GNU Octave / Matlab — Eine Einführung

Prof. Dr. Thorsten Raasch
Dr. Floriane Mefo Kue
Dr. Adam Czapliński

Wintersemester 2018/19

Allgemeines zu Octave und Matlab

Was ist Octave?

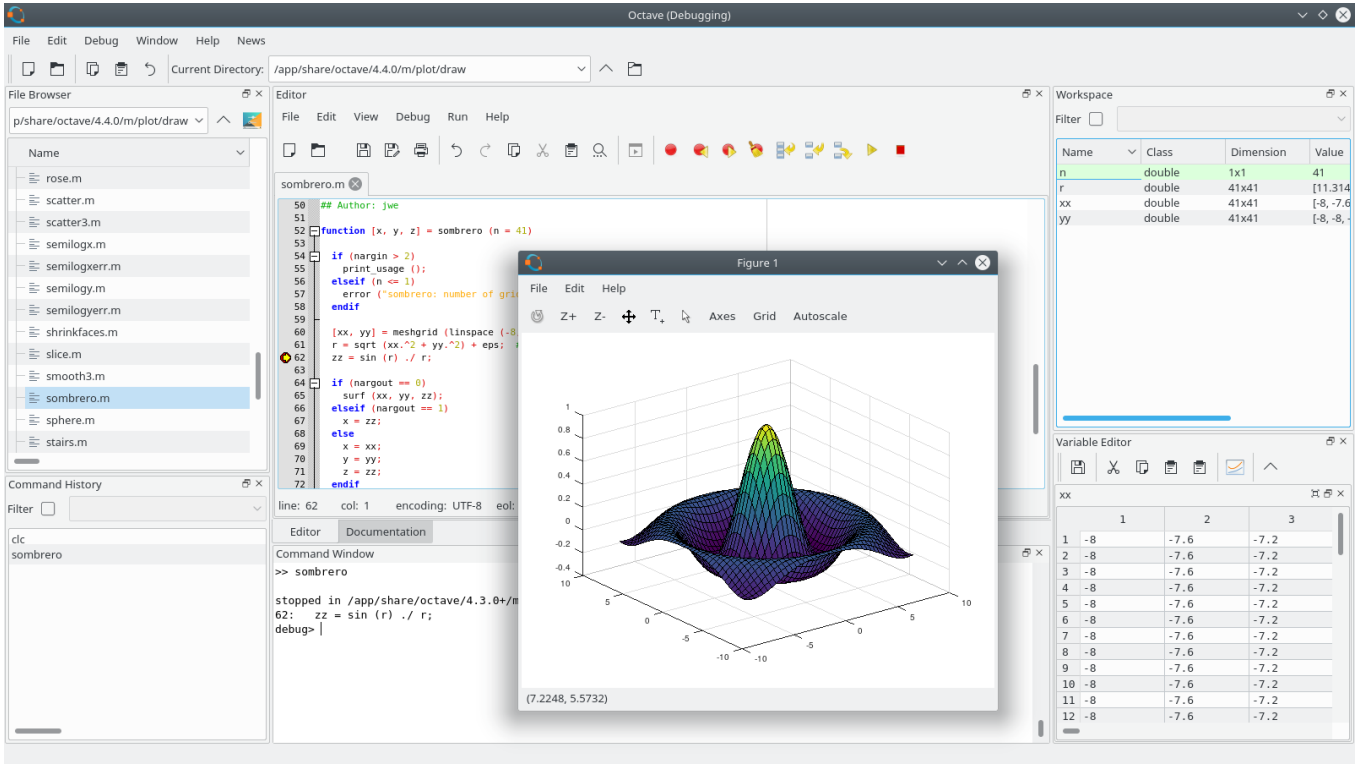
- Programm zur numerischen Lösung mathematischer Probleme
- interaktiv oder skriptgesteuert
- grafische Oberfläche (seit Version 4)
- freie Software (GPL)
- Plattformen: Windows, OS X, Linux, Unix, Solaris, OS/2
- `http://www.gnu.org/software/octave`

Was ist Matlab?

- kommerzielles Gegenstück zu Octave
- Grundfunktionen syntaktisch äquivalent zu Octave
- grafische Oberfläche
- Plattformen: Windows, OS X, Linux, Unix, Solaris
- `http://www.mathworks.de/products/matlab`

Wie arbeitet man mit Octave/Matlab?

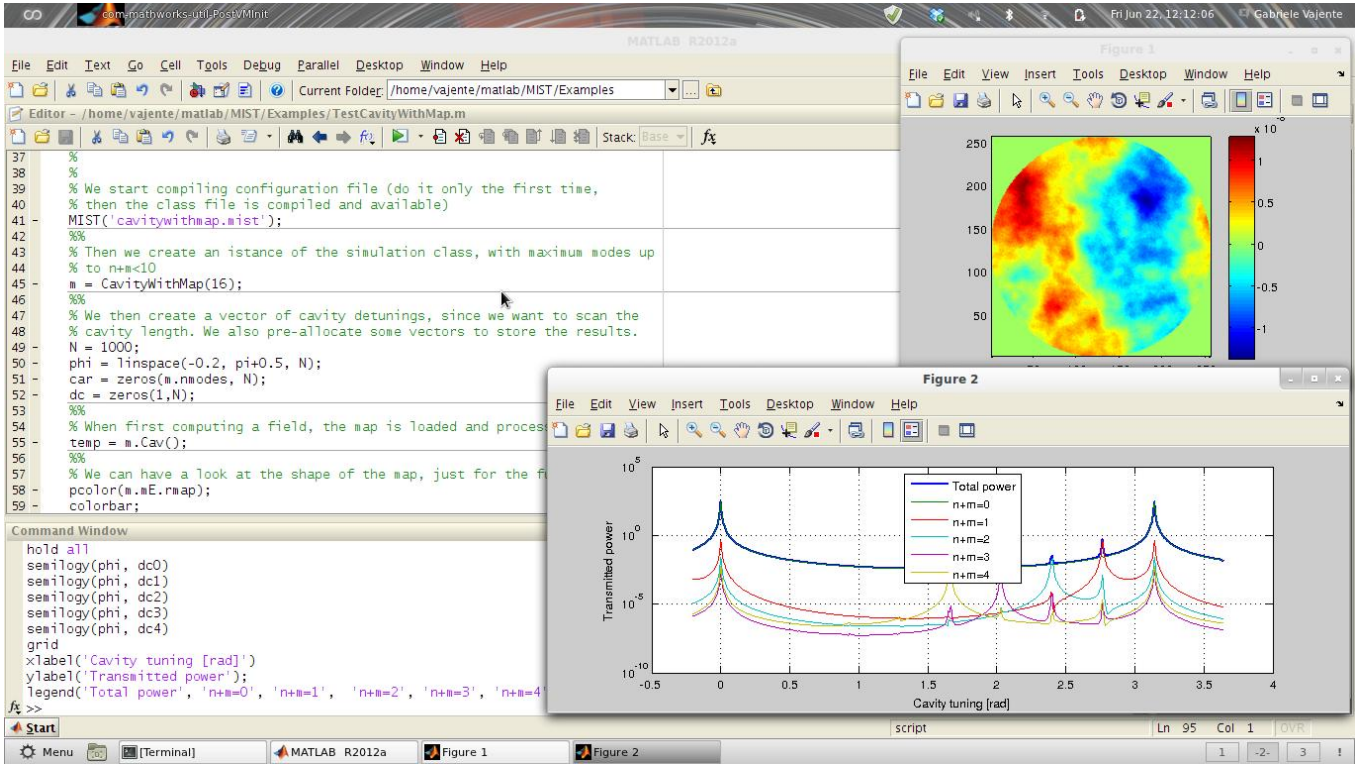
Typische Arbeitsumgebung mit Octave:



- im Hintergrund: integrierte Octave-Umgebung mit Skript-Editor (Mitte oben), Kommandozeile (Mitte unten) und Variablen-Editor (rechts)
- im Vordergrund: grafische Ausgabe

Wie arbeitet man mit Octave/Matlab?

Typische Arbeitsumgebung mit Matlab:



- links im Hintergrund: integrierte Matlab-Umgebung mit Skript-Editor (oben) und Kommandozeile (unten)
- rechts im Vordergrund: Grafikausgabe

Woher bekommt man Octave/Matlab?

Octave:

- Open Source
- Download unter `https://www.gnu.org/software/octave`
- im CIP-Pool schon installiert

Matlab:

- kommerzielles Produkt von Mathworks
- `https://de.mathworks.com/products/matlab.html`
- ZIMT bietet Mathworks-Campuslizenz
  `https://www.zimt.uni-siegen.de/beratung_und_lehre/software/matlab_student/`
  ...für Studierende der Uni Siegen kostenlos!

Starten und Beenden von Octave/Matlab

Unter Linux startet man Octave bzw. Matlab entweder

- mit der Maus über das Startmenü oder
- mit der Tastatur über das Terminal und dem Befehl

  ```
  octave
  ```

  bzw.

  ```
  matlab
  ```

Beenden:

- aus der Kommandozeile von Octave/Matlab mit dem Befehl

  ```
  exit
  ```

- mit der Maus (via Menü oder Fenster schließen)

Erste Schritte in der Octave/Matlab-Kommandozeile

Einige einfache Befehle für die Kommandozeile, danach immer Return drücken (Taschenrechnerfunktion):

- Grundrechenarten `+`, `-`, `*`, `/` (Punkt vor Strich!)
- Kommazahlen werden mit Punkt geschrieben, z.B. `0.5`
- Potenzieren mit `^`, z.B. `2^3` (2×Dach-Taste, oder Dach-&Leertaste)
- Quadratwurzel $\sqrt{\ }$ mit `sqrt`, z.B. `sqrt(2)`
- trigonometrische Funktionen `sin`, `cos`, `tan`, `asin`, `acos`, `atan`,...
- Absolutbetrag: `abs`
- Exponentialfunktion: `exp`
- natürlicher Logarithmus: `log`
- Logarithmus zur Basis 2 bzw. 10: `log2`, `log10`
- Konstante π: `pi`
- Konstante e: `e` (nur in Octave)
- Anzeige aller Nachkommastellen: `format long`
  Anzeige weniger Nachkommastellen: `format short`

→ Aufgabe 1

Verhalten von Octave/Matlab bei Syntaxfehlern

Bei Syntaxfehlern im eingegebenen Befehl (z.B. schließende Klammer fehlt) reagieren Octave und Matlab leicht anders:

- Octave wertet bei fehlenden schließenden Klammern den Befehl noch nicht aus und bietet eine neue Eingabezeile an, um die fehlenden Klammern nachzuliefern. Alternativ kann man Control-C drücken für eine „frische“ Kommandozeile. Im Allgemeinen gibt Octave detaillierte Fehlermeldungen.
- Matlab gibt sofort detaillierte Fehlermeldungen und eine „frische“ Kommandozeile.

Komplexe Zahlen

Definition (Komplexe Zahlen)

Komplexe Zahlen $z \in \mathbb{C}$ haben die Form $z = a + ib$, mit reellen Zahlen $a, b \in \mathbb{R}$ und der imaginären Einheit i. Dabei heißt $a = \mathrm{Re}(z)$ der *Realteil* von z, $b = \mathrm{Im}(z)$ heißt der *Imaginärteil* von z.

Eine komplexe Zahl $z = a + ib$ kann interpretiert werden als Punkt $\binom{a}{b}$ in der Ebene $\mathbb{R}^2$. Die reelle Zahlengerade ist die x-Achse dieser Ebene.

Definition (Rechenoperationen in $\mathbb{C}$)

Komplexe Addition: $(a + ib) + (c + id) := a + c + i(b + d)$
Komplexe Multiplikation: $(a + ib)(c + id) := ac - bd + i(ad + bc)$
Konjugation: $\overline{a + ib} := a - ib$

Beachte: $i^2 = -1$

Definition (Polarform komplexer Zahlen)

Betrag/Länge: $|a + ib| := \sqrt{a^2 + b^2}$
Euler: $re^{i\varphi} := r(\cos\varphi + i\sin\varphi)$, $\varphi \in \mathbb{R}$
Argument/Phase: $\varphi := \arg(z)$ erfüllt $\varphi \in (-\pi, \pi]$ und $z = |z|e^{i\varphi}$

Behandlung komplexer Zahlen in Octave/Matlab

- imaginäre Einheit: `i`
- Eingabe einer komplexen Zahl als `a+i*b`
- Rechengesetze (Addition,...) funktionieren wie erwartet
- ...auch viele Funktionen (trigonometrische Funktionen, Exponentialfunktion, Logarithmus,...)
- Konjugation: `conj`
- Absolutbetrag: `abs`
- Argument: `arg` (in Octave), `angle` (in Matlab)
- Real-/Imaginärteil: `real`, `imag`

→ Aufgabe 2

Hilfe zur Selbsthilfe

Matlab und Octave besitzen eine eingebaute ausführliche *Hilfefunktion*.

- Aufruf der Produktdokumentation in Matlab: F1 drücken oder Hilfe mit der Maus auswählen
- Matlab/Octave: Aufruf einer Kurzhilfe zum Befehl `befehl` aus der Kommandozeile: `help befehl`
- Ausführliche Hilfe in Matlab/Octave: `doc befehl`
- Weitere Hilfsmittel:
  - Die Syntax einer Funktion wird beim Eintippen einer öffnenden Klammer `(` eingeblendet (Matlab).
  - Benutzte Variablen erscheinen im „Workspace“ und können dort mit der Maus inspiziert werden.
  - Funktionsbrowser links neben der Kommandozeile

Variablen

Eine *Variable* ist ein Behälter/Speicherbereich für Rechnungsgrößen, auf den man mit dem *Variablennamen* zugreifen kann.

- Zuweisung/Erzeugung einer Variablen mit `=`, z.B. `x=3`
- Erlaubte Variablennamen: beginnen mit einem Buchstaben, dürfen Ziffern und den Unterstrich `_` enthalten. Groß-/Kleinschreibung wird beachtet und Matlab-Schlüsselworte sind verboten (vgl. `iskeyword`).
- Ausgabe des aktuellen Werts einer Variablen durch Eingabe des Variablennamens, z.B. `x`, oder mit `disp`, z.B. `disp(x)`
- Mit Variablen kann auch gerechnet werden, z.B. `x+2*y`
- Ein Semikolon `;` am Ende eines Befehls unterdrückt die Ausgabe.
- Anzeige der Namen aller benutzten Variablen: `who`
- Mehr Details zu allen benutzten Variablen, z.B. Speicherbedarf: `whos`
- Löschen der Variablen `x` (Speicher freigeben): `clear x`
- Löschen aller Variablen: `clear`

Vektoren

Ein *Vektor* ist für Matlab/Octave ein eindimensionales Zahlenfeld.

- Erzeugen von Zeilenvektoren: mit eckigen Klammern, die Elemente sind durch Leerzeichen oder Kommata abgetrennt, z.B. `[1 2 3]` oder `[1,2,3]`
- Erzeugen von Spaltenvektoren: mit eckigen Klammern, die Elemente sind durch Semikola abgetrennt, z.B. `[1;2;3]`, oder durch *Transponieren* eines Zeilenvektors mit Hochkomma, z.B. `[1 2 3]'`
- Spezielle Vektoren:
  - Nullvektor, z.B. `zeros(1,4)`
  - Vektor aus lauter Einsen, z.B. `ones(3,1)`
  - Laufbereich ganzer Zahlen, z.B. `2:7`
  - Laufbereich mit beliebiger Schrittweite, z.B. `1:0.4:3` oder `10:-1:4`
- Zuweisung zu Variablen geht natürlich, z.B. `x=[1 2 3]`
- Zugriff auf einzelne Einträge mit runder Klammer, z.B. `x(1)` liefert den ersten Eintrag von `x`, `x(end)` liefert den letzten Eintrag von `x`, Schreibzugriff z.B. mit `x(2)=5` (Achtung: verlängert ggf. `x`!)
- Die Länge eines Vektors ermittelt die Funktion `length`.

→ Aufgabe 3

Rechnen mit Vektoren

- Addition/Subtraktion mit `+` bzw. `-`
  (Das Format der beteiligten Vektoren sollte stimmen!)
- skalares Vielfaches mit `*`, z.B. `2*x`
- Elementweise Operationen mit vorangestelltem Punkt, z.B. `x.^2`
  (sehr mächtiges Hilfsmittel!)
- Viele Funktionen erlauben ihre komponentenweise Anwendung auf einen Vektor, z.B. `abs([3 -1.5])` oder `sin([0 2 pi])`
- Aufsummieren aller Einträge eines Vektors: `sum`, z.B. `sum(x)`
- Euklidische Länge $\|x\|_2 = \sqrt{|x_1|^2 + \cdots + |x_n|^2}$ eines Vektors: `norm`, z.B. `norm(x)`
- Innen-/Skalarprodukt zweier Vektoren: `dot`, z.B. `dot(x,y)`
- Kreuzprodukt zweier Vektoren im $\mathbb{R}^3$: `cross`, z.B. `cross(x,y)`

→ Aufgabe 4

Einfache 2D-Plots

Mit einem *Plot* ist die grafische Ausgabe von Matlab/Octave gemeint.

- Plots sind durch eine Menge von Punkten festgelegt.
- 2D-Plots werden meistens mit `plot` erstellt. Die eingegebenen Punkte werden zu einem stetigen Streckenzug verbunden.
- Werte eines Vektors `x` der Länge n gegen $1, 2, \ldots, n$ plotten:

  ```
  plot(x)
  ```

- Werte eines Vektors `y` gegen Werte eines Vektors `x` plotten:

  ```
  plot(x,y)
  ```

- Standardfarbe für den Streckenzug ist Blau. Andere Farben: z.B.

  ```
  plot(x,y,'r')
  ```

  für einen roten Streckenzug. Weitere Farben: Gelb (`y`), Magenta (`m`), Cyan (`c`), Grün (`g`), Blau (`b`), Weiß (`w`), Schwarz (`k`).
- `plot` überschreibt den letzten Plot. Um das zu verhindern, kann man mit `figure` ein neues, leeres Plotfenster erzeugen und danach dorthin plotten.
- Den zuletzt erzeugten Plot hervor holen: `shg` (*show graphics*).

→ Aufgabe 5

Plot-Optionen

Einige Eigenschaften des Graphen lassen sich über Plot-Optionen ändern:

- Einstellen des Linienstils (Matlab): z.B. gepunktete Linien mit `:`

  ```
  plot(x,y,':')
  ```

  Andere Stile: lange Striche (`-`), Strichpunkt (`-.`), durchgezogen (`-`)
- Einstellen des Linienstils in Octave: ähnlich, ggf. aber unmöglich
- Einstellen des Markertyps (Matlab), z.B. Kreise an jedem Datenpunkt mit `o` (nicht miteinander verbunden)

  ```
  plot(x,y,'o')
  ```

  Weitere Marker (Matlab): Pluszeichen (`+`), Stern (`*`), Punkt (`.`), Kreuz (`x`), Quadrat (`s` oder `square`), Raute (`d` oder `diamond`), Dreieck mit Spitze nach oben/unten/links/rechts (`^`/`v`/`<`/`>`), 5-Punkt-Stern (`p` oder `pentagram`), 6-Punkt-Stern (`h` oder `hexagram`), kein Marker (`none`)
- Einstellen des Markertyps in Octave: ähnliche Optionen
- Optionen können überlagert werden, z.B. roter gepunkteter Graph mit Rauten bei jedem Datenpunkt (Reihenfolge der Optionen egal):

  ```
  plot(x,y,'r-d')
  ```

2D-Plots mehrerer Funktionen

Man kann mehrere Funktionen in einem 2D-Plot darstellen:

- Beim Befehl `plot` kann man mehrere Funktionen/Punktmengen übergeben, z.B.

  ```
  plot(x1,y1,x2,y2)
  ```

  Dabei müssen die Knotenvektoren `x1`, `x2` nicht übereinstimmen, die Graphen werden automatisch überlagert.

- Mehrere Plot-Optionen sind möglich, z.B.

  ```
  plot(x1,y1,'r:x',x2,y2,'-p')
  ```

- Alternativ können mehrere Plots mit `hold on` / `hold off` überlagert werden:

  ```
  plot(x1,y1)
  hold on
  plot(x2,y2)
  hold off
  ```

  Dabei verhindert `hold on` das Löschen alter Plotinhalte im aktuellen Plotfenster, `hold off` aktiviert es wieder.

2D-Plots nachträglich beschriften

Plots lassen sich nachträglich beschriften:

- Hinzufügen eines Titeltextes (zentriert) mit `title`, z.B. via

```
title('Dies ist ein Titel')
```

- Hinzufügen von Achsenbeschriftungen mit `xlabel`, `ylabel`:

```
xlabel('x-Achse')
ylabel('y-Achse')
```

- Hinzufügen einer Legende mit `legend`:

```
legend('erster Graph','zweiter Graph')
```

  Dabei kann man die Position mit einer Option steuern, z.B.

```
legend('eins','zwei','Location','NorthWest')
```

- Im Text dürfen Buchstaben hoch-/tiefgestellt werden mit `^` bzw. `_`
- Griechische Buchstaben können z.B. mit `\alpha` erzeugt werden. (funktioniert momentan im CIP-Pool nicht...)

→ Aufgabe 6

Spezielle Plots in 2D

Spezielle Plots in 2D sind:

- 2D-Punktwolke mit `scatter`, z.B.

  ```
  scatter([0 2],[-1 3],'rs')
  ```

  oder

  ```
  scatter([0 2],[-1 3],'r','s')
  ```

  erzeugt zwei rote Quadrate bei $(0, -1)$ und $(2, 3)$.
  In Octave funktioniert nur die zweite Syntax!

- Treppenfunktion mit `stairs`, z.B.

  ```
  stairs(1:4,[2 -1 3 1])
  ```

  Achtung: Die letzte Stufe wird dabei mit „Breite Null“ geplottet, d.h. nur ein vertikaler Strich. Will man diesen vermeiden, muss man den vorletzten y-Wert verdoppeln, z.B.

  ```
  x=0:0.1:pi;
  stairs(x,sin([x(1:end-1) x(end-1)]))
  ```

→ Aufgabe 7

Polynome

Polynome sind Funktionen der Form $p(x) = \sum_{k=0}^{n} a_k x^k$, wobei $a_k \in \mathbb{C}$.

- Polynome sind festgelegt durch die Koeffizienten a_k, $0 \leq k \leq n$
- Eingabe von Polynomen in Octave/Matlab durch den Vektor

  $$[a_n \ a_{n-1} \ \cdots \ a_0]$$

- Grad eines Polynoms mit Koeffizientenvektor `a`:

  ```
  length(a)-1
  ```

- Auswertung von Polynomen an einem (oder mehreren) Punkt(en) mit `polyval`, z.B. für $p(x) = 2x^2 + 3x - 1$

  ```
  polyval([2 3 -1],x)
  ```

  Plotten des Polynoms p über einem Gitter `x` ist also möglich mit

  ```
  plot(x,polyval([2 3 -1],x))
  ```

- Differenzieren eines Polynoms p: mit `polyder`, liefert Koeffn. von p'
- Polynom $(x - x_0)(x - x_1)\cdots(x - x_n)$ zu gegebenen Nullstellen x_k:

  ```
  poly([x_0 x_1 ... x_n])
  ```

Rechnen mit Polynomen

Rechnen mit Polynomen kann auf Koeffizientenebene geschehen:

- Multiplikation eines Polynoms p mit einem Skalar t:
  multipliziere Koeffizientenvektor von p mit t, z.B. `t*a`
- Addition zweier Polynome p, q mit *gleichem* Grad:
  addiere ihre Koeffizientenvektoren, z.B. `a+b`
- Addition zweier Polynome p, q mit *verschiedenen* Graden:
  bestimme den größten Grad (+1), fülle den kürzeren der beiden Koeffizientenvektoren von vorn mit Nullen auf und addiere dann:

```
N=max([length(a) length(b)]);
[zeros(1,N-length(a)) a]+[zeros(1,N-length(b)) b]
```

- Multiplikation zweier Polynome p, q:

$$\Big(\sum_{j=0}^{m} a_j x^j\Big)\Big(\sum_{k=0}^{n} b_k x^k\Big) = \sum_{\ell=0}^{m+n}\Big(\sum_{j=\max\{0,\ell-n\}}^{\min\{m,\ell\}} a_j b_{\ell-j}\Big)x^\ell$$

. . . entspricht einer *Faltung* der Koeffizientenvektoren:

```
conv(a,b)
```

→ Aufgabe 8

Matrizen

Eine *Matrix* ist für Matlab/Octave ein zweidimensionales Zahlenfeld.

- Erzeugen von Matrizen: mit eckigen Klammern, Elemente zeilenweise und durch Leerzeichen oder Kommata abgetrennt, Zeilenende mit Semikolon; z.B.

  `[1 2 3; 4 5 6]` $\quad\leftrightarrow\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

- Matrixeinträge dürfen beliebige Matlab/Octave-Ausdrücke sein, und auch ganze Blöcke sind erlaubt, z.B.

  `[4 -pi/2 5+3*i; zeros(1,3)]` $\quad\leftrightarrow\quad \begin{pmatrix} 4 & -\frac{\pi}{2} & 5+3i \\ 0 & 0 & 0 \end{pmatrix}$

- Spezielle Matrizen:
  - Nullmatrix, z.B. `zeros(2,5)`
  - Matrix aus lauter Einsen, z.B. `ones(3,2)`
  - Einheitsmatrix, z.B. `eye(3)`
- Zugriff auf einzelne Matrixeinträge mit runder Klammer, z.B. `A(2,1)` liefert Eintrag der Matrix `A` aus Zeile 2 und Spalte 1
- Zugriff auf ganze Matrixblöcke durch Laufbereiche von Indizes, z.B. `A(1:2,3:end)` oder `A(3,:)`, funktioniert analog auch für Vektoren

→ Aufgabe 9

Rechnen mit Matrizen

- Addition/Subtraktion mit `+` bzw. `-`
  (Format der beteiligten Matrizen beachten!)
- skalares Vielfaches mit `*`, z.B. `3*A`
- elementweise Operationen mit vorangestelltem Punkt, z.B. `A.^2`
- Matrix-Vektor- bzw. Matrix-Matrix-Multiplikation mit `*`, z.B. `A*v` oder `A*B` (Format beachten!)
- Transponieren von Matrizen mit `'`
- `size(A)` liefert Zeilen-/Spaltenzahl einer Matrix `A` als Zeilenvektor
- `det(A)` liefert Determinante einer quadratischen Matrix `A`
- `inv(A)` liefert Inverse einer invertierbaren quadratischen Matrix `A`
- Lösung des linearen Gleichungssystems $Ax = b$ mit `x=A\b`
  ...sofern `A` invertierbar und quadratisch, ansonsten Lösung des Kleinste-Quadrate-Problems

$$\min_x \sum_{k=1}^{m} (Ax - b)_k^2,$$

vgl. `doc mldivide`

→ Aufgaben 10, 11

Nützliche Matrix-Manipulationen

Beim Erzeugen und Verändern von Matrizen `A` mit Vektoren `v` sind folgende Befehle nützlich:

- `fliplr(A)` bzw. `flipud(A)` spiegeln `A` horizontal bzw. vertikal
- `diag(A)` extrahiert Hauptdiagonale von `A` als Spaltenvektor
- `diag(A,k)` extrahiert die `k`-te Diagonale (0: Hauptdiagonale, 1: erste obere Nebendiagonale, -1: erste untere Nebendiagonale etc.)
- `diag(v)` erzeugt Diagonalmatrix mit `v` auf der Hauptdiagonalen
- `diag(v,k)` erzeugt quadratische Matrix mit `v` auf Diagonale `k`
- `toeplitz(c,r)` mit $r_1 = c_1$ erzeugt die *Toeplitz-Matrix*

$$\begin{pmatrix} c_1 & r_2 & \cdots & \cdots & r_n \\ c_2 & c_1 & r_2 & \cdots & r_{n-1} \\ \vdots & & \ddots & \vdots & \\ \vdots & & & \ddots & r_2 \\ c_n & c_{n-1} & \cdots & \cdots & c_1 \end{pmatrix}$$

und `toeplitz(c)` die entsprechende symmetrische Toeplitz-Matrix

→ Aufgabe 12

Blockmatrizen

Nützliche Befehle für Blockmatrizen `A`:

- `tril(A)` bzw. `triu(A)` extrahieren linke untere bzw. rechte obere Dreiecksmatrix, d.h. `A` ist gleich `tril(A)+triu(A)-diag(diag(A))`
- `tril(A,k)` bzw `triu(A,k)` extrahieren alle Diagonalen auf und unter- bzw. oberhalb der Diagonale `k`
- `blkdiag(A,B,C,...,Z)` erzeugt eine Block-Diagonalmatrix mit `A`,...,`Z` auf der Hauptdiagonalen
- `kron(A,B)` erzeugt das *Kronecker-Produkt*

$$A \otimes B := \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{pmatrix}$$

→ Aufgabe 13

Plotten von Kurven im $\mathbb{R}^2$ und $\mathbb{R}^3$

Wir wollen jetzt 2-/3-dimensionale Kurven der Form

$$t \mapsto \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \quad \text{bzw.} \quad t \mapsto \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$$

plotten, wobei der Parameter t einen gewissen Bereich durchläuft.

- zunächst Laufbereich von t definieren, z.B. `t=0:0.01:2*pi`
- Plotten 2-dimensionaler Kurven mit `plot`, z.B.

  ```
  plot(cos(t),sin(t),'r')
  ```

  für eine rote Kreislinie $(\cos t, \sin t)^\top$
- Plotten 3-dimensionaler Kurven mit `plot3`, z.B.

  ```
  plot3(cos(10*t),sin(10*t),t,'g')
  ```

  für eine grüne Schraubenlinie $(\cos 10t, \sin 10t, t)^\top$
- mehrere Kurven gleichzeitig plotten durch mehrere Argumente, z.B.

  ```
  plot3(t.^2,0*t,sin(t),'b',t,-t,sin(t.^2),'r')
  ```
- Achsenbeschriftungen mit `xlabel`, `ylabel` und ggf. `zlabel`
- Titel mit `title`

→ Aufgaben 14, 15

Einfache 3D-Plots

Plotten von Funktionen in zwei Variablen:

- `surf(x,y,Z)` zeichnet Oberfläche eines Funktionsgraphen auf einem durch *Vektoren* `x`$\in \mathbb{R}^n$ und `y`$\in \mathbb{R}^m$ gegebenen kartesischen Gitter; dabei ist `Z` eine $m \times n$-Matrix aus Funktionswerten an Punkten $(\texttt{x(k)}, \texttt{y(j)})^\top$, $1 \le j \le m$, $1 \le k \le n$, z.B. für $f(x,y) = x^2 \cos(2y)$

```
x=0:0.02:2;
y=0:0.04:4;
Z=cos(2*y)'*x.^2;
surf(x,y,Z)
```

- `mesh(x,y,Z)` zeichnet Drahtgitter (ohne farbige Flächenstücke)
- Alternativ kann Octave/Matlab auch mit *matrixwertigen* x-/y-Koordinaten umgehen, die man mit `meshgrid` erzeugen kann. Dabei sind `X`, `Y` $m \times n$-Matrizen, genau wie `Z`:

```
[X,Y]=meshgrid(0:0.02:2,0:0.04:4);
Z=X.^2.*cos(2*Y);
surf(X,Y,Z) bzw. mesh(X,Y,Z)
```

- Achsenbeschriftung/Titel: `xlabel`, `ylabel` und `zlabel` bzw. `title`

→ Aufgabe 16

Octave/Matlab-Skripte

Octave/Matlab-Skripte sind *Textdateien* mit Dateiendung `.m`, die zeilenweise Octave/Matlab-Befehle enthalten.

- Ein Skript mit Dateinamen `meinskript.m` kann mit dem Befehl

  `meinskript`

  im Kommandofenster ausgeführt werden
- Zur Bearbeitung von Skripten bietet Matlab einen komfortablen Editor, den man über das Menü oder mit `edit` aufrufen kann
- In Octave benutzt man einen externen (Text-)Editor für Skripte
- Skripte können (sollten!) Kommentare enthalten, eingeleitet mit `%`
- Ein typisches Skript hat also die Form

meinskript.m

```
% Dies ist mein erstes Skript
x=0:0.01:2*pi;
plot(x,sin(4*x),'r')
```

- Skripte können alle beim Aufruf existierenden Variablen benutzen, deren Werte verändern und neue Variablen erzeugen
- Tipp: zu Beginn eines Skripts ist es meist ratsam, alle Variablen zu löschen (`clear`) und alle Plotfenster zu schließen (`close all`).

Eigene Octave/Matlab-Funktionen

Eigene Funktionen in einer oder mehreren Variablen kann man auf verschiedene Weisen erzeugen:

- Mit der @-Notation lassen sich *anonyme Funktionen* definieren:

```
f=@(x) sin(3*x);
```

- Auswertung von `f` bei `x` wie erwartet mit `f(x)`
- Falls `f` für Plots verwendet wird, ist die Punkt-Notation nützlich:

```
f=@(x) x.^2+sin(x).*cos(x)./x;
```

- Mehrere Argumente sind auch möglich
- Ist die Funktion komplizierter, kann man sie mit folgender Syntax in eine `.m`-Datei auslagern:

g.m

```
function z=g(x,y)
z=4*x+2*sqrt(y).*x;
```

- Dateiname (ohne `.m`) und Funktionsname müssen übereinstimmen!
- *Mehrere* Rückgabeargumente möglich, z.B. `function [u,v]=h(x)`
- Tipp: In ausgelagerten Funktionen am besten jede Zeile außer der Titelzeile mit `;` abschließen!

→ Aufgabe 17

Bedingte Anweisungen/Verzweigung

Zur bedingten Ausführung bestimmter Skriptabschnitte bieten Octave/Matlab folgende Techniken an:

- Bei einer `if`-Abfrage wird eine logische Bedingung überprüft und davon abhängig der nächste Codeabschnitt ausgeführt:

```
if x>=3
  z=x+1
else
  z=2
end
```

- ...das schließende `end` nicht vergessen!
- Gleichheitstest mit *doppeltem* Gleichheitszeichen `==`, z.B. `if x==4` ...bei einfachem Gleichheitszeichen würde man ja `x` überschreiben!
- Test auf Ungleichheit mit `~=`, z.B. `if x~=7`
- logisches Und mit `&&` und Klammern, z.B. `if (x>0)&&(x<=10)`
- logisches Oder mit `||` und Klammern, z.B. `if (x<=5)||(x^2-x==3)`

→ Aufgabe 18

Schleifen

Zur mehrfachen Ausführung bestimmter Skriptabschnitte bieten Octave/Matlab folgende *Schleifen*techniken an:

- Bei einer `for`-Schleife durchläuft eine Laufvariable (hier `u`) einen gewissen Laufbereich (Zeilenvektor!):

```
for u=1:4
  disp(u^2)
end
```

  (`disp` gibt das Eingabeargument im Kommandofenster aus)

- beliebige Laufbereiche sind möglich, z.B. `for u=[4 2.5 7:3:16]`
- Achtung: die Laufvariable innerhalb der Schleife *nicht* überschreiben!
- Eine etwas allgemeinere Schleifenstruktur als `for` bietet `while`:

```
while Bedingung
  ...
end
```

- vorzeitiges Verlassen der Schleife mit `break` ist jederzeit möglich

→ Aufgabe 19

Zeichenketten

Zeichenketten (Strings) werden in Octave/Matlab als sogenannte Zeichenfelder (*char arrays*) verwaltet und ähnlich wie Vektoren behandelt:

- `s='Hallo'` liefert ein *char array* `s` der Länge 5, jedes Zeichen `s(k)` ist ein *char* und belegt 2 Bytes Platz, vgl. `whos`
- Umlaute sind möglich, z.B. `t='Äußerst praktisch!'`
- Verkleben von Zeichenketten: `strcat('ver','klebt')` oder mit `['ver' 'klebt']` bzw. `['ver','klebt']`
- Nützlich z.B. zum Plotten parameterabhängiger Funktionen:
  - Umwandlung Ganzzahl → Zeichenkette: `int2str(42)`
  - Umwandlung beliebige Zahl → Zeichenkette: `num2str(pi+3*i)`
  - Umwandlung Zeichenkette → Zahl: `str2num('-12.345')`
- Platzieren einer Zeichenkette in einen (gerade erstellten) Plot:

  `text(4,2.5,'Hallo!')`

  ...dabei sind auch LaTeX-Ausdrücke erlaubt:

  `text(1,pi,'\leftarrow Hier ist der Punkt (1,\pi)')`

→ Aufgabe 20

2D-Höhenlinien

Zur Visualisierung von Funktionen in zwei Variablen sind z.B. folgende Befehle hilfreich:

- Plotten von Höhenlinien mit `contour` auf einem 2D-Gitter:

```
x=-2*pi:0.05:2*pi;
y=0:0.05:4*pi;
[X,Y]=meshgrid(x,y);
Z=sin(X)+cos(Y); contour(X,Y,Z)
```

- Anzahl und Beschriftung der Höhenlinien verändern:

```
contour(X,Y,Z,10,'ShowText','on')
```

- Höhenlinienplots mit gefüllten Flächen:

```
contourf(X,Y,Z)
```

- Kombination eines 3D-Funktionsgraphen mit 2D-Höhenlinien:

```
surfc(X,Y,Z)
```

→ Aufgabe 21

2D-Vektorfelder

Ein zweidimensionales *Vektorfeld* ist eine Abbildung

$$f : \mathbb{R}^2 \to \mathbb{R}^2,$$

d.h. an jedem Punkt der Ebene sitzt ein Vektor („Windrichtung“).

- Zeichnen des 2D-Vektorfelds $f(x, y) = (-y, x)^\top$ mit `quiver`:

```
[X,Y]=meshgrid(-1:0.2:1);
U=-Y; V=X;
quiver(X,Y,U,V)
axis equal
axis([-1 1 -1 1])
```

- Octave/Matlab skaliert die Vektoren so, dass sie nicht überlappen. Das kann man mit einem Skalierungsparameter verhindern:

```
s=0; quiver(X,Y,U,V,s)
```

- Ein 2D-Vektorfeld f ist häufig die Richtung des steilsten Anstiegs einer Funktion $g : \mathbb{R}^2 \to \mathbb{R}$, d.h. $f = \nabla g$ („Gradientenfeld“):

```
g=@(x,y) x.^2-y.^2;
Z=g(X,Y);
[DX,DY]=gradient(Z);
quiver(X,Y,DX,DY)
```

→ Aufgabe 22

Polygone

Ein *Polygon* (Vieleck) im $\mathbb{R}^2$ ist eine durch einen geschlossenen Streckenzug zwischen seinen *Ecken* begrenzte Fläche.

- Zeichnen von Polygonen geht mit `patch`, dabei übergibt man z.B. die x- und y-Koordinaten der Ecken als *Vektoren* (linksherum bzw. mathematisch positiv durchlaufen) sowie eine Füllfarbe:

```
patch([0 1 0],[0 0 1],'r')
title('rotes Dreieck mit schwarzem Rand')
```

- Randfarbe des Polygons setzen:

```
patch([0 1 0],[0 0 1],'r','EdgeColor','r')
title('komplett rotes Dreieck')
```

- Polygone müssen nicht unbedingt konvex sein, d.h. einspringende Ecken sind erlaubt:

```
patch([0 2 2 1 1 0],[0 0 1 1 2 2],'g')
title('L-förmiges grünes Polygon')
```

- Mehrere Polygone auf einmal zeichnen: verwende *Matrizen* aus x- und y-Werten, jede Spalte gehört dabei zu einem Polygon:

```
patch([0 1 0; 2 4 2]',[0 0 1; 1 2 3]','b')
title('zwei blaue Dreiecke')
```

→ Aufgabe 23

Datenimport und -export

Octave/Matlab bietet u.a. die folgenden Möglichkeiten zum Laden oder Speichern von Variablen in Dateien:

- Speichern aller aktuellen Variablen in einer Datei `test.mat`:

  `save test.mat` oder `save('test.mat')`
- Speichern aller Variablen im Klartextformat:

  `save -ascii test.mat` oder `save('test.mat','-ascii')`
- Nur Variablen `z` und `M` speichern:

  `save test.mat z M` oder `save('test.mat','z','M')`
- Laden aller in `test.mat` gespeicherten Variablen:

  `load test.mat` oder `load('test.mat')`
- Laden von Klartextdaten:

  `load -ascii test.mat` oder `load('test.mat','-ascii')`
- Ausgeben einer Textdatei auf dem Bildschirm:

  `type datei.txt`

Export in Textdateien

Zur Ausgabe von Klartext (Zeichenketten) in eine Datei existieren folgende Funktionen:

- Datei zum Schreibzugriff (`w` wie *write*) öffnen mit `fopen`:

```
dateihandle = fopen('datei.txt','w')
```

- Eine Zeichenkette (`%s`) ausgeben und einen Zeilenumbruch (`\n`):

```
s='Hallo Welt!';
fprintf(dateihandle,'%s\n',s)
```

  Dabei sind im Formatstring (d.h. im Beispiel `%s\n`) als Platzhalter möglich: `%d` oder `%i` für Ganzzahlen, `%s` für Zeichenketten, `%f` für Gleitkommazahlen und noch viele weitere Varianten, siehe `doc fprintf`. Will man ein einzelnes Backspace-Zeichen ausgeben, muss man im Formatstring `\\` schreiben.

- Datei schließen mit `fclose`:

```
fclose(dateihandle)
```

→ Aufgabe 24

Grafikexport mit Befehlen

Aus Gründen der Wiederverwendbarkeit werden Matlab-Plots meistens über `.m`-Skripte erzeugt. Den Export als Grafikdatei (z.B. JPG) sollte man konsequenterweise auch über Befehle steuern. . . statt mit der Maus:

- aktuellen Plot als Matlab-Figure speichern (geht nicht in Octave):

  ```
  saveas(gcf,'datei.fig')
  ```

- aktuellen Plot als PNG-Bild speichern (Rastergrafik, keine Kompression, Auflösung 600dpi statt Defaultwert 150dpi):

  ```
  print('-dpng','datei.png','-r600')
  ```

- aktuellen Plot als PDF-Datei speichern (Vektorgrafik):

  ```
  print('-dpdf','datei.pdf')
  ```

  Achtung: Die PDF-Datei hat nicht die korrekte enge Bounding-Box, sondern ist eine komplette A4-Seite! Sie muss z.B. mit `pdfcrop` im Terminal beschnitten werden (enthalten in TEX-Distribution).

- aktuellen Plot als EPS-(Level 2-)Farbbild speichern (Vektorgrafik):

  ```
  print('-depsc2','datei.eps')
  ```

  . . . empfehlenswert, z.B. Konvertierung nach PDF mit `epstopdf`

→ Aufgabe 25

Zufallszahlen

Zum Erzeugen „zufälliger“* Zahlen gibt es folgende Funktionen:

- `rand()` erzeugt gleichverteilte `reelle` Zufallszahlen aus $(0, 1)$
- `rand(n)` bzw. `rand(m,n)` erzeugen $n \times n$- bzw. $m \times n$-Matrizen aus reellen Zufallszahlen, gleichverteilt in $(0, 1)$
- `randi(imax)` erzeugt gleichverteilte Zufalls-*Ganzzahlen* zwischen 1 und n („`imax`-seitiger Würfel“)
- `randi(imax,n)` bzw. `randi(imax,m,n)` erzeugen ganzzahlige Matrizen aus gleichverteilten Zufallszahlen zwischen 1 und n
- `randn()` erzeugt standardnormalverteilte `reelle` Zufallszahlen, d.h. Mittelwert 0 und Standardabweichung 1
- `randn(n)` bzw. `randn(m,n)` liefern entsprechende Zufallsmatrizen

→ Aufgabe 26

*Wir diskutieren hier *nicht*, dass ein Computer nur Pseudozufallszahlen erzeugt. . .