Scientific Computing and Matlab

- Matlab is a tool and programming environment used in computational science but also in other areas.

- A simple environment to quickly test ideas and to study the results

- Also available in other tools, such as Freemat, Maple, Mathematica, Comsol Multiphysics, and other programming languages like C/C++, Fortran, Java.
What is MATLAB?

- Developed by MathWorks, Inc. 
  [http://www.mathworks.com](http://www.mathworks.com)
- First version already in late 70s
- Originally MATrix LABoratory.
- Mathematical environment for
  - Numerical computations
  - Graphics
  - Programming
- Many powerful predefined functions with the ability to define your own.
- Own object oriented programming language. Collaboration with C++, Java and Fortran possible.

What is MATLAB?

- Over 25 plug-in (toolboxes) are available for special applications, eg:
  - Signal analysis
  - Image analysis
  - Statistics
  - Symbolic mathematics
  - Financial mathematics
  - Parallel computing
  - ...
- Run under UNIX/Linux, Windows or Macintosh.
Application areas

Wherever there are computational problems

- **Education**: mathematics (specifically linear algebra), computational science, physics, chemistry, engineering, economics, etc.

- **Research**: lab environment to test the solution methods, study, analyze problems, perform calculations and visualize the results.

- **Industry**: used in the same manner as in research, for example on development departments in biotechnology, electronics, automotive, etc.

MATLAB environment

- The development environment (MATLAB desktop) has a plurality of windows, *Command window*, *Current folder*, *Workspace* and *Command History*.

**Task**: Start Matlab on your computer and change to a preferred folder by clicking it in *Current folder* window.
MATLAB environment

- MATLAB is usually controlled from the command window (Command Window), but also from the menus.
- Commands are given by »-prompter and executed when return key is pressed.

Example:

```
>> 42 + 19
ans =
61
```

- Exit matlab:

```
>> exit
```

- In the command window, you can work interactively as an advanced calculator.

> The semicolon suppresses printing.

If no variable names specified, the variable is ans (= answer)

**Task:** Compute the sum and product of a,b,c and store the results in new variables. Note the use of semicolon.

```
>> a = 75
a =
75
>> b = 34;
>> c = a*b
c =
2550
>> a+b
ans =
109
```
Help is available through the Help menu
But even writing demo in the command window provides information on how to use Matlab.

Task: How do you perform square root in Matlab?

MATLAB environment

Help for individual commands can be done directly in the command window

```
>> help command
```

Ex)
What does the command exit?

```
>> help exit
EXIT   Exit from MATLAB.
EXIT terminates MATLAB.
```
Variables in MATLAB

A variable in MATLAB
- can be viewed as containers of a value of a certain type (integer, float, char, ...)
- always have a name. Must begin with a letter. Do not use å, ä, ö, space, hyphen (minus), plus signs etc in the name
- can be assigned a value
- created when needed, without special "declaration" (is of type "matrix").
- Can be predefined, e.g. \texttt{pi,eps,realmax}

\begin{verbatim}
>> a = 3
a =
 3
>> pi
ans =
 3.1416
\end{verbatim}

The variables are displayed in \textit{Workspace}-window

The issued commands are displayed in \textit{Command History}
Variables in MATLAB

- Can also list the variables in the command window using `who`, `whos`

```matlab
>> who
Your variables are:
a    ans
>> whos
Name      Size  Bytes  Class
a         1x1     8  double array
ans       1x1     8  double array
Grand total is 2 elements using 16 bytes
```

- Variables can be displayed in various formats

```matlab
>> y = sin(2*pi/3);
>> y
y =
  0.8660
>> format long; y
y =
  0.86602540378444
>> format long e; y
y =
  8.660254037844387e-001
>> format short e; y
y =
  8.6603e-001
% Standard format
e-001 equals to 10\(^{-1}\)
```

Task: Display `pi` in different formats (note the accuracy is not changed only the format).
Variables in MATLAB

- Variables can be saved to file and loaded up on another MATLAB session later.
- In the menu use the buttons "Save" and "Load".
- Variables will be saved as so-called MAT-files, files with the extension .mat.
- MAT-files are called binary and is not readable or editable.
- Can also use `save` and `load` command directly in the command window.

Save and load variables

- The button "Import Data" can be used to load other formats such as spreadsheets (Excel), audio files, video files, etc.
- Select a file format in the window that opens after clicking the button.
Functions

- Elementary mathematics
- Linear algebra
- Graphics in 2D and 3D
- Integrals and differential equations
- Statistics
- Curve fitting

There are lots of predefined functions, such as for

\[
\text{abs}(x), \sqrt{x}, \sin(x), \log(x), \log10(x), \ldots
\]

(\text{Help} -> \text{Documentation} -> \text{Matlab} -> \text{Mathematics})

There are lot of Toolboxes with specilized functions. You can also create your own functions.

Matrices

- **Matrix** is the basic data type.
- A two-dimensional matrix is a table with **rows** and **columns**.
- A matrix with \( m \) rows and \( n \) columns, has the size \( mxn \) (\( m \) times \( n \)).
- **Vectors** are special cases of the matrices, then the number of rows or number of columns is 1. Example: **row vector**, \( 1 \times n \), and **column vector**, \( m \times 1 \).
- A matrix of size \( 1 \times 1 \) called scalar.
- Each value in the matrix are called **elements**.
Vectors

Column vector and row vector in MATLAB

```
>> vcol = [ 1; 2; 3; 4; 5 ]
>> vrow = [5 6 7 8 ], x = 3
vcol = % Column vector
    1
    2
    3
    4
    5
vrow = % Row vector
    5 6 7 8
x = % Scalar
    3
```

The semicolon into vectors provide new row.

Vectors

Change individual elements

```
>> vrad(1) = -1.1; vrad(2) = 3.5;
>> vrad
vrad =
    -1.1000   3.5000

>> vkol = vrad' %transpose
vkol =
    -1.1000
    3.5000
```

vrad(1) indicates that the index 1 (position 1) to be assigned. The ‘ means transpose.
Generate vectors

- Colon notation is often used to create vectors
  \[ \text{start:stride:stop} \]
  if stride = 1 it can be omitted

\[
\begin{align*}
\text{>> vektor} & = 0:5 \\
\text{vektor} & = \\
& 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5
\end{align*}
\]

\[
\begin{align*}
\text{>> vektor2} & = 0.0:0.05:2.0 \\
\text{vektor2} & = \\
& 0 \quad 0.0500 \quad 0.1000 \quad 0.1500 \ldots \\
& 1.9500 \quad 2.0000
\end{align*}
\]

The command `linspace` also create vectors

\[
\text{linspace(start,stop,elements)}
\]

\[
\begin{align*}
\text{>> v} & = \text{linspace}(0,2,10) \\
\text{v} & = \\
& 0 \quad 0.2222 \quad 0.4444 \quad 0.6667 \quad 0.8889 \\
& 1.1111 \quad 1.3333 \quad 1.5556 \quad 1.7778 \quad 2.0000
\end{align*}
\]

These two methods are used, for example, when creating the x-axis at the graphics.

**Task:** Create a vector in two different ways with equally spaced elements using stride \(2\pi/100\) in the interval \([-\pi, \pi]\).
generate matrices

\[ A = \begin{bmatrix} 1 & 7 \\ 5 & 3 \end{bmatrix} \]

Semicolon or return in the matrix gives a new row
Semicolon in the end suppresses printing

**Task:** Create a 3x4 matrix and then transpose it.

\[
\begin{bmatrix}
1 & 7 \\
5 & 3
\end{bmatrix}
\]

>> A = [1 7; 5 3];
>> A
A =
\[
\begin{bmatrix}
1 & 7 \\
5 & 3
\end{bmatrix}
\]

Creating a Matrix elementwise

\[
A = \begin{bmatrix} 1 & 7 \\ 5 & 3 \end{bmatrix}
\]

>> A(1,1)=1; A(1,2)=7;
>> A(2,1)=5; A(2,2)=3;
>> A
A =
\[
\begin{bmatrix}
1 & 7 \\
5 & 3
\end{bmatrix}
\]
Special matrix commands

- Available built-in features to create the usual (and unusual) matrices
- Can create more advanced matrices by combinations of these

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eye(n)</code></td>
<td>identity-matrix</td>
</tr>
<tr>
<td><code>ones(m,n)</code></td>
<td>one-matrix</td>
</tr>
<tr>
<td><code>zeros(m,n)</code></td>
<td>zero-matrix</td>
</tr>
<tr>
<td><code>rand(m,n)</code></td>
<td>rand-matrix</td>
</tr>
</tbody>
</table>

+ numerous other commands

```
>> B = ones(2,3)
B =
    1     1     1
    1     1     1

>> eye(2)
ans =
    1     0
    0     1

>> C = zeros(2,2)
C =
    0     0
    0     0
```

Extending a matrix

Given \( A = \begin{bmatrix} 1 & 7 \\ 5 & 3 \\ 1 & 7 \end{bmatrix} \)

create \( A = \begin{bmatrix} 3 & 5 & 7 \\ 1 & 2 & 1 \end{bmatrix} \)

```
>> A = [A; 2 1];
alternative
>> temp = [2 1];
>> A = [A; temp];
```

Separate row and column vectors!

```
>> temp2 = [2;1];
>> A = [A; temp2];
??? Error using ==> vertcat
All rows in the bracketed expression must have the same number of columns.
```
Elements, rows and columns

- One can work with individual elements, rows, columns, submatrices

Colon (:) denotes the whole range (all rows or all columns)

```
>> A(2,1)
an = 
    5
>> A(2,:)
an = 
    5 3
>> A(2,:) = [0 0]
A = 
    1 7
    0 0
    2 1
```

Submatrices, colon notation

Submatrices of the matrix A, mxn can be created quickly with colon notation

- A(:,j) j:th column of A
- A(i,:) i:th row of A
- A(i:k,j:m) submatrix, rows i-k and cols j-m

```
A =
 1 7 6
 0 0 5
2 1 3

>> B = A(2:3,1:2)
B =
 0 0
 2 1
```

Example, Picking out this submatrix
Workspace and Array Editor

- If you double-click on a variable in Workspace a new window is opened, Array Editor

- Here you can change the variable value but also size. **Task:** try with one of your matrices.

Size and length of a Matrix

```matlab
>> A = [1 7; 5 3]
>> vkol = [1; 2; 3; 4; 5]
>> size(A), size(vkol), length(vkol)
ans =
    2 2
ans =
    5 1
ans =
    5
```
Matrix operations, arithmetic

- Mathematical operations on matrices can be done directly, $C = \sin(A)$, elementwise.
- Addition/subtraction is OK if same size, e.g., $C = A + B$, where $A$ and $B$ are $m \times n$.
- Matrix multiplication, $C = AB$, only works if the number of columns in $A$ is the same as the number of rows in $B$.
- Matrix division, $C=\frac{A}{B}$, equals $A\cdot\text{inv}(B)$.

Matrix algebra

What if the sizes do not match?

```matlab
>> A=[1,7;5,3];
>> x = [2; 1];
>> A*x
ans =
   9
  13
```

$Ax$ is OK, but $xA$ does not work.

```matlab
>> x*A
??? Error using ==> * Inner matrix dimensions must agree.
```

Mathematical definition of matrix multiplication applies to matrices/vectors!
Elementwise operations

- Operations, e.g., *, /, ^ can be done elementwise

Example:
\[ B^{\times 2} = B \times B \]

but
\[ B^{.^2} = \begin{bmatrix} b_{11}^2 & b_{12}^2 \\ b_{21}^2 & b_{22}^2 \end{bmatrix} \]

Example:
\[ B^2 = B \times B \]

\[ B^{.^2} = 1 \quad 49 \\
25 \quad 9 \]

Elementwise operations

- Similarly, \( B \times C \) "ordinary" matrix multiplication between two matrices while
\[ B \times C = \begin{bmatrix} b_{11}c_{11} & b_{12}c_{12} \\ b_{21}c_{21} & b_{22}c_{22} \end{bmatrix} \]

- Sometimes this can generate errors...

\[ a = [2 \ 4] ; \]
\[ a^{.^2} = \begin{bmatrix} 4 \\ 16 \end{bmatrix} \]

\[ a^2 \]

??? Error using ==> ^
Matrix must be square.
Linear systems of equations

- Backslash-operator, \, is used to solve the linear system of equations, \( Ax=b \)

\[
\begin{align*}
\text{>> } & A = \begin{bmatrix} -2 & 4; 2 & 5 \end{bmatrix}; \quad b=[1; 2]; \\
\text{>> } & x = A\backslash b \\
& x = \\
& 0.1667 \\
& 0.3333
\end{align*}
\]

**Task:** Create a random 5x5 linear system of equations and solve it. Verify the solution by computing the residual \( r=Ax-b \).

Complex numbers

- Complex numbers can be created with `complex`

\[
\begin{align*}
\text{>> } & z = \text{complex}(1.2, 2.5) \\
& z = \\
& 1.2000 + 2.5000i
\end{align*}
\]

- Or in computations

\[
\begin{align*}
\text{>> } & z = \sqrt{-2} \\
& z = \\
& 0 + 1.4142i
\end{align*}
\]
Complex numbers

- Complex works for vectors as well

```
>> z = complex([1.2 -3],[-1 2.5])
z =
  1.2000 - 1.0000i  -3.0000 + 2.5000i
```

- Special functions
  - `real(z)` – real part of \( z \)
  - `imag(z)` – imaginary part of \( z \)
  - `conj(z)` – conjugate to \( z \)
  - `abs(z)` – absolute value of \( z \)
  - `angle(z)` – phase angle in radians

Simple graphics (2D)

- Steps to perform a plot
  - Create a horizontal axis (x-axis)
  - Compute the function values (y-axis)
  - Plot x to y, `plot(x,y)`
  - (Save figure to file to include later in a report)

```
>> x = linspace(0,2*pi,50);
>> y = cos(x)+sin(x);
>> plot(x,y)
>> print -djpeg myfig.jpg
```
**Simple graphics (2D)**

- X-axis is created through
  \[ x = \text{linspace}(x_0, x_1, \text{nelem}); \]
  or
  \[ x = [x_0: \text{step} : x_1]; \]
- The plot command can be extended in many ways, for example:

  \[
  \begin{align*}
  \text{plot}(x, \cos(x), '-', x, \sin(x), 'o') \\
  \text{Alternative:} \\
  \text{plot}(x, \cos(x), 'o') \\
  \text{hold on;} \\
  \text{plot}(x, \sin(x), 'go')
  \end{align*}
  \]

See “help plot” for options.

**Task:** Plot the function \( y = \sin(x^2) \) in the interval \( x=[0,5] \) with steps of 0.05 using a blue dashed line with * markers.
Simple graphics, histogram

- `bar(x,y)` or `bar(y)`
- `bar(A,'stacked')`, `bar(A,'grouped')`
- `bar3(A)`  [Where A is a 6x3 matrix below]

Simple graphics, pie

- `pie(x)`, `pie3(x)`
- `pie3(x,extract,{ 'Bit 1', 'Bit 2', ... })`
3D-graphics

- In 3D we have a x, y och z-axis

- Given the x-axis and y-axis we must create x values for all y values and y values for all x values, i.e., we must create a “grid”.
  This is done with the command `meshgrid`

3D-graphics

We want to plot the function: \( f(x, y) = xe^{-x^2-y^2} \)

- The x- and y-axes can be created as before

```matlab
>> x = 0:0.1:1; y = 0:0.2:1;
```

- The “grid” is created with

```matlab
>> [X, Y] = meshgrid(x,y);
```

- Function values are calculated for each x and y value

```matlab
>> Z = X .* exp(-X.^2 - Y.^2);
```
3D-graphics

- Finally plot

```matlab
>> mesh(X,Y,Z);
```

3D-graphics

- A little better resolution by increasing the number of points in x and y direction

```matlab
>> x=linspace(0,1,50); y=linspace(0,1,50);
>> [X,Y] = meshgrid(x,y);
>> Z = X .* exp(-X.^2 - Y.^2);
>> mesh(X,Y,Z);
```
3D-graphics

- Change mesh to **surf** to get a surface plot

```matlab
>> surf(X,Y,Z);
```

3D-graphics

- Remove the mesh-lines

```matlab
>> shading('interp');
```
Task: Plot the "Twin-peaks" function

\[ z = (x^2 + 3y^2)e^{(1-x^2-y^2)} \], \( x = [-2,2] \), \( y = [-2,2] \)

3D-graphics

- Contour lines

\[
\text{>> contour}(X,Y,Z,20); \\
\text{>> contour3}(X,Y,Z,20);
\]
Graphics-window

- Use the menus on the window to rotate, zoom, move, view, change line styles, add text, etc.

**Task:** Insert x-, y- and z-labels in your Twin-peaks plot. Set also title and rotate the figure for another view. *(Extra, change the colormap of your plot)*

---

Programming, M-files

- A *command file* is a way to "store" commands that would otherwise be written interactively in the command window.

- By *running* the file all the commands in the file are executed and the result is displayed in the command window (or the graphics window).

- The command file must end with `.m`, e.g., `myfile.m`.

- The command files can be written with any editor but MATLAB provides a simple editor.
Creating command files

- M-files are created most easily in MATLAB editor write `edit` in the command window or click on `New Script` in menu

- A new window will open with an editor where you can type in your commands

Creating command files

- Enter the commands exactly the same way as in the command window
Creating command files

- Save the file! Important to have control on the directory in which the m-file is saved.

Note that the working directory and the directory where the file is stored must match or else Matlab will not find the file and can not run it.

Click on the folders in the **Current Folder** or on the path in the menu bar to change the working directory.
Running command files

- Run the code, which performs the commands, using one of the options:
  - in the command window, type the file name without extension (.m)

```
>> Myfile
```
Run the code in Myfile.m

- in editor, click on the run-button

Example, command files

Open MATLAB editor and enter the following code

```matlab
% Lisas program, LisasFile.m
x0 = 0; x1 = 2*pi; n = 100;
x = linspace(x0, x1, n);
y = sin(x);
plot(x,y);
```
Example, command files

Save file and type in the command window

```
>> LisasFile
```

The code runs and gives the result

**Task:** Create a command file with commands for the Twin-peaks plot and run the file.

---

Text strings in MATLAB

- Text strings are written inside `'`

  ```matlab
  >> name = 'Nisse';
  >> greeting = ['Hello ' name '!']
  greeting =
      Hello Nisse!
  >> name(2) = 'a'
  name =
      Nasse
  ```

- The example shows that texts (text strings, strings) works as row vectors
Text strings in MATLAB

When are text strings needed?

- To make better looking prints of results

Example
Given a vector \( x \) print largest element

\[
\begin{align*}
\text{Max value is } & = 9.4248 \\
\end{align*}
\]

- disp – prints the text string that is enclosed in parentheses
- num2str – converts the numeric value to string
- \([s1 \ s2]\) – merges two strings into one

\[\pi\] in a text string is called LaTeX syntax and can be used to write mathematical text

Result...
Text strings in MATLAB

...is this

Task: include labels and title in your Twin-peaks command file

xlabel(...)
ylabel(...)
title(...)

Guide text at input of a value to a variable

\[ x = \text{input('Give a number:')} \]

Using the `input` command in a command file gives the following result at run time

_Give a number:_

The cursor is now waiting at the colon in the running to give a number. This number is then assigned to the variable `x`.

Task: Modify your Twin-peaks program to let the user give x- and y-ranges for the plot at run time using the `input` command.
Repetition, command files

**Task:** Write a program that reads two numbers, calculates the average of these and prints the result with an explaining text.

You will need to use *command file*, *disp*, *input* and *num2str*.

Control structures, If-statement

For alternative paths in a program (conditions) can **if** be used:

```
if logic expression
    path 1
else
    path 2
end
```

The if-statement is a fundamental structure in programming. Used to handle various special cases, alternatives. Which branch of the if-statement that will be used is determined by relational and logical operators. Else-branch can be omitted, and it can also be expanded with several branches.
Relational operators

- The logical expressions use so-called **relational operators**
  - `<` Less than
  - `<=` Less than or equal with
  - `>` Greater than
  - `>=` Greater than or equal to
  - `==` Equal to
  - `~=` Unequal to

- Gives the value **true** (1) or **false** (0)

- In MATLAB, if something is true, it has the value 1 and if something is false, then the value 0.

Logical operators

- Following **logical operators** are defined in MATLAB:
  - `&` and
  - `~` not
  - `|` or
  - `xor` exclusive or

- `U1 & U2` true if both U1 and U2 are true
- `U1 | U2` true if either U1, U2, or both are true
- `~U` true if U is false and false if U is true
- `xor(U1, U2)` true if U1 is true or if U2 is true (not true if both U1 and U2 are true)
**If-statement**

- Important that the if statements cover all the options and the different branches are not logical overlap. Often good that the last else covers "the rest".

**Ex)**

```matlab
x = input('Give a number x: '),
if x > 0
    disp('x is greater than zero!')
elseif x >= 0
    disp('x is greater than or equal to zero!')
end
...not so good, why?
```

**Ex)** We assume that the result has a value assigned to represent the outcome of a roll of the dice.

```matlab
if (result==1 | result==3 | result==5)
    disp('odd number of eyes')
elseif (result==2 | result==4 | result==6)
    disp('even number of eyes')
else
    disp('What kind of dice is this?')
end
```

**Task:** Write a program (modify your previous program) that reads two numbers and calculates the quotient. Are there any exceptions to handle?
For-statement

Repetition with $\textbf{for}$ means a group of commands is performed a predetermined number of times.

$$\text{for} \ \text{variabel} = \text{start:step:stop}$$

$$\text{commands}$$

$$\text{end}$$

For-loop is a fundamental structure in programming. Used to repeat something. A group of commands (between $\textbf{for}$ and $\textbf{end}$) is repeated again and again for the predetermined number of times. If $\text{step} = 1$, it can be omitted. $\text{Step}$ can also be negative (but then $\text{start}>\text{stop}$).

For-statement

The $\textbf{for}$ statement $i = 1: \text{ntime}$ can be read: $\textit{for } i \ \textit{equal to } 1 \ \textit{until } \text{ntime}$

and means that the variable $i$ will start with the value 1 and gradually be increased

$$i=1$$

$$i=2$$

$$...$$

$$i=\text{ntime}$$

in each iteration until it reaches the value $\text{ntime}$. We get $\text{ntime}$ number of iterations (repetitions) of the commands.
For-statement

\textbf{for } x=0:0.5:2

Means that $x$ starts at 0 and is incremented by 0.5 in each iteration until $x = 2$ (we get 5 iterations).

\begin{align*}
x &= 0 \\
x &= 0.5 \\
... \\
x &= 2.0
\end{align*}

\textbf{Task:} Write a program that reads in $n$ numbers (let the user provide $n$ at run time) and then prints them in reverse order.

---

While-statement ("as long as")

Repetition with \texttt{while} is used when we in advance do not know how many times a group of commands will be performed

\begin{verbatim}
while logic expression
  commands
end
\end{verbatim}

While loop is a fundamental structure in programming. Used to repeat something as long as a certain condition is met.
While-statement

Example problem:

Write a program that reads numbers from input, calculates the sum and average of the numbers and print the result.

We do not know beforehand how many numbers to be read, but stop when an empty <enter> is given.

If a variable, e.g., variable t is empty, it can in Matlab be tested with `isempty(t)`

```matlab
% Program that computes the average and sum
% Input is stopped with a blank <enter>

disp('Give numbers, terminate with blank <enter>');
counter = 0;  % Counter of numbers
num=[];  % Vector for numbers (empty)
t = input('Give the first number: ');

while ~isempty(t)  % Iterate as long as t non-empty
    num = [num t];  % Place t in the num-vector
    counter = counter + 1;  % increase counter
    t = input('Give a new number: ');
end

if counter == 0
    disp('No numbers have been processed.');
else
    average = sum(num)/counter;
    disp(['Sum = ', num2str(sum(num)),...
         ' och average = ', num2str(average)]);
end
```
While-statement

Task:

• Load (copy and paste) the example program, run and verify that it works as intended.

• Modify the program so that the input instead ends when one gives -9999.

• Modify the program so that a blank <enter> is ignored and the program continues asking for new numbers (until -9999 is given).

Summary

- In programming we use some basic control structures, in Matlab we have
  - Alternatives: if
  - Iterations: for and while
- The structures have a matching end in Matlab
- To clearly see where the different statements begin and end in the code use indentation, e.g.
  ```matlab
  if t~=0
    disp('…');
    kvot = a/t;
  …
  end
  ```
Self study

Goto to the course page and start working with self study material on Matlab:

http://www.it.uu.se/edu/course/homepage/bridging/ht13/lab