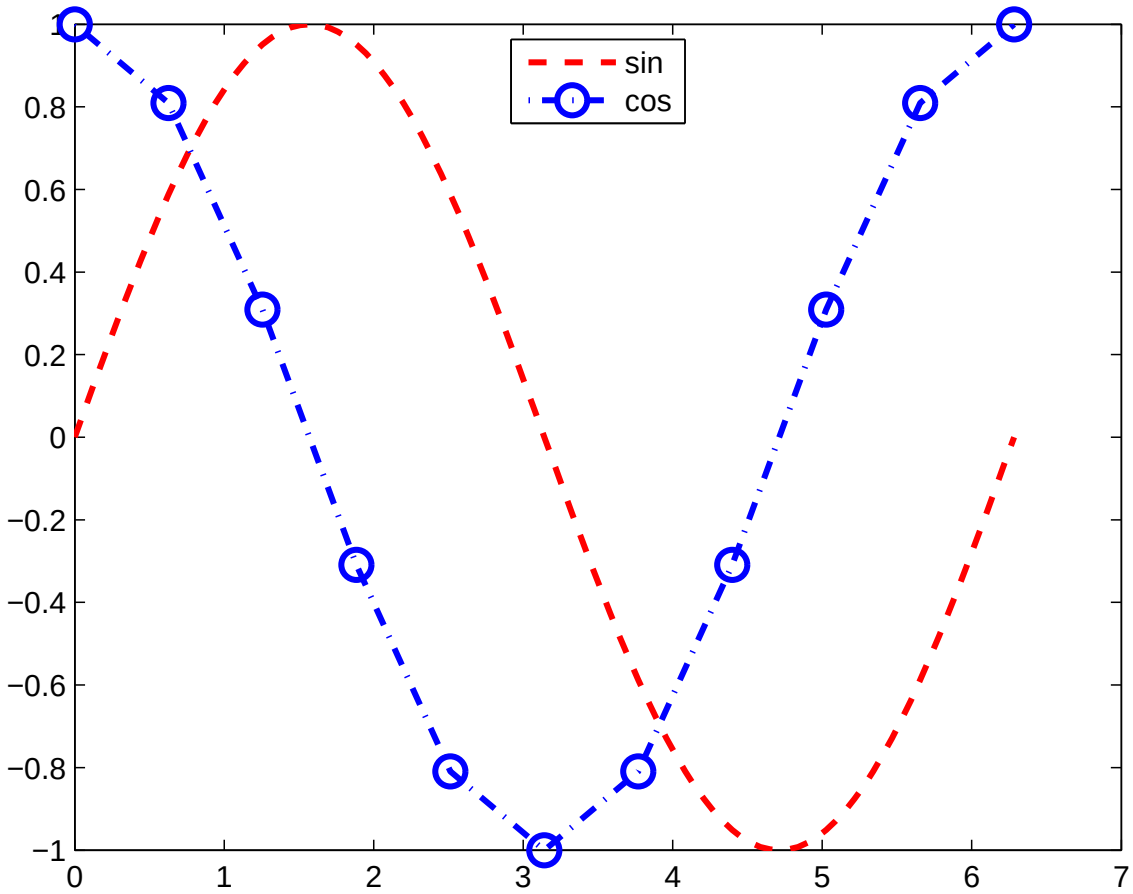
Beispiel

```
1 plot(x1, y1, 'r--', x2, y2, 'b-.o', ...
2      'Linewidth',2, 'MarkerSize',10)
```



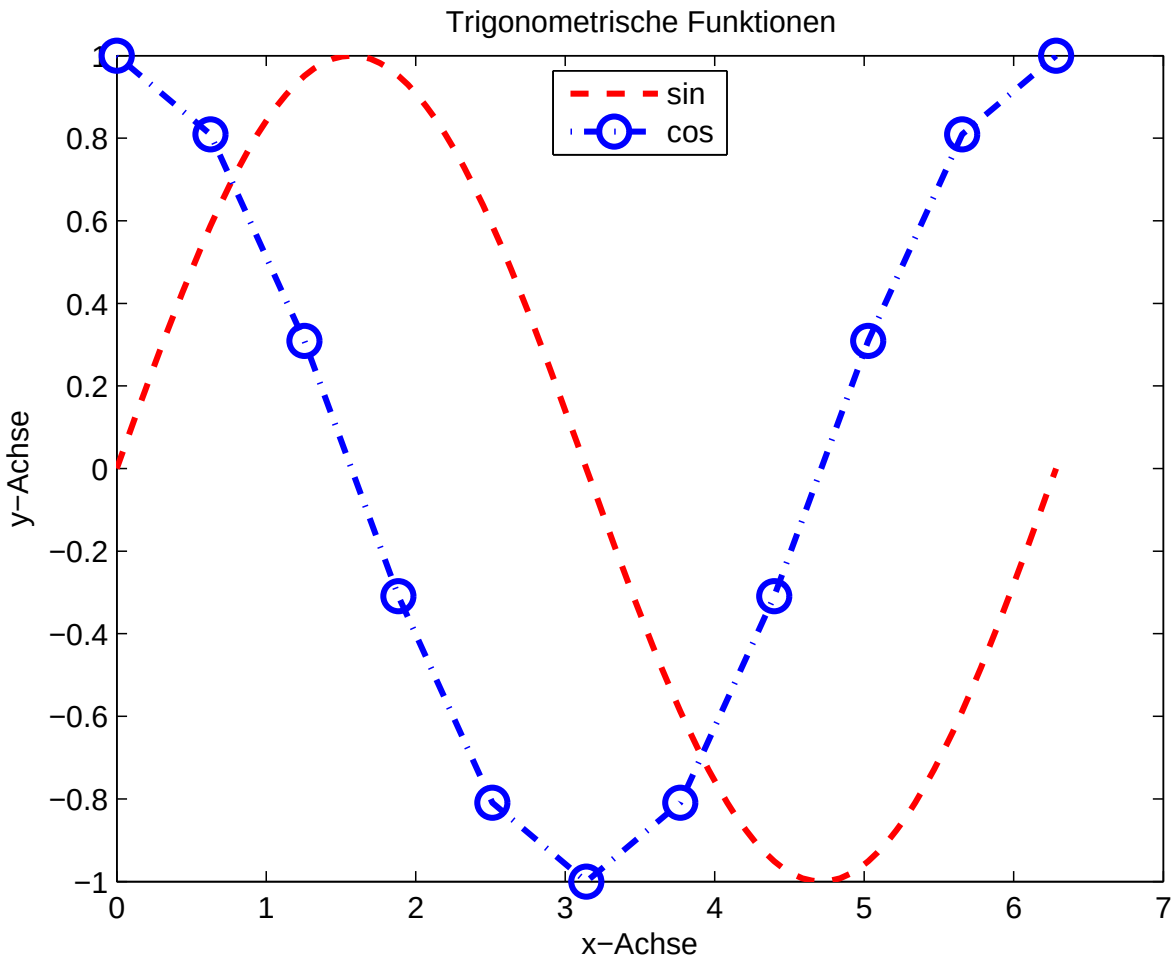
Beispiel

```
1 plot(x1, y1, 'r--', x2, y2, 'b-.o', ...
2      'Linewidth',2, 'MarkerSize',10)
3 legend('sin','cos','Location','North')
```



Beispiel

```
1  plot(x1, y1, 'r--', x2, y2, 'b-.o', ...
2       'Linewidth',2, 'MarkerSize',10)
3  legend('sin','cos','Location','North')
4  title('Trigonometrische Funktionen')
5  xlabel('x-Achse')
6  ylabel('y-Achse')
```



Logarithmische Plots

Listing 9: Beispiel für logarithmische Plots.

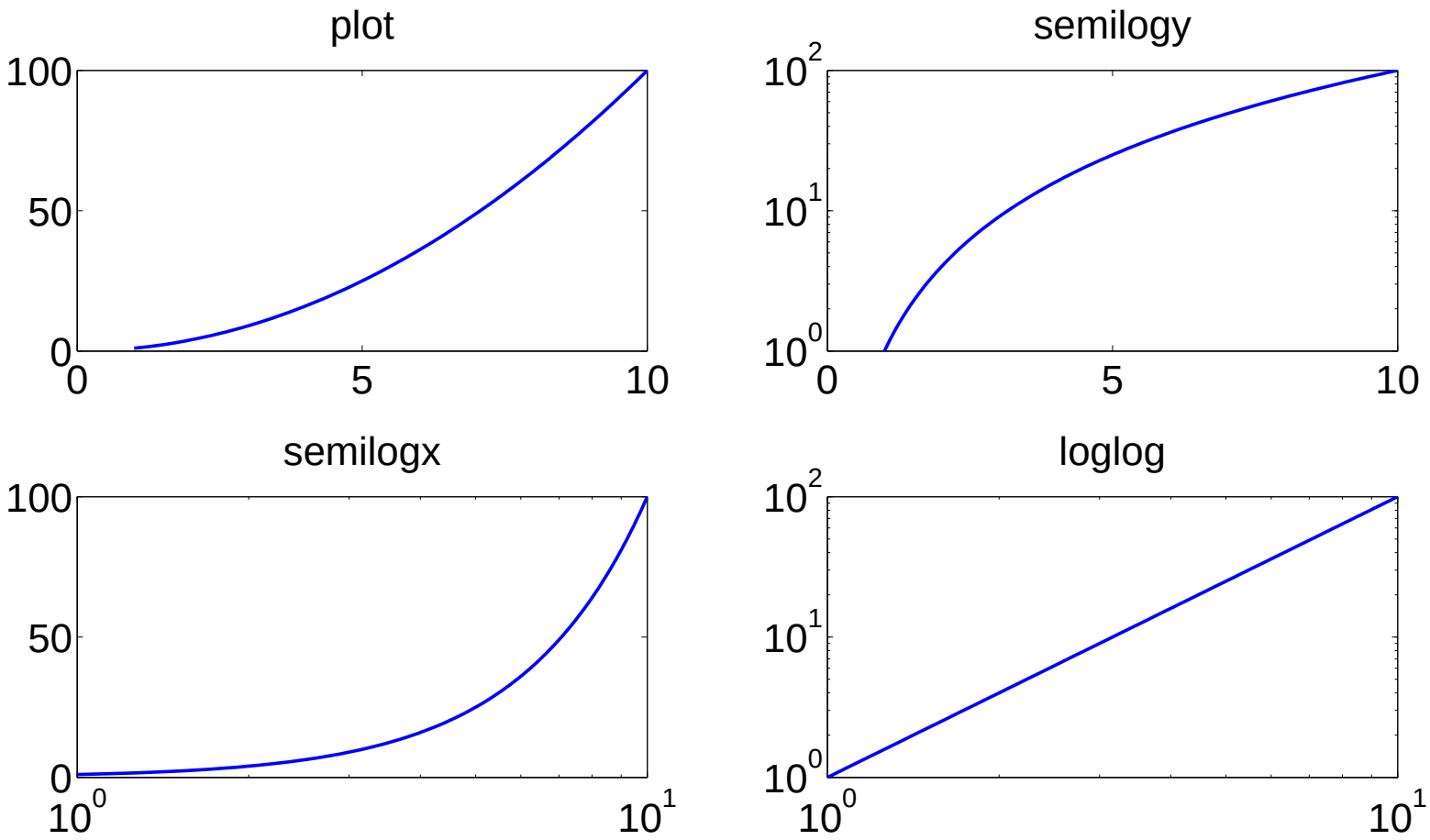
```
clear all
close all

x = linspace(1,10,101);
y = x.^2;

figure(1)
subplot(2,2,1), plot(x,y),     title('plot')
subplot(2,2,2), semilogy(x,y), title('semilogy')
subplot(2,2,3), semilogx(x,y), title('semilogx')
subplot(2,2,4), loglog(x,y),   title('loglog')
```

- Syntax analog zu `plot`.
- Achsen nicht mit äquidistanten Skalen sondern mit logarithmischen Skalen.

Logarithmische Plots



Ein paar Befehle

- `hold on` und `hold off`: Bestimmt, ob der nächste Plotbefehl den letzten (in dem aktuellen Frame) überschreibt oder sich dazuschreibt.
- `axis`: Bestimmt den Achsenausschnitt.
- `figure`: Öffnet neues Fenster, dieses ist auch aktiv. `figure()` mit einer Zahl als Argument aktiviert oder falls es noch nicht existiert öffnet das entsprechende Fenster.
- `subplot` erzeugt in einem Fenster mehrere Untergraphiken.

Aufgaben

- Finden Sie Beispiele für Anwendungen der unterschiedlichen logarithmischen Plotbefehle.
- Informieren Sie sich über die Funktionsweise der in diesem Abschnitt genannten Funktionen.
- Erzeugen Sie einen schönen Plot Ihrer Lieblingsfunktion.