Discrete Event Simulation

IN2045

Dr. Alexander Klein
Stephan Günther
Prof. Dr.-Ing Georg Carle

Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
http://www.net.in.tum.de

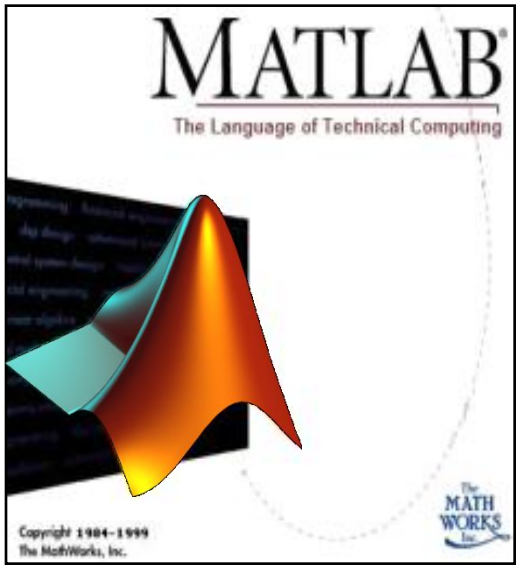
Matlab - Tutorial

- Introduction
- Basics
- Plots

Exercise Block A

- Fitting
- Practical Examples
- Advanced Topics

Exercise Block B



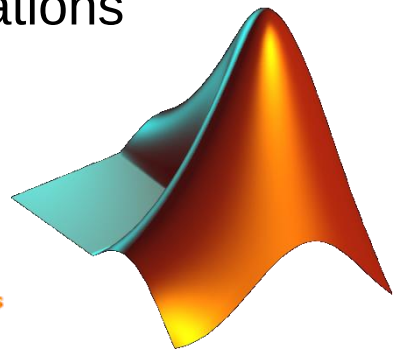
Alternatives:

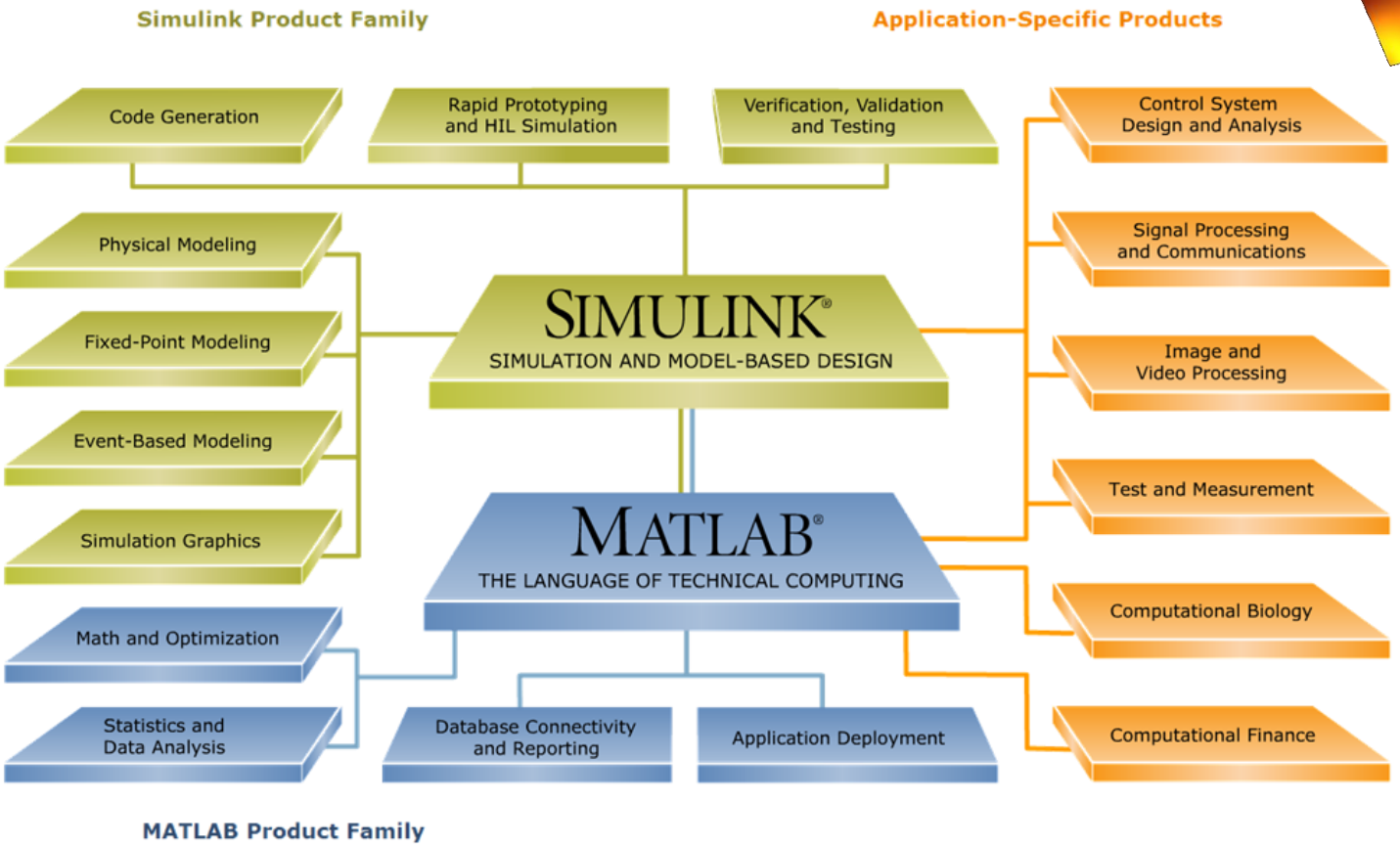
- Commercial
  - Mathematica
    - Large library
  - Excel
    - Limited number of mathematical functions
    - Only efficient on small data sets
- Maple
  - Large library

- Free
  - Octave
    - Free software
    - Partially compatible to MATLAB (available as plugin)
  - R (Project)
    - Free software
    - Powerful function library
    - http://www.r-project.org/
  - Freemat
    - Matlab clone
    - Limited function library

Matlab - Tutorial

- One of the most popular pograms for numeric calculations
- Large library of mathematic functions
- Provides methods for statistical evaluation and visualization





- Vectors:
  - **Generation:**
    - Example 1:
      - Matlab: a = [1 2 3 4 5 6]
      - Output: a = 1 2 3 4 5 6
    - Example 2:
      - Matlab: a = [1 2 3 4 5 6];
      - Output: none

Semi-colon prevents output on the console

    - Example 3:
      - Matlab: a = 1:6
      - Output: a = 1 2 3 4 5 6

Colon represents the most practical way to generate vectors and matrices

    - Example 4:
      - Matlab: a(1:6) = 1
      - Output: a = 1 1 1 1 1 1

Index 1,2,..,6 of vector a are set to 1

Matlab – Tutorial – Basics

- Vectors:
  - **Operation:**
    - a = [1 2 3 4 5 6]; b = [1 1 1 2 2 2];
    - Example 1 – Addition / Subtraction:
      - Matlab: c = a + b
      - Output: c = 2 3 4 6 7 8
    - Example 2 – Addition / Subtraction:
      - Matlab: c = a – 1
      - Output: c = 0 1 2 3 4 5
    - Example 3 – Transpose:
      - Matlab: a‘
      - Output: a =

        1

        …

        6

- Vectors:
  - **Operation:**
    - a = [1 2 3 4 5 6]; b = [1 1 1 2 2 2];
    - Example 4 – Multiplication:
      - Matlab: c = 2 * a
      - Output: c = [2 4 6 8 10 12]

The scalar operation is performed on every index of the vector

    - Example 5 – Multiplication:
      - Matlab: c = a*b‘
      - Output: c = 36
    - Example 6 – Multiplication:
      - Matlab: c = a .* b
      - Output: c = 1 2 3 8 10 12

Dot indicates index-wise multiplication
c(1)=a(1)*b(1), …. c(n) = a(n)*b(n)

Analogue - Division

- **: - Colon:**
  - **Usage:**
    - Start value : Maximum Value

      or
    - Start value : Step size : Maximum Value

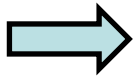
    - Matlab assumes a step size of one if no step size is given
    - Matlab adds step size to the start value as long as the calculated value is smaller than the maximum value

    - Example 1:
      - Matlab: a = 2:2:10
      - Output: a = 2 4 6 8 10

    - Example 2:
      - Matlab: a = 2:2:9
      - Output: a = 2 4 6 8

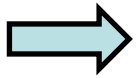
- **Matrices:**
  - Generation:
    - Example 1:
      - Matlab: `m = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]`
      - Output:
        ```
        m = 1  2  3
            4  5  6
            7  8  9
        ```
    - Example 2:
      - Matlab: `m(1:3,1:3) = 1`
      - Output:
        ```
        m = 1  1  1
            1  1  1
            1  1  1
        ```

Matrices:

- Access:
  - a = 1 2 3
        4 5 6
        7 8 9  a = [1 2 3 4 5 6 7 8 9]
  - Element: a (i , j)
  - Row: a (i , :)
  - Column: a (: , j)
  - Sub matrix: a (1:2, 2:3) = 2 3
                               5 6

Note: It is also possible to access matrix elements like vectors

Basic functions:

- Sum (vector), mean (vector):
  - a = 1:9  a = [1 2 3 4 5 6 7 8 9]
  - Example 1:
    - Matlab: c = sum(a)
    - Output: c = 45
  - Example 2:
    - Matlab: c = mean(a)
    - Output: c = 5

Note: Matlab supports a large number of very useful basic functions! Explore the help doc before rewriting a function.

Basic functions:

- a = [9 3 4 3 6]

- Find (condition) – Returns a vector of indices of elements which hold the given condition

- Example 1:
  - Matlab: c = find (a > 5) ~= a(a>5)
  - Output: c = 1 5

- Example 2:
  - Matlab: c = a(find (a > 5))
  - Output: c = 9 6 ⇨ Contains all values of a which are larger than 5

- **Basic functions:**
  - a = [9 3 4 3 6]
  - **Minimum** min(**vector**) / **Maximum** min(**vector**)
    - Example 1:
      - Matlab: c = min (a)
      - Output: c = 3
  - a = [3 2 5
    4 3 2
    5 1 2]
    - Example 2:
      - Matlab: c = min (a)
      - Output: c = 3 1 2

Min and Max return the minimum / maximum element of each row if a matrix is passed as argument to the function

- **Basic functions:**
  - a = [9 3 4 3 6]
  - **Sort** - Sort(vector)
    - Example 1:
      - Matlab: c = sort (a)
      - Output: c = 3 3 4 6 9
  - a = [3 2 5
    5 3 2
    4 1 2]
    - Example 2:
      - Matlab: c = sort (a)
      - Output: c = 3 1 2
        4 2 2
        5 3 5

Sort returns the sorted column vectors if a matrix is passed as argument to the function

- **Basic functions:**
  - a = [1 2 3
    4 5 6
    7 8 9]

    b = [1 0 0
    0 1 0
    0 0 1]

  - **Matrix concatenation:**
    - Example 1:
      - Matlab: c = [a b]
      - Output:
```
c =   1  2  3  1  0  0
      4  5  6  0  1  0
      7  8  9  0  0  1
```
    - Example 2:
      - Matlab: c = [a ; b]
      - Output:
```
c =   1  2  3
      4  5  6
      7  8  9
      1  0  0
      0  1  0
      0  0  1
```

Basic functions:

- a = [0 1 2 3 4 5 6]

b = [0 1
1 0]

- **Save a variable** – save('absolute_file_path', 'name_of_variable', options)
  - Example 1:
    - Matlab: save ('d:\workdir\variable_a.txt', 'a', '-ascii')
    - Creates a file named variable_a.txt in the workdir folder with the following content:

» 0.0000000e+000 1.0000000e+000 2.0000000e+000 3.0000000e+000 4.0000000e+000 5.0000000e+000 6.0000000e+000 (\n)

Note that Matlab does not store the name of the variable or any other related information in the file if the –ascii option is used!

- **Basic functions:**
  - a = [0 1 2 3 4 5 6]

    b = [0 1
    1 0]

  - **Save a variable** – save('absolute_file_path', 'name_of_variable', options)
    - Example 2:
      - Matlab: save ('d:\workdir\variable_a.mat', 'a', 'b', '-mat')
      - Creates a file named variable_a.mat in the workdir folder which contains both variables in a Matlab format
      - The variables can be loaded directly from the file into the workspace

Note that Matlab will create or overwrite variables in the workspace which have the same name as those stored in the file!

Basic functions:

- **Load a variable** – load(‘absolute_file_path‘)
  - Example 1:
    - Matlab: load (‘d:\workdir\variable_a.mat‘)
    - Generates variables the stored variables in the workspace.
    - It also possible to only partially load the file
    - See ‘help load‘ for more information

Note that Matlab supports a large number of different file types such as xls or other spreadsheet formats

Basic functions:

Random numbers

- **Rand** (#rows, #columns) ⇒ Returns uniform distributed values

$$RV\ X \sim U(0,1)$$

- **Randn** (#rows, #columns) ⇒ Returns normal distributed values

$$RV\ X \sim N(0,1) \quad (\mu = 1, \sigma = 1)$$

- Example 1:
  - Matlab: c = rand(100,1)
  - Output: c = [0.2344 ….]

- Example 2:
  - Matlab: c = rand(m,n)
  - Output: c = Matrix (m,n) containing uniform distributed random numbers

- **Basic functions:**

  sum(Vector), prod(Vector)

  - s = [1 2 3 4 5]
    - Examples:
      - sum(s) = $\sum_{i=1}^{i=\text{length}(s)} s(i) = 15$
      - prod(s) = $\prod_{i=1}^{i=\text{length}(s)} s(i) = 120$

  ceil(value)

  Rounds the element to the nearest integer greater than or equal to the element.

  floor(value)

  Rounds the element to the nearest integer lower than or equal to the element.

- **Basic functions:**

  cumsum(Vector), cumprod(Vector)

  Cumsum and cumprod return the cumulative sum / product of the elements of the input vector.

  - s = [1 2 3 4 5]

    - Example 1:
      - Matlab: result = cumsum(s)
      - Output: result = 1 3 6 10 15

    - Example 2:
      - Matlab: result = cumprod(s)
      - Output: result = 1 2 6 24 120

  Both functions are very useful, especially cumsum which is typically used to calculate cumulative density functions.

Basic functions:

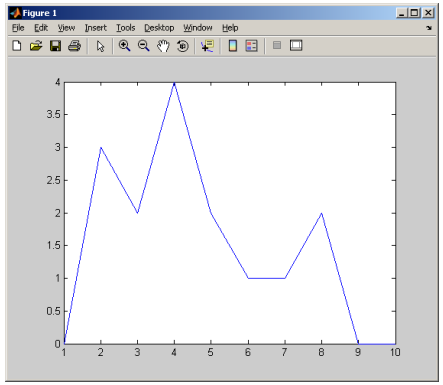
Histc(Vector sample, intervals)

Histc counts the number of values that are within a certain interval

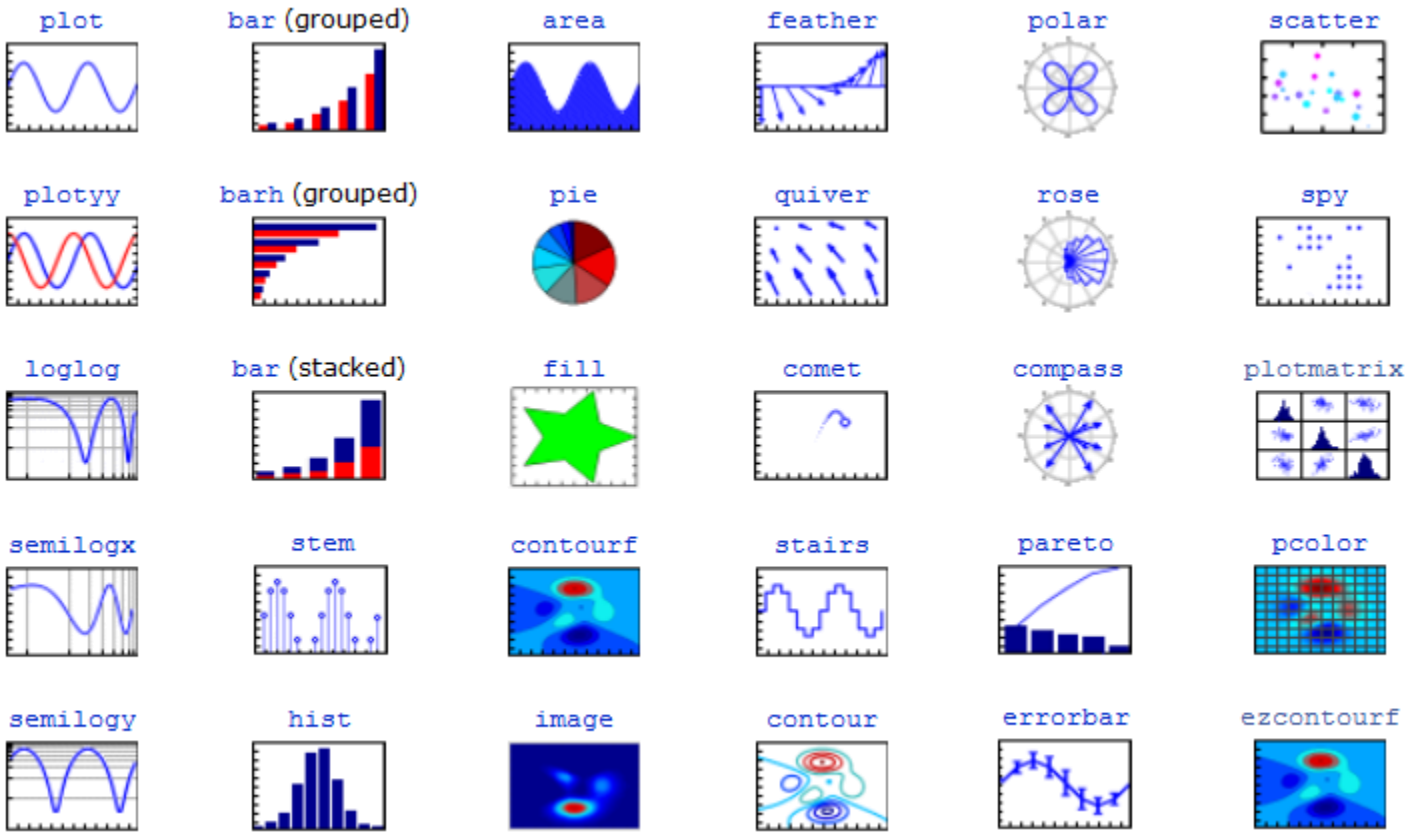
- Example:
  - Matlab: sample = [2 2 4 5 4 4 2 3 3 4 6 7 8 8 5];
  - Matlab: result = histc(sample, 1:10)

Intervals [1;2[, [2;3[,…[9;10[,[10; ∞ [

  - Output: result = [0 3 2 4 2 1 1 2 0 0]
  - Matlab: plot(result);
  - Output:

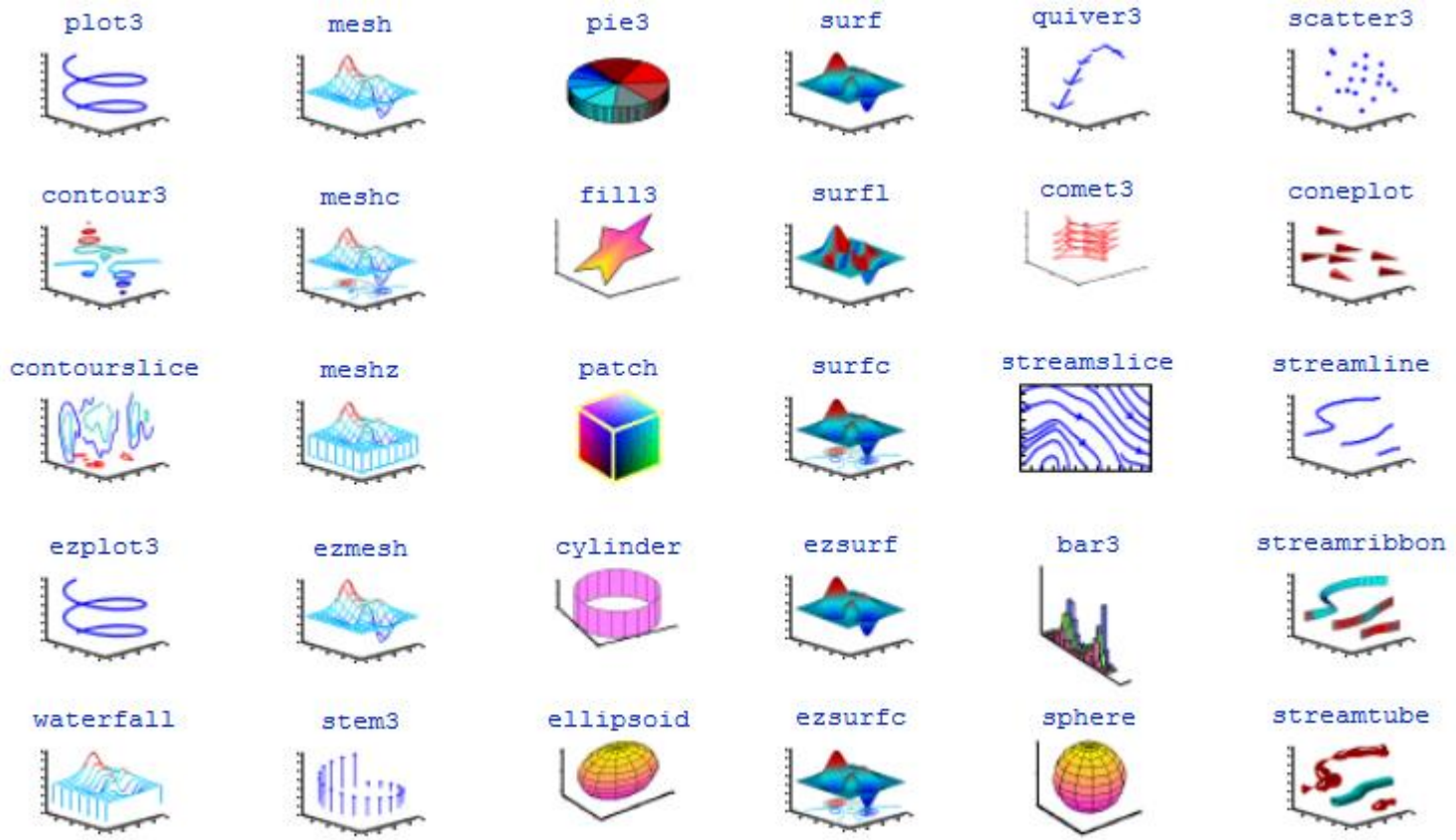


- Plots:



http://www.mathworks.com/help/techdoc/creating_plots/f9-53405.html

- Plots:

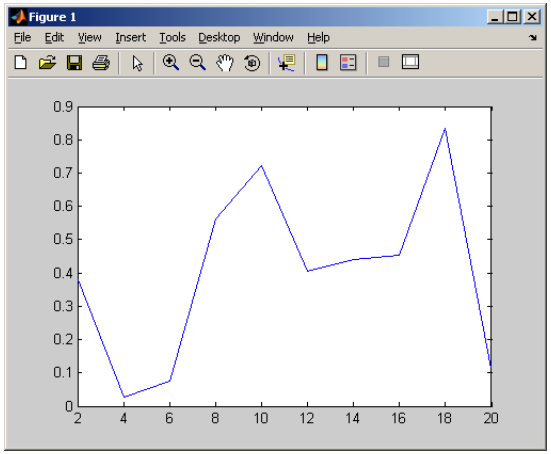


http://www.mathworks.com/help/techdoc/creating_plots/f9-53405.html

Matlab – Tutorial – Plots

- Plots:
  - **Standard 2D-Line Plot** **plot(Vector x, Vector y, …Layout….)**
    - Example 1:
      - Matlab: x = 2:2:20
      - Output: x = 2 4 6 8 10 12 14 16 18 20
      - Matlab: y = rand (10,1)
      - Output: y = 0.2553 0.3418 … 0.7943
      - Matlab: plot (x,y)
      - Output:

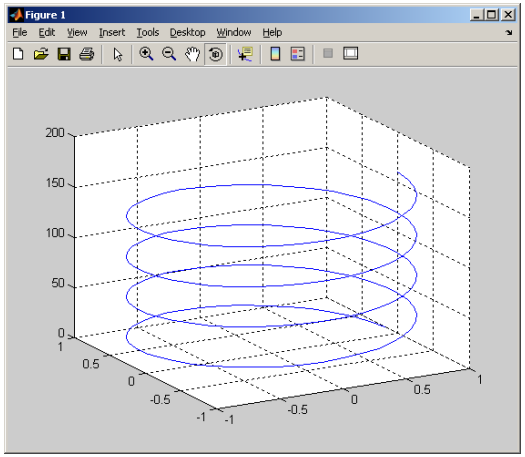
Will be introduced at the end of the section



Matlab draws the lines between the points (x(1),y(1)), … , (x(n),y(n))

Matlab – Tutorial – Plots

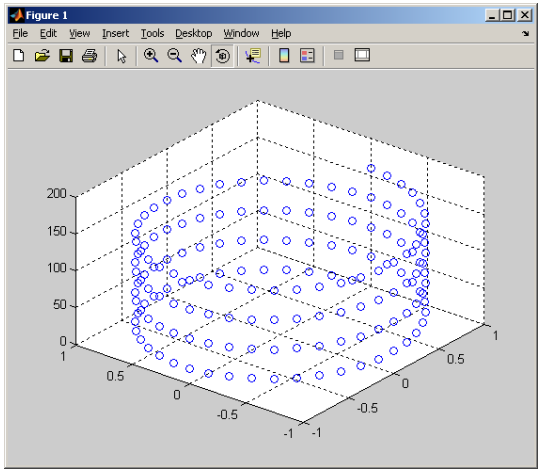
- Plots:
  - **Standard 3D-Line Plot plot3(Vector x, Vector y, Vector z)**
    - Example 1:
      - Matlab: x = cos(pi/20:pi/20:8*pi); %Vector with 160 elements
      - Matlab: y = sin(pi/20:pi/20:8*pi); %Vector with 160 elements
      - Matlab: z = 1:160;
      - Matlab: plot3(x,y,z)
      - Matlab: grid; %Adds a nice grid to the figure
      - Output:



Matlab draws the lines between the points (x(1),y(1)), … , (x(n),y(n))

Matlab – Tutorial – Plots

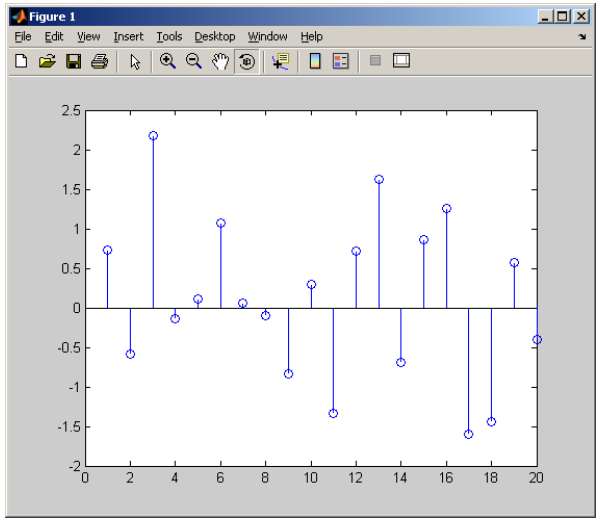
- Plots:
  - **Scatter plots are similar to standard line plots**
    - Example 1:
      - Matlab: x = cos(pi/20:pi/20:8*pi); %Vector with 160 elements
      - Matlab: y = cos(pi/20:pi/20:8*pi); %Vector with 160 elements
      - Matlab: z = 1:160;
      - Matlab: scatter3(x,y,z)
      - Matlab: grid; %Adds a nice grid to the figure
      - Output:



Scatter plots are often used to visualize point fields, point densities and autocorrelation plots

Matlab – Tutorial – Plots

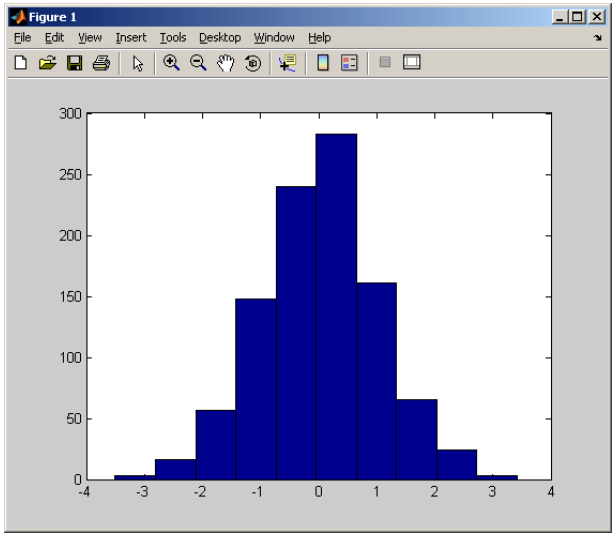
- Plots:
  - **Stem - stem(Vector x, Vector y) / stem3(Vector x, Vector y, Vector z)**
    - Example 1:
      - Matlab: x = 1:20; %Vector with 20 elements
      - Matlab: y = randn(20,1);%Vector with 20 normal distributed elements
      - Matlab: stem (x, y)
      - Output:



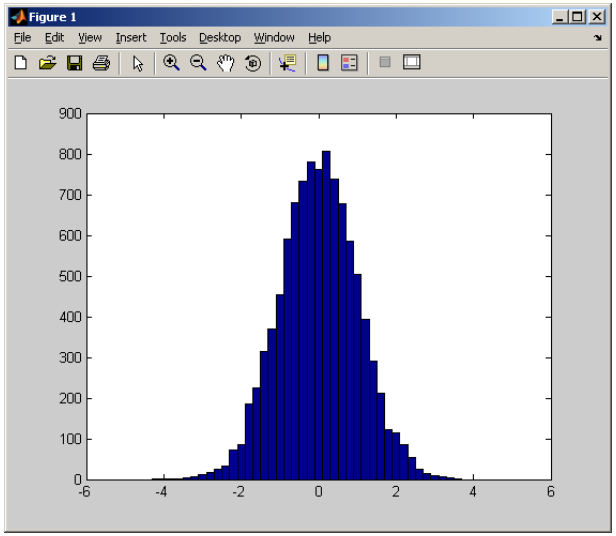
Stem plots are optimal for visualization of errors or discrete random variable

Matlab – Tutorial – Plots

- Plots:
  - **Histogram – hist (Vector x) / hist (Vector x, #bins) / hist(Vector x, Vector bins)**
    - Example 1:
      - Matlab: x = randn(1000,1); %Vector with 1000 elements
      - Matlab: hist(x)
      - Output: histogram with 10 bins

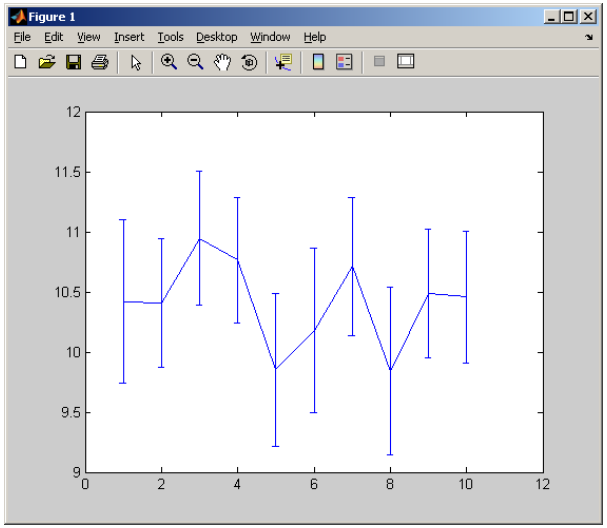


    - Example 2:
      - Matlab: hist(randn(10000,1),-5:0.2:5)
      - Output: histogram with 51 bins with bin size 0.2 starting from -5 to +5



Matlab – Tutorial – Plots

- Plots:
  - **Errorbar - errorbar(Vector sample, Vector error)**
    - Example:
      - Matlab: sample = 0.5*randn(10,1)+10;
      - Matlab: error = 0.2*rand(10,1)+0.5;
      - Matlab: errorbar (sample, error)
      - Output:

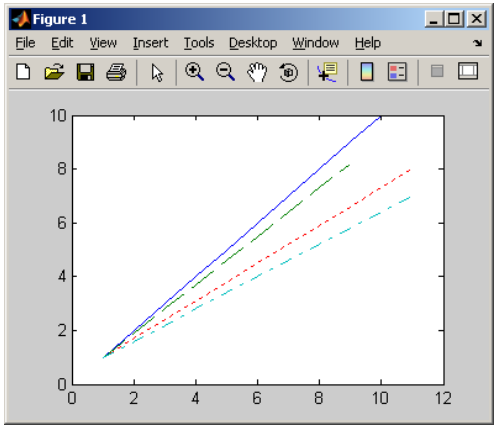


The shown error bars reflect twice the error value. Typical values are standard deviation or confidence intervals of the samples.

Layout - Basics

Matlab supports many features to modify an existing plot. This tutorial just gives a brief overview of the most important ones.

- **Line style:**
  - - Solid line (default)
  - -- Dashed line
  - Example:
    - Matlab: plot (1:10,‘LineStyle‘,‘-‘); hold all;
    - Matlab: plot (1:0.9:9,‘LineStyle‘,‘--‘);
      plot (1:0.7:8,‘LineStyle‘,‘:‘); plot (1:0.6:7,‘LineStyle‘,‘-.‘);
  - : Dotted line
  - -. Dash-dot line



Use different line styles since many colors cannot be distinguished if they are printed in b/w.

Layout - Basics

Matlab supports many features to modify an existing plot. This tutorial just gives a brief overview of the most important ones.

- Line style
  - Matlab: plot(x, 'LineStyle','--');
- Line width
  - Matlab: plot(x, 'LineStyle','--', 'LineWidth',2);
- Color
  - Matlab: plot(x, 'LineStyle','--', 'LineWidth',2, 'Color', 'red');
- Marker type
  - Matlab: plot(x, 'LineStyle','--', 'LineWidth',2, 'Marker','diamond');
- Marker size
  - Matlab: plot(x, 'LineStyle','--', 'Marker','diamond', 'MarkerSize', 10);

Always use a line width of 2 since it looks much better in most plots

Take a look at Matlab Help ‘linespec’ for more information

Layout - Basics

- Brief command description:

  - Command: **axis normal**

    Stretches the current figure such that it fits fills the frame.

  - Command: **set(gcf, 'Position', [200 200 800 640]);**

    Moves the current figure to position [200;200] and sets its size to 800x640 pixels

  - Command: **grid**

    A grid is plotted which simplifies reading the graphs that are plotted. The grid is removed it is already present.

  - Command: **box**

    A box is plotted around the figure which is strongly recommended for 2D plots. The box is removed by the command if already present.

Layout - Basics

- Brief command description:

  - Command: **xlabel('This is the label of the x axis')**
    Sets the name of the x axis to the string argument.(ylabel / zlabel)

  - Command: **xlim([0 10]); ylim([5 10]); zlim([3 10]);**
    Sets the limit of the axis to the given interval.

  - Command: **set(gca, 'FontSize', 14);**
    Sets the font size of the axis label and tick labels to the given value.

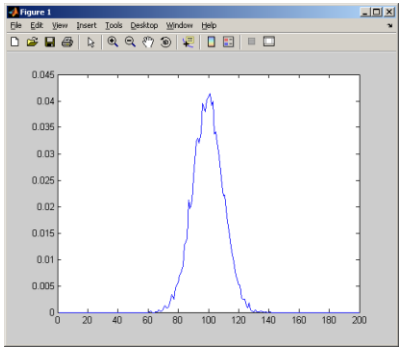
  - Command: **set(gca, 'XTickLabel', {'0','5','10','15','20','25','30','35'}, 'XTick',[-1 4 9 14 19 24 29 34]);**
    Sets the labels and the position of the labels to the given values. XTick describes the position of the label whereas XTickLabel holds the string of the label. Here it is used to shift the graph by one unit.

Probability Density Function – Sample Vector

- Vector x is the vector which holds all elements of the sample /measurement
  - Example:
    - Matlab: x = 100+10*randn(10000,1);%Returns a vector with 10000 random values with a mean of 100
    - Matlab: result = histc(x, 1:200);

Intervals [1;2[, [2;3[,…[199;200[,[200; ∞ [

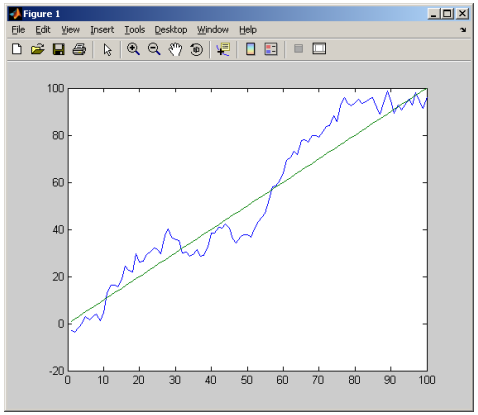
    - Matlab: result = result / length(x); %Normalize the result
    - Matlab: plot(result);
    - Output:



P(robability)P(robability)-Plot

Plot two sample vectors against each other to outline their relative difference in a single plot. (Typically only done with distributions)

- Example 1:
  - Matlab: i = 1:100; % Generate a reference vector with 100 elements
  - Matlab: j = 1:100; % Second vector
  - Matlab: b(1:10) = 0.1; %Generate a filter vector
  - Matlab: k = 20*randn(1,100);
  - Matlab: k = filter(b,1,k);%Smoothed random vector
  - Matlab: plot (i, j+k);hold all;plot(1:100);%Plot i against j+k. Then hold the figure and plot the reference graph

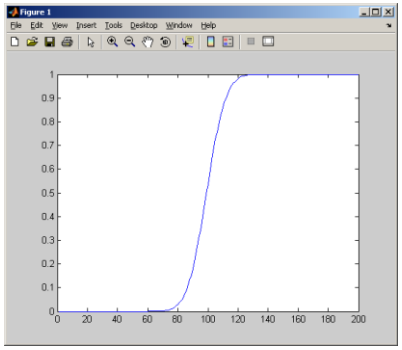


Cumulative Density Function – Sample Vector

- Vector x is the vector which holds all elements of the sample /measurement
  - Example:
    - Matlab: x = 100+10*randn(10000,1);%Returns a vector with 10000 random values with a mean of 100
    - Matlab: result = histc(x, 1:200);

Intervals [1;2[, [2;3[,…[199;200[,[200; ∞ [

    - Matlab: result = result / length(x); %Normalize the result
    - Matlab: plot(**cumsum**(result));
    - Output:



Calculates the sum of the elements which corresponds to the CDF if the input vector represents the PDF

p-quantil

The p-quantile for a random variable is the value *x* such that the probability that the random variable will be less than *x* is at most *p*

- sample = randn(10000,1)+10;

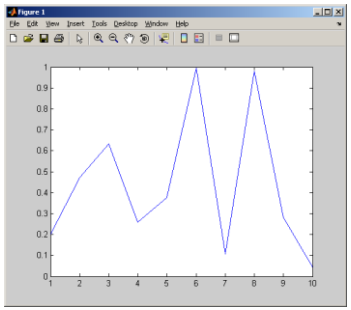
- Example:
  - Matlab: p = 0.95;%Probability that a randomly chosen sample value is lower than the p-quantile
  - Matlab: size = length(sample);%Sometimes the length of the sample is not known in advance. Thus, length should be used.
  - Matlab: result = sort(sample);%Sort the samples
  - Matlab: index = ceil(size*p);%Calculates the corresponding index
  - Matlab: p_quantile = result (index);

The p-quantile can also be directly read from the CDF which should be done if it is already calculated.

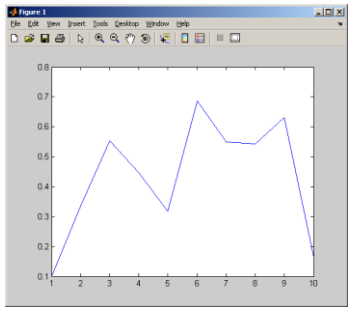
Moving Average Filter – filter (Vector b, Vector a, Vector x)

$$y(n) = b(1) \cdot x(n) + b(2) \cdot x(3) + \cdots + b(k) \cdot x(n-k)$$

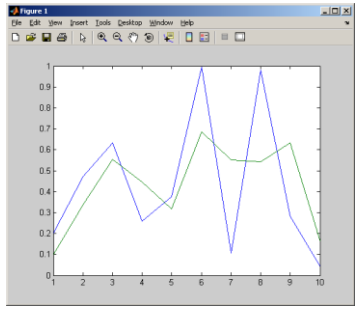
- Vector x is the sample vector that has to be smoothed
- Variable a should be set to 1 c.f. Matlab Help
- Vector b defines the length and the weights for the smoothing vector
  - Example 1:
    - x = rand(10,1);%Vector with 10 uniform distributed elements
    - a =1;% Set a to 1
    - b = [0.5 0.5];%Will result in a smoothing over the last two values
    - y = filter(b, a, x);%Store the smoothed vector in variable y
    - plot(x);hold all; plot(y); %Plots x and y in the same figure



Vector x



Vector y



Output