**GNU Octave – Einführung**

Link: <https://www.youtube.com/watch?v=TqwSlEsbObg>

**Variablen**

Variablen werden mit ihrem zugewiesenen Wert wird mit der Eingabe „*Variable = Wert*“ festgelegt. Der Variablentyp, also „*double*“, „*char*“, usw. wird dabei automatisch zugewiesen.

>> a=7  
a = 7

>> b=7.56  
b = 7.5600

>> text='Hallo!'  
text = Hallo!

>> c=300; % Das Semikolon „;“ unterdrückt das erneute Anzeigen   
>> der definierten Variable

Beispiele zur Einagbe von Variablen.

Dabei wird zwischen groß- und kleingeschrieben Variablen unterschieden

>> x='kleines x'  
x = kleines x

>> X = "grosses X"  
X = grosses X

>>

Unterscheídung von Groß- und Kleinschreibung.

Um anzuzeigen, welche Variablen definiert wurden, verwendet man die Befehle „*who*“ und „*whos*“. Dabei wird im letzteren Fall ein detaillierter Überblick über die Variablen gezeigt.

>> who  
Variables visible from the current scope:

X a ans b c text x

>>

Anzeigen einer Variablenübersicht mittels dem Befehl „who“.

Um Variablen zu löschen, verwendet man den Befehl „*clear*“.

>> clear a b % löscht Variable „a“ und „b“  
  
>> clear all % löscht alle Variablen

>>

Löschen von ausgewählten Variablen oder allen Variablen.

>> whos  
Variables visible from the current scope:

variables in scope: top scope

Attr Name Size Bytes Class

==== ==== ==== ===== =====

X 1x9 9 char

a 1x1 8 double

ans 1x1 8 double

b 1x1 8 double

c 1x1 8 double

text 1x6 6 char

x 1x9 9 char

Total is 28 elements using 56 bytes

>>

Anzeigen einer Variablenübersicht mittels dem Befehl „who“.

**Rechenoperationen**

Die Grundrechenarten werden mit den Symbolen „+, -, \*, /, ^“ durchgeführt.

>> a + b % Addition   
>> a - b % Subtraktion  
>> a \* b % Multiplikation  
>> a ^ b (oder) a \*\* b % Exponent  
>> a / b % Division

>>

Symbole für die Grundrechenarten.

**Weitere Befehle**

>> date % zeigt aktuelles Datum an  
ans = 14-Jan-2023  
  
>> clc % bereinigt das Befehlsfenster  
>> home % bereinigt das Befehlsfenster  
  
>> help Befehl % Informationen/Hilfe zu Funktionen und Befehlen  
>> lookfor Wort % sucht nach allen Befehlen bzw. Funktionen, die „Wort“ enthalten

>> pi % gibt die Zahl Pi an  
ans = 3.1416

>>

Einige nützliche Befehle.

**Zeilenvektoren**

In GNU können Vektoren definiert werden. In Programmiersprachen entspricht das den „*Arrays*“. Ein Zeilenvektor wird wie folgt definiert.

>> a = [3 5 -2 0 -1] % Zeilenvektor mit 5 Einträgen anlegen  
a =

3 5 -2 0 -1

>>

Definiton eines Zeilenvektors.

Wenn man einen bestimmten Wert des Vektors ausgegeben habe möchte, muss man die Stelle, an der der Wert gewünschte Wert steht in Klammern hinter den Variablennamen schrieben: „*Variable(Stelle)*“.

>> a(3) % 3. Eintrag aus Zeilenvektor angeben

ans = -2

>>

Ausgabe des Wertes an der 3. Stelle eines Zeilenvektors.

Einen Wert in einem Zeilenvektor ersetzt man mit der Eingabe „*Variable(Stelle) = Wert*“.

>> a(3) = 10 % 3. Eintrag im Zeilenvektor wird durch „10“ ersetzt  
a =

3 5 10 0 -1

>>

Ersetzen des Wertes an der 3. Stelle durch einen neuen Wert.

Anfügen eines zusätzlichen Wertes geht analog zum Ersetzen. Es muss nur eine Stelle angegeben werden, die noch nicht angelegt wurde.

>> a(6) = -4 % ein 6. Eintrag wird angefügt  
a =

3 5 10 0 -1 -4

>>

Anfügen eines zusätzlichen Eintrags für einen Zeilenvektor.

Um mehrere Einträge eines Zeilenvektors aufzurufen, werden die gesuchten Stellen in eckigen Klammern geschrieben: „*Variable([Stelle1 Stelle2 …])*“.

>> a([1 4]) % 1. Und 4. Eintrag werden aufgerufen  
ans =

3 0

>>

Aufrufen von mehrern Einträgen eines Zeilenvektors.

Der Befehl für das Ersetzen mehrere Einträge hat folgende Syntax:   
„*Variable([Stelle1 Stelle2 …]) = [Wert1 Wert2 …]*“.

>> ([1 4]) = [5 9] % 1. Und 4. Eintrag werden durch „5“ und „9“ ersetzt   
‬a =

5 5 10 9 -1 -4

>>

Ersetzen von mehrern Einträgen eines Zeilenvektors.

Einen Zeilenvektor anlegen, deren Einträge die Schrittweite „1“ haben, kann mit dem Befehl:  
„*Variable =* *Anfangswert : Endwert*“.

>> b = 1:5 % erzeugt einen Zeilenvektor mit den Einträgen von 1 bis 5 mit der Schrittweite „1“   
b =

1 2 3 4 5

>>

Erstellen eines Zeilenvektors mit Einträgen mit einer Schrittweite von „1“.

Um die Einträge in einem bestimmten Bereich aufzurufen, wird der Aufruf mithilfe des Befehls:  
„*Variable(Stelle\_X : Stelle\_Y)*“ durchgeführt.

>> b(2:4)  
ans =

2 3 4

>>

Aufrufen von mehrern Einträgen in einem bestimmten Bereich eines Zeilenvektors.

Die Schrittweite kann auch beliebig festgelegt werden:   
„*Variable =* *Anfangswert : Schrittweite : Endwert*“

>> c = 0:0.1:0.5 % erzeugt einen Zeilenvektor mit Einträgen  
 von 0 bis 0,5 mit der Schrittweite „0,1“   
  
c =

0 0.1000 0.2000 0.3000 0.4000 0.5000

>>  
>> d = 5:-1:0 % erzeugt einen Zeilenvektor mit Einträgen  
 von 5 bis 0 mit der Schrittweite „-1“  
d =

5 4 3 2 1 0

>>

Erstellen eines Zeilenvektors mit Einträgen mit einer beliebigen Schrittweite.

Mit dem Befehl „*linspace(Startwert, Endwert, Anzahl der enthaltenen Punkte)*“ können Intervalle mit festgelegter Punktanzahl definiert werden.

>> f = linspace(1,5,4) % Intervall von 1 bis 5 mit 4 Punkten  
f =

1.0000 2.3333 3.6667 5.0000

>>

Anlegen eines Intervalls mit festgelegter Anzahl an Punkten.

**Rechenoperationen mit Zeilenvektoren**

>> A=[1 -1 2]  
A =

1 -1 2

>> 3\*A % jeder Eintrag im Zeilenvektor mit Faktor 3 multipli- ziert  
ans =

3 -3 6

>>

>>

Vervielfachen von jedem Eintrag um einen bestimmten Faktor.

>> C=A.^2 % Einträge der Matrix werden quadriert

C =

1 1 4

>>

Quadrieren von jedem Eintrag des Zeilenvektors.

>> A  
A =

1 -1 2

>> B  
B =

2 1 -1

>> A.\*B % elementweise Multiplikation

ans =

2 -1 -2

>> A.\B % elementweise Division

ans =

2.0000 -1.0000 -0.5000

>> A.^B % elementweise Potenzierung

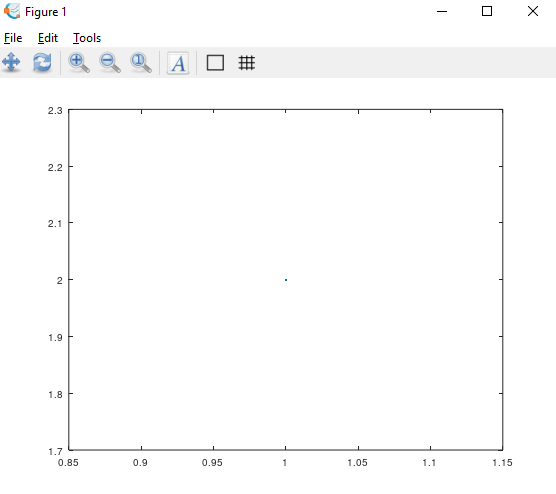
ans =

1.0000 -1.0000 0.5000

>>

Elementweises Rechnen.

**Darstellen von Daten: Diagramme**



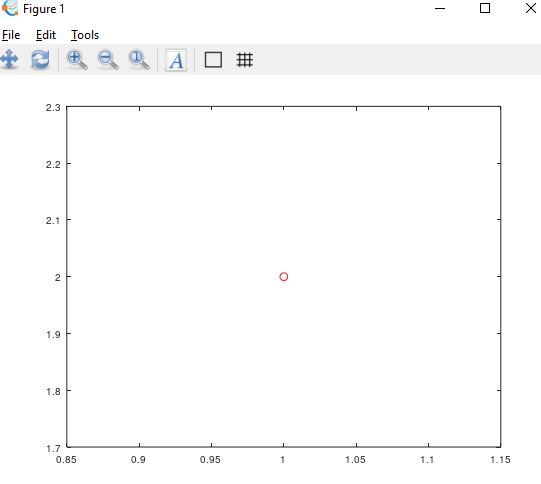
>> x=1  
x = 1

>> y=2;

>> plot(x,y)

>>

Darstellung von Punkten.



>> x=1  
x = 1

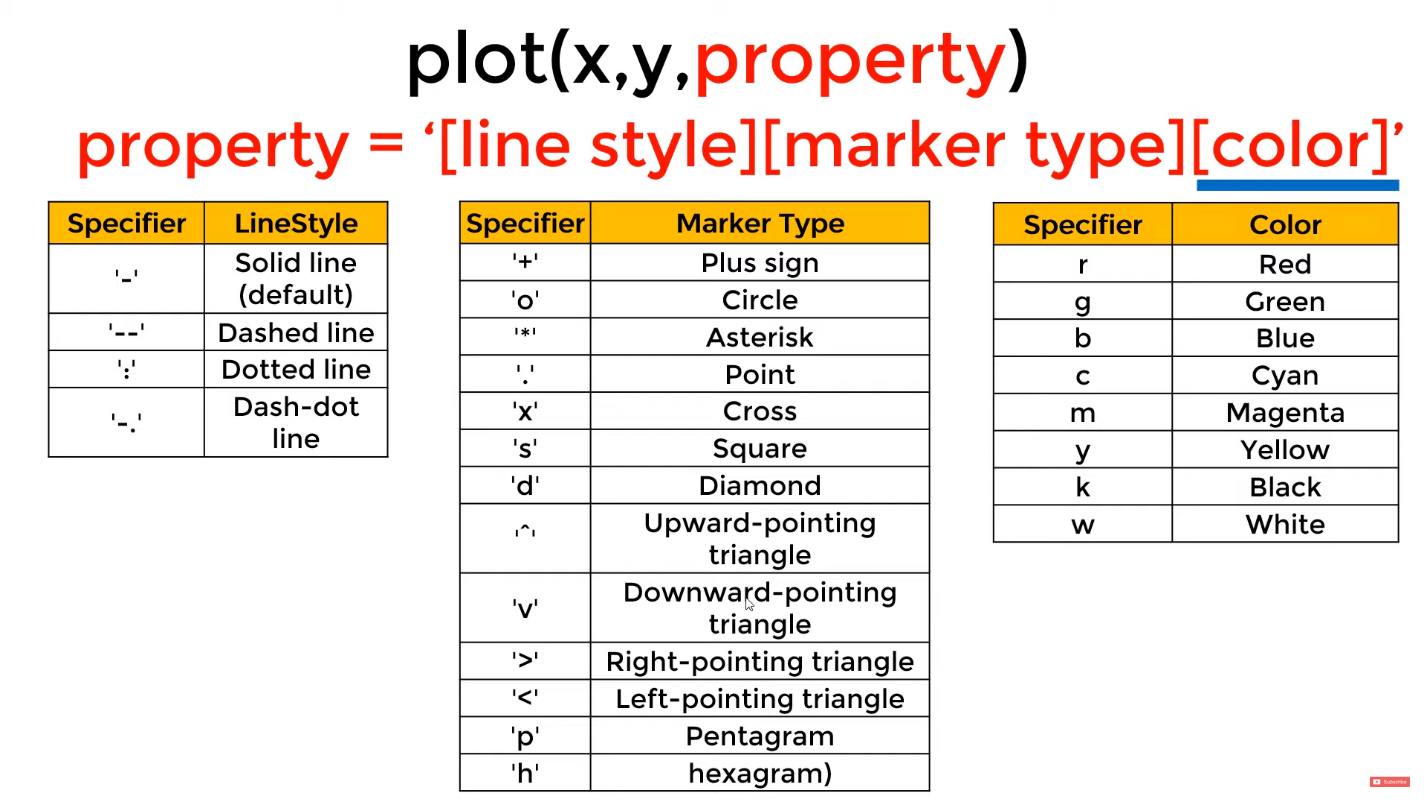
>> y=2;

>> plot(x,y,‘or‘) % Punkte (x,y) als Kreise in rot

>>

>>

Eigenschaften von Punkten.



>> X=[1 3 6];

>> Y=[2 4 1];

>> plot(X,Y,'or','MarkerSize',12) % Punkte (x,y) als Kreise in rot mit der Größe 12 in einem Diagramm darstellen

>> grid on % Gitternetzlinien anzeigen

>> title('Graph: y(x)') % Diagrammtitel hinzufügen

>> set(gca,'fontsize',24) % Schrfitgröße auf 24

>> xlabel('Varialbe x') % Beschriftung x-Achse

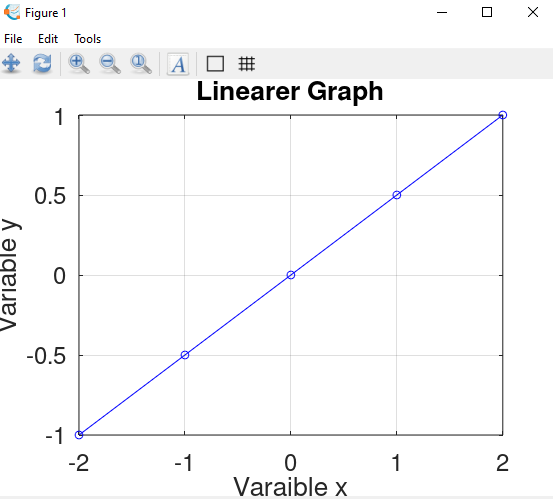
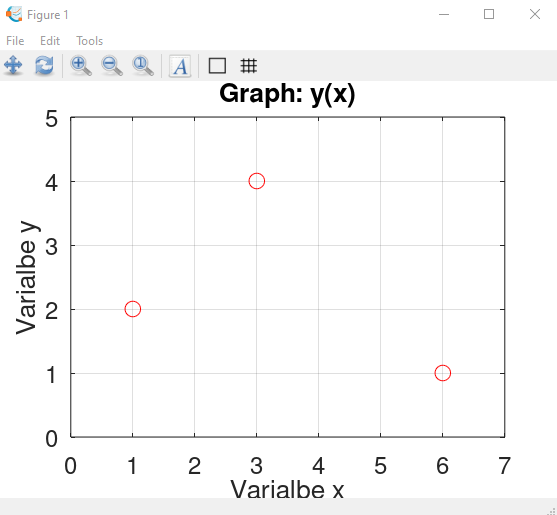
>> ylabel('Varialbe y') % Beschriftung y-Achse

>> axis([0 7 0 5]) % x-Achse-Bereich: 0 bis 7  
 y-Achse-Bereich: 0 bis 5

>>

>>

Diagrammlayout bearbeiten.



>> X1=linspace(-2,2,5)

X1 =

-2 -1 0 1 2

>> Y1=0.5\*X1

Y1 =

-1.0000 -0.5000 0 0.5000 1.0000

>> plot(X1,Y1,'bo-')

>> grid on

>> title('Linearer Graph')

>> xlabel('Varaible x')

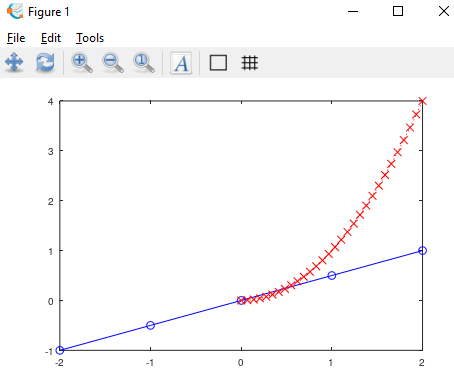
>> ylabel('Variable y')

>> set(gca, 'Fontsize',24)

>>

>>

Einen linear Graph darstellen.



>> X2=linspace(0,2,30);

>> Y1=0.5\*X1;

>> Y2=X2.^2;

>> plot(X1,Y1,'bo-')

>> hold on

>> plot(X2,Y2,'rx:')>>

Quadratischen Graph hinzufügen durch den Befehl „hold on“.

>> X2=linspace(0,2,30);

>> Y2=X2.^2;

>> plot(X1,Y1,'bo-',X2,Y2,'rx:')

>>

Quadratischen Graph hinzufügen.

>> X1=linspace(-2\*pi,2\*pi);

>> X2=linspace(-pi,2\*pi);

>> Y1=cos(X1); % Kosinus von X1

>> Y2=cos(X2).^2; % jeder Eintrag von cos(X2) wird quadriert

>> plot(X1,Y1,'b-',X2,Y2,'r:')

>> title('Kosinus und Kosinus^2')

>> set(gca,'Fontsize',24)

>> set(gca,'Fontsize',24,'XTick',[-2\*pi -pi 0 pi 2\*pi])  
 % Werte, die auf der x-Achse angezeigt werden

>> set(gca,'Fontsize',24,'XTicklabel',{'-2\pi','-\pi','0',  
'\pi','2\pi'}) % Beschriftung der Werte, die auf der x-Achse angezeigt werden

>> axis([-2\*pi 2\*pi -1.1 1.1])

>>

Trigonometrische Funktionen.

