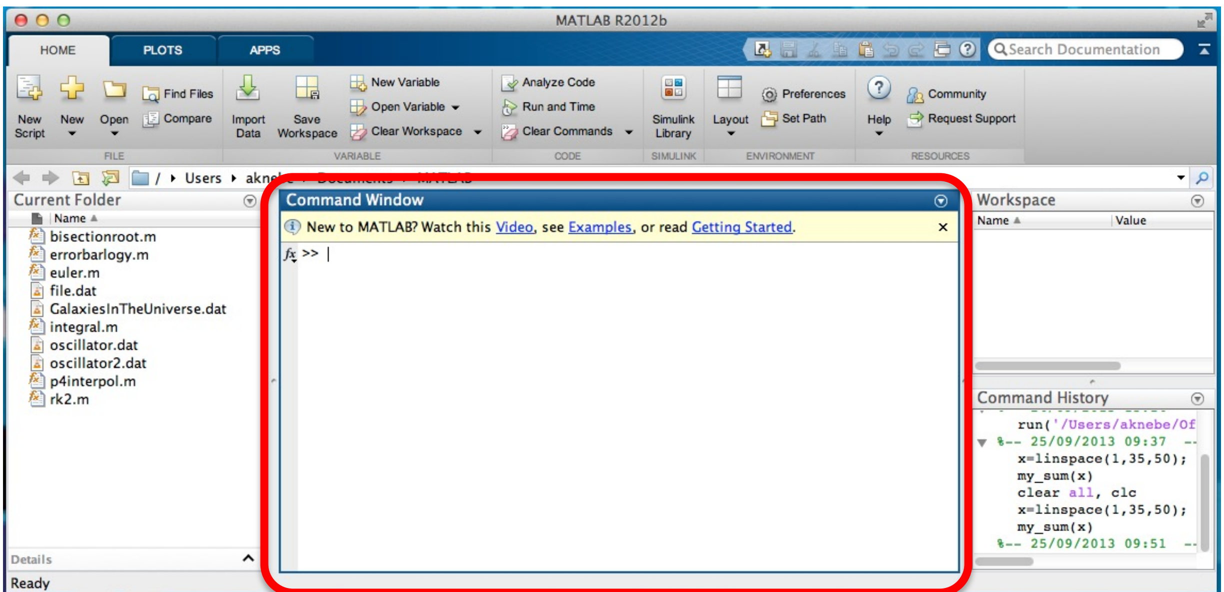
Unit 1

Basic Numerical Concepts & First Applications

MATLAB *...in general*

- MATLAB...
  - ...is an interactive program to perform calculations
  - ...uses a high level programming language
  - ...is highly tuned for vector operations
  - ...can be operated via a user-friendly interface:



primary command window

- MATLAB can perform the following calculations...

+	addition,	e.g. 2.5+3.1
-	subtraction,	e.g. 3.7-1.6
*	multiplication,	e.g. 5.1*8.2
/	division,	e.g. 7.3/3.2
^	power,	e.g. 3.1^2.5 (=$3.1^{2.5}$)

...following the elementary arithmetic rules

- MATLAB has a library of functions (e.g. sin(), log(), tan(), ...), but allows for user-defined functions, too

- MATLAB comes with an in-built help system:

```
>> help name
```

...where *name* is the function you like to know more about.

➢ **exercise:**

- try >> `help help`
- try >> `help sin`
- try >> `help exp`

- **Note**: whenever you are uncertain about anything in MATLAB, use `help` to find out about it!

MATLAB *...as a calculator*

- **exercise:**

• use MATLAB as a calculator to perform the following calculations (in the command window)...

```
>> 4+5/12
>> (4+5)/12
>> 1000000000+1
>> 1000000000-1
>> 1.278e-2+0.23
>> 1.5^5.8
>> -2^2
>>(-2)^2
>> 2*pi
>> 103^2.4+4*3.7e-2-1.2/4^1.2
>> ...
```

• is MATLAB always obeying the correct rules?
• understand the results you obtain
• try to find examples where the answer is "wrong", i.e. not what you expected!
• ***Note***: 3.7e-2 is a shortcut for 3.7 x 10^{-2}

- **exercise:**

• use `help` to find out which mathematical functions (e.g. `sin()`, `asin()`, `sqrt()`, ...) are available in MATLAB

- **exercise:**

• perform the following calculations:

$\log_3(108)$	$\exp(\log(9))$	$\sin(a\sin(0.7))$	$\tan(0.7) - \frac{\sin(0.7)}{\cos(0.7)}$
$3 - \frac{3 * 5^{1.5}}{\left(5 - \frac{4}{3}\right)}$	$3^{2*(4-\frac{1}{3})}$	$\sqrt{e^{-2*34} - \pi * \ln(3.5)}$	$\sin(\pi/2) * \cos(2.4 * 10^{-3})$

• do you get the expected answer?
• what happens when you increase the accuracy of your calculations?

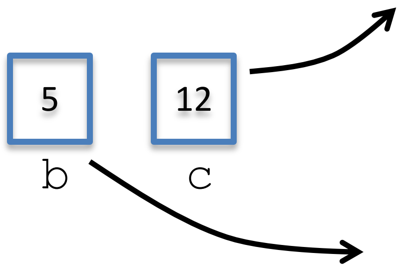
- **exercise:**

• how to increase the accuracy of your calculations? try >> `help format`
• perform some more calculations switching between various `format`s, e.g. switch on >> `format long` and try `1000000000+1` again

MATLAB *variables*

- MATLAB can store numbers in variables

```
>> a=4
>> b=5
>> c=12
>> a+b/c
```



a "piece" of computer memory (RAM) will be called "`c`" and is reserved for storing any number, e.g. "12"

a "piece" of computer memory (RAM) will be called "`b`" and is reserved for storing any number, e.g. "5"

you can choose whatever name you prefer for variables:
ALWAYS USE MEANINGFUL VARIABLE NAMES!

➢ **exercise:**

- repeat your previous calculations utilizing variables this time…

➢ **exercise:**

- why is
  ```
  >> 4=a
  ```
  not working?

- useful commands

```
>> clear variable
>> clear all
>> who
>> clc
```

(try `help` on them, i.e. `help clear`)

➢ **exercise:**

- are there any predefined variables in MATLAB?
- what happens when you use MATLAB's variables as your own?
- how can you recover MATLAB's values?

MATLAB

complex variables

- MATLAB can also store complex numbers in variables

```
>> c=4.5+14.75i
>> d=complex(4.5,14.75)
>> real(c)
>> imag(c)
```

➢ **exercise:**

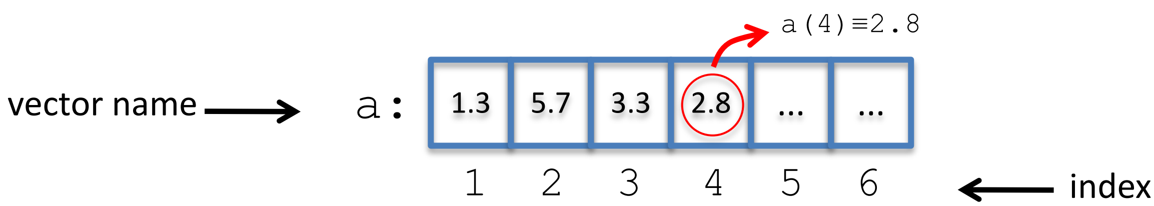
- define various complex variables
- perform mathematical operations with those complex variables

vectors

- MATLAB can store multiple numbers as a vector:
  - "[]" generates a vector and
  - ", " or "; " separates its elements (coma creates row vector, semi-colon creates column vector)

```
>> a = [1.3, 5.7, 3.3, 2.8, sqrt(9.67), 0.67, 4.23]
>> b = [3,  4, 2]
>> c = [3; -4; 2]
>> d = [15, 73, 65]
>> e = [d, b]
```

- a vector is a consecutive row/column of variables that all have the same name, e.g. "a"



- the individual elements of the vector can be obtained by properly "indexing" the vector name

```
>> a(1)
ans = 1.3
>> a(4)
ans = 2.8
```

Note: the index of a vector is **always** an integer number!!

- the number of elements of a vector are obtained with `length()`

```
>> length(a)
```

- the last element of a vector can be easily accessed in two different ways:

```
>> a(length(a))
>> a(end)
```

- you can easily add elements to an existing vector, e.g. add 5 as the new last element:

```
>> a = [a, 5]
```

- you can also remove any element from a vector, e.g. completely remove element at position `i`

```
>> a(i)=[]
```

MATLAB *vectors*

➢ **exercise:**

- define various vectors, e.g.

```
>> a = [15.7, sin(0.7), exp(-1.5), 2*pi]
>> b = [2/3, a, 104.7]
>> c = [9.5; 7.14; cos(0.23)]
>> d = [tan(0), -3.5, 15.4*2.9]
```

- perform several mathematical operations on them, e.g.

```
>> 2*a
>> a-pi
>> c/2-2.3*d
>> sin(5.34*d)
>> sort(c)
>> norm(a)
>> max(b)
>> min(d)
>> sum(a)
>> …
```

- you can also define vectors for accessing certain elements of another vector

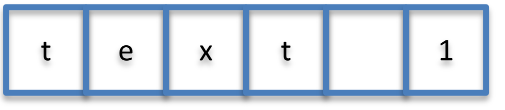
```
>> a = [1.3, 5.7, 3.3, 2.8, sqrt(9.67), 0.67, 4.23]
>> i = [3, 4, 2]
>> a(i)
```

- ***Note:*** the index of a vector is **always** an integer number!
- ***Note:*** **MATLAB favours the use of vectors and hence we will use such index vectors very often!**

- MATLAB also allows the definition of text vectors (called ‘strings’)

```
>> s = [‘text 1’]
>> c = [‘text 2’]
>> t = [s, ‘ together with ’, c]
```

s:



- you can also access individual elements of a string

```
>> s(3),t(15)
```

- you can ‘add’ strings together

```
>> newstring = strcat(s,c)
```

- and you can convert numbers to strings (which can be useful when generating legends!)

```
>> n = num2str(5.78)
```

MATLAB *vectors*

- MATLAB distinguishes between column and row vectors

```
>> a = [1, 5, 10]          (row vector)
>> b = [1; 5; 10]          (column vector)
>> transpose(a)
>> a’
```

➢ **exercise:**

- perform several mathematical operations on row and column vectors
- are there new operations possible that previously were not allowed?

- MATLAB distinguishes between ***mathematical*** and ***numerical*** vector products:

>> a * b	is a mathematical vector multiplication	(result = either scalar or matrix)
>> a .* b	is a component-wise multiplication	(result = vector)

a * b

```
>> a = [5.3, 7.8, 1.9];
>> b = [9.8, 3.7, 2.6];

>> a*b’
ans =
  85.74
```
← *a(1)b(1)+a(2)b(2)+a(3)b(3)*

```
>> a'*b
ans =
  51.94  19.61  13.78
  76.44  28.86  20.28
  18.62   7.03   4.94
```
← *matrix multiplication*

```
>> a*b
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

a .* b

```
>> a = [5.3, 7.8, 1.9];
>> b = [9.8, 3.7, 2.6];

>> a.*b
ans =
  51.94  28.86  4.94
```
↑ *a(1)b(1)* ↑ *a(2)b(2)* ↑ *a(3)b(3)*

➢ **exercise:**

- try various possibilities and understand the results
- when is it possible to perform a*b and when not?

- ***Advise!***

  - inner vector product: `dot(a,b)`
  - outer vector product: `cross(a,b)` vs. component-wise multiplication: a.*b

- MATLAB also performs component-wise division and powers:

```
>> a ./ b
>> a .^ b
```

a ./ b

```
>> a = [5.3, 7.8, 1.9];
>> b = [9.8, 3.7, 2.6];

>> a ./ b
ans =
   0.5408    2.1081    0.7308
```

a(1)/b(1) *a(2)/b(2)* *a(3)/b(3)*

a .^ b

```
>> a = [5.3, 7.8, 1.9];
>> b = [0.5, 1.1, 0.7];

>> a .^ b
ans =
   2.3022    9.5786    1.5672
```

$a(1)^{b(1)}$ $a(2)^{b(2)}$ $a(3)^{b(3)}$

➢ **exercise:**

- perform several component-wise divisions and powers of row and column vectors
- are there new operations possible that previously were not allowed?

- ***Note:*** a/b (without the 'dot') is a special matrix operation to be explained later!

MATLAB *vectors*

- MATLAB can generate vectors containing equally spaced values, **option 1**: "**:**" (colon operator)

```
>> a = [0:2:10]
>> b = [0.5:0.1:1.2]
>> c = [1:3:10]
>> help colon
>> help length
```

➢ **exercise:**

- generate a vector running from 100 down to 0 in steps of 2
- extract every 10th element of that vector (in one command only!)
- calculate the product of all those 10th elements (hint: `help prod`)

- ***Advice:*** sometimes it is more convenient to generate an index vector:

```
>> a = [0:2:10]
>> i = [1:2:5]
>> a(i)
```

we will use index vectors a lot throughout the course!

➢ **exercise:**

- repeat the previous exercise using an index vector
- generate an index vector that accesses every element but the first and last
- **Note**: you might have done the previous exercise already like this, then there is no need to repeat it again...

MATLAB *vectors*

- MATLAB can generate vectors containing equally spaced values, **option 2**: `linspace()`

```
>> x = linspace(xmin, xmax, N)
>> …
>> help linspace
```

- MATLAB uses vectors like variables in functions

```
>> x = linspace(0, 2*pi, 75)
```

=> x

0	...	...	...	2pi

(75 elements)

```
>> y = sin(x)
```

=> y

sin(0)	...	...	...	sin(2pi)

(75 elements)

➢ **exercise:**

- is there a difference between `sin(x)` and `sin(x')`?
- why do we need to write `x.^3.5` and not `x^3.5`?

- ***Note:*** `linspace()` is a very useful function that will be frequently used throughout the course!

- there is also a function in MATLAB that generates logarithmically spaced vectors: `logspace()`

```
>> x = logspace(log10(xmin), log10(xmax), N)
>> …
>> help logspace

>> x = logspace(log10(1), log10(100), 75)
>> y = x.^3.5
```

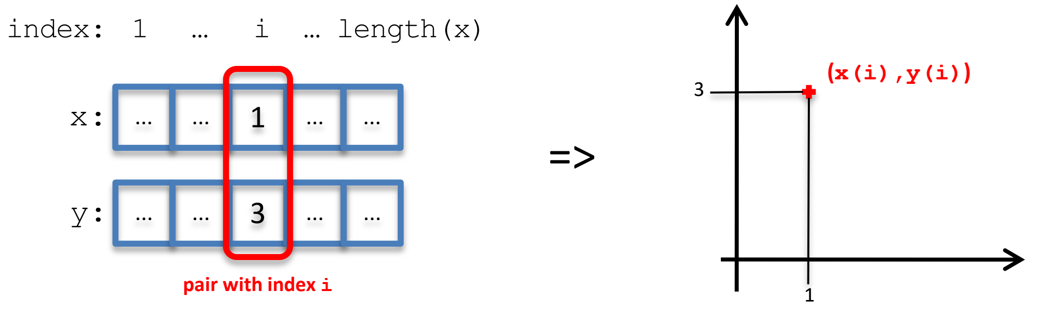
not relevant for exams...

MATLAB *plotting*

- MATLAB can plot vectors for you

```
>> x = linspace(0,2*pi,100)
>> y = sin(x)
>> plot(x,y)
```

- `plot()` is plotting a point (dot, cross, ...) for each and every *pair of variables* stored in the two vectors



- ***Notes***:
  - both vector must have the same lengths (otherwise there are obviously no pairs)
  - the first vector will be used as x-axis, the second as y-axis
  - `plot()` connects the points with a line (unless you specify this differently)
  - `plot()` takes various additional arguments to specify line-style and line-colour

```
>> plot(x,sin(x),'y+')
```

- there are various commands to add legends, labels, etc.

```
>> legend('sine curve')
>> title('MATLAB course example')
>> xlabel('angle [rad]')
>> ylabel('sin(angle)')
```

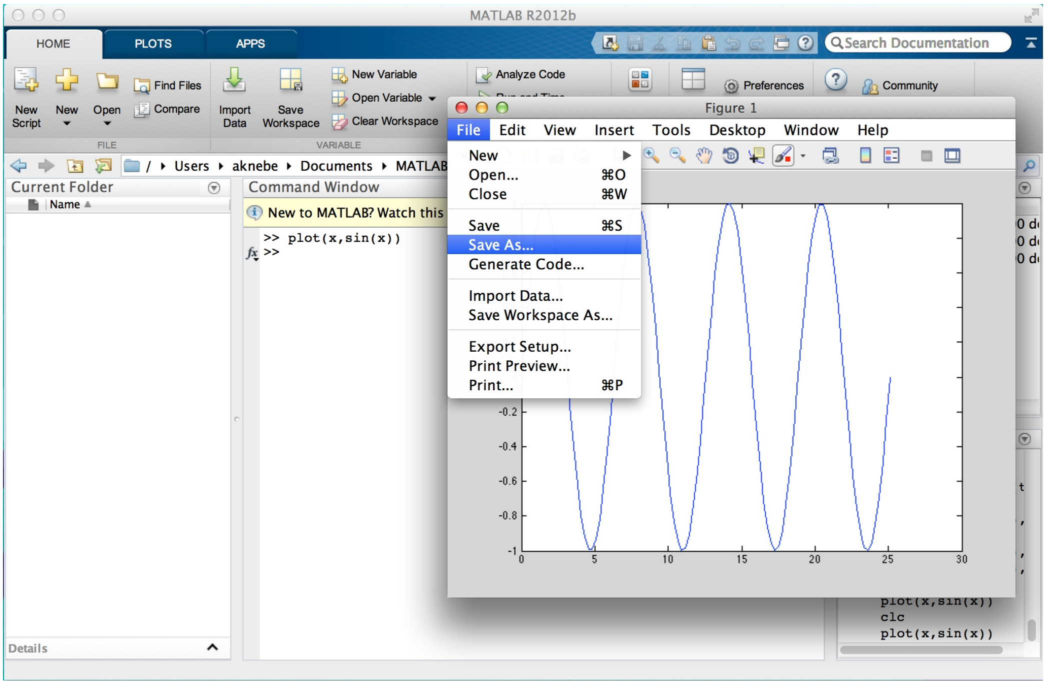
➢ **exercise:**

- use `help` to learn more about `plot()`
- plot `sin(x)` where the x-axis is the angle in degrees and label the plot correctly
- plot various other mathematical functions, e.g. e^x, $log_3(x)$, $5x^3$, $\cos(x)$ / $\sin(x)$, ...

➢ **exercise:**

- generate a plot that shows $f(x)=x^{3.5}$ for 20 points on the intervall $x=[1,100]$ using `linspace()`
- add to the same plot $f(x)=x^{3.5}$ on the same intervall, but now using `logspace()`
- what is the difference between using `linspace()` and `logspace()`?

- you can save the plot in various formats to disk by using 'Save As...' the 'File' menu from the figure window:



- as mentioned before, power-laws are ***very*** common in physics and best plotted on a logarithmic scale

```
>> x = logspace(log10(1),log10(1000),100)
>> plot(log10(x),log10(x.^3),'o')
>> loglog(x,x.^3,'+')
```

➢ **exercise:**

- why is the logarithmically scaled plot better than a linear plot?
- what is the difference between `plot(log10(x),log10(y))` and `loglog(x,y)`?
- what happens when you use `linspace()` instead of `logspace()`?
- how can you determine the power-law exponent from the log-plot?

advanced exercise (voluntary!)

MATLAB *plotting*

- `figure(n)` generates a new window `#n` for a plot, e.g.

```
>> x = linspace(0,2*pi,100);
>> figure(1)
>> plot(x,sin(x))
>> figure(2)
>> plot(x,cos(x))
```

- `close(n)` closes window `#n` again.

- `hold on/off` allows to use multiple `plot()` commands in a sequence for the same figure, e.g.

```
>> x = linspace(0,2*pi,100);
>> plot(x,sin(x))
>> hold on
>> plot(x,cos(x))
>> hold off
```

- `axis` controls the axis scaling and appearance

- `xlim`, `ylim`, and `zlim` set (or even retrieve) the axis limits

- `grid` places a grid on top of an existing plot

- `subplot()` (followed by `plot()`) generates multiple plot windows in one figure window, e.g.

```
>> x = linspace(0,2*pi,100);
>> subplot(1,2,1), plot(x,sin(x))
>> subplot(1,2,2), plot(x,cos(x))
```

where the first argument is the total number of rows, the second the total number of columns and the third the identifier of the actual subplot

- `plot3(x,y,z)` connects the points (x,y,z) in a 3D plot, e.g.

```
>> t = linspace(0,8*pi,100);
>> plot3(cos(t), sin(t), t)
```

- experimental data with error bars is best plotted with `errorbar(x,y,e)`, e.g.

```
>> x = linspace(0,8*pi,100);
>> y = sin(x);
>> e = randn(1,length(x));     we assign random error values using MATLAB's randn() function (more later!)
>> errorbar(x,y,e)
```

➢ **exercise:**

- repeat the exercise with the errorbars assigning 0.1 as the error for every single point

MATLAB **from now write a script for every exercise!** *scripts*

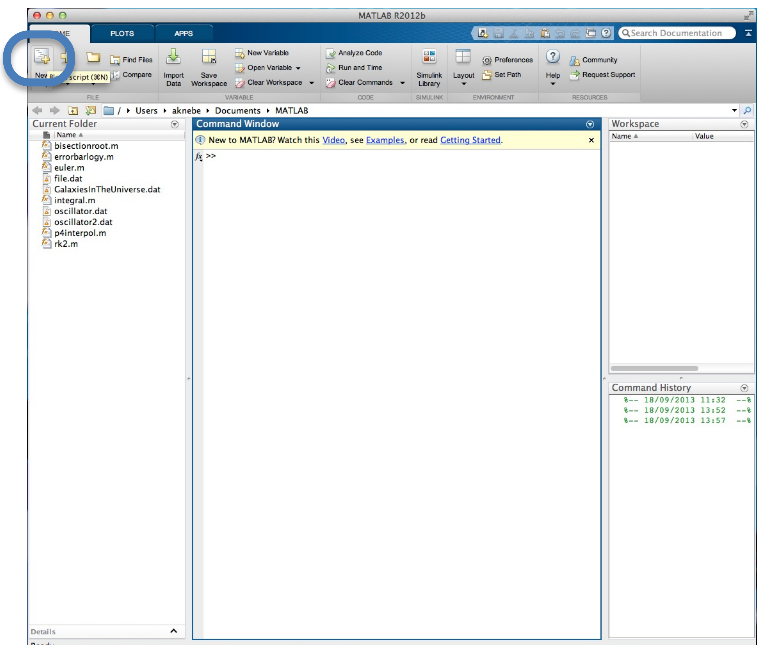
- MATLAB reads and executes multiple commands from a given file with extension `*.m`

➢ **exercise:**

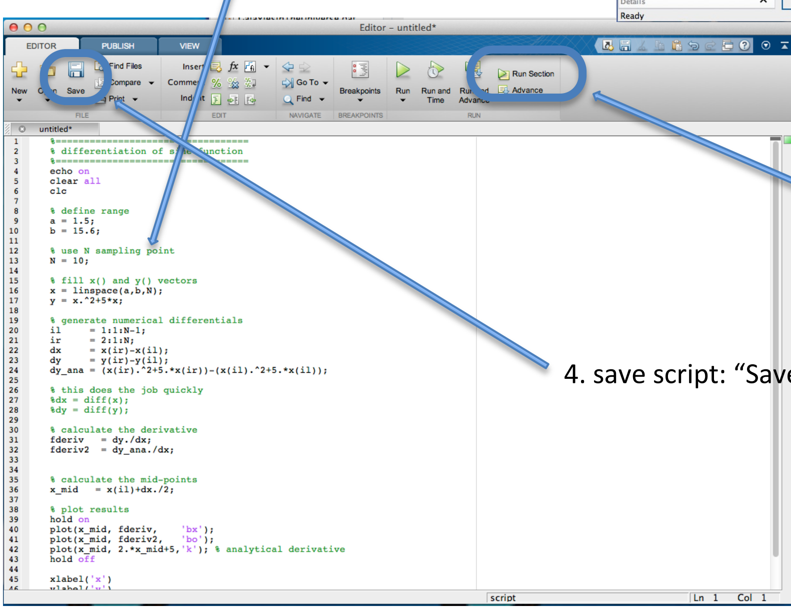
> • write a script that plots sin(x) and cos(x) in one figure window next to each other
> • ***hint***: use `figure()` and `subplot()`

- `pause` causes a script to wait until you press any key (can be useful for debugging...)
- `%` is used for placing comments into the script
- `%%` structures the script into blocks (also very useful for debugging your script!)

1. open the editor “New Script”



2. write commands into script



3. evaluate script “Run Section”
(“Run Section” executes until the next %%)

4. save script: “Save”

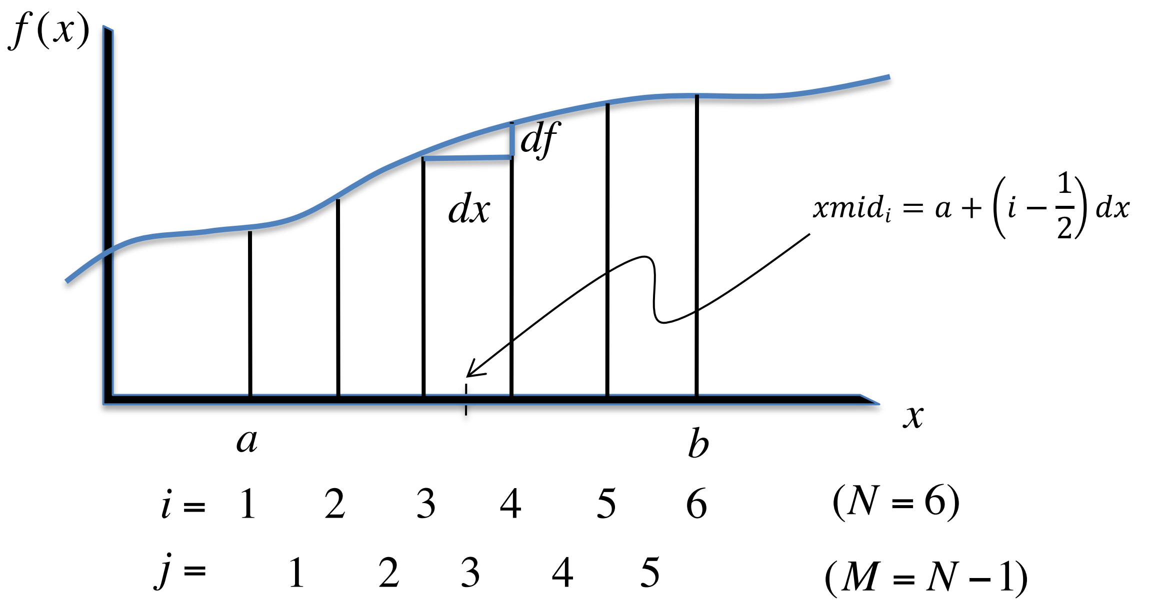
<u>Please start every script with the lines</u>

```
clear all    → removes all variables
close all    → closes all plot windows
clc          → clears command window
```

▪ **Numerical Derivatives – arbitrary functions**

Given two data vectors `x()` and `f()` we aim at determining the derivative

$$f'(x) = df/dx$$



▪ **Note**:

- the function in this example is tabulated in an array at N=6 points, i.e. i=1:6
- the derivative can only be tabulated in an array at M=N-1 points, i.e. j=1:5
- the derivative will be given at the midpoints $xmid_i = a + (i\text{-}0.5)\ dx$
- the midpoint can also be calculated as $xmid_i = (x_{i+1}+x_i)/2$

➢ **exercise:** (solution on next page...)

- numerically differentiate the function $f(x)=x^2+5x$, defined on interval [a,b] with a=1.5 and b=15.6
- ***hints***:
  - define two new vectors `df` and `dx` to then calculate `f_deriv = df./dx`
  - remember the usage of index vectors to access `x()` and `f()`
  - remember that the length of `df`, `dx`, and `f_deriv` will be N-1
  - remember that the derivative $f'(x)$ will be given at the midpoints

➢ **exercise:**

- numerically differentiate the function $x(t)=5\sin(t)\cos^2(t)$ on interval [a,b] with a=-π/4 and b=3π/4

▪ Numerical Derivatives – how to realise the project

- define interval $[a,b]$
- choose number of sampling points N

- generate vector $\vec{x}$ where $x_i \in [a,b], \forall i \in N$
- generate vector $\vec{f}$ where $f_i = f(x_i), \forall i \in N$

- calculate vector $d\vec{x}$ where $dx_i = x_{i+1} - x_i, \forall i \in N-1$
- calculate vector $d\vec{f}$ where $df_i = f_{i+1} - f_i, \forall i \in N-1$
- calculate vector $\vec{f}^{deriv}$ where $f_i^{deriv} = df_i / dx_i, \forall i \in N-1$
- calculate vector $\vec{x}^{mid}$ where $x_i^{mid} = (x_{i+1} + x_i)/2, \forall i \in N-1$

- plot the pairs $(x_i^{mid}, f_i^{deriv}), \forall i \in N-1$
- plot analytical derivative at $\vec{x}^{mid}$

1. algorithm
(independent of programming language!)

"translation"

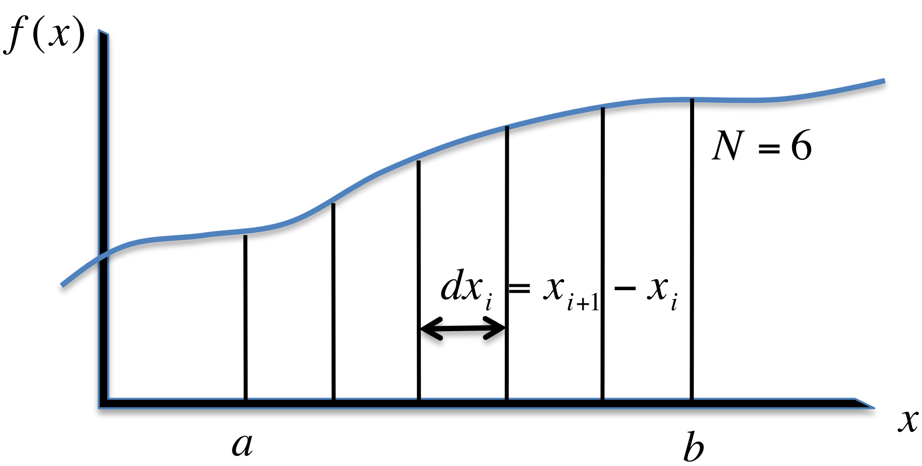
```
% parameters
a = 1.5;
b = 15.6;
N = 10;

% the function
x = linspace(a,b,N);
f = x.^2+5.*x;

% the derivative
il = 1:N-1;
ir = 2:N;
dx = x(ir)-x(il);
df = f(ir)-f(il);

fderiv = df./dx;
xmid   = (x(ir)+x(il))./2;

% plot numerical derivative
plot(xmid,fderiv,'r'), hold on

% and compare to analytical result
plot(xmid,2.*xmid+5,'bo')
```

2. program
(depends on programming language!)

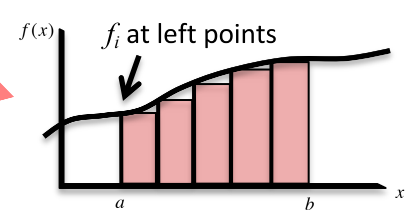
- **Numerical Integration**

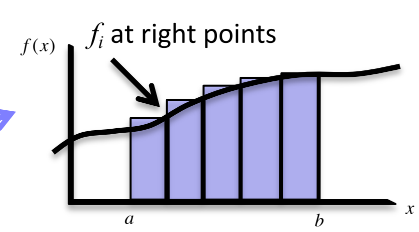
Given two data vectors `x()` and `f()` we aim at determining the integral

$$I = \int_a^b f(x)dx$$



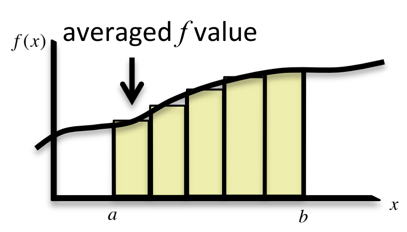
$$I_{num}^{edge} \approx \sum_{i=1}^{N-1} f(x_i)dx_i \approx \sum_{i=2}^{N} f(x_i)dx_{i-1}$$





mid-point integration:

$$I_{num}^{mid} \approx \sum_{i=1}^{N-1} \frac{f(x_i)+f(x_{i+1})}{2} dx_i$$



▪ Numerical Integration

➢ **exercise:**

- write a script that evaluates the integral of $\cos(x)$ between two angles $0<a<b<2\pi$.
- use all three options to numerically evaluate the integral.
- compare those three options against each other.
- which methods gives the best results and why?
- what happens when you in-/decrease the number of sampling points N?

- ***hints***:
  - remember `sum()`
  - to access the vector `y()` generate an index vector `i()`
  - the difference between consecutive elements of a vector is best calculated with `diff()`

```
>> dx = diff(x)
```

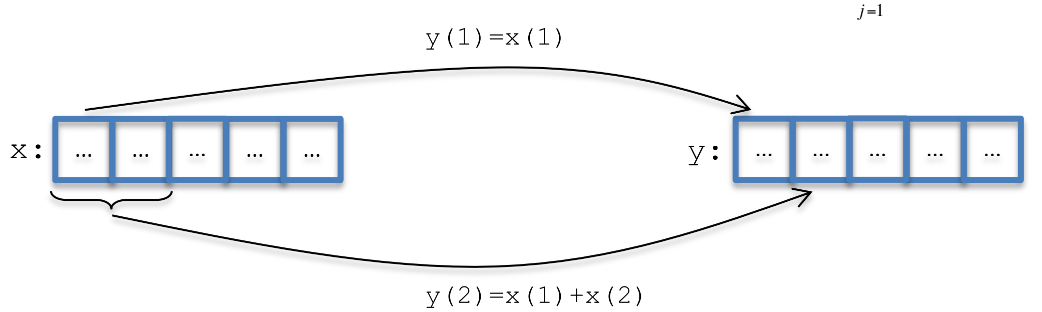
- **Numerical Integration**

Given two data vectors `x()` and `f()` we aim at constructing the antiderivative

$$y(x) = \int_a^x f(s)\,ds$$

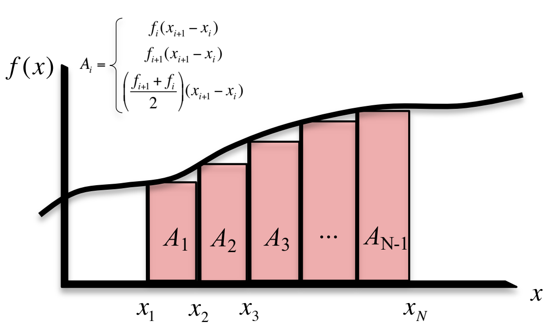
- MATLAB has an in-built function helping to solve for antiderivatives of functions: `cumsum()`

`>> y=cumsum(x)` generates a vector `y()` with the elements: $\mathtt{y(i)} = \sum_{j=1}^{i} \mathtt{x(j)} \quad \forall \mathtt{i} \in [1, \mathtt{length(x)}]$



etc.

- `cumsum()` can be used to numerically construct the antiderivative:



$$y(x_2) = \sum_{i=1}^{1} A_i = A_1 \quad \rightarrow \mathtt{y(1)}$$

$$y(x_3) = \sum_{i=1}^{2} A_i = A_1 + A_2 \quad \rightarrow \mathtt{y(2)}$$

$$y(x_2) = \sum_{i=1}^{3} A_i = A_1 + A_2 + A_3 \quad \rightarrow \mathtt{y(3)}$$

... ...

$$y(x_N) = \sum_{i=1}^{N-1} A_i \quad \rightarrow \mathtt{y(N-1)}$$

➢ **exercise:**

- write a script **integratecos.m** that calculates the antiderivative of $\cos(x)$ on an interval $x \in [a,b]$

$$g(x) = \int_a^x \cos(s)\,ds$$

- use the formula I_{num}^{mid} together with `cumsum()` and compare the results to the analytical solution.

- **Note**:
  - when using `cumsum()` like described above, you get the anti-derivative at the points $x_2,\dots,x_N$
  - if you want the anti-derivative at all points x_i, you can add a 0 as the first element to the vector:

```
>> y = [0, cumsum()];
```

  - `y()` now contains the same number of points as `x()` where the first element is correctly 0

MATLAB *application – study of function*

We want to study the function

$$f(x) = -5x^2 \sin(x) + e^{x/2} \sqrt{x}$$

and its (anti-)derivative on the interval $[x_1, x_2]$.

➢ **exercise:**

- write a script with the name **function.m** in which you calculate and plot $f(x)$ on the interval $x_1=3,\ x_2=10$ using N=200 points.

➢ **exercise:**

- calculate the numerical derivative of $f(x)$ inside your script and plot the numerical derivative into the same figure using a line with a different colour.

➢ **exercise:**

- write a script-function **antiderivative.m** that returns the numerically determined anti-derivative for a function provided in a vector f(); the values of the anti-derivative should be given at the same values x() as the function f(). The function should work as follows:

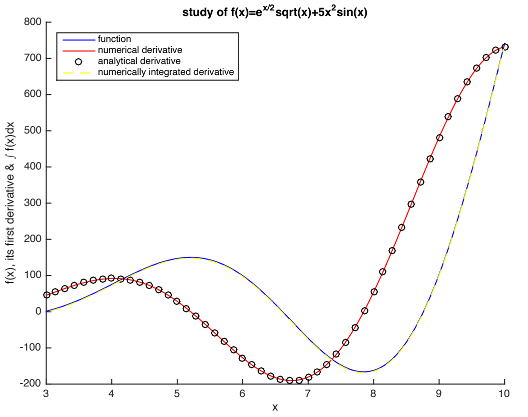
```
function [antideriv] = antiderivative(f, x)
% x() : vector containing the values of the independent variable x_i
% f() : vector containing the function values f(x_i)
% antideriv() : vector containing the value of the anti-derivative at all x_i
```

- use that function to calculate (and plot into the same figure) the anti-derivative

$$f(x) = \int \frac{df}{dx} dx$$

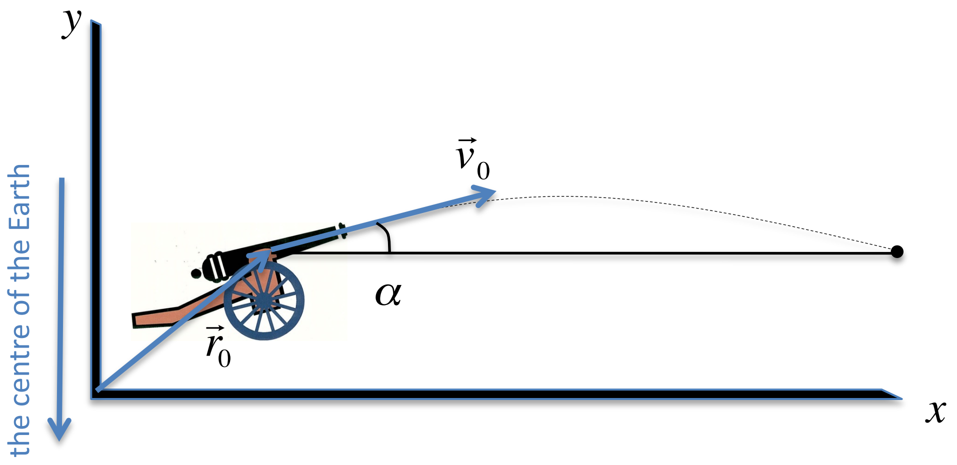
of your numerically obtained df/dx

▪ **Note:** your final plot should look like this



- **Cannonball Fever**

We intend to calculate and plot the flight path of a cannon ball



$$\vec{F} = -G\frac{mM}{r^2}\vec{e}_r \qquad \text{law of gravity}$$

$$\vec{F} = m\frac{d^2\vec{r}}{dt^2} \qquad \text{equation of motion}$$

$$\frac{d^2\vec{r}}{dt^2} = -G\frac{M}{r^2}\vec{e}_r$$

- the solution to the equations of motions for the cannonball is:

$$\vec{r}(t) = \vec{r}_0 + \vec{v}_0 t + \frac{1}{2}\vec{g}t^2$$
$$\vec{v}(t) = \vec{v}_0 + \vec{g}t$$

- an initial position $\vec{r}_0$
- an initial velocity $\vec{v}_0 = v_0\begin{pmatrix}\cos\alpha \\ \sin\alpha\end{pmatrix}$
- an initial angle α

- the value of the acceleration $\vec{g}$ felt by any object on the surface of the earth

$$\vec{g} = \begin{pmatrix} 0 \ km/s^2 \\ -9.81 \ km/s^2 \end{pmatrix}$$

- **Cannonball Fever**

We intend to calculate and plot the flight path of a cannon ball

➢ **exercise:**

• using the analytial solution (see previous page), plot the cannonball trajectory using a time intervall $t \in [0, -2v_{y,0}/g_y]$.
• plot $x(t)$ and $y(t)$ into a different figure.
• using the analytical solution for the velocityies, verify energy conservation:

$$E(t) = \frac{1}{2}mv^2(t) - mgy(t) = const.$$

➢ **exercise:**

• use your MATLAB vectors for $x(t)$ & $y(t)$ to obtain their numerical derivatives, i.e. $v_x(t)$ & $v_y(t)$
• compare the numerical velocities to the analytical ones by plotting them in the same figure.
• use your numerical vectors for velocities and positions to verify energy conservation again.

➢ **exercise:**

• use the numerical velocities to obtain the accelerations, too.
• compare the numerical accelerations against the analytical solution $a_x = 0, a_y = -g_y$
• ***hint***: you may have to use `ones()`

➢ **exercise:**

• place your cannon on the Moon and calculate the flight path
• compare the flight paths on Earth and Moon

➢ **Note:** use your favourite values for x_0, y_0, v_0, m, and α, but all different from zero.

- **trajectory of two particles**

Consider two particles with the following parametrized trajectories:

particle #1:

$$x_a = v \cdot t$$

$$y_a = 0$$

particle #2:

$$x_b = v \cdot t - D \cdot \tanh\left(\frac{v \cdot t}{D}\right)$$

$$y_b = D \cdot \operatorname{sech}\left(\frac{v \cdot t}{D}\right)$$

➢ **exercise:**

• write a script that calculates the position, velocity, and acceleration; velocities and accelerations have to be calculated numerically!

➢ **exercise:**

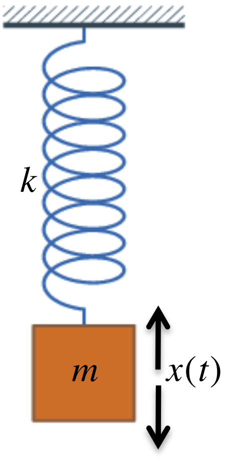
• plot the trajectories into one figure, the absolute value of the velocities in a second figure, and the absolute value of the acceleration in a third figure; use v=0.5m/s, D=5.0m, t_f=50s.

➢ **exercise:**

• calculate and plot the distance traversed by both particles as a function of time. This distance is calculated by either of the following two integrals:

$$d(t) = \int_0^t |d\vec{r}| = \int_0^t |\vec{v}(t')|\, dt'$$

▪ harmonic oscillator



- a harmonic oscillator obeys a 2nd order ordinary differential equation:

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x$$

...which describes the motion of a mass m which, displaced from its equilibrium position, experiences a restoring force proportional to the displacement x, i.e. $F = -kx$ (Hooke's law).

- the initial conditions x_0 and v_0 need to specified as follows

$$x_0 = x(t=0)$$
$$v_0 = v(t=0) = \left.\frac{dx}{dt}\right|_{t=0}$$

- the solution for the velocity is given by

$$v(t) = A\omega_0 \cos(\omega_0 t + \varphi) \quad \text{with} \quad \omega_0 = \sqrt{\frac{k}{m}}, \quad A = \sqrt{x_0^2 + \left(\frac{v_0}{\omega_0}\right)^2}, \quad \varphi = \arctan\left(\omega_0 \frac{x_0}{v_0}\right)$$

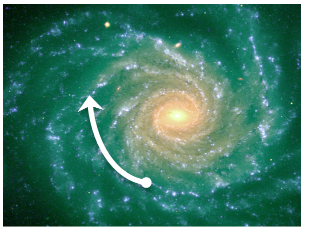
➢ **exercises:**

- plot $v(t)$ for t_0=0s, x_0=1.2m, v_0=3.7m/s, k=1.5kg/s², m=1.4kg
- numerically obtain $x(t) = x_0 + \int_{t_0}^{t} v(s)ds$ and plot it into the same figure
- numerically obtain $a(t) = dv/dt$ and plot it into the same figure
- plot the analytical $x(t)$ and $a(t)$ into the same figure, too

MATLAB *application – galaxy observations*

- **Tully-Fisher relation in Astronomy**

It has been observed that the more luminous a galaxy is the faster it rotates!



L_{galaxy1} < L_{galaxy2}

v_{galaxy1} < v_{galaxy2}

	M31	M33	M81	NGC2403	NGC4236	IC 2574	NGC 2366	NGC 5585	NGC 5204	Ho IV
v	546	242	530	306	202	126	129	214	151	110
M	–20.96	–18.66	–20.01	–19.17	–17.53	–16.69	–16.34	–18.05	–17.78	–16.35

original data from Tully & Fisher (1977)

➢ **exercises:**

- plot the data in an appropriate way
- try to determine the exponent p of the power-law $L \propto v^p$
- is there a way to sort the data prior to plotting?

- ***Note***: the absolute magnitude M relates to the luminosity like $M \propto -\log_{10}(L)$

mathematical/numerical operations							
+	-	*	/	^	.*	./	.^
in-built functions							
sqrt()	log()	log10()	exp()	sin()/asin()	cos()/acos()	tan()/atan()	rem()
floor()	ceil()	complex()	real()	img()	sign()	abs()	
in-built variables							
pi	eps	inf	nan	i	ans		
vector operations							
=	:	()	[]	linspace()	length()	diff()	sum()
max()	min()	prod()	cumsum()				
interactive/script commands							
;	,	%	clear	clc	home	pause	who
help	format	close					
plotting							
figure	close	plot	subplot	hold	legend	xlabel	ylabel
axis	xlim	ylim	grid	errorbar	plot3	text	
advanced commands							
logspace	loglog	semilogx	semilogy	fplot			

- **please familiarize yourself with all of these commands, functions, variables, etc., even if they have not been discussed in class:**

from now on you must know how to use them all!

MATLAB *summary*

- you further need to know the following...

  - dealing with vectors
    - generating vectors using “:” as well as linspace()
    - vectors as arguments of functions, e.g. sin(x)
    - selecting and modifying individual elements of a vector

  - writing and executing scripts

  - calculating a numerical derivative

  - calculating a numerical integral

  - plotting functions (of one variable)
    - using plot(), subplot(), hold
    - adjusting and polishing a figure, e.g. axis, grid, …