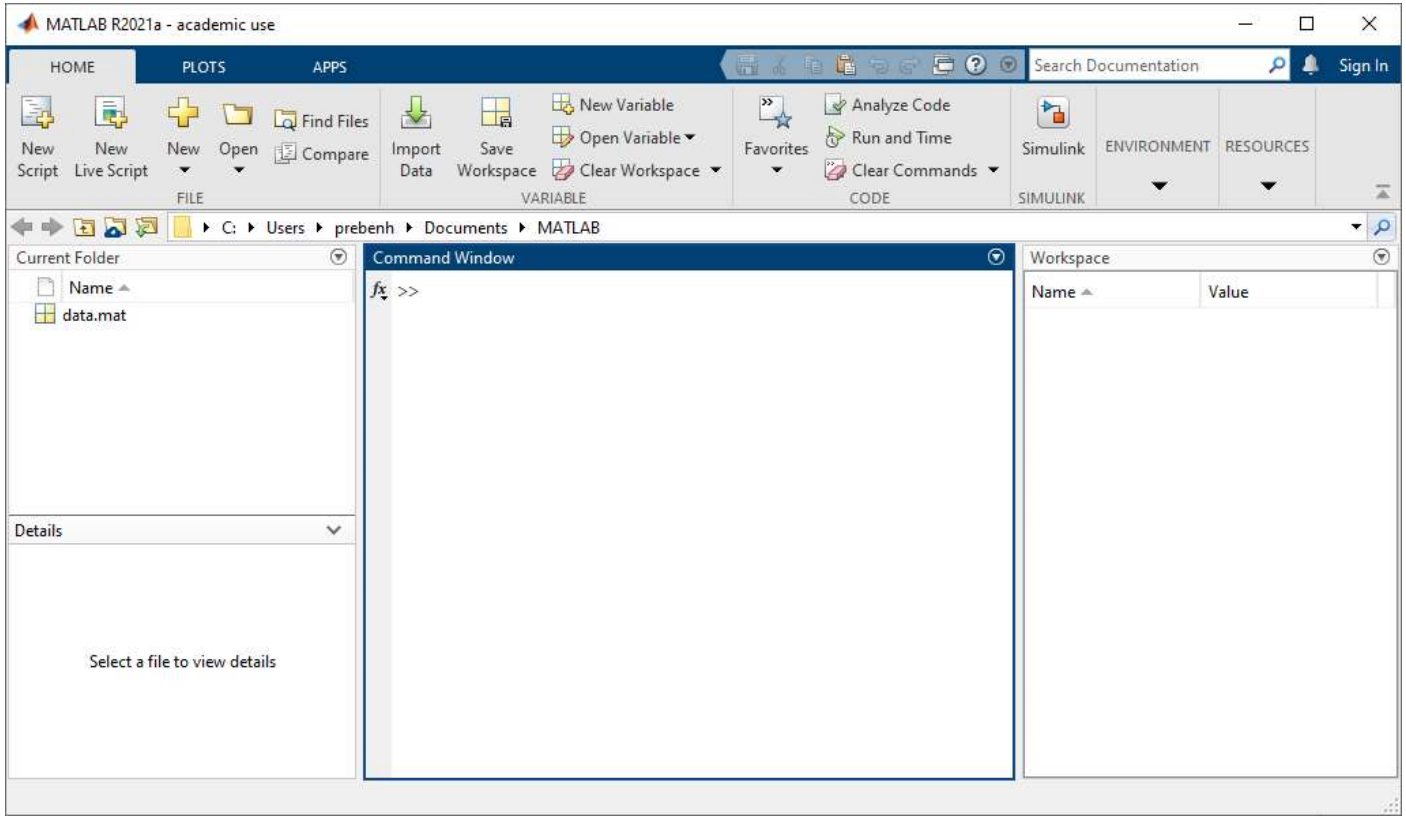
Matematiske metoder i programmering (MMP)

Lektion 1 - Matlab

Matlab


MATLAB R2021a - academic use
HOME
PLOTS
APPS
Search Documentation
Sign In
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
Analyze Code
Run and Time
Clear Commands
CODE
Simulink
SIMULINK
ENVIRONMENT
RESOURCES
C: ▸ Users ▸ prebenh ▸ Documents ▸ MATLAB
Current Folder
Name
data.mat
Details
Select a file to view details
Command Window
>>
Workspace
Name
Value

Entering commands

→ Basic multiplication

→
```
>> 3*5
ans =
    15
```

→ Assigning variables

→
```
>> m = 3*5
m =
    15
```

→
```
>> m = m+1
m =
    16
```

→
```
>> y = m/2
y =
    8
```

→
```
>> k = k-2;
>>
```

Command window

→ Previous command
  → Pil op ↑
→ View variable content
  → `>> y`

```
y =
     8
```

→ Assigning variables with variables
  → `>> avgAa = (a+A)/2`

```
avgAa =
     3
```

→ Variabelnavne
  → `>> 3sq=9`

```
 3sq=9
  ↑
Invalid expression. Check for missing multiplication operator, missing or unbalanced delimit
or other syntax error. To construct matrices, use brackets instead of parentheses.
```

Workspace

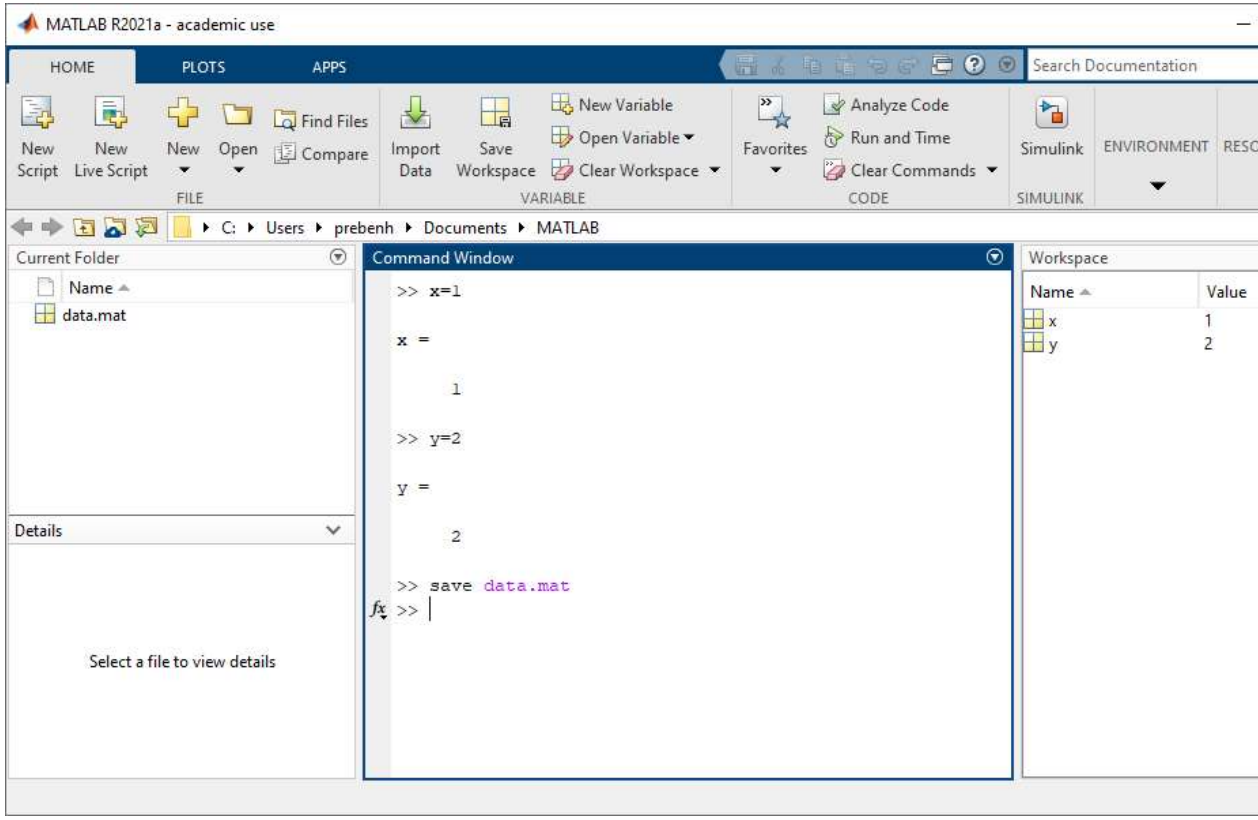
→ Workspaces can be saved
  → `>> save data.mat`

→ ”Content” is saved in working directory
  → Default
    → C:\users\prebenh\Documents\MATLAB

→ Loading the workspace
  → `>> load data.mat`
  → `>> load 'data.mat'`

→ Clear the workspace
  → clear

→ Clear command-window (right click)

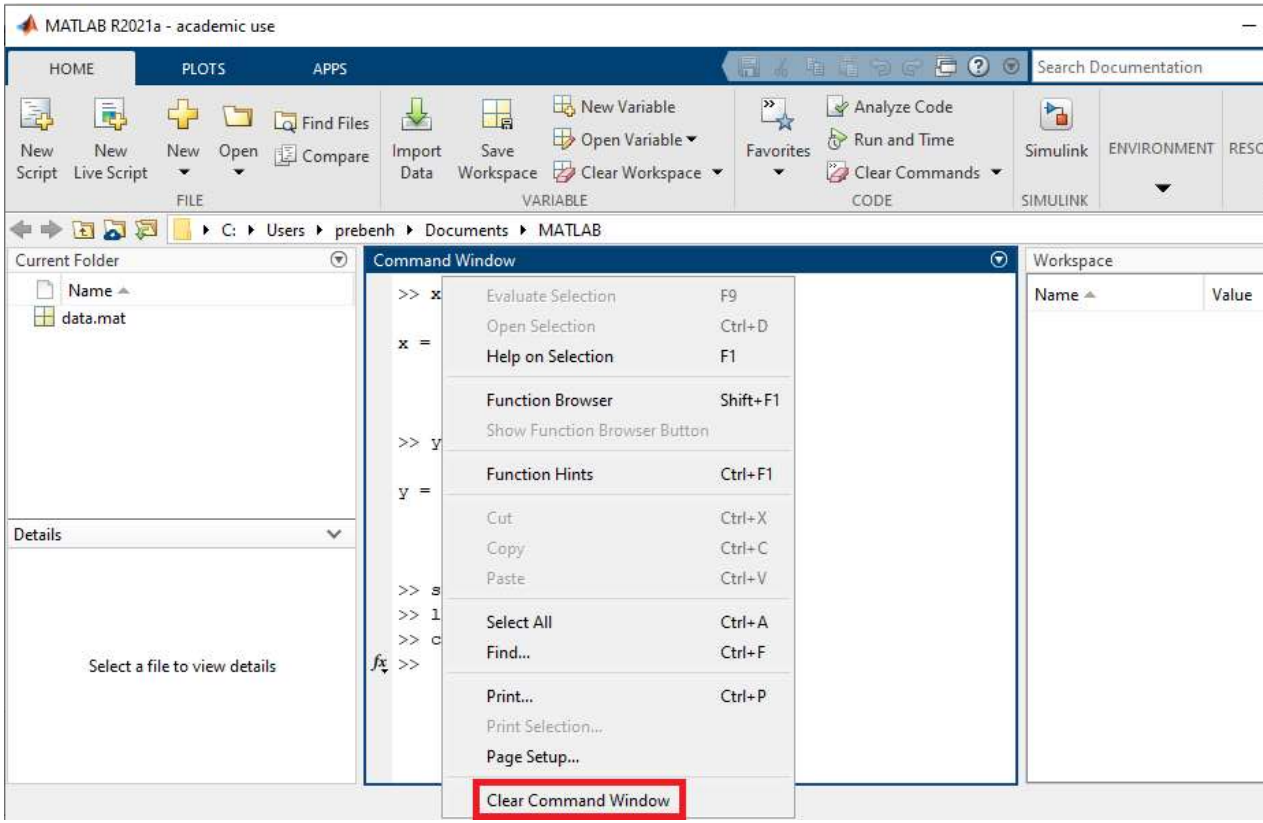


SDU

Assignment 1

→ Open Matlab
→ Put into console:
  → `x=1`
  → `y=2`
  → `save 'data.mat'`
  → `clear`
  → right click and choose "Clear command window"
    → or just type: `clc`
  → notice the right part of window "Workspace"
  → `load 'data.mat'`
  → `x`
  → `y`

→ Check that the variables have been saved



More workspace

→ Clear a single variable
  → `>> clear var`

→ Save a single variable to file
  → `>> save datafile.mat var`

→ Load a single variable from a file
  → Try this now:

```
a = 2;
b = 3;
save datafile.mat
clear
load datafile.mat b
```

Built in constants and functions

→ Constant π
  → `>> a = pi`
  ```
  a =
      3.1416
  ```

→ Trigonometric functions
  → `>> cos(2)`
  ```
  ans =
      -0.4161
  ```
  → `>> sin(pi)`
  ```
  ans =
      1.2246e-16        (practically zero, but matlab is a numeric tool)
  ```

→ Other functions
  → sqrt(2), abs(-2)

Numerisk præcision og visning

→ Sqrt(2) umiddelbart upræcis

→ `>> z = sqrt(2)`

```
z =
    1.4142

>> format long
z
z = 1.414213562373095
```

→ Format SHORTENG

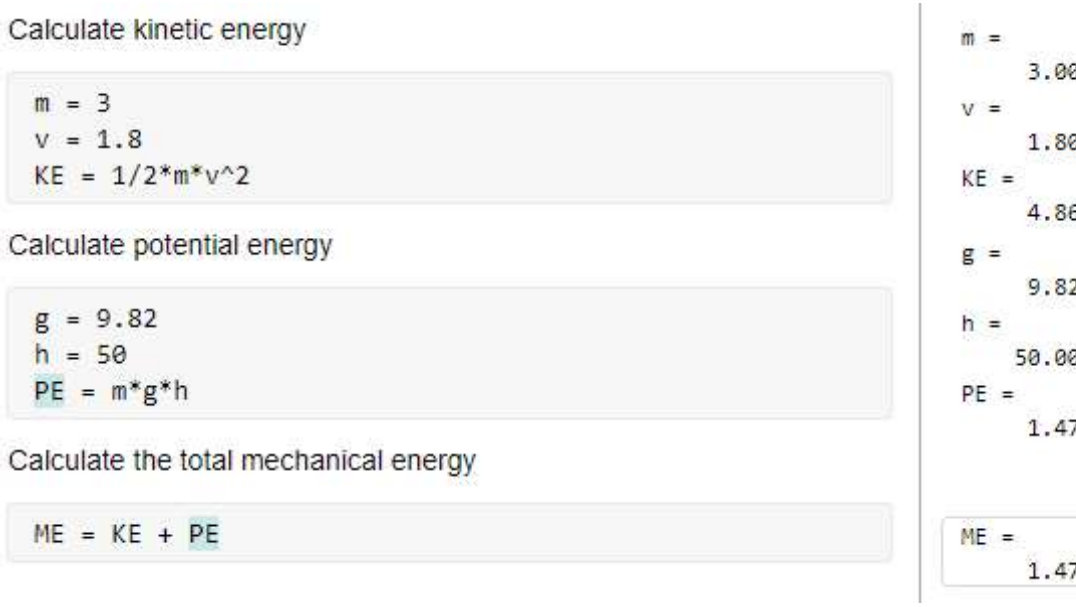
→ `>> format shorteng`

```
>> z
z =
     1.4142e+000
```

→ Standard: `format short`

Live script

→ Pænere udsende
→ Virker som en notesblok
med udregninger mellem "formler"
→ Relativt nyt i matlab

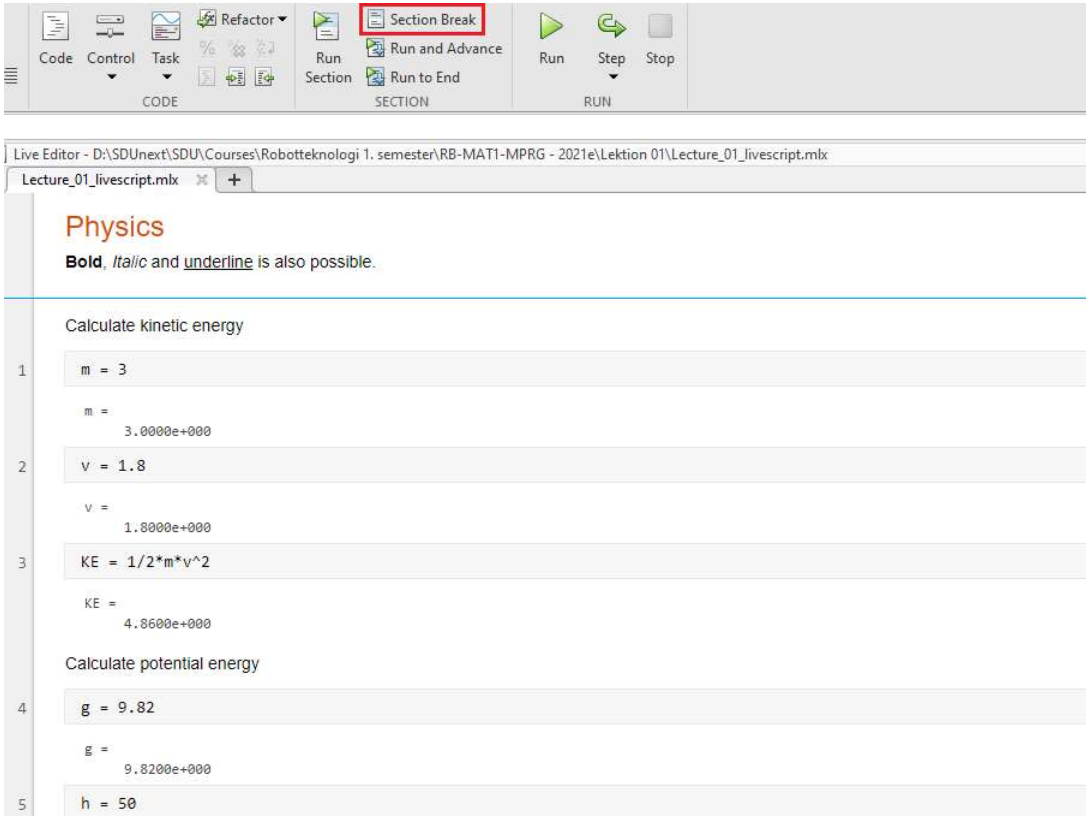


Live script output

→ Output in different modes

Formatting the live script

→ Headings
→ Bold
→ Italic
→ Underline
→ Bullet points
→ Sections



SDU

Arrays (vectors and matrices)

→ A single number: scalar
  → Could also be an array of dimension `1x1` (1 row, 1 column structure)
→ An array is usually a collection of numbers in a "row" (`1 x n` elements forms a row vector)
  → Like {1,2,3,4}
  →
```
>> x = [1, 2, 3, 4]
x =
     1     2     3     4
```
→ An array can also be like a column vector (`n x 1` elements)
  → E.g. $x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$
  →
```
>> x = [1; 2; 3]
x =
     1
     2
     3
```

Arrays (matrices)

→ A matrix is a collection of elements in a `m x n` structure

→
```
>> x=[5 6 7; 6 7 8]
x =
     5     6     7
     6     7     8
```

→
```
>> x = [sqrt(10) pi^2]
x =
    3.1623    9.8696
```

Evenly spaced vectors

→ A row vector {1, 2, 3, 4, 5, 6}

→ `>> x = 1:6`

```
x =
     1     2     3     4     5     6
```

→ A row vector of {20, 22, 24, 26, 28}

→ `>> x = 20:2:28`

```
x =
    20    22    24    26    28
```

→ Linspace: a fixed number of points evenly spaced (from 0 to 1, 5 elements)

→ `>> x = linspace(0,1,5)`

```
x =
    0    0.250    0.500    0.750    1.000
```

Transpose of vectors and matrices

→ Convert "rows to columns" AND "columns to rows"

→
```
>> x = 1:3;
>> x = x'
x =
     1
     2
     3
```

→ Create a transposed vector in one line

→
```
>> x = [1 3 5]'
x =
     1
     3
     5
```

Assignment

→ Create an evenly spaced row vector with 19 elements starting at 1 and ending at 10
→ Save the result in a variable x
→ Transpose the matrix to get a column vector and save the result in y
→ Print the result

More matrix functions

→ Random matrix, matrix of zeros and ones

→ >> rand(2)

```
ans =
    0.9575    0.1576
    0.9649    0.9706
```

→ >> rand(2, 3)

```
ans =
    0.9572    0.8003    0.4218
    0.4854    0.1419    0.9157
```

→ >> zeros(1,2)

```
ans =
     0     0
```

→ >> ones(2, 3)

```
ans =
    1     1     1
    1     1     1
```

Matrix size and indexing

→ Size of a matrix

→
```
>> x = [1 1 1; 1 1 1];
>> size(x)
ans =
     2     3
```

→ Indexing

→
```
>> x = rand(4)
x =
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    0.3816
    0.7060    0.0971    0.9502    0.7655
```

→
```
x(3,2)
ans =
    0.0462
```

Matrix indexing continued

→ End of row or column
  → `>> x(end,1)`
  ```
  ans =
      0.7060
  ```
  → `>> x(1,end)`
  ```
  ans =
      0.0344
  ```

→ Using end in other ways
  → `>> x(end-1, end-2)`
  ```
  ans =
      0.0462
  ```

→ One index (first element of row)
  → `>> x(2)`
  ```
  ans =
      0.6555
  ```

```
x =
    0.3922    0.0318    0.8235
    0.6555    0.2769    0.6948
    0.1712    0.0462    0.3171
    0.7060    0.0971    0.9502
```

Matrix indexing continued

→ Row of matrix

→ `>> x(2,:)`

```
ans =
    0.6555    0.2769    0.6948    0.4387
```

→ Column of matrix

→ `>> x(:,3)`

```
ans =
    0.8235
    0.6948
    0.3171
    0.9502
```

```
x =
    0.3922    0.0318    0.8235
    0.6555    0.2769    0.6948
    0.1712    0.0462    0.3171
    0.7060    0.0971    0.9502
```

Matrix indexing continued

→ Part of a row/column of a matrix

→ `>> x(1:3,:)`

```
ans =
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    0.3816
```

→ `>> x(2:3,1:2)`

```
ans =
    0.6555    0.2769
    0.1712    0.0462
```

→ Multiple elements of matrix in out of order structure

→ `>> x([3 4 2], [1 2 2])`

```
ans =
    0.1712    0.0462    0.0462
    0.7060    0.0971    0.0971
    0.6555    0.2769    0.2769
```

```
x =
    0.3922    0.0318    0.8235
    0.6555    0.2769    0.6948
    0.1712    0.0462    0.3171
    0.7060    0.0971    0.9502
```

Matrix indexing continued

→ Single elements of matrix

→ `>> x(3,4)`

```
ans =
    0.3816
```

→ Changing the value of a matrix element

→ `>> x(3,4) = 2.1`

```
x =
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    2.1000
    0.7060    0.0971    0.9502    0.7655
```

→ `>> x(3,4) = x(3,2)`

```
x =
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    0.0462
    0.7060    0.0971    0.9502    0.7655
```

```
x =
    0.3922    0.0318    0.8235
    0.6555    0.2769    0.6948
    0.1712    0.0462    0.3171
    0.7060    0.0971    0.9502
```

Operations on arrays (matrices)

→ Add a value to each element in array

→ `>> x+1`

```
ans =
    1.3922    1.0318    1.8235    1.0344
    1.6555    1.2769    1.6948    1.4387
    1.1712    1.0462    1.3171    1.3816
    1.7060    1.0971    1.9502    1.7655
```

→ Add two vectors or matrices

→ `>> v1+v2`

```
ans =
     9
    17
    13
    10
```

```
x =
    0.3922    0.0318    0.8235
    0.6555    0.2769    0.6948
    0.1712    0.0462    0.3171
    0.7060    0.0971    0.9502

v1 =
     7
     9
    10
     5

v2 =
     2
     8
     3
     5
```

Operations on arrays (matrices)

→ Division and multipliation

→
```
>> (v1+v2)/2
ans =
    4.5000
    8.5000
    6.5000
    5.0000
```

→ Max/min numbers

→
```
>> avgMax = max((v1+v2)/2)
avgMax =
    8.5000
```

→ Afrunding

→
```
>> round(avgMax)
ans =
    9
```

Operations on arrays (matrices)

→ Rounding elements in matrix

→ `>> round(v1+v2)`

```
ans =
     9
    17
    13
    10
```

→ Element wise product of two vectors

→ `>> v1 .* v2`

```
ans =
    14
    72
    30
    25
```

```
v1 =

v2 =
```

Multiple outputs from functions

→ "Row wise" multiplication

→ `>> m.*v1`

```
ans =
     7    14
    27    36
    50    60
    35    40
```

```
v1 =
     7
     9
    10
     5

m =
     1
     3
     5
     7
```

→ Naming variables with multiple output functions

→ `>> [mrow mcol] = size(m)`

```
mrow =
     4
mcol =
     2
```

Multiple outputs from functions

→ Maximum value with index

→ `>> [v1Max, v1Idx] = max(v1)`

```
v1Max =
    10
v1Idx =
     3
```

→ Omit the actual maximum value

→ `>> [~, v1Idx] = max(v1)`

```
v1Idx =
     3
```

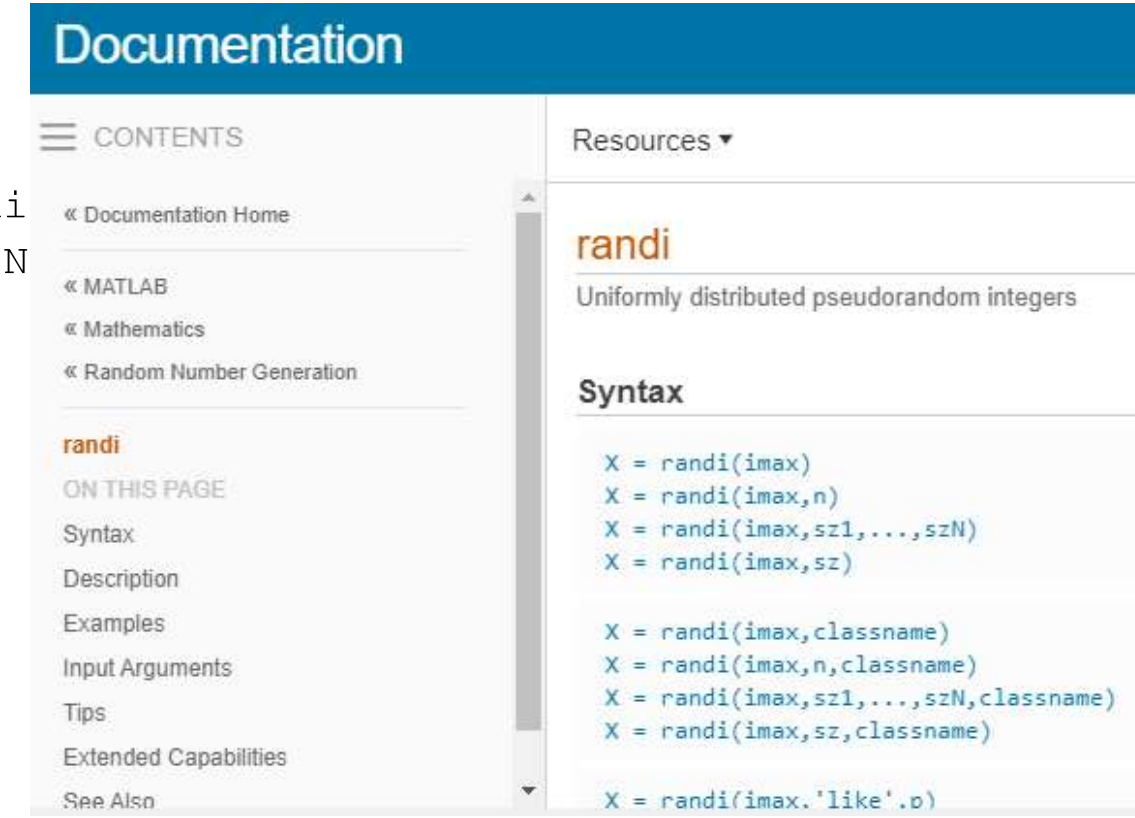
```
v1 =
     7
     9
    10
     5
```

Documentation help

→ >> doc Randi

→ >> help Randi

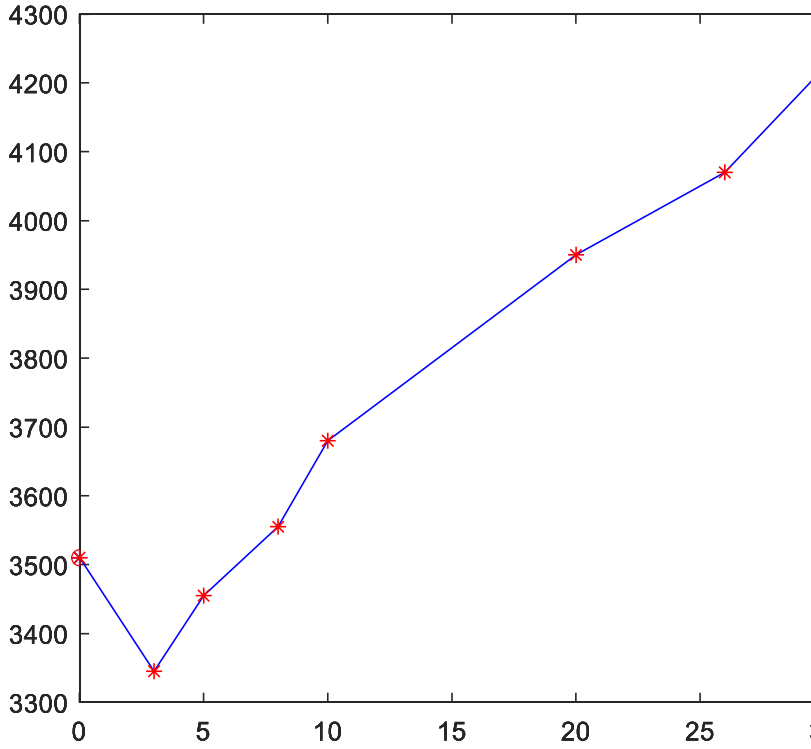
```
   randi Pseudorandom integers from a uni
      R = randi(IMAX,N) returns an N-by-N
      matrix containing pseudorandom
      integer values drawn from the
      discrete uniform distribution on
      1:IMAX. randi(IMAX,M,N) or
      randi(IMAX,[M,N]) returns an
      M-by-N matrix.

      ...
```



Figures

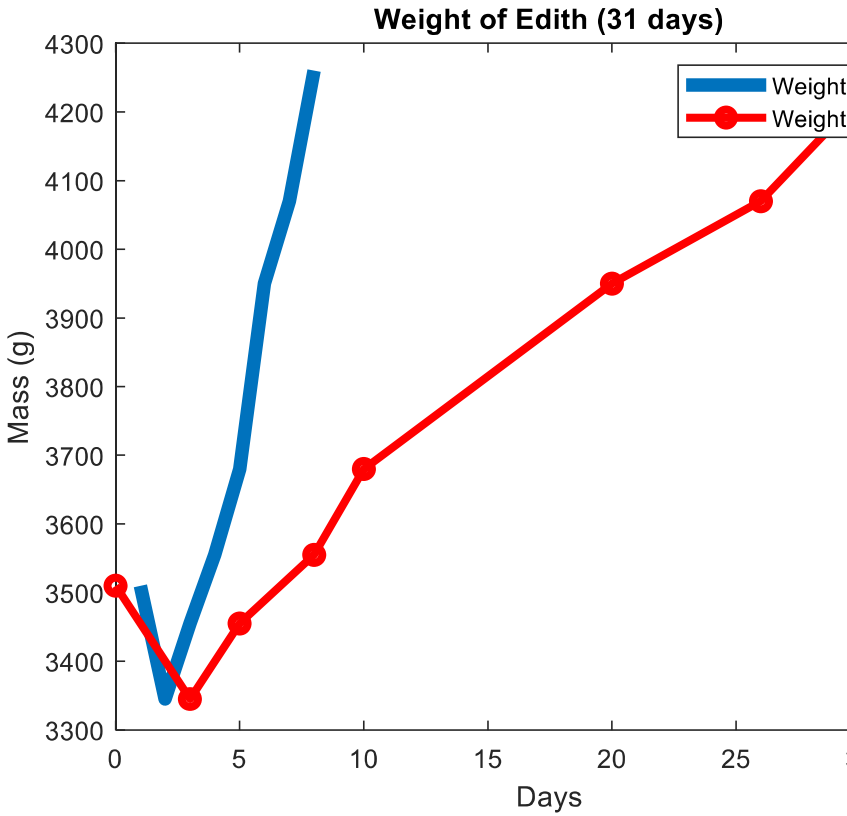
→ Figures

```
→ >> weight=[3510 3345 3455 3555 3680 3950 4070 4260];
→ >> days=[0 3 5 8 10 20 26 31];
→ >> plot(days, weight)
→ >> figure
→ >> plot(days, weight, "r--o")
→ >> figure(1)
→ >> plot(days, weight, "b*")
→ >> figure(2)
→ >> plot(days, weight, "b-")
→ >> hold on
→ >> plot(days, weight, "r*")
→ >> hold off
```



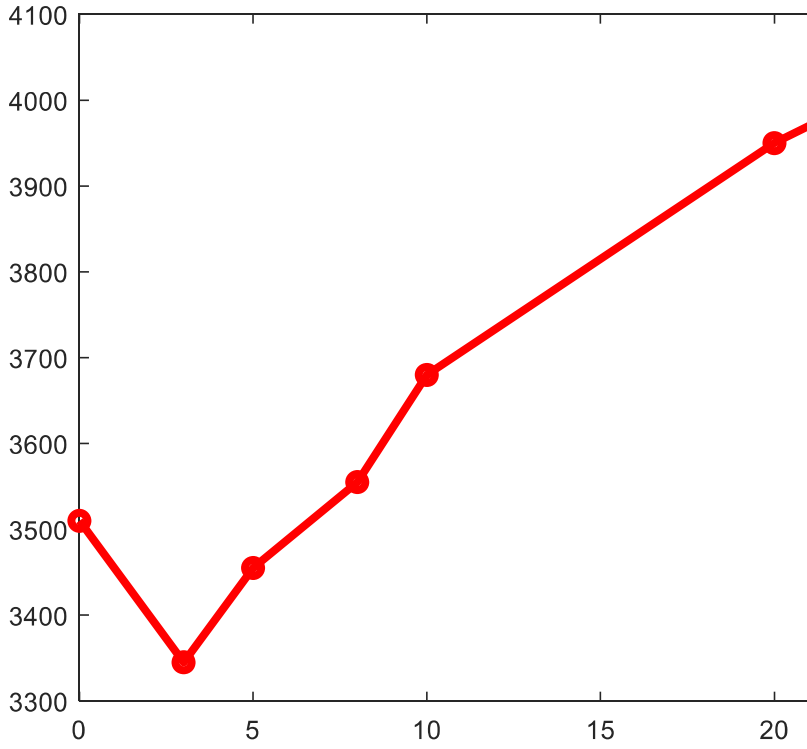
Plotting vector data

- → x-axis is just a vector with range from 1:n
  - → `>> plot(weight)`
- → Properties
  - → Linewidth
    - → `>> plot(weight, "LineWidth", 5)`
  - → Combined
    - → `>> plot(days, weight, "ro-", "LineWidth", 3)`
- → Multiple graphs, axes labels and legends
  - → `>> plot(weight, "LineWidth", 5)`
  - → `>> hold on`
  - → `>> plot(days, weight, "ro-", "LineWidth", 3)`
  - → `>> hold off`
  - → `>> ylabel("Mass (g)")`
  - → `>> xlabel("Days")`
  - → `>> title("Weight of Edith (" + max(days) + " days)")`
  - → `>> legend("Weight no days", "Weight with days")`



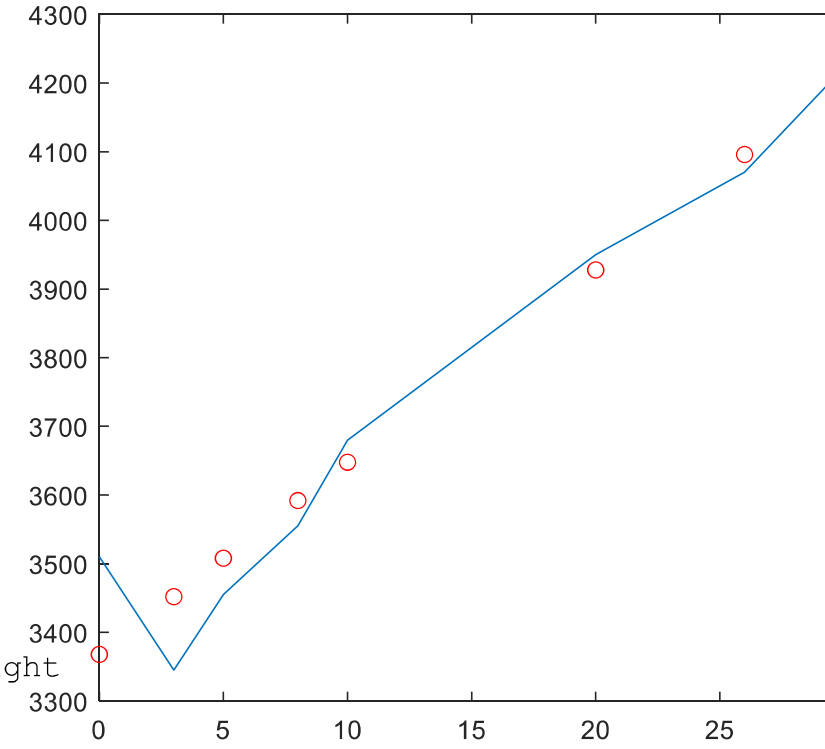
Plotting vector data

→ Limit the x-axis
  → `>> plot(days, weight, "ro-", "LineWidth", 3)`
  → `>> xlim([0,25])`



Working with Tables

- In class tutorial
  - Load babyweight.xlsx
  - Change variable name from Weight to w using the editor
  - Change back the variable name using
    - `>> babyweight.Properties.VariableNames{2} = 'Weight';`
  - Plot the data
    - `>> x=babyweight.Days`
      `>> y=babyweight.Weight`
    - `>> plot(x,y)`



- You can add columns to the table
  - Model 28*x + 3367.8
    - `>> babyweight.Linear = 28*x + 3367.8`
    - `>> babyweight.LinearDiff = babyweight.Linear - babyweight.Weight`
    - `>> hold on`
    - `>> plot(babyweight.Days, babyweight.Linear, "ro")`
    - `>> hold off`
  - Show babyweight table

Working with Tables

```
>> element = ["Hydrogen" "Helium" "Lithium" "Beryllium" "Boron" "Carbon" "Nitrogen" "Oxygen"
>> density = [0.0899 0.1785 535 1848 2460 2260 1251 1429]'
>> elements = table(element, density)
```

element	density
"Hydrogen"	0.0899
"Helium"	0.1785
"Lithium"	535
"Beryllium"	1848
"Boron"	2460
"Carbon"	2260
"Nitrogen"	1251
"Oxygen"	1429

Working with Tables

→ Sorting

→ `>> sortrows(elements, 'density')`

```
ans =
      element       density
    ____________    _______

    "Hydrogen"      0.0899
    "Helium"        0.1785
    "Lithium"          535
    "Nitrogen"        1251
    "Oxygen"          1429
    "Beryllium"       1848
    "Carbon"          2260
    "Boron"           2460
```

Logical tests on arrays

→ Testing elements of inequality

→
```
>> r = randi(10,[1 10])
r =
     7     8     1    10     8     5     5     5     4     6
>> r < 5
ans =
    0   0   1   0   0   0   0   0   1   0
>> r(r<5)
ans =
     1     4
>> r(r<5) = 0
r =
     7     8     0    10     8     5     5     5     0     6
```

→ Testing example 2

→
```
>> r(r==5) = 0
r =
     7     8     0    10     8     0     0     0     0     6
```

Logical tests on arrays

```
→ >> r = [7 8 1 10 8 5 5 5 4 6];
  >> r(r<5 | r==5) = 0
  r =
       7     8     0    10     8     0     0     0     0     6
```

Homework 1: Arrays and vector (easy)

1. Create a column vector of the integers 1 to 5
   → By entering each value manually
   → Using the colon operator
2. Create a 5 by 5 matrix where each element is equal to its row number
   → Using row/column indexing with the colon operator to assign values
   → Using the vector created in step 1 and the concatenation operator (square brackets)
3. Recreate the array
   → 0 0 0 0 0 0 0
   0 1 1 1 1 1 0
   0 1 2 2 2 1 0
   0 1 2 3 2 1 0
   0 1 2 2 2 1 0
   0 1 1 1 1 1 0
   0 0 0 0 0 0 0
4. Load the file randomChars.mat and change a portion of the array to the string 'MATLAB'
5. Display the array from step 3 as an image with imagesc(array)

Homework 2: Cassini 1 (easy/medium)

→ The Cassini-Huygens was a mission to study the planet Saturn and its moons. The payload consisted of the Cas orbiter and the Huygens probe. The craft was launched in October 1997 and reached Saturn in 2004. Cassini-Huygens' path consisted of four different gravity assist maneuvers.

1. Load the file cassiniData1.mat
2. Separate the columns of the data array into inidividual arrays for Time, Year, Month, Day, Radius, Latitude and Longitude (Ti years and fractions of years, Radius is distance to sun measured in AU)
3. Plot Radius vs. Time
4. Find the index and minimum radius of the sun
    1. Use the index to find the year, month and day when the distance to the sun was at a minimum

Homework 3: Cassini 3 (harder)

1. Load cassiniData2
   → X, Y, Z Coordinates are available, but not radius
   Calcuate the Radius: sqrt(x^2+y^2+z^2)
2. The planets in our solar system has a radius to earth represented by these arrays:
   → planetNames = {'Mercury','Venus','Earth','Mars','Jupiter','Saturn','Uranus','Neptune'};
   planetRadii = [0.39,0.72,1.0,1.51,5,9.04,19.18,30.06];
   → Find the index'es of when the Cassini orbiter passes the radius of the planets.
   → Use the index'es to find the date when the planets (radius') are passed
3. Use the calculated radius to make a plot where each year is represented by a new color (e.g. 1997 path being 1998 path being green, etc).
4. Add a circle for each planet to the plot where the radius of the planet is used.
   → HINT: create a unit circle and multiply by radius

Homework 4: Natick Hourly Temperatures (medium

1. Load the file natickData.mat and read the description
2. Calculate the mean daily temperature for each month
3. Find the standard deviation for each month and determine which month had the largest standard deviation
4. Determine the month with the largest temperature range
5. Plot the temperature versus time for each month. Add a legend

Homework 5: Edinburgh marriages (medium)

→ Add *edinburgh_marriages.xls* to working directory for Matlab
→ Import *edinburgh_marriages.xls* to Matlab
→ Open the file using the Variable Editor i Matlab. Replace the missing value in year 1695 with the average data-va
  1694 and 1696
→ Create new variables *Year* and *M* from the two columns of *data*
→ Plot the *M* variable on the y-axis and use *Year* as the x-axis
  → Add legends, labels on axes and a title
  → Save the output in a png-file and an svg file
→ Try to use the right click option in Workspace to plot the variables *year*, *M*
  → Click both and right click
  → Try reversing the selection order
→ Save *year* and *M* in a .mat-file
→ In the plot window, choose *Data Statistics* from the *Toolbox*-menu to find the standard deviation