Grundlagenlabor Nachrichtentechnik

Dr.-Ing. Dirk Wübben
Institute for Telecommunications and High-Frequency Techniques
Department of Communications Engineering
Room: N2300, Phone: 0421/218-62385
wuebben@ant.uni-bremen.de

www.ant.uni-bremen.de/courses/glab/

Überblick

- Organisation
  - Termine
  - Überblick zum Ablauf des Labors
  - Einteilung der Gruppen

- Einführung in Matlab (Octave)

Organisation

4 statt bisher 6 Versuche

- Durchführung des Praktikums an 4 Terminen (A, B, C, D)
  - Das Labor findet im Zweiwochen-Rhythmus Donnerstags und Freitags per Zoom statt. Zeiten sind auf den Punkt zu verstehen (s.t.), kein Zeitpuffer

Gruppe	Tag	1	2	3	4
A	Do, 1. Woche	14:00	15:00	16:00	17:00
B	Fr, 1. Woche	8:00	9:00	10:00	11:00
C	Do, 2. Woche	14:00	15:00	16:00	17:00
D	Fr, 2. Woche	8:00	9:00	10:00	11:00

  - Bei Krankheit ist schnellstmöglich der entsprechende Betreuer zu kontaktieren
- 4 Versuche (aktuelle Planung – ggf. Anpassung um synchron mit Vorlesung)

Gruppe	Betreuer	Termine
Signale in Matlab	Shengdi Wang	26.11., 27.11.,3.12., 4.12.
Lineare Systeme	Tobias Monsees	10.12., 11.12., 17.12., 18.12.
Zeitdiskrete Signale und Systeme	Edgar Beck	21.1., 22.1., 28.1., 29.1.
Codierung	Matthias Hummert	4.2., 5.2., 11.2.,12.2.

Organisation

- Webpage: www.ant.uni-bremen.de/courses/glab/
  - Aktuelle Informationen, Termine, Einteilung der Gruppen, …
  - Versuchsbeschreibung (PDF) mit
    - Beschreibung zu allen Versuchen
    - Vorbereitungsaufgaben
    - Beschreibung zur Durchführung der Labore
    - Matlab Kurzreferenz
  - Zip-File mit allen benötigten Matlab-Routinen
  - Tutorials, Skripte zu Matlab

„Literatur“

- Per Stud.IP werden Informationen zur Vorlesung und zu dem Labor mitgeteilt

Philosophie des Praktikums

- Begleitend zur Vorlesung Grundlagen Nachrichtentechnik
- Es sollen schrittweise die Grundlagen zur analogen Datenübertragung (mit Ausblick auf die digitale Übertragung) erarbeitet werden. Hier steht die **selbständige Problemlösung** im Vordergrund.
- Hierzu bearbeiten die Gruppenmitglieder vorab die Laboraufgaben und präsentieren dann in dem Labortermin mit dem Betreuer die Aufgaben
  - Vorbereitungsaufgaben und Aufgaben bearbeiten
  - Programmierung vorab im Team durchführen
  - Skript mit allen Programmieraufgaben um nacheinander zügig den Versuch live durchzuführen (Kommentare zur Strukturierung der Teilaufgaben sinnvoll)
  - Beantwortung der Aufgaben vorbereiten – Finalisierung während des Labors
  - Nach Abschluss des Versuchs Programme & Antworten an Betreuer senden
- **Sehr gute Vorbereitung jedes Teilnehmers** unbedingt erforderlich!
  → Vorlesungsinhalte verstehen & Aufgaben lösen

„Vorbereitungsaufgaben“ und „Aufgaben“ sind gleich zu behandeln

Nützliche Hinweise

- Die Versuche bauen teilweise aufeinander auf → die Ergebnisse (bzw. Matlab-Files) vorheriger Versuche sind auch für die nachfolgenden Versuche wichtig!

- Folgt den Vorschlägen zur Programmstruktur und zur Bezeichnung von Variablen
  - Programme werden „zukunftssicher“ (also für die nächsten Versuche)
  - Programme sind für euch und uns lesbar (zum Debuggen nicht unwichtig)

- Verfolgt die Vorlesung & Übung

- Aufgaben sind vor dem Versuch zu bearbeiten
  - Es reicht sicherlich nicht, sie einen Tag vorher kurz anzuschauen
  - Bei Problemen steht der jeweilige Betreuer auch vor dem Versuch für Fragen zur Verfügung

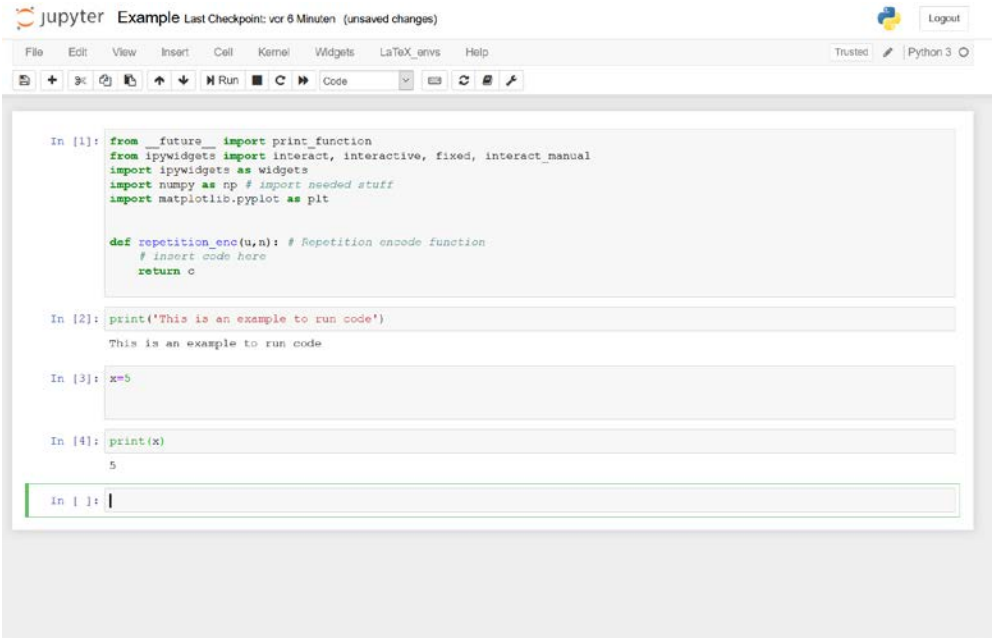
Software

- Mathworks (https://de.mathworks.com)
  - MATLAB and Simulink Student Suite für 69€
  - MATLAB Student (unbundeled) für 35€
- Open Source Alternative „Octave“ (https://www.gnu.org/software/octave/)
- JupyterHub
  - Octave per JupyterHub des ANT möglich
  - Keine Installation, allerdings Performanz limitiert

jupyterhub

- https://www.ant.uni-bremen.de/jupyterhub/
- Juypter Hub is used for programming Exercises
  - Jupyter Hub gives you access to computational environments and resources without installation and maintenance tasks
  - Available online, running on our servers with Octave
  - No local installation of Octave/Matlab needed – Code runs online on our servers

Email an Matthias Hummert falls jemand Zugang benötigt!

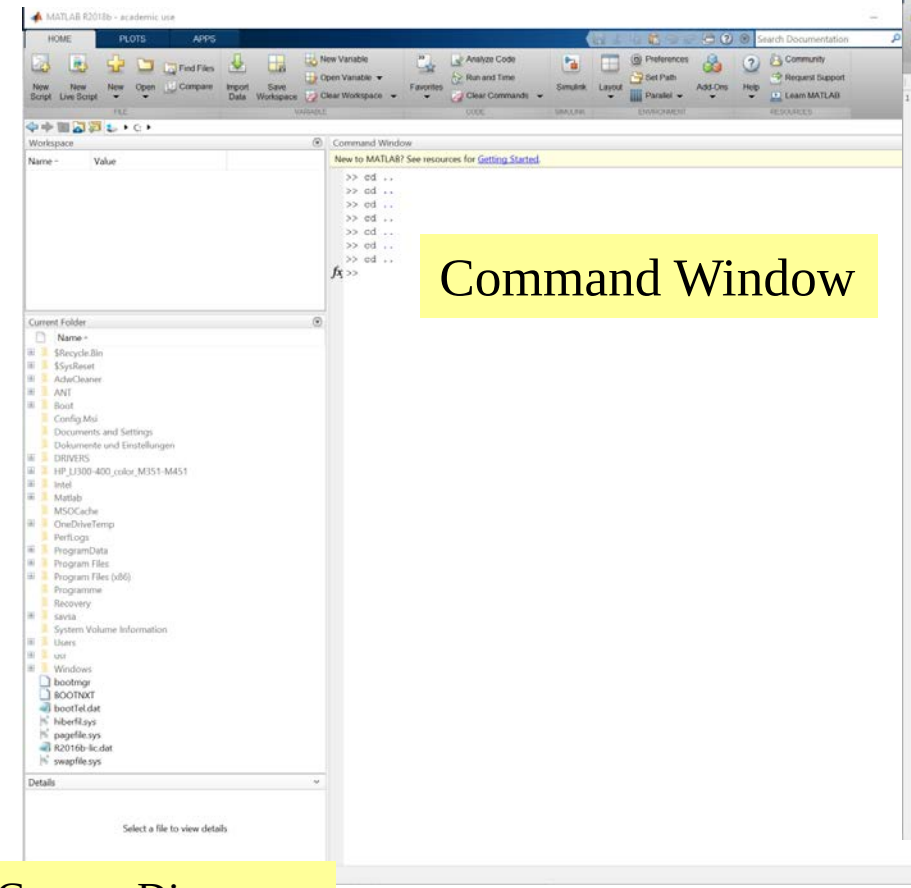


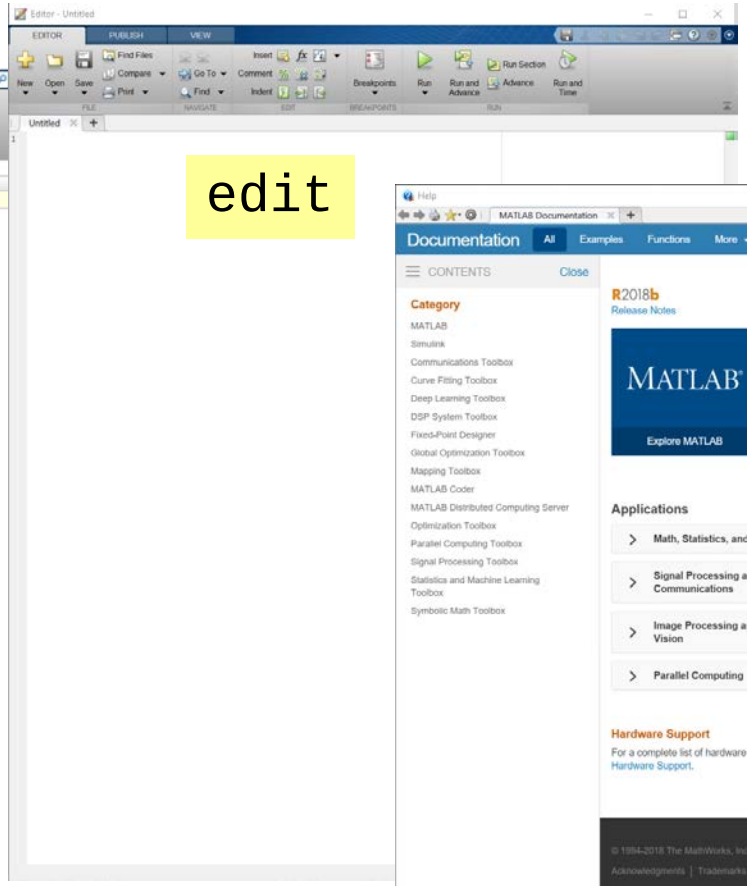
Kurzeinführung in Matlab / Octave

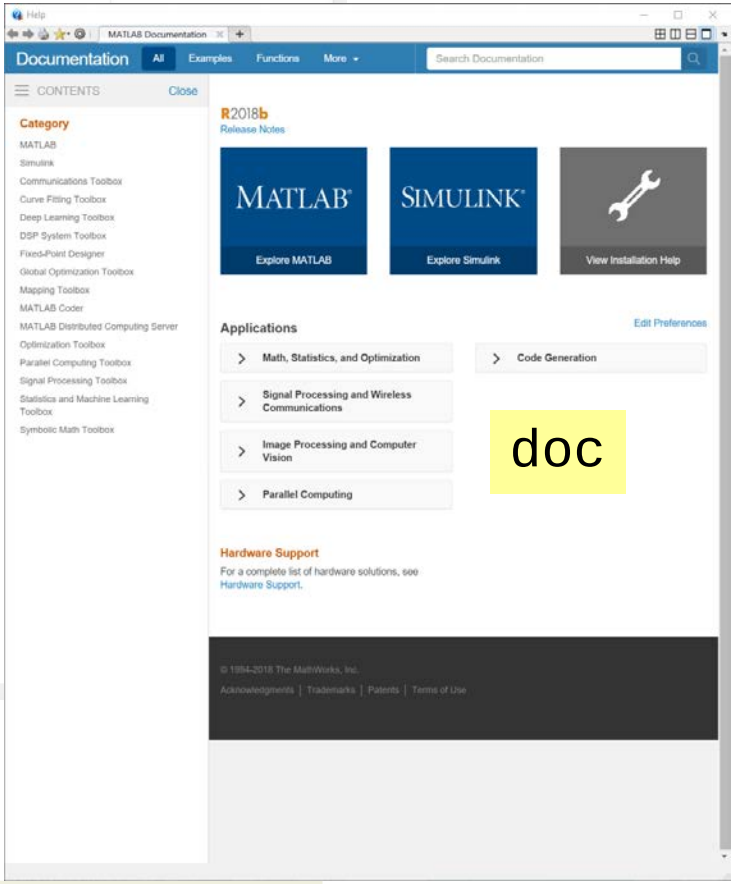
Matlab

- Was ist Matlab?
  - Das Programm Matlab (MATrix-LABoratory) ist ein vielseitiges und doch einfach zu erlernendes Werkzeug zur Erstellung mathematischer Berechnungen
  - Ausführliche Informationen unter www.mathworks.de
- Vielzahl an ausführlichen und detaillierten Einführungen im Internet
- Zahlreiche Ebooks / Bücher in der SUUB, z.B.
  - Attaway: MATLAB - A Practical Introduction to Programming and Problem Solving
  - Hahn, Valentine: Essential MATLAB for Engineers and Scientists
  - Hunt et al.: A guide to MATLAB for beginners and experienced users
- Kleine Auswahl unter www.ant.uni-bremen.de/courses/glab/
  - Thomas Schubiger: Ausführliches Skript der ETH Zürich
  - Peter Arbenz: Einführung in MATLAB, (Matlab-Kurs ETH Zürich)
  - Günter M. Gramlich: Eine Einführung in MATLAB, (Skript Hochschule Ulm)
  - Harald Loose: Einführung in MATLAB, (Folien-Vortrag)
  - Susanne Teschl: MATLAB - Eine Einführung, (Skript)
  - Kermit Sigmon: MATLAB Primer 3rd ed, (Standardwerk, 39 Seiten)
  - Mark S. Gockenbach: Practical Introduction to Matlab, (Skript, 33 Seiten)
  - D.J. Higham und N.J. Higham: MATLAB Guide, (Buch, 302 Seiten)
  - Edward Neuman: MATLAB Tutorials
  - Cleve Moler: Numerical Computing with MATLAB (Buch in PDF-Format)
  - Mathworks Help-Desk (alle Benutzerhandbücher im PDF-Format)

Matlab starten







Anweisungen

- Anweisungen
  - Kommandos, Datenzuweisungen, Funktionen, mathematische / logische Ausdrücke, … werden nach dem Prompt eingegeben **>>**
  - Ausführung nach **Enter**
  - Ergebnis wird als **ans** im **Workspace** gesichert und im **Command Window** dargestellt.

```
>>   3.45
ans =
  3.4500
```

- Ausdruck `sqrt(1.44)` verwendet MATLAB Funktion **sqrt**

```
>>   sqrt(1.44)
ans =
  1.2000
```

- Vorherige im **Command Window** angegebene Anweisungen werden gespeichert → durch ↑anzeigen

Anweisungen

- Daten können Namen zugewiesen werden → Variablen
- Mehrere durch Komma getrennte Anweisungen können nach einem Prompt eingegeben werden
- Variablen-Namen sind Case-sensitive → **mb** und **Mb** sind zwei Variablen

```
>>   a=16,  b=sqrt(a)
a =  16
b =  4
```

- Sind alle Anweisungen durch Semikolon (;) abgeschlossen, so werden diese zwar ausgeführt aber das Ergebnis nicht im **Command Window** dargestellt. Diese Werte lassen sich nachfolgend durch Angabe des Namens am Prompt ausgeben

```
>>   c=25;  d=sqrt(b)+2.5;
>>   ans,  a,  b,  c,  d
ans =      1.2000
a =  16
b =  4
c =  25
d =  4.5000
```

Belegter Speicher & Hilfe

- Zur Feststellung der gespeicherten Variablen und ihrer Größe dient der Befehl **whos**.

```
>>   whos
 Name        Size            Bytes  Class
  a           1x1                8  double array
  ans         1x1                8  double array
  b           1x1                8  double array
  c           1x1                8  double array
  d           1x1                8  double array

Grand total is 5 elements using 40 bytes
```

- Matlab stellt ***mehrere*** ausführliche Hilfen an

```
help sin
doc sin
```

Komplexe Zahlen

- Komplexe Zahlen bestehen aus Real- und Imaginärteil. Zur Kennzeichnung des Imaginärteils können **`i`** oder **`j`** verwendet werden. (Ausgabe immer mit **`i`**).
- Der Realteil, der Imaginärteil, die Amplitude und der Winkel in rad einer komplexen Zahl lassen sich durch die Funktionen **`real`**, **`imag`**, **`abs`** und **`angle`** bestimmen.

```
>> a=3 - 4j, b=real(a), c=imag(a), d=abs(a),  e=angle(a)
a =  3.0000 - 4.0000i
b =  3
c = -4
d =  5
e = -0.9273
```

- Eingabe einer komplexen Zahl durch **`a=3-j*4`** mit Multiplikation **`*`**
  → notwendig, wenn Imaginärteil durch Funktion generiert wird.

```
>> f=4; g=9; h=sqrt(f)+j*sqrt(g)
h =  2.0000+3.0000i
```

- Falls **`j`** durch Wert überschrieben wurde (z.B. Zähler), so kann durch **`j = sqrt(-1)`** die Definition als Imaginär-Einheit wiederhergestellt werden

Matrizen

- **Alle numerischen Werte werden als Matrix gespeichert**! Skala → 1x1 Matrix
- Die Eingabe einer n x m Matrix (n Zeilen, m Spalten) erfolgt durch zeilenweise Eingabe der einzelnen Elemente innerhalb von eckigen Klammern. Zur Trennung zwischen Zeilen dient das Semikolon (**;**)

```
>>   a=[3  4; 2  1]
a =
  3   4
  2   1
>> b=[1.5  -2.4  3.5  0.7;  -6.2 3.1  -5.5  4.1; 1.1  2.2  -0.1  0]
b =
  1.5000  -2.4000   3.5000   0.7000
 -6.2000   3.1000  -5.5000   4.1000
  1.1000   2.2000  -0.1000        0
```

- Einzelne Matrixelemente oder eine Teilmatrix lassen sich durch Angabe der entsprechenden Zeilen und Spalten in zwei einzeilige Matrizen erhalten.

```
>>   e=b(2,  3),  f=b([2  3], [1  3]),  g=b(2, [3  4])
e = -5.5000
f = -6.2000   -5.5000
     1.1000   -0.1000
g = -5.5000    4.1000
```

Matrizen

- Eine Matrix kann aus mehreren Teilmatrizen erstellt werden:

```
>>   h=[1  2  3],  k=[4;  7],  m=[5  6;  8  9]
h =   1   2   3
k =
  4
  7
m =
  5   6
  8   9
>>   n=[h;  k  m]
n =
  1   2   3
  4   5   6
  7   8   9
```

- Alle Teilmatrizen in einer Zeile müssen die gleiche Anzahl an Zielen aufweisen. Ebenso muss die Summe an Spalten aller Teilmatrizen in allen Zeilen gleich sein.

- Ansonsten **Fehlermeldung**:

```
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must have the
 same number of rows.
```

Vektoren

- Ein Zeilenvektor ist eine Matrix mit einer Zeile und ein Spaltenvektor eine Matrix mit einer Spalte
  → Eingabe wie bei Matrizen

```
>>   a=[3  5  9],  b=[3;  5;  9]
a = 3   5   9
b =
  3
  5
  9
```

- Doppelpunkt (`:`) zur Erstellung eines Zeilenvektors mit Elementen, die den gleichen Abstand aufweisen

```
>>   c=2:5,  d=3:2:9
c = 2   3   4   5
d = 3   5   7   9
```

Start:delta:End → Benachbarte Element weisen Differenz delta auf
Start:End → Hochzählen von Start bis End in Einerschritten
→ Start:1:End

- Vektoren mit Doppelpunkt-Notation sind nützlich um eine mathematische Funktion über ein Intervall an Werten auszuwerten.

  Beispiel: Die Funktion $y(x) = x^{1/2}$ soll über das Intervall *0.5 … 2.0* an Vielfachen von *0.25* ausgewertet werden

```
>>   x=0.5:0.25:2.0; >>   y=sqrt(x);
>>   x,  y
x =   0.5000   0.7500   1.0000   1.2500   1.5000   1.7500   2.0000
y =   0.7071   0.8660   1.0000   1.1180   1.2247   1.3229   1.4142
```

Indexierung von Elementen

- Indexierung von einzelnen Vektorelementen

```
>>   f=[10  5  4  7  9  0],  g=[2  5  6];  h=f(g)
f = 10   5   4   7   9   0
h =  5   9   0
```

- Beispiel: Auswahl jedes dritten Wertes der Variablen `x` and dem zugehörigen Funktionswert `y = sqrt(x)`:

```
>>   k=1:3:7;  x1=x(k),  y1=y(k)
x1 =  0.5000   1.2500   2.0000
y1 =  0.7071   1.1180   1.4142
```

- Definition von Teilmatrizen mit Hilfe der Dopplepunkt-Notation

```
m =
  1.5000   -2.4000    3.5000   0.7000
 -6.2000    3.1000   -5.5000   4.1000
  1.1000    2.2000   -0.1000        0
>>   n=m(1:2,2:4), o=m(:, 1:2), p=m(2, :)
n = -2.4000    3.5000   0.7000
     3.1000   -5.5000   4.1000
o =  1.5000   -2.4000
    -6.2000    3.1000
    1.1000    2.2000
p = -6.2000   3.1000   -5.5000    4.1000
```

Beachte: Ein einzelner Doppelpunkt referenziert alle Zeilen bzw. alle Spalten

Arrays und spezielle Zahlen

- Arrays: Verallgemeinerung von Matrizen auf mehr als zwei Dimensionen

```
>> A = [1 2; 3 4]; A(1,1,2) = 5
A(:,:,1) =
     1     2
     3     4
A(:,:,2) =
     5     0
     0     0
```

- Spezielle Zahlen: In Matlab sind spezielle Zahlen wie pi, Inf, NaN, eps definiert

```
>> pi, Inf, NaN, eps
ans =    3.1416
ans =   Inf
ans =   NaN
ans  =  2.2204e-016
```

- Operationen wie 1/0 oder sin(Inf) mit nichtdefiniertem Ergebnis führen zur Ausgaben **NaN**, (*Not a Number)*

```
>> 1/0
Warning: Divide by zero.
ans =   NaN
```

Arithmetische Operationen

- Matrix Addition, Subtraktion, Multiplikation, Potenzieren, und Transponieren mit den Operatoren **+**, **-**, *****, **^** und **'**
- Zusätzlich spezifizieren **b/c** und **c\b** die Multiplikation der Matrix **b** mit der Inversen der quadratischen Matrix **c** von rechts bzw. von links
- Die Dimensionen der Matrizen müssen stimmen

```
>>   a=[1  2;  3   4];  b=[3  1;  7  8];  c=[2  4];
>>   d=a+b,  e=c*a,  f=a^2,  g=c'
d =    4     3
      10    12
e =   14    20
f =          7   10
      15    22
g =    2
       4

>>   h=a\b,  k=b\a
h =   1.0000   6.0000
      1.0000  -2.5000
k =   -4.5000   2.5000
      -2.0000   3.0000
```

Elementweise Operation

- Element-bei-Element Operation von Arrays
  - Array-Addition und Subtraktion entspricht Elementweiser Matrix-Addition bzw. Subtraktion
  - **Punkt-Operator** zur Elementweisen Multiplikation, Division, Potenzierung von Arrays: **.***, **./**, **.'** und **.^**
  - Beachte richtige Dimensionierung

```
>>   m=a.*b,  n=b./a,  o=b.^a
m =   3    2
     21   32
n =   3.0000   0.5000
      2.3333   2.0000
o =     3      1
      343   4096
```

- Achtung: Bei Multiplikation, Division, Addition, Subtraktion mit einer Konstanten wird die Dimensionierung nicht beachtet → erfolgt elementweise

```
>>   p=c+2,  q=c - 2,  r=2.*c,  s=c./2,  t=2*c,  u=c/2
p =  4   6
q =  0   2
r =  4   8
s =  1   2
t =  4   8
u =  1   2
```

Ergebnisse sind gleich

Logische Ausdrücke

- Die logischen Operationen „und“, „oder”, and „nicht“ sind durch **&**, **|** und **~** gegeben. Sie lassen sich in Verbindung mit den folgenden Vergleichsoperatoren

<	kleiner als	**= =**	gleich
<=	kleiner als oder gleich	**~ =**	nicht gleich
>	größer als		
> =	größer als oder gleich		

zur Erstellung von Matrizen bestehend aus Nullen (Falsch) und Einsen (Wahr) verwenden.

```
>>   a=[1  3  2;  4  6  5],  b=a>2&a<=5
a =  1   3   2
     4   6   5
b =  0   1   0
     1   0   1
>>   c=[1  5  3  4  7  8],  d=c>4
c =  1   5   3   4   7   8
d =  0   1   0   0   1   1
```

Mathematische Funktionen

- Die folgenden in Matlab implementierten Funktionen werden einzeln auf die Elemente eines Arrays angewendet und liefern ein Array gleicher Dimension

`sqrt`	Quadratwurzel
`real`	Realteil einer komplexen Zahl
`imag`	Imaginärteil einer komplexen Zahl
`abs`	Absolutbetrag einer komplexen Zahl bzw. einer reellen Zahl
`angle`	Winkel einer komplexen Zahl (in rad)

- Weitere Funktionen

`exp`	Exponent zur Basis e
`log`, `log10`	Logarithmus zur Basis e, 10, !
`sin`, `cos`	Sinus, Kosinus
`tan`, `cot`	Tangens, Cotangens
`asin`, `acos`	Arkussinus, Arkuskosinus
`atan`	Arkustangens
`round`	Runden zum nächsten Integer-Wert
`floor`	Runden Richtung -∞
`ceil`	Runden Richtung ∞

- Alle Trigonometrischen Funktionen beziehen sich auf Winkel in rad!

Ablaufsteuerung

- `for`-Schleife: Mehrfache Ausführung der gleichen Anweisungen
- Beispiel: Summe der Quadrate aller ungeraden Zahlen von 1 bis 9

```
x=0;
for k=1:2:9;
   x=x+k^2;
end;
```

- `while` Anweisung: Im Gegensatz zur for-Schleife ist Anzahl an Durchläufen nicht notwendig vorab bekannt.
  Die Ausführung stoppt sobald der logische Ausdruck erfüllt ist
- Beispiel: Bestimme größtes Vielfaches von 2 kleiner als 5000:

```
>>   n=1;
>>   while 2*n<5000;   n=2*n;   end;
>>   n
n =  4096
```

If-Anweisung

- Bei der `if` Anweisung werden je nach Ergebnis des logischen Ausdrucks die Anweisungen ausgeführt

```
for k=1:4;
   if k==1;
     x(k)=3*k;
   elseif k==2|k==4;
     x(k)=k/2;
   else;
     x(k)=2*k;
   end;
 end;
```

```
x =   3   1   6   2
```

- Es können String-Variablen wie 't' und 'f' in mit den logischen Ausdrücken == und ~= verwendet werden

```
c='t';   n=2;
if c=='f'; c='false';  y=NaN; end;
d=0.1:0.1:0.4;

if c=='t';
   if n==2;
     y=10*d(n);
   else;
     y=0
   end;
end;
```

```
c = 't' n=2;    → c=t      y =  2
c = 't' n~=2    → c=t      y = 0
c = 'f' n~=2    → c=false  y = NaN
```

Numerische Funktionen

- `find(A)` → Zeilenvektor mit Indizes der Nicht-Nullelemente eines eindimension. Arrays

```
a=[1  0  2  3  0  4];   b=find(a)        → b =  1   3   4   6
n=find(a>2)                              → n =  4   6
```

- `size(A,i)` Bestimmt die Anzahl an Zeilen in `A` wenn `i=1` oder die Anzahl an Spalten wenn `i=2`. Ohne Parameter `i` werden Anzahl Zeilen und Spalten bestimm.
- `zeros(m,n)` Erstellt ein (*m* x *n*) Array mit Nullen. Beispiel `zeros(size(A))`
- `max(A)` Liefert maximalen Wert eines eindimensionalen Arrays. `max(max(A))` liefert maximalen Wert eines zweidimensionalen Arrays.
- `min(A)` Liefert entsprechen den kleinsten Werte
- `mean(A)` Liefert den Mittelwert eines eindimensionalen Arrays
- `sum(A)` Liefert die Summe der Element von `A` wenn `A` ein eindimensionales Array ist. Bei zweidimensionaler Matrix wird Zeilenvektor mit Spaltensumme erstellt

```
>>   e=[1  2  3];  es=sum(e)          →    es =  6
>>   f=[4;  5;  6],  fs=sum(f)        →    fs =  15
>>   g=[1  2  3;  4  5  6],  gs=sum(g)
g =  1   2   3
     4   5   6
gs = 5   7   9
```

Plotting-Funktionen

- **`plot(x,y)`**: Darstellung der Variablen **`y`** über **`x`** , wobei die Datenwerte verbunden werden.
  Vielzahl an Optionen: **`help plot`**
- **`bar`**: Balkendiagramm
- **`xlabel('text')`**: Benennt x-Achse mit dem Text **`'text'`**
- **`ylabel('text')`**: Benennt y-Achse mit dem Text **`'text'`**
- **`text(x,y,'text')`**: Fügt den Text **`'text'`** zu der Grafik an der Position **`(x, y)`**, wobei **`x`** und **`y`** die horizontale und die vertikale Achse bezeichnen
- **`figure`**: Öffnet eine neues **Figure Window**.
- **`subplot(a,b,c)`** : Unterteilt aktuelles Figure in **`a`** Zeilen und **`b`** Spalten und greift auf das **`c`**-te Unterfenster zu
- Viele weitere Plot-Möglichkeiten: **`Help-Windows/Matlab/Graphics`**

M-Files

- Generell: Im Matlab-Verzeichnis (oder in den Toolbox-Verzeichnissen) gespeicherte Dateien mit der Endung **.m** sind ausführbare Dateien und werden m-Files genannt. Es ist zwischen *Script* und *Function* zu unterscheiden.

- Script-Files: Anstatt Befehle nacheinander im **Command Window** einzugeben und mit Enter auszuführen lassen sich diese Befehle auch in einem Script zusammenfassen, so dass auch Änderungen leichter vorzunehmen sind. Zum Ausführen der Befehle wird das m-File aufgerufen.

- Function-Files: Funktion mit definierten Eingabe- und Ausgabevariablen. Innerhalb der Funktion sind die Variablen lokal (erscheinen nicht im Workspace). Erste Zeile des M-Files enthält Funktionsname und Eingangs- und Ausgangsvariablen.
  Beispiel: `function [z,w] = abcd(x,y)`

Aber!

- Hauptbestandteil des Vorlesung Grundlagen der Nachrichtentechnik bilden **analoge** Signale, Systeme und Modulationsformen, …
- Die Darstellung auf dem PC erfolgt natürlich **digital und zeitdiskret**
- Interpretation der digitalen Signale als analoge Signale
- Beispiel: $s(t) = \sin(2\pi \cdot t)$

```
t = 0:1/20:2-1/20;
s = sin(2*pi*t);
plot(t,s,'.')
hold on
plot(t,s,'g')
hold off
```

„Hausaufgaben“

- Labor-Skript, Matlab-Files von www.ant.uni-bremen.de/courses/glab/ herunterladen und Skript ausdrucken
- Matlab-Einführung des Vertrauens herunterladen (ausdrucken)

- Matlab / Octave
  - Installieren und Spielen
  - Matlab-Einführung praktisch bearbeiten

- Versuchsbeschreibung für den ersten Versuch durchlesen, Vorbereitungsaufgaben bearbeiten