Einführung in die Programmierung mit MATLAB

Dominik Edelmann

Numerische Analysis, Eberhard Karls Universität Tübingen

Wintersemester 2018/2019

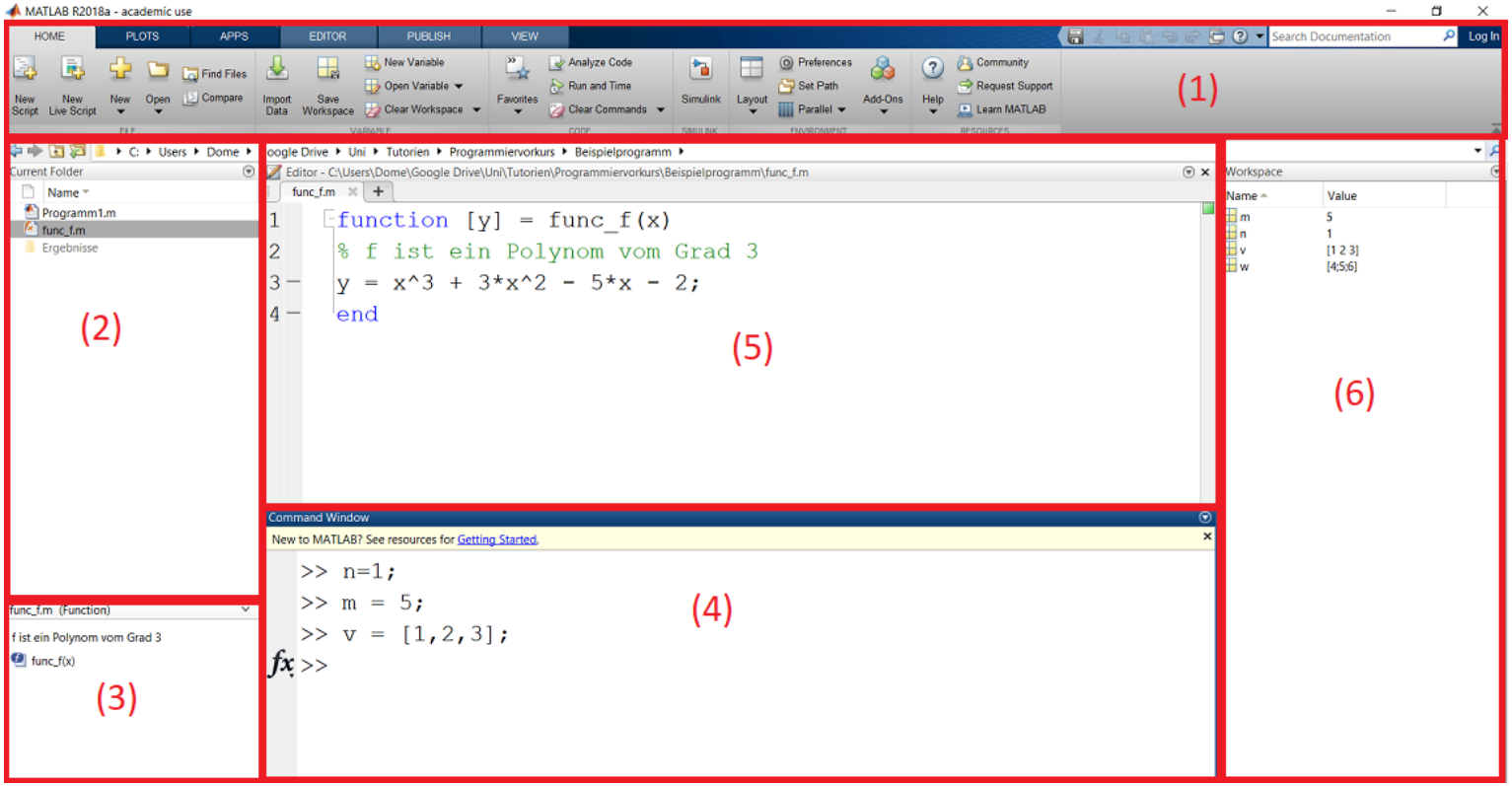
Was bedeutet Programmieren?

- Übersetzung von umgangssprachlichen Anweisungen in Computersprache
- Erstellen von Computerprogrammen

Matlab ist

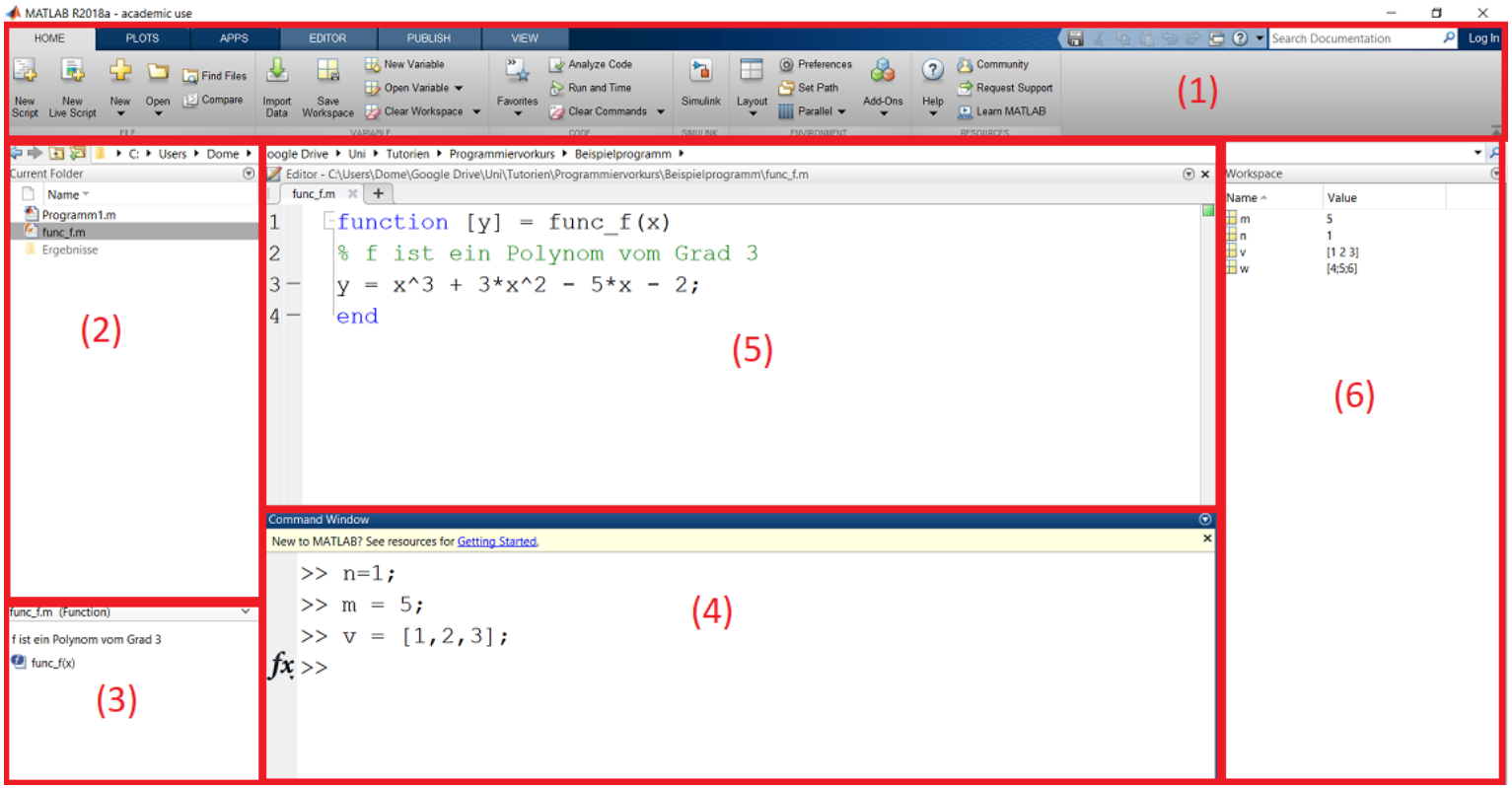
- Ein umfangreiches Software-Paket zur numerischen Lösung mathematischer Probleme
- Eine Programmiersprache, in der die Lösungsalgorithmen implementiert werden

Aufbau von Matlab (Beispiel)



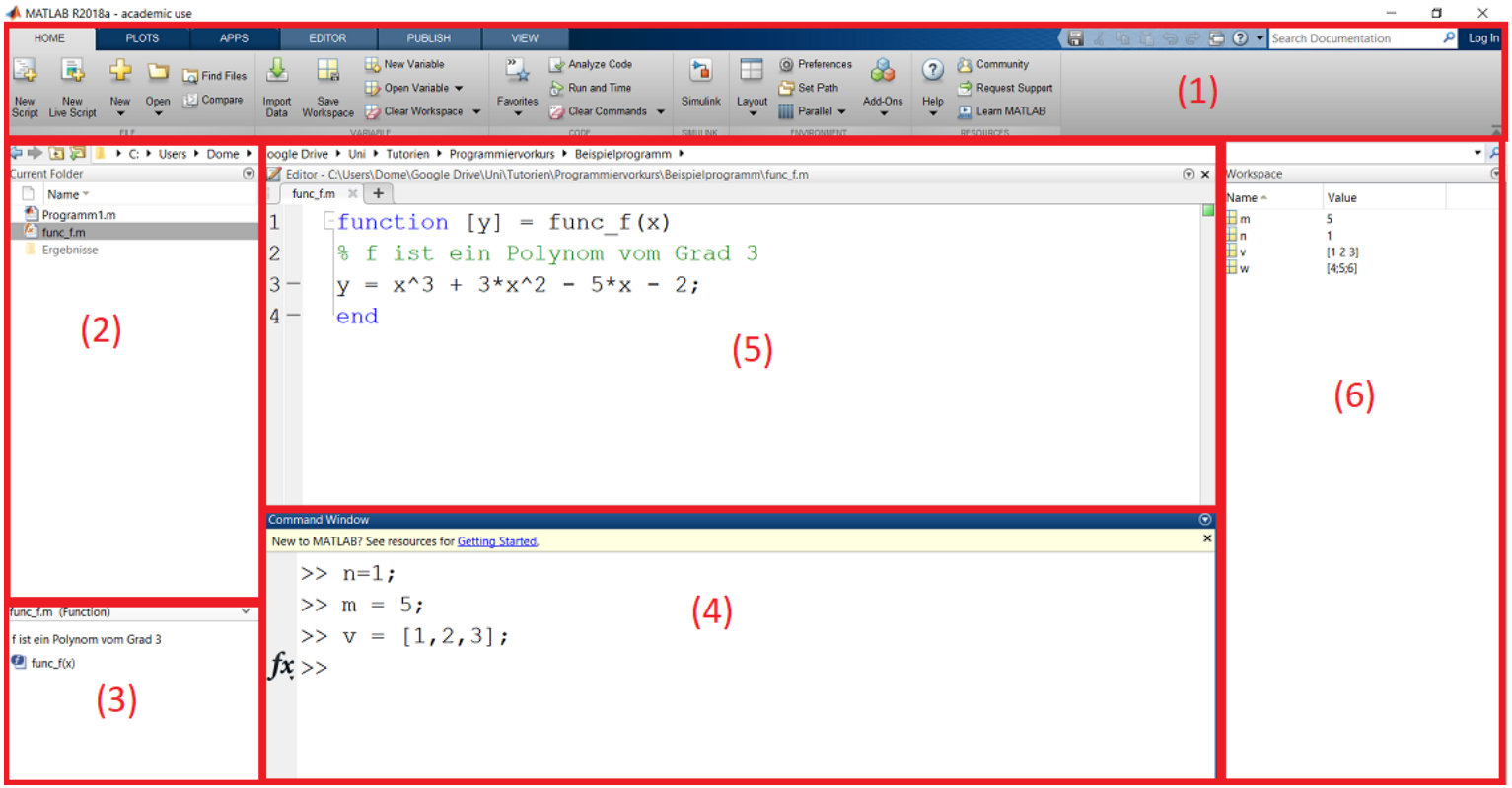
(1) enthält das Menü. Hier können Sie z. B. Einstellungen ändern oder die Hilfefunktion aufrufen.

Aufbau von Matlab (Beispiel)



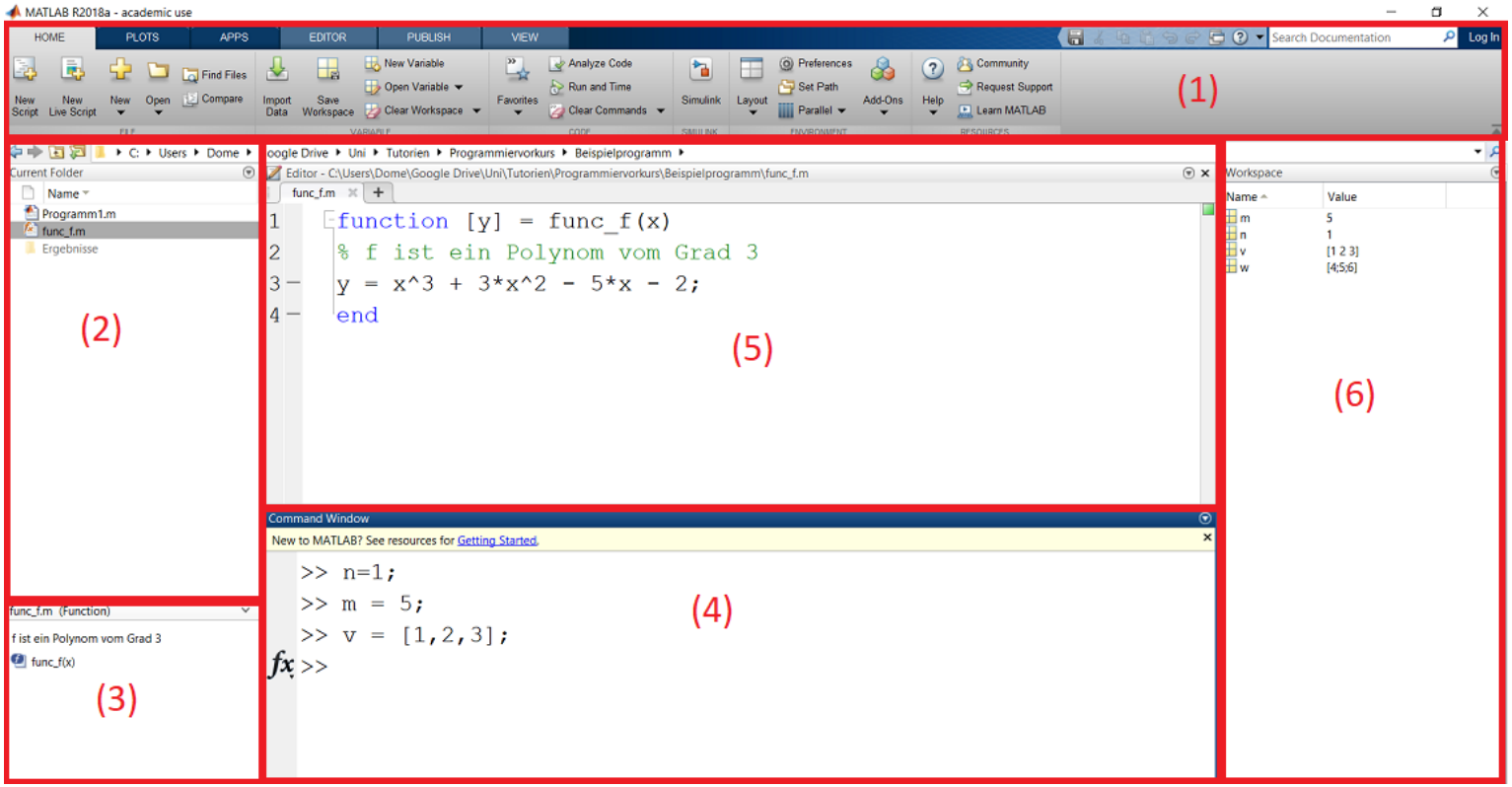
In (2) sehen Sie die Dateien und Unterordner, die sich in dem Ordner befinden, in dem Sie gerade arbeiten.

Aufbau von Matlab (Beispiel)



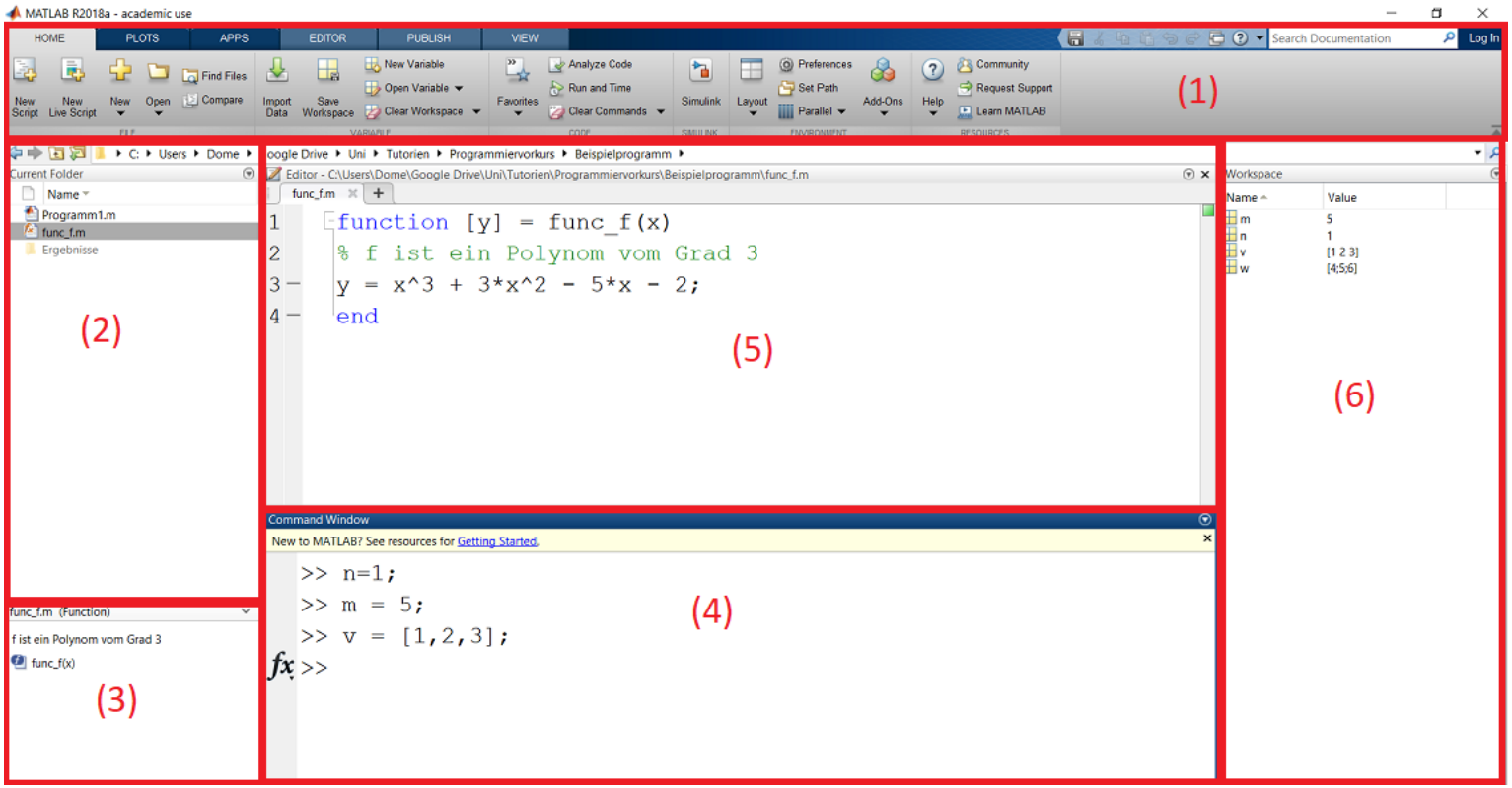
In (3) können Sie eine Kurzbeschreibung der Datei sehen, die in (2) gerade ausgewählt ist (sofern diese Datei eine Beschreibung enthält).

Aufbau von Matlab (Beispiel)



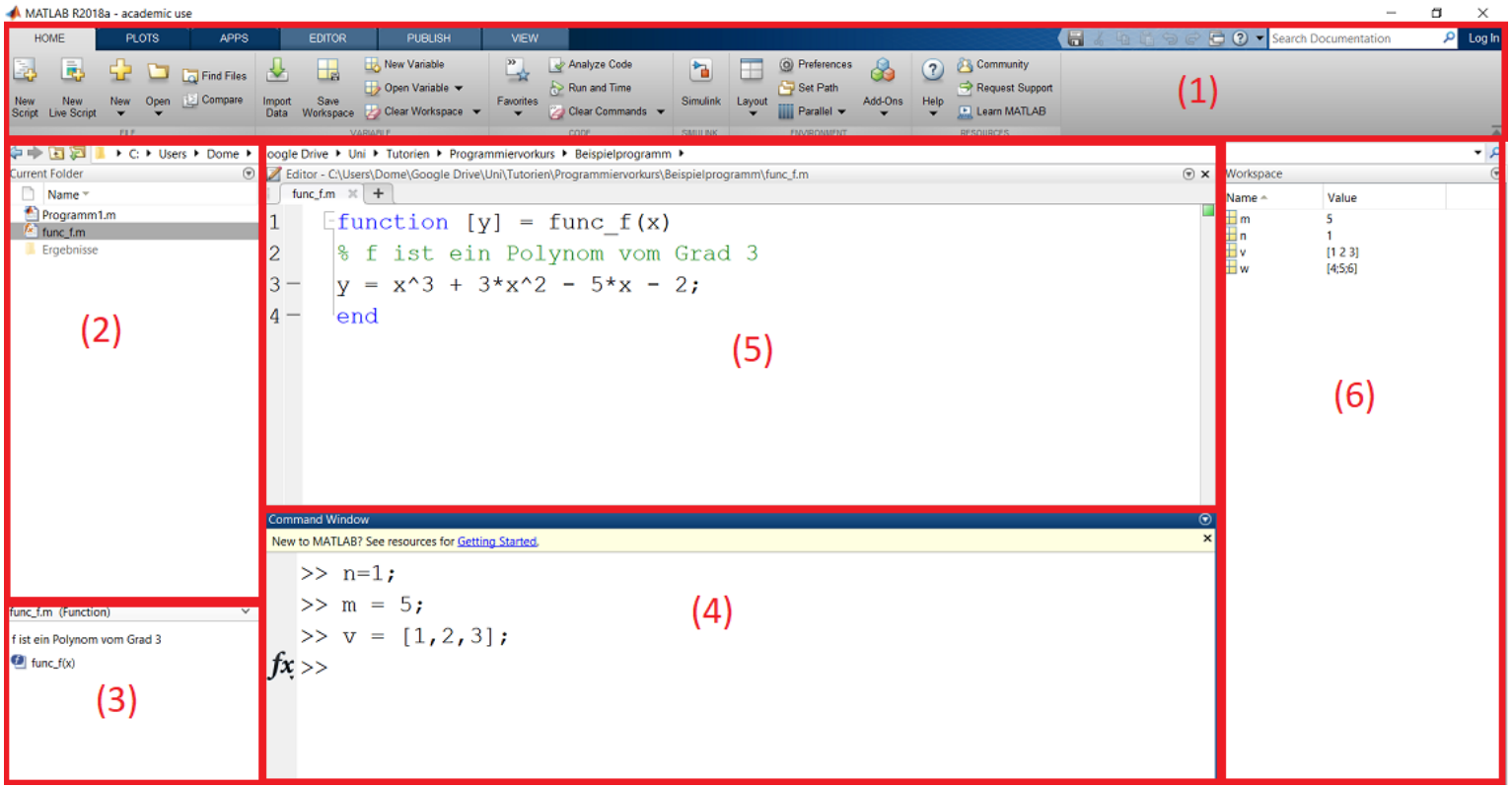
(4) ist das sog. *Command Window*, indem Sie direkt (Programmier-)Befehle eingeben können. Hier werden drei Variablen deklariert: zwei Zahlen $n = 1$ und $m = 5$, sowie ein Zeilenvektor $v = (1, 2, 3)$.

Aufbau von Matlab (Beispiel)



(5) ist der Editor und mit (4) das wichtigste Element. Hier sehen und modifizieren Sie den Code der Datei, die Sie in (2) geöffnet haben.

Aufbau von Matlab (Beispiel)



In (6) sehen Sie den *Workspace*. Das ist eine Liste mit Variablen, die zum aktuellen Zeitpunkt gespeichert sind.

Aufgaben

Aufgabe 1

a. Öffnen Sie Matlab.

b. Deklarieren Sie im Workspace einige Variablen, z. B. `n=3` und `m=5`.

c. Geben Sie beispielsweise die folgenden Befehle ein: `n+m`, `n+m;`, und `k=n+m`. Machen Sie sich die Unterschiede klar.

Syntax

- Damit ein Programmierbefehl durchgeführt werden kann, muss die richtige *Syntax* verwendet werden:
  `a = 3;`
  `b = 2a;` $\Leftarrow$ Nicht lauffähig!
- Jeder Befehl wird typischerweise mit einem Semikolon beendet, um die Ausgabe des Ergebnisses zu unterdrücken[1].
- Mit dem Prozentzeichen können Sie Kommentare in Ihr Programm einfügen:
  `p = 3.14; % a ist ein Näherungswert für pi`
  $\Rightarrow$ Alles ab dem Prozentzeichen bis zum Ende der Zeile wird von Matlab bei der Durchführung des Programms ignoriert.

[1] Manchmal ist es aber hilfreich, wenn bestimmte Werte ausgegeben werden.

Syntax

- Vordefinierte Konstanten, z. B. `pi`, `eps` (Maschinengenauigkeit). Nicht: `e` (man verwendet `exp(1)`).
- Vordefinierte Funktionen, z. B. `sin`, `exp` usw.
- Strings (Zeichenketten) werden mit einfachen Anführungszeichen umschlossen:
  ```
  string = 'Hallo.'
  ```
- Code kann mit dem 3-Punkt-Operator auf mehrere Zeilen verteilt werden (fortgeschritten)
  ```
  summe = 1 + 2 + 3 + 4 + 5 ...
  + 6 + 7 + 8 + 9 + 10;
  ```

Variablen

- Variablen und Operatoren (`+`, `-`, `*`, `/`, `...`) sind die wichtigsten Bausteine beim Programmieren.
- Variablen werden durch Ausdrücke wie `a=3+5`, `v = [1 2 3]` usw. deklariert.
- Das Gleichheitszeichen ist wie ein $:=$ in mathematischen Texten zu verstehen: Der Ausdruck rechts vom Gleichheitszeichen wird berechnet (falls möglich), und dann als Variable gespeichert, deren Namen links vom Gleichheitszeichen steht.
  $\Rightarrow$ `n = n+1` erhöht den Wert der Variable `n` um 1.

Variablennamen

- Variablennamen dürfen aus Klein- und Großbuchstaben, Zahlen und Unterstrichen gebildet werden.
- Darüber hinaus dürfen Sie nur mit Buchstaben beginnen.
- Matlab unterscheidet zwischen Groß- und Kleinschreibung!
- Es gehört zu einem guten Programmierstil, weitgehend selbsterklärende, intuitive und möglichst kurze Variablennamen zu verwenden.
- Halten Sie sich möglichst auch an Konventionen, die Sie sonst aus der Mathematik kennen: z. B. $n, m, \ldots$ für natürliche Zahlen, v, w für Vektoren und A für Matrizen etc.

Aufgaben

Aufgabe 2

Welche der folgenden Variablennamen sind in Matlab zulässig?

a) `2nach1`
b) `die erste`
c) `dieZweite`
d) `die_dritte`
e) `variable-nummer-1`
f) `variable-nummer.2`
g) `zins_in_%`

Aufgaben

Aufgabe 3

Welchen Wert hat die Variable `n` nach Zeile 4 und nach Zeile 8?

```
1:  n = 3;
2:  n = n+1;
3:  n = n/2;
4:  n = n^3;
5:  n = n-4;
6:  n = n/4;
7:  n = 5;
8:  n = n+1;
```

Skripte

- Ein (einfaches) Matlab-Programm besteht aus mehreren Funktionen und Skripten.
- Technisch gesehen sind beide eine `.m`-Datei.
- Es gibt jedoch einen großen Unterschied: Ein Skript *kennt* alle Variablen, die bereits im Workspace vorhanden sind. Alles was während der Durchführung des Skripts an Variablen deklariert, verändert etc. wird, ist auch nach der Durchführung im Workspace gespeichert.
  (Im Gegensatz dazu hat bei einer Funktion jeder Funktionsaufruf seinen eigenen Workspace. Dazu später mehr.)
  $\Rightarrow$ Der Aufruf eines Skriptes ist äquivalent dazu, die dort enthaltenen Befehle nacheinander in das Command Window einzugeben.

Aufgaben

Aufgabe 4

a. Per Rechtsklick können Sie in der linken Spalte *Current Folder* einen Ordner mit dem Namen *Programmierkurs* anlegen oder zu diesem navigieren.

b. Erstellen Sie per Rechtsklick ein neues Skript (Rechtsklick → New File → New Script) mit dem Namen `meinErstesSkript.m`

c. Öffnen Sie dieses Skript per Doppelklick. Die leere Datei erscheint im Editor.

d. Programmieren Sie nun (im Editor) einige der Befehle, die Sie bisher kennen. Deklarieren Sie nach Belieben einige Variablen und führen einige Rechenoperationen durch.

e. Rufen Sie ihr Skript auf, indem Sie im Command Window den Befehl `meinErstesSkript` eingeben.

Aufgaben

Aufgabe 5

1. Geben Sie, nachdem Sie das Skript aus der vorigen Aufgabe aufgerufen haben, den Befehl `clear all` ein. Was passiert mit den im Workspace gespeicherten Variablen?
2. Geben Sie den Befehl `clc` ein. Was passiert?

$\Rightarrow$ In den meisten Fällen ist es sinnvoll, diese beiden Befehle an den Anfang eines Skripts zu stellen.

Relationale Operatoren

- Neben dem Zuweisungsoperator = und den arithmetischen Operatoren `+ - * / ^` gibt es weitere Operatoren.
- Die relationalen Operatoren sind `== ~= > >= < <=`.

```
n = (3 ~= 4);
disp(n); % Ausgabe:  true
```

- Diese werden in der Regel verwendet, um bestimmte Bedingungen abzufragen.
- Auch hier gilt: Die Operation rechts von =, also hier die Abfrage, ob $3 \neq 4$ gilt, wird zuerst durchgeführt. Das Ergebnis `true` wird in n gespeichert.

Logische Operatoren

- Neben den relationalen Operatoren gibt es auch logische Operatoren: `&&` `||` `~` stehen für and, or bzw. not.
- Mit logischen Operatoren werden logische Variablen (also `true` und `false`) verknüft. Bei `||` um ein sog. *einschließendes Oder*.
  Beispiel:

```
x = 5;
y = (x < 6) || (x > 4);
disp(y); % Ausgabe true
z = (x > 0) && ~(x > 3);
disp(z); % Ausgabe false
```

if-Abfragen

- In sehr vielen Programmen gibt es einzelne Schritte, die nur unter bestimmten Bedingungen durchgeführt werden.
- Dafür werden if-Abfragen gepaart mit logischen Ausdrücken verwendet. Die Struktur ist dabei immer Folgende:

if *logischer Ausdruck*
  Anweisung(en)
elseif *logischer Ausdruck*
  Anweisung(en)
⋮
else
Anweisung(en)
end

if-Abfragen

- Das letzte else darf keinen logischen Ausdruck haben.
- Die Zeilen mit `if`, `elseif`, `else` und `end` werden **nicht** mit einem Semikolon beendet!
- `elseif` und `else` sind nicht erforderlich:

```
if x ~= 0
  y = 1/x;
end
```

  $\Rightarrow$ Da keine Alternative mit `else` angegeben wurde, werden diese Zeilen einfach übersprungen, falls $x = 0$ ist.
- if-else-Strukturen können beliebig ineinander geschachtelt werden. Matlab rückt die Anweisungen zwischen `if` und `else` um vier Leerzeichen ein. Behalten Sie diese Einrückung bei! Dadurch wird der Code übersichtlicher.

Aufgaben

Aufgabe 6

- Schreiben Sie ein Skript, welches für eine gegebene ganze Zahl $x \geq 0$ die Anzahl deren Stellen zurück gibt. Das Programm soll eine Fehlermeldung ausgeben, falls $x < 0$ ist, die exakte Anzahl der Stellen, falls $x < 1000$ ist und *mehr als drei Stellen*, falls $x \geq 1000$.
- Was gibt das Skript aus, wenn Sie es mit einer reellen Zahl $x \in (0, 1000)$ aufrufen, die keine ganze Zahl ist?

Schleifen

- Zur wiederholten Durchführung bestimmter Operationen.
- Schleifen sind die wichtigste Kontrollstruktur beim Programmieren.
- Die Themen der Vorlesung sind (zumindest anfangs) grob danach gegliedert, wie viele Schleifen ineinander geschachtelt werden müssen.

for-Schleifen

- for-Schleifen haben die Struktur
  for *Variable* = *Startwert:Endwert*
  Anweisung(en)
  end
- Beispiel: Berechnung von $S = \sum_{i=0}^{n} \frac{(-1)^i}{2i+1}$ für gegebenes n, z. B. $n = 5$:

```
n = 5;
S = 0; % Variable für die Summe
for i=0:n
  S = S + (-1)^i / (2*i+1);
end
```

for-Schleifen

Erläuterung:

- In der Zeile `for i=0:5` wird zunächst eine Variable `i=0` definiert.
- Alle Anweisungen die bis `end` auftreten werden durchgeführt, wobei die Variable *i* den Wert 0 hat.
- Dann wird *i* um 1 erhöht, hat also nun den Wert 1.
- Alle Anweisungen werden erneut durchgeführt, diesmal mit `i=1`
- Dann wird *i* um 1 erhöht und die Anweisungen erneut durchgeführt
- . . . so lange bis `i=5` ist, wobei alle Anweisungen dann ein letztes Mal durchgeführt werden.

for-Schleifen

- Manchmal möchte man nicht, dass die Schleifenvariable in jedem Schritt um 1 erhöht wird.
- Beispiel: Berechnung von $7! = 7 \cdot 6 \cdots 2 \cdot 1$

```
n = 7;
x = n;
for i=(n-1):-1:1
  x = x*i;
end
```

- Erklärung: `i` hat anfangs den Wert `n-1 = 6` und wird in jedem Schleifendurchlauf um `-1` erhöht, solange, bis die Schleife mit `i = 1` das letzte Mal durchgeführt wird.

Aufgaben

Aufgabe 7

1. Schreiben Sie ein Programm, welches die Zahlen von 1 bis 10 ausgibt.
2. Ändern Sie Ihr Programm so ab, dass es die Zahlen $k = 2i + 1$ für $i = 0, \ldots, 10$ ausgibt.

Aufgabe 8

Schreiben Sie ein Programm, welches

$$S = \sum_{i=0}^{n} q^i$$

für gegebenes q und n berechnet.
Welche Bedeutung hat diese Formel? Was ist der Grenzwert für $n \to \infty$ (und für welche q)?

while-Schleifen

- Manchmal ist nicht von Vornherein klar, wie oft die Anweisungen in einer Schleife wiederholt werden müssen.
- Beispiel: Man kann zeigen, dass die rekursiv definierte Folge

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{2}{x_k}\right), \quad x_1 = 2$$

  gegen $\sqrt{2}$ konvergiert.
- Möchte man dieses Verfahren realisieren, berechnet man nacheinander diese Folge, bis sich zwei aufeinanderfolgende Folgenglieder nicht mehr stark unterscheiden, also z. B. solange, bis $|x_{k+1} - x_k| < \texttt{tol}$.
- Es ist nun nicht mehr klar, wie viele Iterationen erforderlich sind.

while-Schleifen

- Eine while-Schleife hat die Struktur
  while *logischer Ausdruck*
  Anweisung(en)
  end
- Vor jedem Schleifendurchlauf wird gecheckt, ob *logischer Ausdruck* den Wert `true` liefert. Dann werden die Anweisungen durchgeführt und *logischer Ausdruck* wird erneut gecheckt.

while-Schleifen

```
x = [0 2]; %Startwerte
tolerance = 10^(-8);
k = 1; %Zähle Anzahl Iterationen
while abs(x(k+1)-x(k))>tolerance
  x(k+2) = 0.5*(x(k+1)+2/x(k+1))
  k = k+1;
end
disp(k);
```

Aufgaben

Aufgabe 9

Man kann zeigen, dass $\lim_{n\to\infty} \sum_{i=0}^{n} (-1)^i/(2i+1) = \pi/4$ gilt. Man kann also die Zahl π approximieren, indem man $4S_n$ für großes n berechnet, wobei S_n die n-te Partialsumme bezeichne.
Schreiben Sie ein Programm, welches das kleinste n bestimmt, so dass $|4S_n - \pi| < 10^{-8}$ gilt. Sie dürfen die in Matlab gespeicherte Zahl `pi` in der Abbruchbedingung verwenden.
Warum ist ihre Abbruchbedingung eigentlich unsinnig, wenn Sie die Zahl π approximieren möchten?
Überlegen Sie sich, ob obige Formel (aus numerischer Sicht) sinnvoll ist, um einen Näherungswert für π zu bestimmen.

break und continue in Schleifen

- `continue`: Der aktuelle Schleifendurchlauf wird unterbrochen und es wird an den Schleifenanfang gesprungen.
- `break`: Die komplette Schleife wird unterbrochen und die Abarbeitung wird nach der Schleife fortgesetzt.

```
for i=1:10
  if i == 4
    continue
  end
  if i == 8
    break
  end
  disp(i)
end
```

Welche Zahlen gibt das Programm aus?

Skripte und Funktionen

- Funktionen sind das Hauptwerkzeug zur Strukturierung von Programmen.
- Im Gegensatz zu Skripten können Funktionen so angelegt werden, dass sie *Funktionswerte* erwarten und *Rückgabewerte* liefern.

```
function [ar, per] = area(a,b)
%Berechne Fläche und Umfang eines Rechtecks mit
Seitenlängen a,b.
ar = a*b;
per = 2*a+2*b;
end
```

Skripte und Funktionen

- Der Dateiname muss mit dem Funktionsname übereinstimmen: In obigem Beispiel muss der Dateiname `area.m` sein.
- Die Funktion erwartet beim Aufruf zwei Variablen.
- Die Rückgabewerte müssen nicht speziell gesetzt werden.
- Benötigt eine Funktion keine Eingabewerte, so ist die runde Klammer hinter dem Funktionsnamen leer: `()`
- Liefert eine Funktion keine Rückgabewerte, so bleibt die eckige Klammer hinter `function` leer: `[]`.

Wichtige Funktionen in Matlab

- Trigonometrische Funktionen `sin, cos, tan`
- Exponentialfunktion und natürlicher Logarithmus `exp, log`
- Wurzelfunktion `sqrt`
- Absolutbetrag `abs`
- Verwaltungsfunktionen `length, size`
- Laden, Speichern `load, save` (wird nicht besprochen)
- Grafische Funktionen `plot` (später)
- *Mächtigere* Funktionen `lu, qr, norm` ... (wird nicht besprochen)

Aufgaben

Aufgabe 10

Temperaturen in Grad Celsius können mit der Formel $t_{Fahr} = \frac{9}{5} t_{Cel} + 32$ in Grad Fahrenheit umgerechnet werden. Schreiben Sie dafür eine Matlab-Funktion `function [tFahr] = celinfahr(tCel)`.

Aufgabe 11

Schreiben Sie eine Funktion `function S = geomSum(q,N)`, welche die N-te Partialsumme der geometrischen Reihe berechnet:

$$S = \sum_{k=0}^{N} q^i \, .$$

Aufgaben

Aufgabe 12

Schreiben Sie eine Funktion `function [ar,per] = area(r)`, welche die Fläche (= area) und den Umfang (= perimeter) eines Kreises mit Radius r zurückgibt.
Rufen Sie dann die Funktion (im Command Window) auf unterschiedliche Weisen auf (ohne Semikolon ; am Ende). Betrachten Sie nach jedem Aufruf den Workspace.

- `area(r)`
- `[ar,per] = area(r)`
- `x = area(r)`

Welchen Wert bekommt `x`? Machen Sie sich die unterschiedlichen Bedeutungen der Aufrufe klar.

Vektoren und Matrizen

- Matlab basiert auf Matrizen. (Matlab steht für **Mat**rix **Lab**oratory)
- Skalare sind (1×1)-Matrizen.
- Spalten- bzw. Zeilenvektoren sind $(n \times 1)$- bzw. $(1 \times n)$-Matrizen.

Vektoren: Erzeugung

- `a = [1, 2, 3, 4];` erzeugt einen Zeilenvektor der Dimension 4.
- `b = [1 2 3 4];` ist identisch zu `a`.
- `c = [1;2;3];` erzeugt einen Spaltenvektor der Dimension 3.
- `d = 1:0.5:3;` erzeugt den Vektor `[1 1.5 2 2.5 3]`.
- `e = [];` erzeugt einen *leeren Vektor*.

Vektoren: Zugriff

- `a = [1 2 5 9 3 7];` erzeugt Zeilenvektor der Dimension 6.
- Elementweiser Zugriff:
  `g = a(1);` erzeugt Skalar mit dem Wert $a_1 = 1$.
  `i = 3; g = a(i);` erzeugt Skalar mit dem Wert $a_i = a_3 = 5$.
  `g = a(end);` erzeugt Skalar mit dem letzten Wert von a, d. h. $g = 7$.
- Bereichswahl:
  `g = a(1:3);` enthält die ersten drei Werte von a.
  `v = [1 2 4]; g = a(v);` $\Rightarrow$ Was ist g?

Vektoren: Operationen

Wichtig: Indizes können nur positive natürliche Zahlen sein!

- Transponieren: Ist `c` ein Spaltenvektor, so ist `d = c'` ein Zeilenvektor.
- Länge eines Vektors ermitteln, z. B. für Schleifen:

```
n = length(a);
for i = 1:n
  disp(a(i));
end
```

Aufgaben

Aufgabe 13

Schreiben Sie mittels einer for-Schleife eine Funktion `function [c] = add(a,b)`, welche zwei Vektoren `a` und `b` addiert.

Aufgabe 14

Gegeben seien zwei Vektoren $a, b \in \mathbb{R}^n$. Welche Arten der Multiplikation von Vektoren kennen Sie? Welche Dimensionen haben die jeweiligen Resultate? Schreiben Sie eine Funktion `function [y] = innerProduct(a,b)`, welche das Skalarprodukt zweier Vektoren berechnet.

Aufgaben

Aufgabe 15

Schreiben Sie eine Matlab-Funktion `function [y] = mean(v)`, die das arithmetische Mittel der Elemente von `v` berechnet.

Aufgabe 16

Schreiben Sie eine Matlab-Funktion `function [n1,n2,nInf] = norms(v)`, die die $\|\cdot\|_1$-, $\|\cdot\|_2$- sowie die $\|\cdot\|_\infty$-Norm des Vektors `v` berechnen.

Vektoren: Operationen

Operatoren wirken auf Vektoren möglicherweise anders als auf Skalare.

- Zuweisungsoperator: =
  `a = b`: $a(i) = b(i)$ für alle i.
- Arithmetische Operatoren: + −
  `a + b`: $a(i) + b(i)$ für alle i.

Vorsicht!

Matlab addiert auch Vektoren und Skalare sowie Vektoren unterschiedlicher Dimensionen, insbesondere auch Zeilen- und Spaltenvektoren selber Länge. Dies kann zu unerwarteten Ergebnissen führen.

- Arithmetische Operatoren: *
  `a * b` funktioniert nur dann, wenn `a` ein Zeilen- und `b` ein Spaltenvektor ist (Skalarprodukt) oder umgekehrt (dyadisches Produkt).
  $\Rightarrow$ entspricht Matrix-Multiplikation.

Vektoren: Operationen

- Relationale Operatoren werden elementweise ausgeführt: `a == b` liefert nicht `true` oder `false`, sondern einen Vektor, deren Einträge `true` oder `false` sind:

```
a = [1 2 3 4];
b = [1 9 3 4];
c = (a == b); % c = [1 0 1 1]
```

- Entsprechendes gilt für >, >=, usw.
- if-Abfragen werden nur dann durchgeführt, wenn alle Werte `true` sind:

```
if a == b disp(a); end
```

liefert keine Ausgabe.

Matrizen: Erzeugung

- `A = [1 2 3 4; 5 6 7 8]; % 2 x 4 - Matrix`

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

- `B = [1 5; 2 6; 3 7; 4 8]'; % identisch zu A`
- `B(2,3) = 15; % ändert Element von B`

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 15 & 8 \end{pmatrix}$$

Matrizen: Erzeugung

- `I = eye(n); % n x n - Einheitsmatrix`
- `A = zeros(n,m); % n x m - Matrix mit Nullen`
- `B = ones(n,m); % n x m - Matrix mit Einsen`
- `v = [1 2 3]; D = diag(v);` erzeugt die Matrix

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

- `M = diag(v,1);` erzeugt die Matrix

$$D = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrizen: Zugriff

- Als Ganzes: `B = A;`.
- Elementweise:
  `b = A(3,1);`
  `i = 2; j = 3; g = A(i,j);`
  `g = A(end,1);` `g` ist Skalar und enthält das erste Element aus der letzten Zeile von `A`.
- Bereichswahl:
  `B = A(2:4,2:4);` ist (3×3)-Untermatrix

$$B = \begin{pmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Für Vektoren v, w mit positiven ganzen Zahlen liefert $A(v, w)$ die Matrix mit den Einträgen a_{v_i, w_j} (fortgeschritten).

Matrizen: Operationen

- Transponieren:
  ```
  B = A'; % B ist Transponierte von A
  ```
- Größe ermitteln:
  ```
  [n m] = size(A); % n Zeilen, m Spalten
  ```
  (gemäß der Notation $A \in \mathbb{R}^{n \times m}$)

Aufgaben

Aufgabe 17

Schreiben Sie eine Funktion `function [C] = matMult(A,B)`, welche zwei Matrizen $A \in \mathbb{R}^{n \times m}$ und $B \in \mathbb{R}^{m \times k}$ multipliziert.
Hinweise: Die Elemente von C sind

$$c_{ij} = \sum_{l=1}^{m} a_{il} b_{lj}, \quad i = 1, \ldots, n \quad j = 1, \ldots, k$$

Verwenden Sie die `size`-Funktion zur Ermittlung von `m`, `n` und `k`. Achten Sie auf die richtige Initialisierung von `C`.

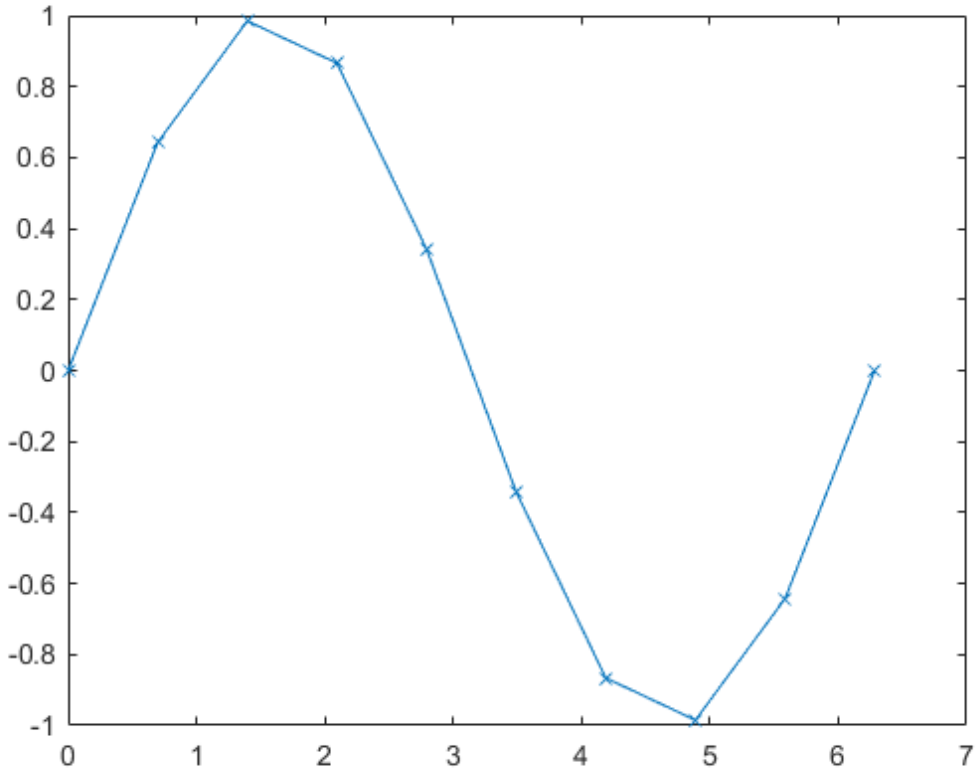
Aufgabe 18

Überlegen Sie sich, wie man obige Funktion verwenden kann, um das Produkt Av einer Matrix $A \in \mathbb{R}^{n \times m}$ und einem Vektor $v \in \mathbb{R}^m$ berechnen kann.

2d Plots

- Matlab plottet nicht *stetig*, sondern interpoliert Punktmengen $(x(i), y(i))_{i=1}^{N}$.

```
x = linspace(0,2*pi,10); y = sin(x);
plot(x,y); % Plotte Vektoren der Dimension 10
```



Vorgehen beim Plotten

1. Falls die zu plottende Funktion nicht bereits als Punktmenge vorliegt, muss diese in eine überführt werden.
   Lege Intervallgrenzen $[a, b]$ fest und erstelle Vektor `x` mit *Stützstellen* (Auswertungspunkte). Bsp:
   ```
   x = -1:0.1:1;
   % Intervall [-1,1], Abstand 0.1 zwischen zwei Punkten
   x2 = linspace(-1,1,21); % entspricht x
   ```
2. Werte die Funktion in allen Stützstellen aus, beispielsweise mit einer Schleife.
   ```
   for i=1:length(x) y(i)=f(x(i)); end
   ```
3. Die Punktmenge wird an Matlab zum Plotten übergeben
   ```
   plot(x,y,'Marker','x')
   ```
4. Nach Wunsch können an dem Plot noch verschiedene Einstellungen vorgenommen werden.

Plotten: Beispiel

```
% Plotte sin(x) fuer x in [0,2pi]

% Wertetabelle erstellen

% Vektor mit 50 aequidistanten Stuetzstellen erzeugen
x = linspace(0,2*pi,50);
y = zeros(50,1); % Initialisierung

% Berechne sin(x(i)) fuer alle x(i)
for i=1:length(x)
y(i) = sin(x(i));
end

% Plotten mit roter Farbe
% Die Punkte (x(i),y(i)) werden mit einem Kreis markiert
plot(x,y,'r','Marker','o');
```

Plotten: Beispiel

```
18 % Ueberschrift und Achsenbeschriftungen!  Matlab kann
auch LaTeX
19 title('Mein erster plot');
20 xlabel('0 \leq x \leq 2 \pi');
21 ylabel('sin(x)');
22
23 % Lege Achsen fest:  [xmin,xmax,ymin,ymax]
24 axis([0,2*pi,-1.1,1.1]);
25
26 % Speichere Plot als png-Datei
27 % Erstes Argument:  Dateiformat
28 % Zweites Argument:  Aufloesung
29 % Letztes Argument:  Dateiname
30 print('-dpng','-r100','sin_plot.png');
```

Mehrere Plots in einem Schaubild

- Eine Möglichkeit:

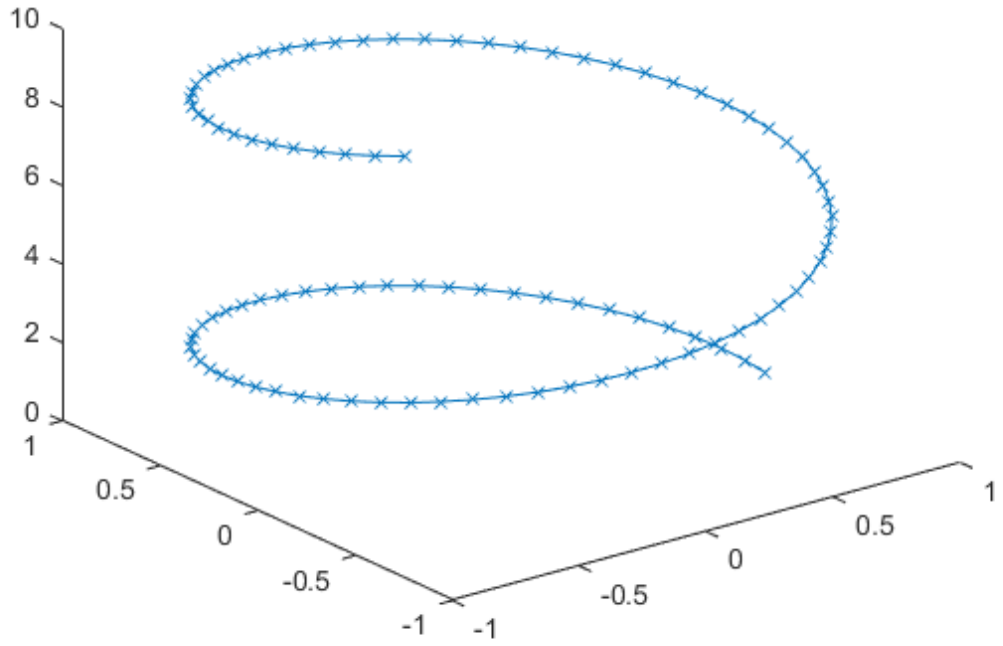
```
% x1, y1 Vektoren der Dimension n
% x2, y2 Vektoren der Dimension m
plot(x1,y1);
hold on; % Garantiert, dass nochmal in dieselbe Figur
geplottet wird.
plot(x2,y2);
legend('Erste Linie','Zweite Linie');
```

- Andere Möglichkeit: `plot(x1,x2,y1,y2)`

3D-Plot einer Kurve

- 3D-Plot einer Kurve $\gamma : \mathbb{R} \supset I \to \mathbb{R}^3$

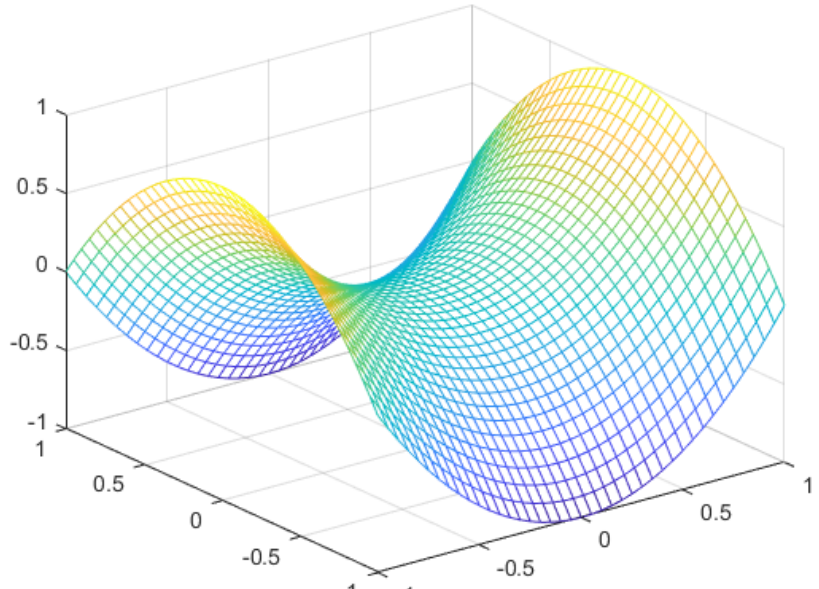
```
plot3(x,y,z);
% x, y, z Vektoren der Dimension n
```



3D-Plot eines Graphen

- 3D-Plot einer Funktion $f : [a_1, b_1] \times [a_2, b_2] \to \mathbb{R}$
- Geplottet wird die Punktmenge $(x_i, y_j, f(x_i, y_j))$

```
1:  a = -1:0.05:1; % Diskretisierung von [a1,b1]
2:  b = -1:0.05:1; % Diskretisierung von [a2,b2]
3:  [x,y] = meshgrid(a,b) % Generiere Gitterpunkte
4:  z = x.^2 - y.^2;
5:  mesh(x,y,z);
```



Aufgaben

Aufgabe 19

Plotten Sie die Funktion

$$f : [-2, 2] \to \mathbb{R}, \quad x \mapsto \begin{cases} -x^2, & x < 0 \\ x, & 0 \leq x \leq 1 \\ x^2 + 1, & x > 1 \end{cases}$$

Schreiben Sie dafür eine Funktion `func_f`, welche obige Funktion realisiert, sowie ein Skript `plot_f`, in der die obige Funktion ausgewertet und geplottet wird.
Achten Sie auf eine sinnvolle Beschriftung des Plots.
Wie unterscheidet sich das Verhalten des Plots vom Verhalten der Funktion f im Punkt $x = 1$?

Aufgaben

Aufgabe 20

Schreiben Sie eine Funktion `y = fschar(x,b)`, welche die Parameter x und b erwartet und den Wert $y(x) = bx^2$ zurück gibt. Schreiben Sie ein Skript `plot_fschar` welches diese Funktion für $x \in [-1, 1]$ und $b \in \{1/2, 1, 3/2, 2\}$ in ein Schaubild plottet. Achten Sie auf eine sinnvolle Beschriftung.

Aufgabe 21

Schreiben Sie ein Skript `schaubenplot.m`, welches die Funktion

$$f : [0, 10] \to \mathbb{R}^3, \quad t \mapsto (\cos(t), \sin(t), t)$$

plottet. Wie lässt sich die Anzahl der *Windungen* erhöhen?
Hinweis: Die eingebauten Matlab-Funktionen wirken komponentenweise, wenn man Vektoren übergibt.

Aufgabe 22

Schreiben Sie ein Skript `graphenplot.m`, welches die Funktion

$$f : [-3,3] \times [-3,3] \to \mathbb{R}, \quad (x,y) \mapsto \exp(x)\sin(y^2)$$

graphisch darstellt.
Hinweis: Die eingebauten Matlab-Funktionen wirken komponentenweise, wenn man Vektoren übergibt. Vektoren können mit `.*` komponentenweise multipliziert werden.

Aufbau eines (kleinen) Progamms

- Wir haben kennengelernt, wie man mithilfe von Skripten und Funktionen (kleine) Programme aufbauen kann.
- Es ist eine Frage des Stils, wie viele verschiedene Funktionen und Skripte verwendet.
- Typisch: Ein *Hauptskript*, in dem der Benutzer Parameter eingibt, dazu in der Regel mindestens eine Funktion, die im Skript aufgerufen wird.
- Drei *Blöcke*:
  - *Preprocessing:* Bspw. Eingabe von Parametern.
  - *Processing:* Code, in dem der Algorithmus durchgeführt wird.
  - *Postprocessing:* Ausgabe, Speicherung und Visualisierung von Ergebnissen.
- In einem *fertigen* Programm sollte der Benutzer nur noch die Input-Parameter im *Hauptskript* eingeben. Das Processing und Postprocessing sollten dann automatisch nach Start des Programms ablaufen.

Einschub: Function Handles

- Man kann mithilfe des Codes

  ```
  f = @(x)(x^2+1)
  ```

  eine Funktion (genauer: ein *function handle* definieren, wobei in der letzten Klammer der Funktionsausdruck steht).
- Diese Funktion kann, nachdem sie definiert wurde, genauso aufgerufen werden, wie die Funktionen, die als Dateien gespeichert wurden.
- Nachdem der Workspace gelöscht oder Matlab beendet wurde ist die Funktion nicht mehr verfügbar.
- Vorteil: Function handles können wie Variablen an andere Funktionen übergeben und dort verwendet werden, sofern die Syntax dort für eine Funktion ausgelegt ist.

Beispiel: Numerische Integration

- Man kann das Integral $\int_a^b f(x)dx$ mithilfe der sog. *Rechtecksregel* approximieren. Dabei wird das Intervall $[a, b]$ aufgeteilt in N kleine Teilintervalle $[a, a+h], [a+h, a+2h], \ldots, [a+(N-1)h, b]$ der Länge h (d. h. $a + Nh = b$).
- Es gilt nun

$$\int_a^b f(x)dx = \sum_{k=1}^{N} \int_{a+(k-1)h}^{a+kh} f(x)dx\,.$$

- Die Idee ist nun, dass auf einem *kleinen* Intervall

$$\int_c^{c+h} f(x)dx \approx hf(c)$$

- Daraus ergibt sich die Rechtecksregel

$$I := \int_a^b f(x)dx \approx \sum_{k=1}^{N} hf(a+(k-1)h) =: I(h)$$

Beispiel: Numerische Integration

```
1  % Funktion, die integriert werden soll
2  f = @(x) exp(-x.^2);
3
4  % Intervall I = [a,b], ueber das integriert wird
5  a = -0.5;
6  b = 0.5;
7
8  % Anzahl Teilintervalle
9  N = 100;
10
11 % Aufruf der Funktion, die die Summe in der
Rechtecksregel berechnet:
12 % Die Funktion f wird als Parameter uebergeben.
13 Ih = integrate_rect(f,a,b,N);
14 disp(Ih);
```

Beispiel: Numerische Integration

```
 1 function [Ih] = integrate_rect(f,a,b,N)
 2 % Approximiert das Integral von f ueber das Intervall
I=[a,b] mit der
 3 % Rechtecksregel auf N aequidistanten Teilintervallen.
 4
 5 h = (b-a)/N; % Laenge jedes Teilintervalls
 6 Ih = 0; % Initialisiere Rechteckssumme auf 0
 7
 8 for i=1:N
 9 Ih = Ih + f(a+(i-1)*h);
10 end
11 Ih = h*Ih; % Multiplikation mit h, wurde ausgeklammert.
12
13 end
```

Fehlersuche

- Hat man einen Code fertiggestellt, wird dieser in den meisten Fällen nicht auf Anhieb funktionieren.
- Es ist normal, einen großen Teil der Zeit mit der Fehlersuche (Debugging) zu verbringen.
- Es gibt verschiedene Arten von Fehlern:
  - Syntax-Fehler, z. B. `A = * A`: Leicht zu beheben, da Matlab euch sagt, was falsch ist.
  - Logische Fehler: Schwer zu finden, da das Programm aus Sicht des Computers fehlerfrei ist, aber nicht das richtige Ergebnis liefert.

Typische Fehlerquellen

- Falsche Syntax, z. B. `y = 2x+1;`
- Unvollständiges Programm, z. B. wenn man sich nicht genug Gedanken über negative Zahlen macht.
- Unerwartete Argumente für Funktionen, z. B. wenn man einer Funktion, die Skalare erwartet, Vektoren übergibt.
- Unerwarteter Zustand von Daten, z. B. eine Variable, die später noch gebraucht wird, wird aus Versehen überschrieben.
- Logische Fehler, z. B. falsche Bedingungen in if-Abfragen.

Fehlersuche

Ein paar Tipps:

- Bleiben Sie ruhig: Fehler beim Programmieren sind absolut normal.
- Systematik: Gehen Sie systematisch vor!
- Fehlermeldungen: Lesen Sie die Fehlermeldungen! Zumindest die Zeilenangaben sind verständlich. Viele Fehler lassen sich dadurch leicht beheben.
- Fehlereinschätzung: Ist es eher ein einfacher Syntax-Fehler?
- Fehlereingrenzung: Isolieren Sie den Fehler:
  - Kommentieren Sie Teile des Codes aus, die nicht unbedingt nötig sind. Taucht der Fehler immernoch auf?
  - Hat der Code früher bereits funktioniert? Was genau haben Sie verändert?
- Fragen Sie andere! Das ist meistens die schnellste Methode.

Performance

- Performance ist das *Zeitverhalten* eines Programms.
- Hier: Performance ist wichtig, wenn es in der Programmieraufgabe ausdrücklich verlangt ist.
- Performance sollte erst zum Schluss optimiert werden, wenn das Programm bereits lauffähig ist und richtige Ergebnisse liefert.
- Performance kann leicht mit `tic` und `toc` gemessen werden (zurückgegeben wird die Zeit, die zwischen `tic` und `toc` vergangen ist), oder mit dem Matlab-Profiler (fortgeschritten, wird nicht besprochen).

Beispiel: Fibonacci-Zahlenfolge

Die Fibonacci-Zahlenfolge ist definiert durch[2]

$$f_{n+2} = f_{n+1} + f_n\,, \quad f_1 = f_2 = 1\,.$$

Man kann leicht eine *rekursive Funktion* schreiben:

```
1 function [fn] = fib_recursive(n)
2 % Berechnet das n-te Element der Fibonacci-Folge rekursiv
3 if n <= 2
4 fn = 1;
5 else
6 fn = fib_recursive(n-1)+fib_recursive(n-2);
7 end
```

[2] Index-Verschiebung, da in Matlab bei 1 begonnen wird zu zählen.

Aufgaben

Aufgabe 23

Obiges rekursives Programm benötigt bereits ca. eine Minute, um f_{40} zu berechnen. Überlegen Sie sich grob, warum der Rechenaufwand des rekursiven Programms immens ist.

Aufgabe 24

Schreiben Sie ein (nicht-rekursives) Programm, um die n-te Fibonacci-Zahl zu berechnen. Testen Sie die Performance.
Hinweis: Verwenden Sie einen Vektor der Länge n sowie eine Schleife.

Aufgaben

Aufgabe 25

In dieser Aufgabe optimieren Sie Ihre Funktion zur Berechnung der geometrischen Summe aus Aufgabe 11.
Nehmen Sie zunächst an, Matlab kenne den Befehl zur Berechnung der Potenz und ändern Sie ihren Code so, dass nur Additionen und Multiplikationen verwendet werden.
Weisen Sie mithilfe von `tic` und `toc` nach, dass die Laufzeit zur Berechnung der N-ten Partialsumme in etwa proportional zu N^2 ist. Können Sie sich das erklären? Wie viele *Operationen* sind notwendig um die N-te Partialsumme zu berechnen? Hinweis: Eine *Operation* entspricht einer Addition oder einer Multiplikation.
Optimieren Sie ihr Programm so, dass die Laufzeit nur noch proportional zu N ist.

Allgemeines zum Übungsbetrieb

- Es wird voraussichtlich 6 Programmieraufgaben geben.
- Es wird empfohlen, zu zweit oder zu dritt abzugeben.
- Geben Sie Ihr Programm in einem .zip-Ordner ab!
- Abgabe per Mail an `progtutor@na.uni-tuebingen.de`
- Zulassungskriterium zur Klausur und zur Scheinvergabe (Programmierpraktikum):
  50 % der Programmieraufgaben müssen abgegeben und akzeptiert sein.
- Nur lauffähige Programme werden bewertet.

Alle Infos finden Sie unter
`http://na.uni-tuebingen.de/`

Format der Abgabe

Abzugebende Funktionen und Skripte müssen folgendes Format haben:

```
1   function [C] = funktion(A,B)
2   % Ein paar Worte, was die Funktion macht
3   % Input:
4   % A: Beschreibung
5   % B: Beschreibung
6   % Output:
7   % C: Beschreibung
8   %
9   % Von:  Name1, Name2, Name3
10  % Email:  Email1, Email2, Email3
11  % Datum:  xx.xx.xx
```

Halten Sie sich an die in der Aufgabenstellung geforderten Funktionsnamen.

Fragen?