MATLAB / Simulink Lab Course

Optimization & Statistics Toolbox

Objectives & Preparation "Optimization & Statistics Toolbox"

- Which MathWorks products are covered?
  - ⇨ Optimization Toolbox
  - ⇨ Statistics and Machine Learning Toolbox

- What skills are learnt?
  - ⇨ Finding optimal solutions to continuous and discrete problems
  - ⇨ Incorporating constraints into an optimization problem
  - ⇨ Describe, analyzing, and modeling data
  - ⇨ Generating random numbers

- How to prepare for the session?
  - ⇨ MathWorks Webinars:
    - https://de.mathworks.com/videos/tips-and-tricks-getting-started-using-optimization-with-matlab-81594.html
    - https://de.mathworks.com/videos/data-driven-fitting-with-matlab-81809.html

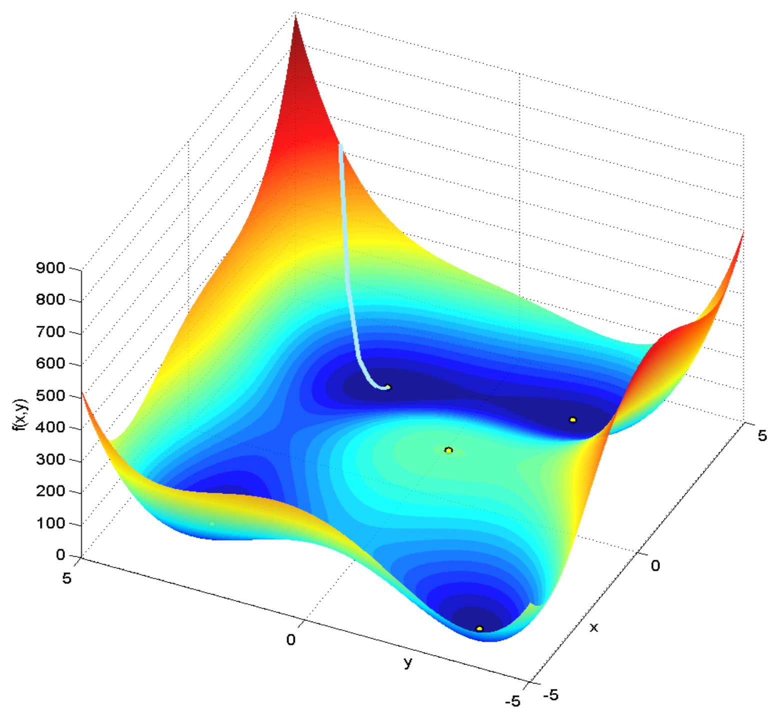
Outline

- Optimization Toolbox
  - Introduction
  - Linear programming
  - Quadratic programming
  - Nonlinear optimization
  - Nonlinear least squares
  - Outlook

- Statistics Toolbox
  - Introduction
  - Data Import
  - Exploratory Data Analysis
  - Fitting Distribution Objects
  - Generate Random Numbers
  - Hypothesis Tests
  - Parametric Regression Analysis

- List of Commands

Optimization Toolbox



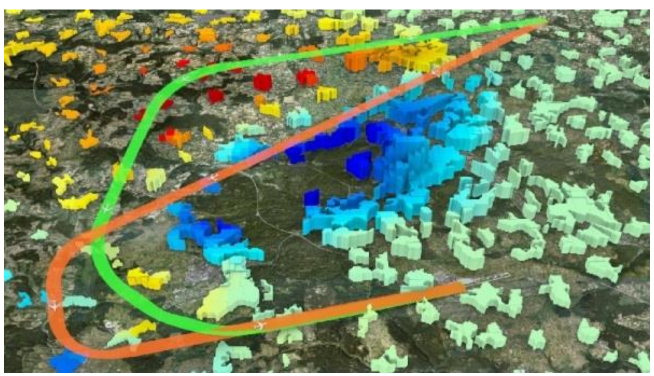
Optimization Toolbox – Motivation

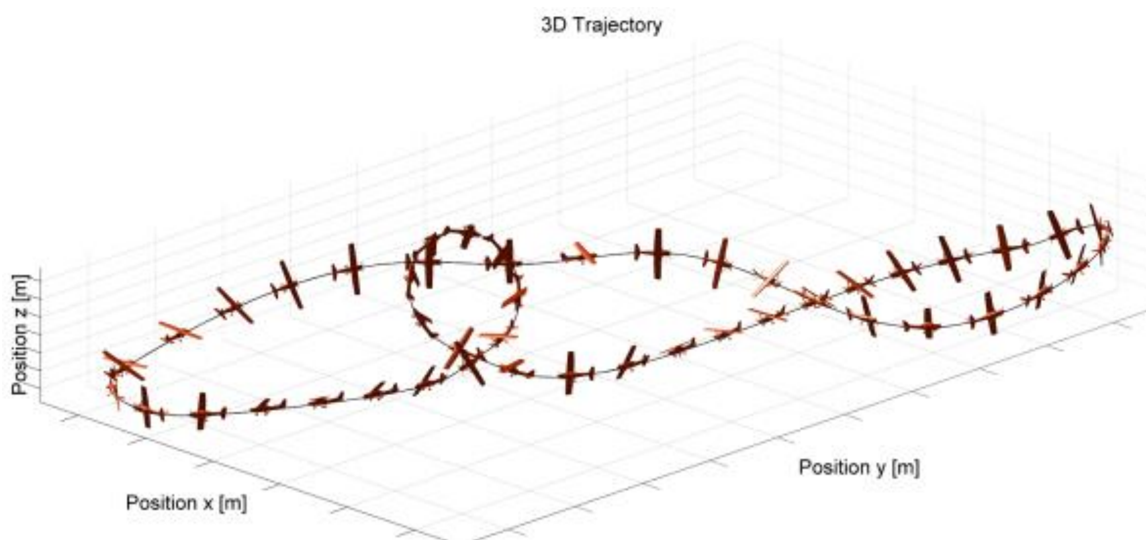
Trajectory Optimization

How can we calculate noise minimal approaches and departure routes?

How can an air race track be designed so that it is fair for two different aircraft types?

How can fuel minimal trajectories be calculated?





Lehrstuhl für
Flugsystemdynamik

Optimization Toolbox – Introduction

- The **Optimization Toolbox** provides functions for finding parameters that minimize or maximize objectives while satisfying constraints.

- Available solvers for different kind of optimization problems:
  - **linear programming**,
  - **mixed-integer linear programming**,
  - **quadratic programming**,
  - **nonlinear optimization**
  - **nonlinear least-squares**.

- You can use these solvers to **find optimal solutions** to continuous and discrete problems, **perform tradeoff analyses**, and **incorporate optimization** methods into algorithms and applications.

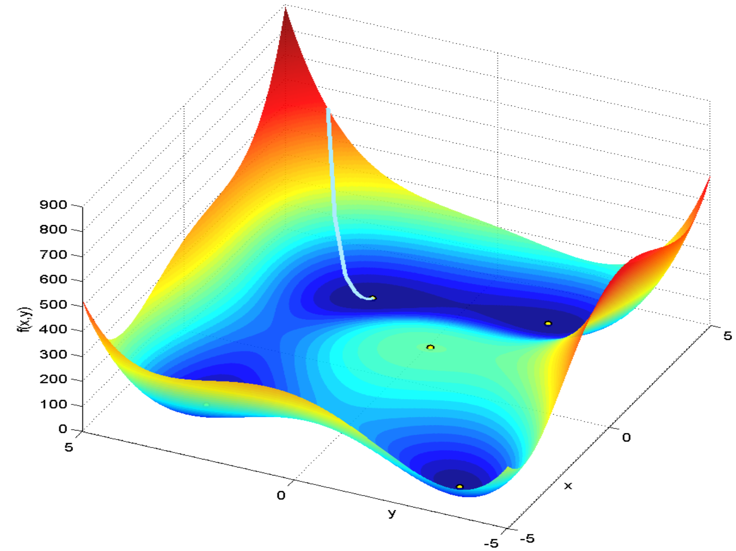
- In general, optimization is an **iterative process** (compare Newton's method)

- Considering minimization problems is sufficient (**maximization** of f is minimization of $-f$)

Source: http://www.mathworks.com/products/optimization/

Optimization Toolbox – Introduction

An optimization problem consists of two key components:

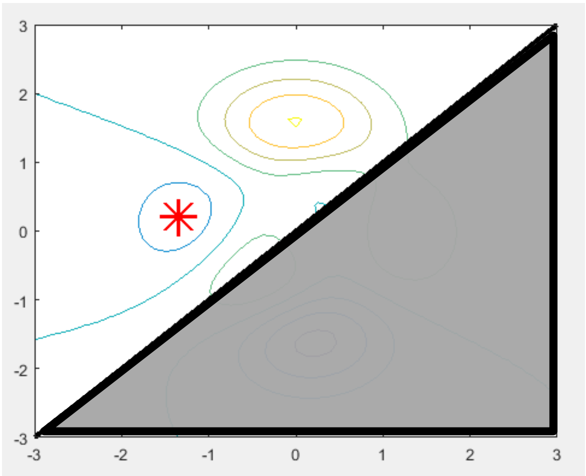


- Objective function (also called cost function)

  The goal is to find a (local) minimum of this function.

- Constraints (optional)

  The solution (i.e. local minimum) is restricted to be in a certain area of the domain.



Optimization Toolbox – (Mixed-Integer) Linear Programming / (MI)LP

- **Goal**: Find the minimum of a linear problem $\min_{\boldsymbol{x}} \boldsymbol{f}^T \cdot \boldsymbol{x}$ such that
  - $\boldsymbol{A}\,\boldsymbol{x} \leq \boldsymbol{b}$ (linear inequality constraints)
  - $\boldsymbol{A}_{\mathrm{eq}}\,\boldsymbol{x} = \boldsymbol{b}_{\mathrm{eq}}$ (linear equality constraints)
  - $\boldsymbol{x}_{\mathrm{lb}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\mathrm{ub}}$ (bound constraints)

  where $\boldsymbol{f}$, $\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{b}_{\mathrm{eq}}$, $\boldsymbol{x}_{\mathrm{lb}}$ and $\boldsymbol{x}_{\mathrm{ub}}$ are vectors with suitable lengths and $\boldsymbol{A}$ and $\boldsymbol{A}_{\mathrm{eq}}$ are matrices of suitable dimensions.

  Note: The linear objective is defined by a coefficient vector, not a function handle.
- LP solver: **linprog**

```
[x_opt, f_opt, status] = linprog(f, A, b, A_eq, b_eq, x_lb, x_ub);
```

- MILP solver: **intlinprog**

```
[x_opt, f_opt, status] = intlinprog(f, intcon, A, b, A_eq, b_eq, x_lb, x_ub);
```

  The index vector `intcon` describes which components `x(intcon)` are integers.

Note: Always check for `(status > 0)` after calling a solver!

Source: http://www.mathworks.com/help/optim/ug/linprog.html

Optimization Toolbox – (Mixed-Integer) Linear Programming

LP example: $\min_{x}(-x_1 - x_2)$ $= \min_{x} [-1, -1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

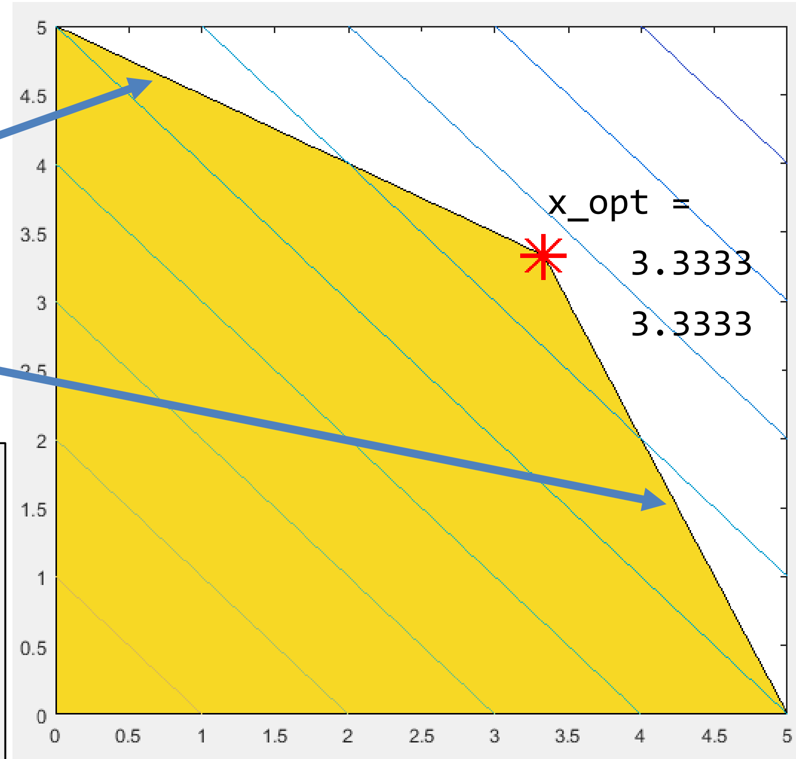
such that

$\begin{bmatrix} 0.5 & 1 \\ 2 & 1 \end{bmatrix} x \le \begin{bmatrix} 5 \\ 10 \end{bmatrix} \Big\{$

- $x_2 \le -0.5 * x_1 + 5$
- $x_2 \le -2 * x_1 + 10$

$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \le \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Big\{$

- $-x_1 \le 0$
- $-x_2 \le 0$




```
A = [0.5  1    % 0.5 x_1 + 1 x_2 <= b_1
     2    1];  %   2 x_1 + 1 x_2 <= b_2
b = [5 10];
A_eq = []; b_eq = [];
x_lb = [0, 0]; x_ub = [inf, inf];
f = [-1 -1];
[x_opt, f_opt, status] = ...
    linprog(f, A, b, A_eq, b_eq, x_lb, x_ub)
if ~(status > 0); error('Failed'); end
```

Note: The bounds can also be specified as generic inequality constraints in A, but it can be more efficient to pass them separately.

Optimization Toolbox – Quadratic Programming / QP

- **Goal**: Find the minimum of a problem specified by

$$\min_{\boldsymbol{x}} \left(\frac{1}{2}\, \boldsymbol{x}^T\, \boldsymbol{H}\, \boldsymbol{x} + \boldsymbol{f}^T\, \boldsymbol{x}\right)$$

  such that

  - $\boldsymbol{A}\,\boldsymbol{x} \leq \boldsymbol{b}$
  - $\boldsymbol{A}_{\mathrm{eq}}\,\boldsymbol{x} = \boldsymbol{b}_{\mathrm{eq}}$
  - $\boldsymbol{x}_{\mathrm{lb}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\mathrm{ub}}$

  where $\boldsymbol{f}$, $\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{b}_{\mathrm{eq}}$, $\boldsymbol{x}_{\mathrm{lb}}$ and $\boldsymbol{x}_{\mathrm{ub}}$ are vectors with suitable lengths and $\boldsymbol{H}$, $\boldsymbol{A}$ and $\boldsymbol{A}_{\mathrm{eq}}$ are matrices of suitable dimensions.

- QP solver: **quadprog**

```
[x_opt, f_opt, status] = quadprog(H, f, A, b, A_eq, b_eq, x_lb, x_ub);
```

- Note: The quadprog solver only handles linear and bound constraints, even though the cost function is nonlinear

Source: http://www.mathworks.com/help/optim/ug/quadprog.html

Optimization Toolbox – Quadratic Programming / QP

QP example:

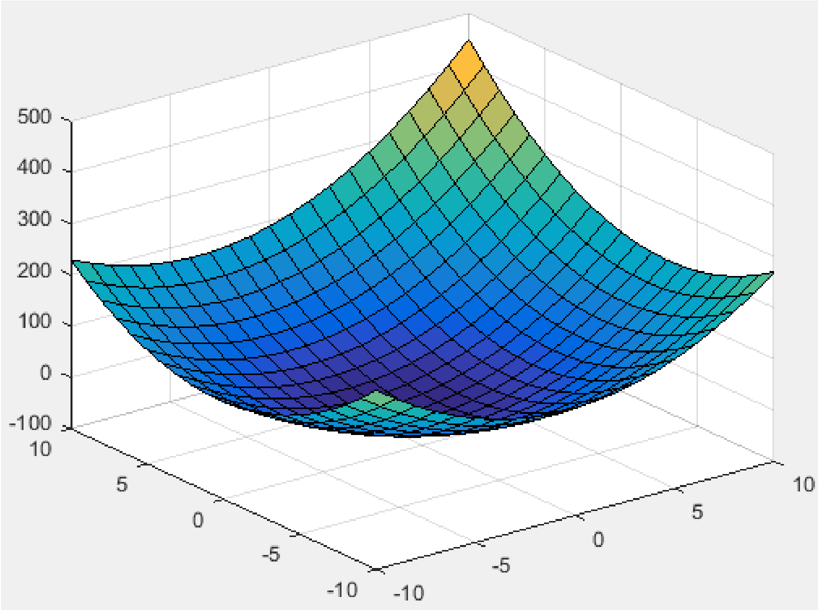
$$\min_{x} \left(\frac{3}{2} x_1^{\ 2} + \frac{1}{2} x_1 x_2 + \frac{3}{2} x_2^{\ 2} + 6 x_1 + 4 x_2 \right)$$

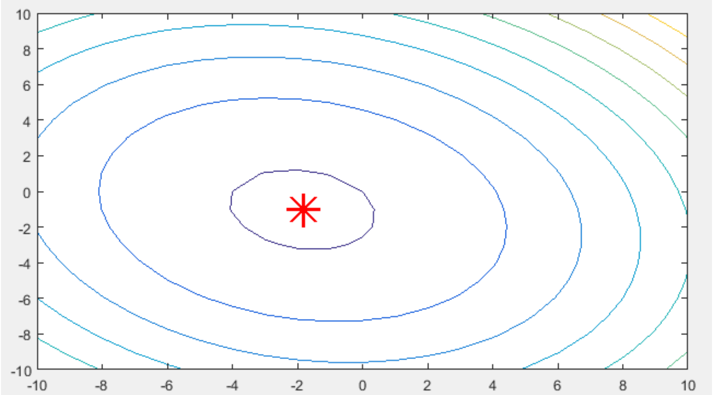
without constraints.

$\underline{x}^T \begin{bmatrix} 3/2 & 1/4 \\ 1/4 & 3/2 \end{bmatrix} \underline{x}$ $\quad [6 \ \ 4]\,\underline{x}$

```
H = [3    0.5
     0.5  3];

f = [6; 4];

[x_opt, f_opt, status] = quadprog(H,f)
if ~(status > 0); error('Failed'); end
```

```
x_opt =
   -1.8286
   -1.0286
```





Optimization Toolbox – Nonlinear Programming / NLP (unconstrained)

- **Goal**: Find the minimum of a problem specified by

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

$f \in C^2$

where $f: \mathbb{R}^n \to \mathbb{R}$ is a smooth nonlinear function.

The problem is unconstrained, i.e. there are no restrictions for $\boldsymbol{x}$.

- In general (also applies to the constrained case; exceptions exist for some problem types):
  - Nonlinear optimization is an iterative process.
  - There are no guarantees for global convergence and/or global optimality.
  - Typical (local) NLP solvers are based on the Newton-Raphson method and thus require f to be at least twice continuously differentiable.
  - An initial guess x_0 must be specified; the quality of this guess can be crucial.

- Unconstrained NLP solver: **fminunc**

```
[x_opt, f_opt, status] = fminunc(fun, x_0, options);
```

In the optional argument `options`, further settings about the optimization (algorithm, max. iterations, …) can be set. To obtain `options` use the `optimoptions` function.

Source: http://www.mathworks.com/help/optim/ug/fminunc.html

Optimization Toolbox – Nonlinear Programming / NLP (unconstrained)

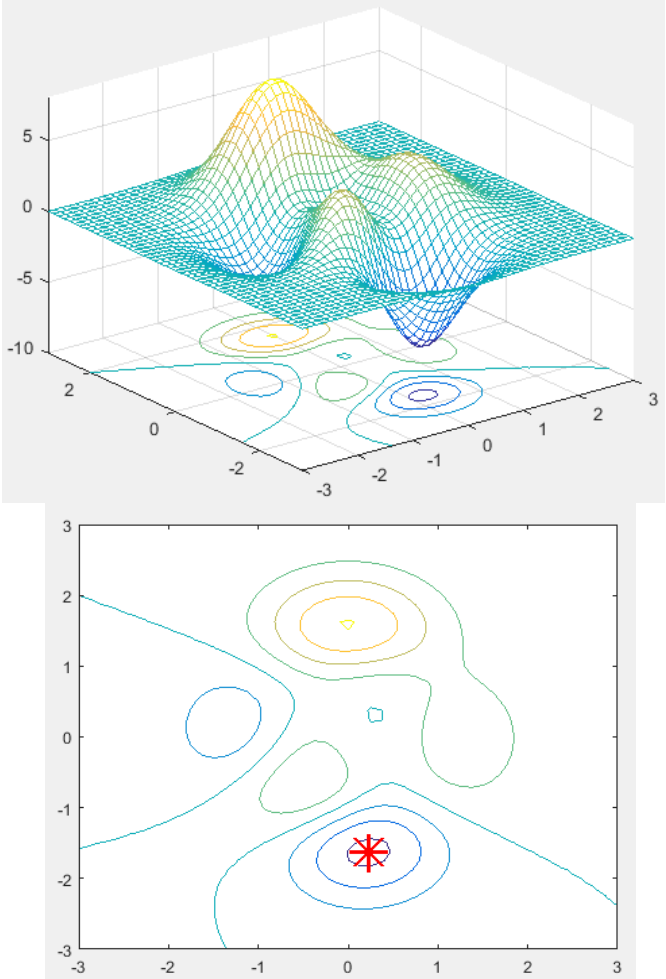
Unconstrained NLP example:

minimize MATLAB's Peaks function without constraints.

x is a vector

```
myPeaks = @(x) 3*(1-x(1))^2.*exp(-(x(1)^2) - (x(2)+1)^2)...
    - 10*(x(1)/5 - x(1)^3 - x(2)^5).*exp(-x(1)^2-x(2)^2)...
    - 1/3*exp(-(x(1)+1)^2 - x(2)^2);

x_0 = [-1, -2];

[x_opt, f_opt, status] = fminunc(myPeaks, x_0)
if ~(status > 0); error('Failed'); end
```

```
x_opt =
 0.2283   -1.6255
```



Source: http://www.mathworks.com/help/matlab/ref/peaks.html

Optimization Toolbox – Nonlinear Programming / NLP (constrained)

- **Goal**: Find the minimum of a problem specified by $\min_{\boldsymbol{x}} f(\boldsymbol{x})$

  such that
  - $\boldsymbol{c}(\boldsymbol{x}) \leq \boldsymbol{0}$ (nonlinear inequality constraints)
  - $\boldsymbol{c}_{\text{eq}}(\boldsymbol{x}) = \boldsymbol{0}$ (nonlinear equality constraints)
  - $\boldsymbol{A}\,\boldsymbol{x} \leq \boldsymbol{b}$ (linear inequality constraints)
  - $\boldsymbol{A}_{\text{eq}}\,\boldsymbol{x} = \boldsymbol{b}_{\text{eq}}$ (linear equality constraints)
  - $\boldsymbol{x}_{\text{lb}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\text{ub}}$ (bound constraints)

  with functions $f,\ \boldsymbol{c},\ \boldsymbol{c}_{\text{eq}}: \mathbb{R}^n \to \mathbb{R}$, vectors $\boldsymbol{b}$, $\boldsymbol{b}_{\text{eq}}$, $\boldsymbol{x}_{\text{lb}}$ and $\boldsymbol{x}_{\text{ub}}$ with suitable lengths and matrices $\boldsymbol{A}$ and $\boldsymbol{A}_{\text{eq}}$ of suitable dimensions.

- Constrained NLP solver: **fmincon**

```
[x_opt, f_opt, status] = ...
          fmincon(fun, x_0, A, b, A_eq, b_eq, x_lb, x_ub, nonlcon, options);
```

Here, `nonlcon` is a function that calculates nonlinear inequalities and equalities:

```
function [ c, c_eq ] = nonlcon(x)
…
```

@(x) deal(c(x), c_eq(x))

Source: http://www.mathworks.com/help/optim/ug/fmincon.html

Optimization Toolbox – Nonlinear Programming / NLP (constrained)

Constrained NLP example:

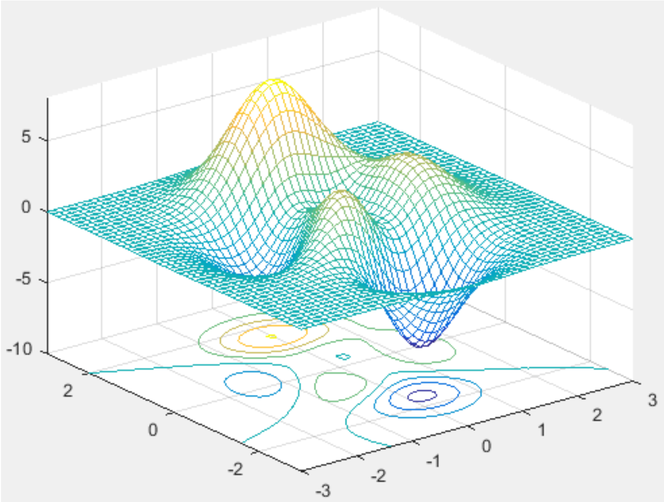
minimize MATLAB's Peaks function

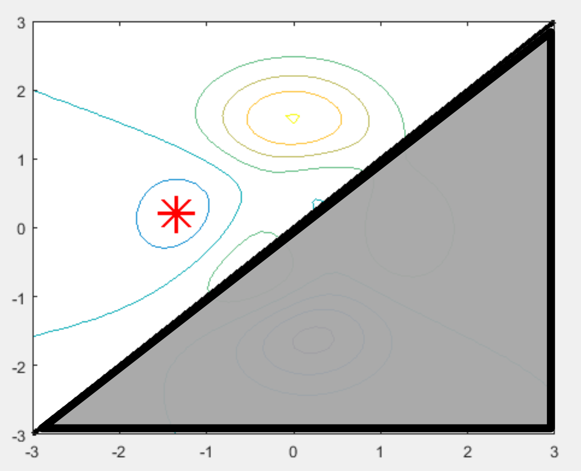
such that:

- $x_1 \leq x_2$

```
myPeaks = @(x) 3*(1-x(1))^2.*exp(-(x(1)^2) - (x(2)+1)^2)...
    - 10*(x(1)/5 - x(1)^3 - x(2)^5).*exp(-x(1)^2-x(2)^2)...
    - 1/3*exp(-(x(1)+1)^2 - x(2)^2);

x_0 = [-3, -1];

A = [1, -1];
b = 0;

[x_opt, f_opt, status] = fmincon(myPeaks, x_0, A, b)
if ~(status > 0); error('Failed'); end
```

```
x_opt =
 -1.3474    0.2045
```





Source: http://www.mathworks.com/help/matlab/ref/peaks.html

Optimization Toolbox – Nonlinear Least-Squares / NLS

- **Goal**: Find the minimum of a problem specified by

$$\min_{\boldsymbol{x}} \|f(\boldsymbol{x})\|_2^2 = \min_{\boldsymbol{x}} \left(f_1(\boldsymbol{x})^2 + \cdots + f_n(\boldsymbol{x})^2\right)$$

subject to $\boldsymbol{x}_{\mathrm{lb}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\mathrm{ub}}$

with $\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n$. (smooth)

vector-valued nonlinear function handle

- Specialized NLS solver: **`lsqnonlin`**

```
[x_opt, resnorm, residual, status] = lsqnonlin(f, x_0, x_lb, x_ub);
```

Here, `resnorm` is $\|\boldsymbol{f}(\boldsymbol{x}_{\mathrm{opt}})\|_2^2$ and `residual` provides the vector $\boldsymbol{f}(\boldsymbol{x}_{\mathrm{opt}})$.
Note: `f` is a vector-valued function!

- NLS problems typically arise from data fitting applications.
- **`lsqnonlin`** implements algorithms that are tailored to this specific type of cost function; however, you can also apply `fmincon` (or `fminunc`, in the unconstrained case).
- **`lsqnonlin`** only handles bound constraints; if that is not sufficient, use `fmincon`

Source: http://www.mathworks.com/help/optim/ug/lsqnonlin.html

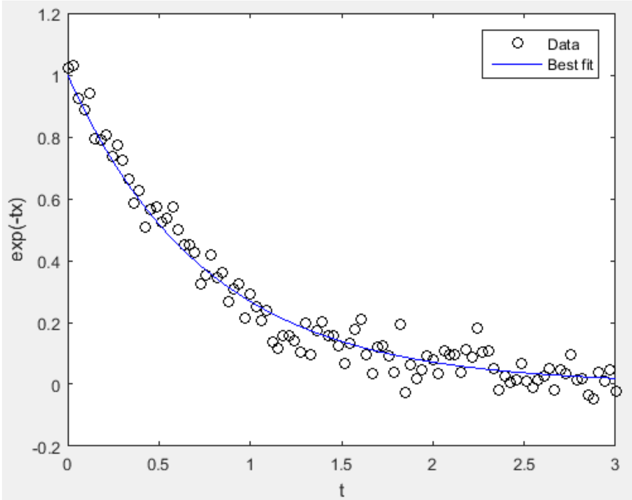
Optimization Toolbox – Nonlinear Least-Squares / NLS

NLS example:

- Fit a simple exponential decay curve to data.
- **Goal**: Find optimal exponential decay rate based on sample data.
- Generate data from an exponential decay model plus noise. The model is: $y = e^{-1.3*t} + \varepsilon$.
- t is ranging from 0 to 3 and ε is normally distributed noise with mean 0 and standard deviation 0.05.

```
d = linspace(0,3);
y = exp(-1.3 .* d) + 0.05 .* randn(size(d));

f = @(r) exp(-r .* d) - y;  % residual = model - data  残差

x_0 = 4;
[x_opt, resnorm, residual, status] = lsqnonlin(f, x_0)
if ~(status > 0); error('Failed'); end
```

```
x_opt =
 1.3169
```

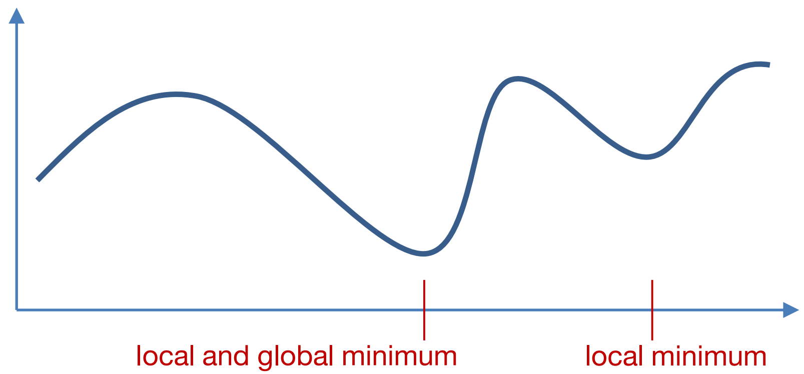


Source: http://www.mathworks.com/help/optim/ug/lsqnonlin.html

Optimization Toolbox – Outlook: Global and Non-Smooth Optimization

So far we talked about finding local minima – but a local minimum is not always a **global minimum**. MATLAB offers the following concepts:

- Multistart
- Globalsearch



For **non-smooth optimization problems** (e.g. non-continuous objective function) there are the following methods:

- Pattern search
- Simulated Annealing
- Genetic Algorithm

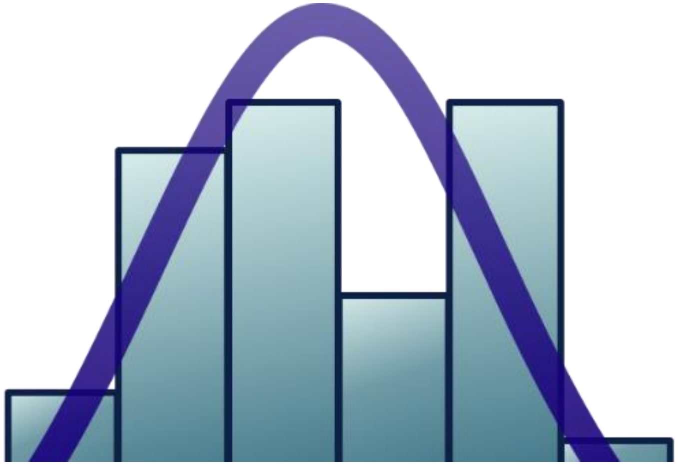
Optimization Toolbox – Outlook: Optimization Decision Table

- To find a suitable solver for a given problem, the **Optimization Decision Table** can be useful:

Constraint Type	Objective Type				
	Linear	**Quadratic**	**Least Squares**	**Smooth nonlinear**	**Nonsmooth**
None	n/a (f = const, or min = $-\infty$)	quadprog, Information	\, lsqcurvefit, lsqnonlin, Information	fminsearch, fminunc, Information	fminsearch, *
Bound	linprog, Information	quadprog, Information	lsqcurvefit, lsqlin, lsqnonlin, lsqnonneg, Information	fminbnd, fmincon, fseminf, Information	fminbnd, *
Linear	linprog, Information	quadprog, Information	lsqlin, Information	fmincon, fseminf, Information	*
General smooth	fmincon, Information	fmincon, Information	fmincon, Information	fmincon, fseminf, Information	*
Discrete	intlinprog, Information	*	*	*	*

Source: http://www.mathworks.com/help/optim/optimization-decision-table.html

Statistics Toolbox

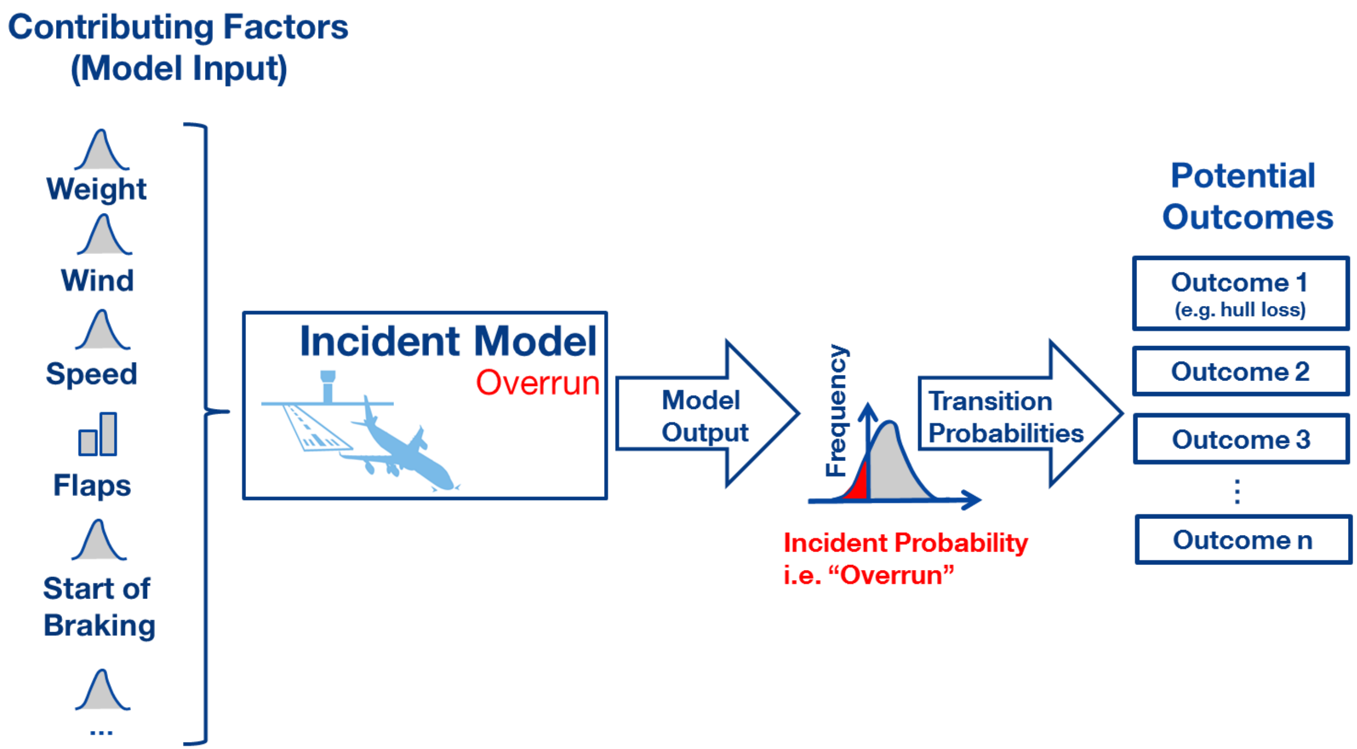


Statistics Toolbox – Motivation

Flight Safety

What is the probability of certain incidents (e.g. runway overrun) within my flight operation?

To answer this question, a lot of statistics is necessary and probability distributions have to be fitted to recorded data.



Statistics Toolbox – Introduction

- The **Statistics and Machine Learning Toolbox** provides functions and apps to describe, analyze, and model data using statistics and machine learning.

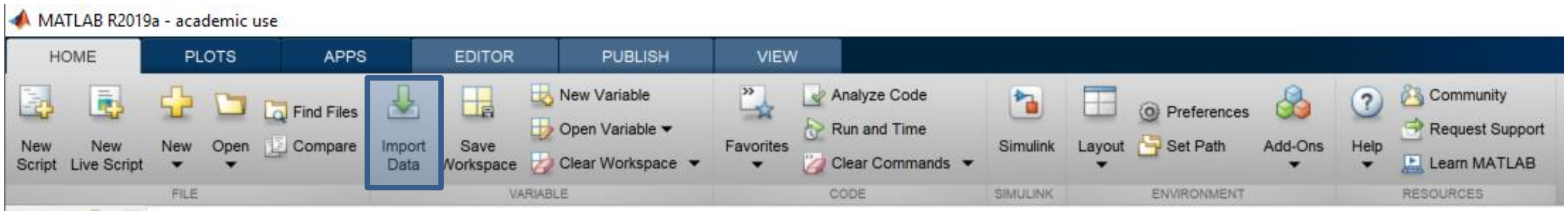
- You can use descriptive statistics and plots for **exploratory data analysis**, **fit probability distributions to data**, **generate random numbers** for Monte Carlo simulations, and perform **hypothesis tests**.

- **Regression and classification algorithms** let you draw inferences from data and build predictive models.

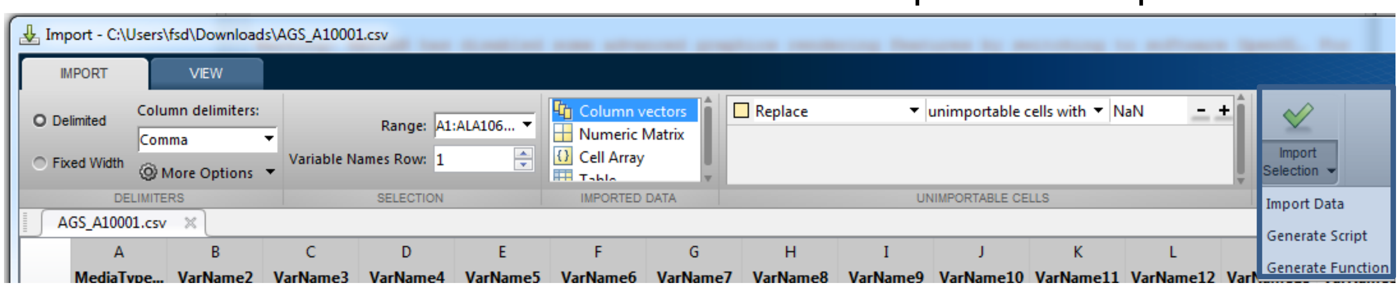
Source: http://www.mathworks.com/products/statistics/

Statistics Toolbox – Data Import

- Every statistical analysis works with data which has to be imported into MATLAB first.
- A very convenient way to import data is the "**Import Data**" button in the "HOME" tab.



- After choosing the file, MATLAB looks for the suitable commands automatically.
- The chosen commands can be considered with the help of the "Import Selection".



- Alternatively, the right command for the specific file can be applied directly:

  `textscan`, **`readtable`**, **`xlsread`**, **`csvread`**, **`dlmread`**, **`webread`** …

Statistics Toolbox – Exploratory Data Analysis

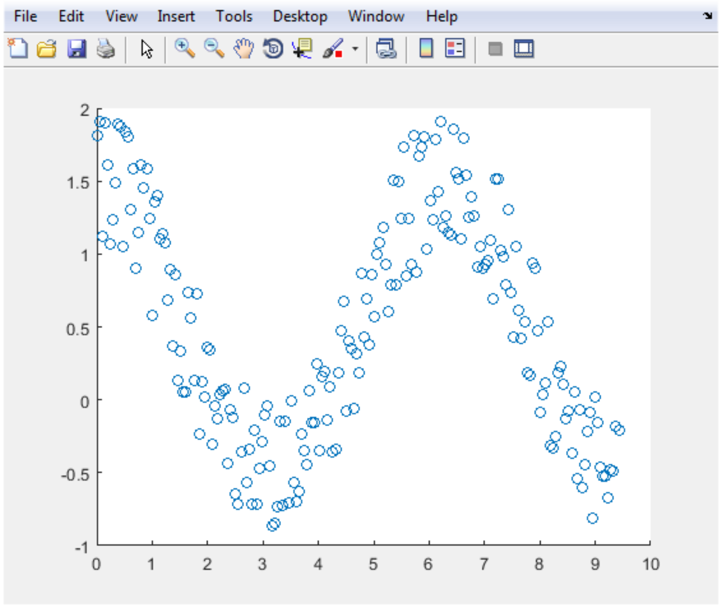
- **Scatterplots** are a very basic tool exploratory data analysis, i.e. graphical representation of data

```
scatter(x, y)
```

- Example:

```
x = linspace(0, 3*pi, 200);
y = cos(x) + rand(1, 200);
scatter(x, y)
```

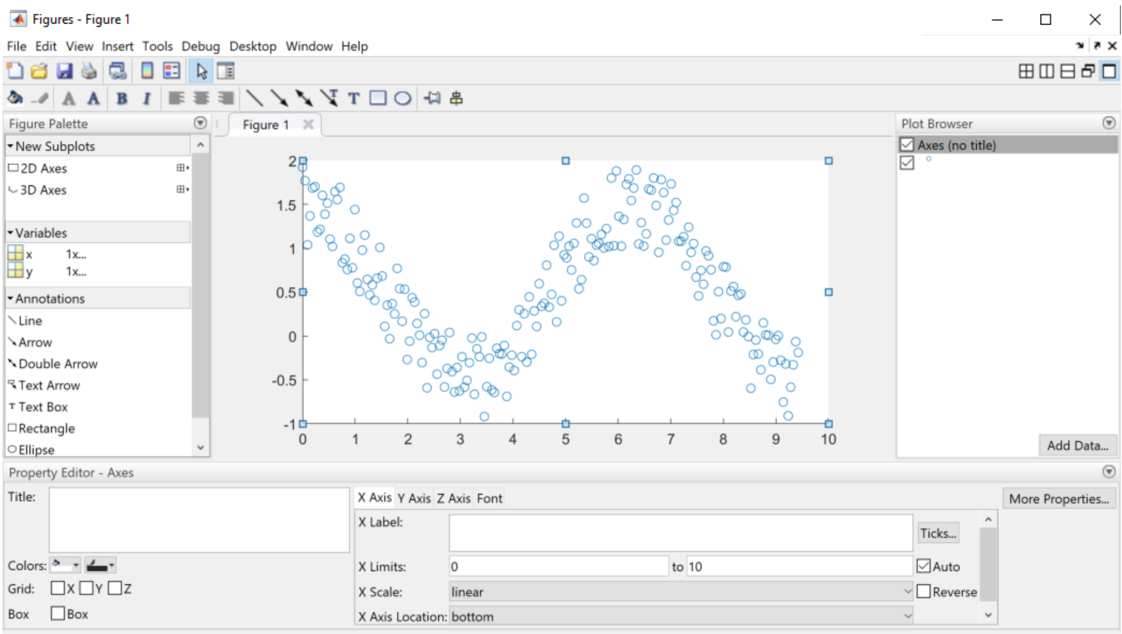
  - `linspace(0, 3*pi, 200)` returns a row vector of 200 evenly spaced points between `0` and `3*pi`
  - `rand(1, 200)` returns a row vector of 200 single *uniformly distributed* random numbers in the interval `(0,1)`

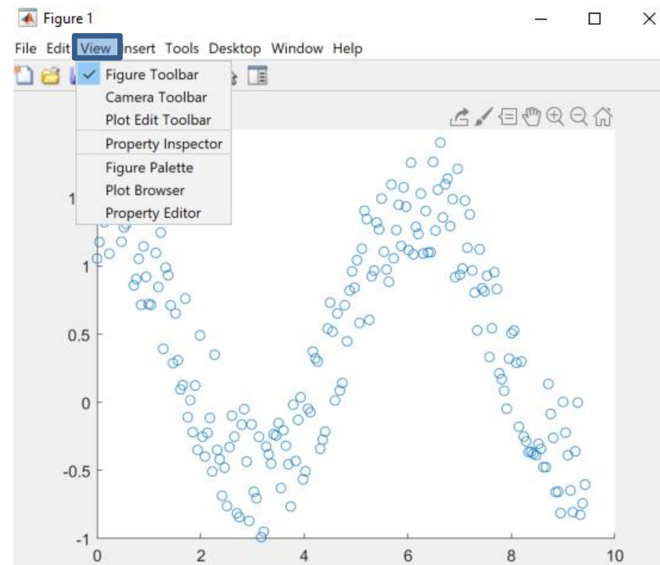


Source: http://www.mathworks.com/help/matlab/ref/scatter.html

Statistics Toolbox – Exploratory Data Analysis

- **Change properties** of the plot view the *View* menu
- Label, colors and further options can be set

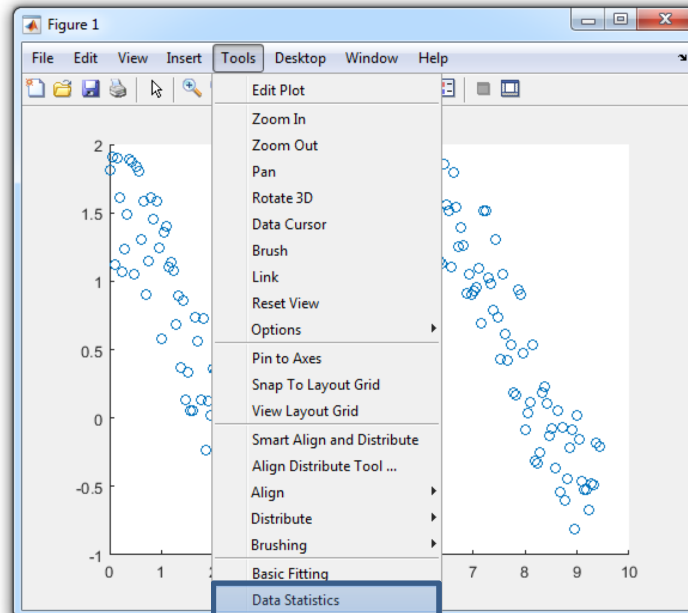




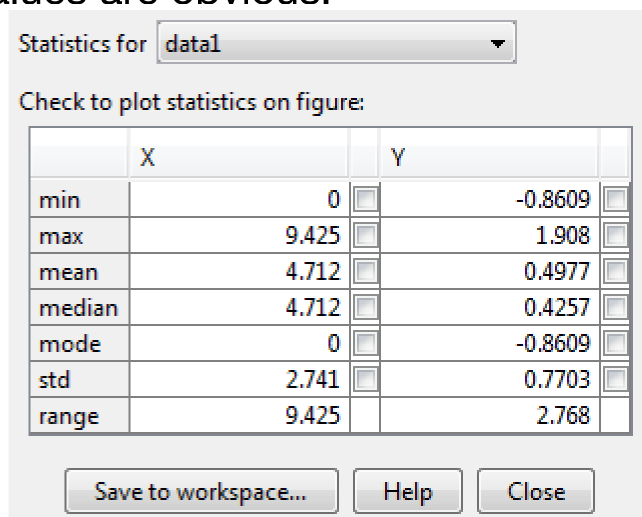
- After modification, you can generate Matlab code to recreate the figure later:
File → Generate Code
- For repeated analysis, it is preferable to configure the plot programmatically, using the functions **`axes`**, **`xlabel`**, **`xlim`**, …, among others.

Statistics Toolbox – Exploratory Data Analysis

- To get a basic statistical overview of the considered data, the “**Data Statistics**” command is useful.



- The statistics window gives the overview for the individual dimensions.
  The following values are obvious:
  - min
  - max
  - range



	X	Y
min	0	-0.8609
max	9.425	1.908
mean	4.712	0.4977
median	4.712	0.4257
mode	0	-0.8609
std	2.741	0.7703
range	9.425	2.768

- **mean**, **median**, **mode** and **std** are described in the following.

Statistics Toolbox – Exploratory Data Analysis

- **mean** calculates the average or mean value of arrays.

```
M = mean(A);
```

If A = $(a_1, \dots, a_n)$ is a vector, then M is the mean of its elements:

$$\mathrm{M} = \frac{1}{n}\sum_{i=1}^{n} a_i$$

It is also possible to calculate the mean along a single dimension of matrices and multidimensional arrays (see documentation).

Source: http://www.mathworks.com/help/matlab/ref/mean.html

- **median** returns a number separating the higher half of a data sample from the lower half. If there is an even number of observations, then the median is defined as the mean of the two middle values.

```
M = median(A);
```

If A = $(a_0, \dots, a_n)$ is a (sorted!) vector, then M is the median of its elements:

$$\mathrm{M} = \begin{cases} a_{n/2} & \text{for } n \text{ even} \\ \dfrac{a_{n-1/2} + a_{n+1/2}}{2} & \text{for } n \text{ odd} \end{cases}$$

Versions for matrices and multidimensional arrays are again available.

Source: http://www.mathworks.com/help/matlab/ref/median.html

Statistics Toolbox – Exploratory Data Analysis

- **mode** returns the most frequent value in an array (only makes sense for discrete data).

```
M = mode(A);
```

It is also possible to calculate the mode for matrices and for multidimensional arrays (see documentation).
Source: http://www.mathworks.com/help/matlab/ref/mode.html

- **std** returns the standard deviation of an array, which is a measure of its dispersion or spread.

```
S = std(A);
```

If A = $(a_1, \ldots, a_n)$ is a vector, then the (empirical) standard deviation is defined as:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mathbf{mean}(a))^2}$$

Versions for matrices and multidimensional arrays are again available (see documentation).
http://www.mathworks.com/help/matlab/ref/std.html

- **var** calculates the variance of an array and is also widely used in statistics. $\mathrm{var} = \mathrm{std}^2$

```
V = var(A);
```

http://www.mathworks.com/help/matlab/ref/var.html

Statistics Toolbox – Exploratory Data Analysis

- **cov** calculates the covariance matrix of / between arrays.

```
C = cov(A, B);
```

If `A` = $(a_1, \ldots, a_n)$, `B` = $(b_1, \ldots, b_n)$ are vectors, then `C` is the covariance matrix of `A` and `B`.

This is the following 2x2 matrix:

$$
\mathrm{C} = \begin{pmatrix} \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mathbf{mean}(\mathrm{A}))\cdot(a_i - \mathbf{mean}(\mathrm{A})) & \frac{1}{n-1}\sum_{i=1}^{n}(\mathrm{b}_i - \mathbf{mean}(\mathrm{B}))\cdot(a_i - \mathbf{mean}(\mathrm{A})) \\ \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mathbf{mean}(\mathrm{A}))\cdot(b_i - \mathbf{mean}(\mathrm{B})) & \frac{1}{n-1}\sum_{i=1}^{n}(b_i - \mathbf{mean}(\mathrm{B}))\cdot(b_i - \mathbf{mean}(\mathrm{B})) \end{pmatrix}
$$

Observe that this matrix is symmetric along its main diagonal. Furthermore, `C(1, 1) == `**`var`**`(A)` and `C(2, 2) == `**`var`**`(B)`.

It is also possible to calculate the covariance for matrices and further objects (see documentation). For a detailed <u>AN</u>alysis <u>O</u>f <u>VA</u>riance and Covariance the so called ANOVA methods can be useful.

Source: http://www.mathworks.com/help/matlab/ref/cov.html, http://www.mathworks.com/help/stats/analysis-of-variance-and-covariance.html

Statistics Toolbox – Exploratory Data Analysis

- **`corrcoef`** calculates the correlation coefficients of / between arrays.

```
R = corrcoef(A,B);
```

If `A` = $(a_1, \dots, a_n)$, `B` = $(b_1, \dots, b_n)$ are vectors, then `R` is the correlation matrix of `A` and `B`.

$$R = \begin{pmatrix} \frac{\frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mathbf{mean}(\mathrm{A}))\cdot(a_i - \mathbf{mean}(\mathrm{A}))}{\mathbf{std}(A)\cdot\mathbf{std}(A)} & \frac{\frac{1}{n-1}\sum_{i=1}^{n}(\mathrm{b}_i - \mathbf{mean}(\mathrm{B}))\cdot(a_i - \mathbf{mean}(\mathrm{A}))}{\mathbf{std}(\mathrm{B})\cdot\mathbf{std}(A)} \\ \frac{\frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mathbf{mean}(\mathrm{A}))\cdot(\mathrm{b}_i - \mathbf{mean}(\mathrm{B}))}{\mathbf{std}(A)\cdot\mathbf{std}(\mathrm{B})} & \frac{\frac{1}{n-1}\sum_{i=1}^{n}(\mathrm{b}_i - \mathbf{mean}(\mathrm{B}))\cdot(\mathrm{b}_i - \mathbf{mean}(\mathrm{B}))}{\mathbf{std}(\mathrm{B})\cdot\mathbf{std}(\mathrm{B})} \end{pmatrix}$$

Observe that this matrix is again symmetric along its main diagonal. Furthermore, `R(1, 1) == R(2, 2) ==` 1.

It is also possible to calculate the covariance for matrices and further objects (see documentation).

Source: http://www.mathworks.com/help/matlab/ref/corrcoef.html

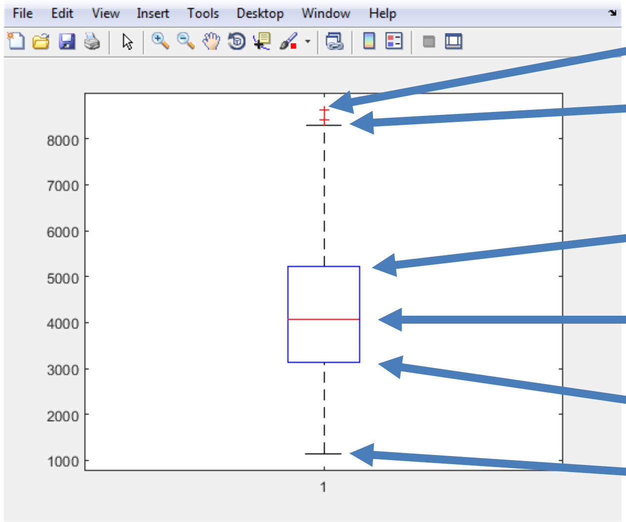
Statistics Toolbox – Exploratory Data Analysis

- **`boxplot`** is a further useful and widely used visualization tool.

```
boxplot(X);
```

If X is a matrix, there is one box per column; if X is a vector, there is just one box.

```
load cities.mat
boxplot(ratings(:, 5))
```



single data points, "outliers"

upper "whisker", by default this is 75 % quantile + 1.5 * (75 % quantile – 25 % quantile)

75 % quantile, i.e. 75 % of the data below that value

median (= 50 % quantile)

25 % quantile

lower "whisker"

Source: http://www.mathworks.com/help/stats/boxplot.html

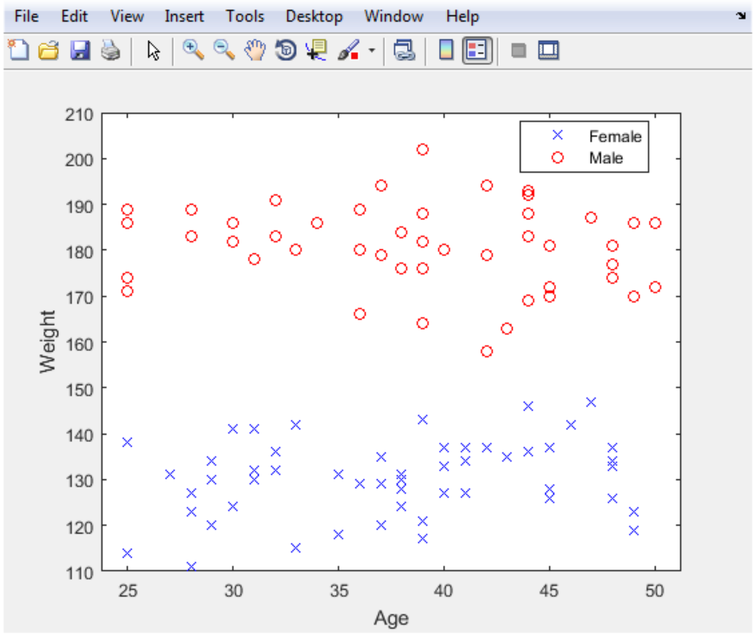
Statistics Toolbox – Exploratory Data Analysis

- **`grpstats`** and **`gscatter`** are tools to analyze data categorized in different groups.

```
load hospital.mat
ds = hospital(:, {'Sex', 'Age', 'Weight', 'Smoker'});
statarray = grpstats(ds, 'Sex')
```

```
statarray =

              Sex       GroupCount    mean_Age    mean_Weight    mean_Smoker
    Female    Female    53            37.717      130.47         0.24528
    Male      Male      47            38.915      180.53         0.44681
```

```
figure;
gscatter(ds.Age, ds.Weight, ds.Sex, 'br', 'xo')
xlabel('Age');
ylabel('Weight');
```

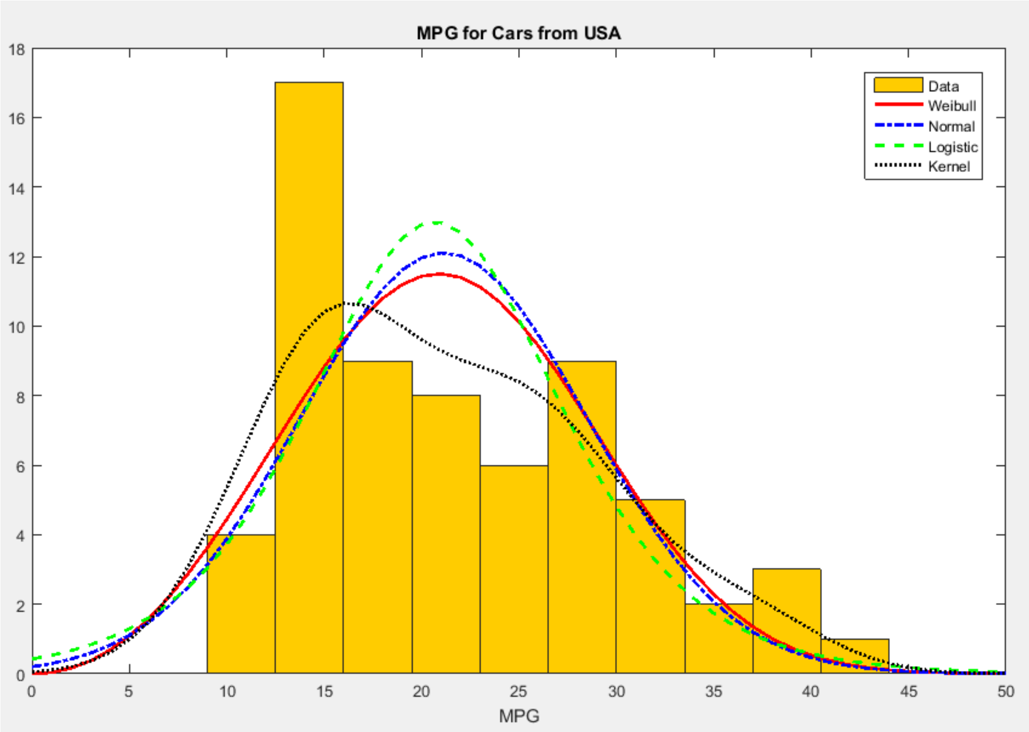


Source: https://de.mathworks.com/help/stats/gscatter.html

Lehrstuhl für Flugsystemdynamik

Statistics Toolbox – Fitting Distribution Objects

- Fit probability distributions to sample data
- Evaluate probability functions such as pdf and cdf
- Calculate summary statistics such as mean and median
- Visualize sample data
- Generate random numbers



- To generate MATLAB Distribution Objects **`makedist`** and **`fitdist`** can be used

Source: http://www.mathworks.com/help/stats/probability-distributions-1.html, http://www.mathworks.com/help/stats/compare-multiple-distribution-fits.html

Statistics Toolbox – Fitting Distribution Objects

- **makedist** can be used to create probability distribution objects.

```
pd = makedist(distname, Name, Value);
```

- This command creates a probability distribution object for the **distribution** `distname` with one or more distribution parameter values specified by the name-value pair arguments `Name, Value`.

- Available distributions can be seen in the table on the right.

- The command can be used to generate both **discrete** and **continuous** distributions.

- The output `pd` is a MATLAB **object** of the specified distribution class, e.g. `prob.BetaDistribution` for the Beta distribution.

Source: http://www.mathworks.com/help/stats/makedist.html

Distribution Name	Description
'Beta'	Beta distribution
'Binomial'	Binomial distribution
'BirnbaumSaunders'	Birnbaum-Saunders distribution
'Burr'	Burr distribution
'Exponential'	Exponential distribution
'ExtremeValue'	Extreme Value distribution
'Gamma'	Gamma distribution
'GeneralizedExtremeValue'	Generalized Extreme Value distribution
'GeneralizedPareto'	Generalized Pareto distribution
'InverseGaussian'	Inverse Gaussian distribution
'Logistic'	Logistic distribution
'Loglogistic'	Loglogistic distribution
'Lognormal'	Lognormal distribution
'Multinomial'	Multinomial distribution
'Nakagami'	Nakagami distribution
'NegativeBinomial'	Negative Binomial distribution
'Normal'	Normal distribution
'PiecewiseLinear'	Piecewise Linear distribution
'Poisson'	Poisson distribution
'Rayleigh'	Rayleigh distribution
'Rician'	Rician distribution
'tLocationScale'	*t* Location-Scale distribution
'Triangular'	Triangular distribution
'Uniform'	Uniform distribution
'Weibull'	Weibull distribution

Statistics Toolbox – Fitting Distribution Objects

- **`fitdist`** fits a probability distribution object to data.

```
pd = fitdist(x, distname, Name, Value);
```

- This command creates a probability distribution object `pd` by fitting the distribution specified by `distname` to the data in column vector `x`.

- Additional options can be specified by one or more name-value pair arguments `Name, Value`. For example, you can indicate censored data or specify control parameters for the iterative fitting algorithm.

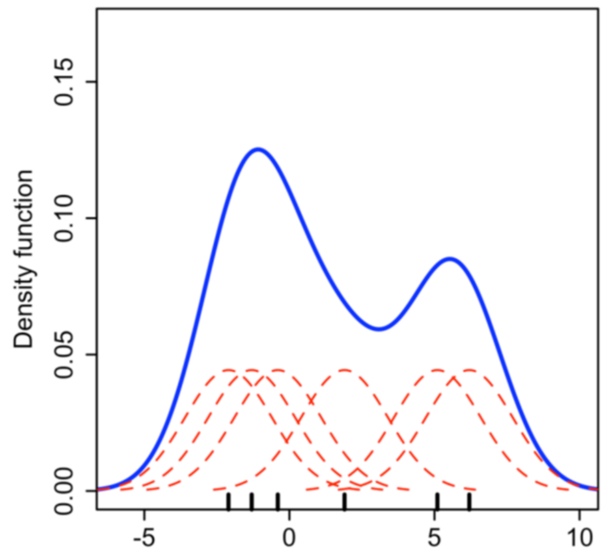
- For a given distribution object, the following commands are very useful:
  - `cdf`
  - `icdf`
  - `iqr`
  - `median`
  - `pdf`
  - `random`
  - `truncate`
  - `mean`
  - `negloglik`
  - `paramci`
  - `proflik`
  - `std`
  - `var`

Source: http://www.mathworks.com/help/stats/fitdist.html

Statistics Toolbox – Fitting Distribution Objects

- **`fitdist`** also implements **Kernel Density Estimation (KDE)**, providing a means to create arbitrary, non-standard distributions from a dataset.
- KDE is a non-parametric way to estimate the probability distribution. Thereby, a so called kernel function is centered at every data point. Subsequently, all the collected kernel functions are summed up and normalized suitably.

```
pd = fitdist(x, 'Kernel');
```



Source: http://www.mathworks.com/help/stats/fitdist.html, https://en.wikipedia.org/wiki/Kernel_density_estimation

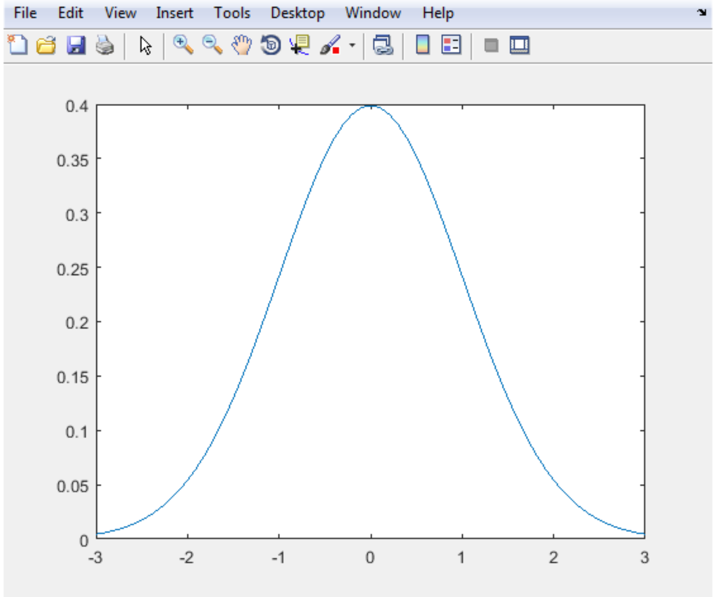
Statistics Toolbox – Fitting Distribution Objects

- **pdf** returns the *probability density function* at the specific value.

```
y = pdf(pd, x);
```

- The `pdf` describes the relative likelihood for the associated random variable to take on a given value.

```
pd = makedist('Normal', 0,1);
x = -3:0.1:3;
y = pdf(pd, x);
plot(x, y)
```



Source: https://de.mathworks.com/help/stats/prob.normaldistribution.pdf.html

Statistics Toolbox – Fitting Distribution Objects

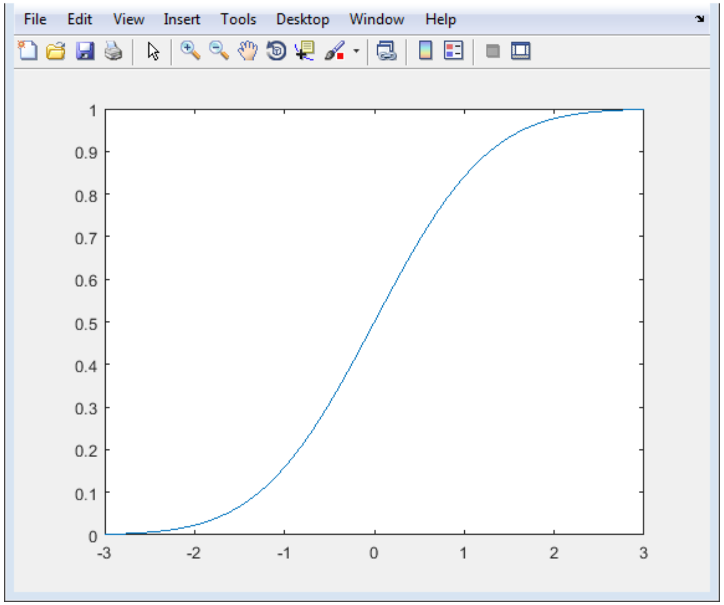
- **cdf** returns the cumulative distribution function at the specific value; it is the integral of the pdf.

```
y = cdf(pd, x);
```

- The `cdf` describes the probability that values less or equal the given value occur.

$$\mathrm{cdf}(x) = \int_{-\infty}^{x} \mathrm{pdf}(z)\, dz$$

```
pd = makedist('Normal',0,1);
x = -3:0.1:3;
y = cdf(pd,x);
plot(x,y)
```



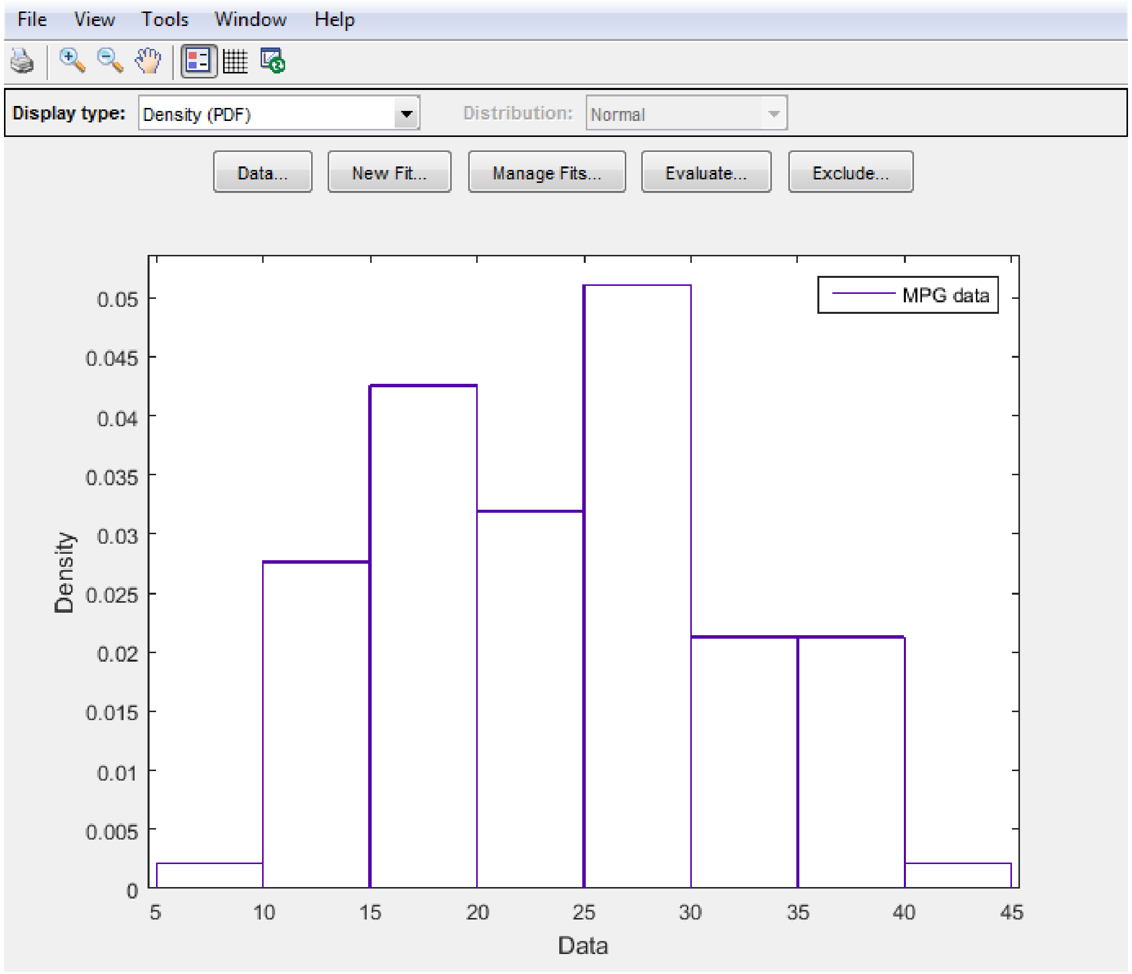
Source: https://de.mathworks.com/help/stats/prob.normaldistribution.cdf.html

Statistics Toolbox – Fitting Distribution Objects

- The **Distribution Fitting Tool** is a nice tool to fit distributions to data

```
dfittool
```

- This command opens the Distribution Fitting app



Source:
http://www.mathworks.com/help/stats/dfittool.html

Statistics Toolbox – Generate Random Numbers

- In MATLAB, there are various methods to generate random numbers.

- **`random`** returns a random number `Y` from the distribution specified by the probability distribution object `pd`.

```
Y = random(pd);
```

- Applying the `cdf` function always transfers the random numbers of any distribution to random numbers of the uniform distribution on the interval $[0, 1]$.

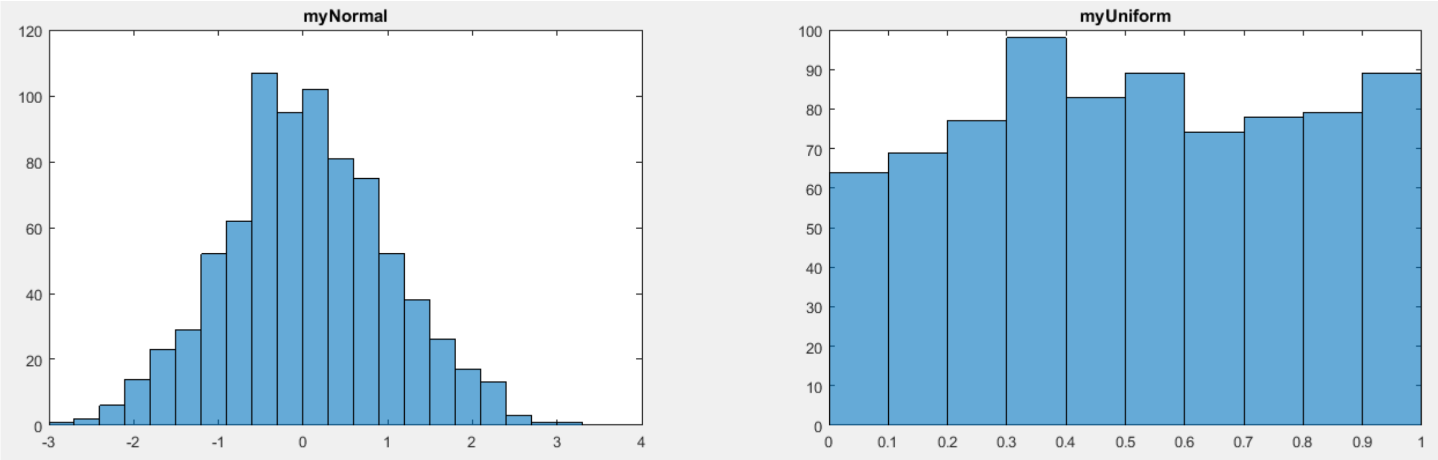
```
pd = makedist('Normal', 0, 1);
Y_normal = random(pd)
Y_uniform = cdf(pd, Y_normal)
```

```
Y_normal =
 -0.7873


Y_uniform =
 0.2156
```

Source: https://de.mathworks.com/help/stats/prob.normaldistribution.random.html

Statistics Toolbox – Generate Random Numbers

```
pd = makedist('Normal', 0, 1);
myNormal = random(pd, 800, 1);
myUniform = cdf(pd,myNormal);
subplot(1, 2, 1);
histogram(myNormal);
title('myNormal')
subplot(1, 2, 2);
histogram(myUniform);
title('myUniform')
```



Statistics Toolbox – Hypothesis Tests

- A statistical **hypothesis test** is a tool to verify certain statistical assumptions.
- A **null hypothesis** H_0 is compared to the statistical **alternative** H_1.
- Statistical hypothesis tests are performed with respect to a **significance level** which can be considered as a threshold probability.
- The **result of the test** is given by rejecting, or failing to reject, the null hypothesis for a pre-specified significance level.
- Various hypothesis tests for different situations are available in MATLAB:
  - **One-sample and paired-sample t-test**
  - **z-test**
  - **Lilliefors test**
  - **One-sample Kolmogorov-Smirnov test**

Source: http://www.mathworks.com/help/stats/hypothesis-testing.html

Statistics Toolbox – Hypothesis Tests

- **One-sample Kolmogorov-Smirnov test**

```
[h, p] = kstest(x, Name, Value);
```

- The command returns a test decision for the **null hypothesis** that the data in vector x comes from a standard normal distribution, against the **alternative** that it does not come from such a distribution, using the one-sample Kolmogorov-Smirnov test. The result h is 1 if the test rejects the null hypothesis at the 5% significance level, or 0 otherwise.

- With the `Name, Value` arguments, **additional options** specified by one or more name-value pair arguments can be given. For example, you can test for a distribution other than standard normal, change the significance level, or conduct a one-sided test.

- An optional output is the ***p*-value**. It indicates the probability under the null hypothesis of observing a value as extreme or more extreme of the ks-statistic computed from the sample.

Source: http://www.mathworks.com/help/stats/hypothesis-testing.html, http://www.mathworks.com/help/stats/kstest.html

Statistics Toolbox – Hypothesis Tests

```
pdNormal = makedist('Normal');
samplesNormal = random(pdNormal, 500, 1);

pdEV = makedist('ExtremeValue', -1, 2);
samplesEV = random(pdEV, 500, 1);

result1 = kstest(samplesNormal, 'CDF', pdNormal)
result2 = kstest(samplesNormal, 'CDF', pdEV)
result3 = kstest(samplesEV, 'CDF', pdNormal)
result4 = kstest(samplesEV, 'CDF', pdEV)
```

```
result1 = 0   % samplesNormal might be from pdNormal (failed to reject null hypothesis)

result2 = 1   % samplesNormal are not from pdEV (at the given significance level)

result3 = 1   % samplesEV are not from pdNormal (at the given significance level)

result4 = 0   % samplesEV might be from pdEV (failed to reject null hypothesis)
```

Statistics Toolbox – Parametric Regression Analysis

- Regression is the process of **fitting models to data**.
- If a model is parametric, **regression estimates the parameters** from the data.
- For a **linear model**, estimation of the model parameters is based on linear algebra.
- For a **nonlinear model**, estimation is based on iterative optimization methods that minimize the norm of a residual vector (see also NLS/`lsqnonlin` in the optimization chapter).
- Models of data with categorical response (e.g. "Safe Flights", "Critical Flights") are called **classifiers**. The related problem is called "Parametric Classification".
- Depending on the regression model and on the result of interest, several methods and algorithms are available in MATLAB. They are summarized on the next slide.

Source: http://www.mathworks.com/help/stats/introduction-to-parametric-regression-analysis.html, http://www.mathworks.com/help/stats/introduction-to-parametric-classification.html

Statistics Toolbox – Parametric Regression Analysis

Choose a Regression Function

You have:	You want:	Use this:
Continuous or categorical predictors, continuous response, linear model	Fitted model coefficients	`fitlm`. See Linear Regression.
Continuous or categorical predictors, continuous response, linear model of unknown complexity	Fitted model and fitted coefficients	`stepwiselm`. See Stepwise Regression.
Continuous or categorical predictors, response possibly with restrictions such as nonnegative or integer-valued, generalized linear model	Fitted generalized linear model coefficients	`fitglm` or `stepwiseglm`. See Generalized Linear Models.
Continuous predictors with a continuous nonlinear response, parametrized nonlinear model	Fitted nonlinear model coefficients	`fitnlm`. See Nonlinear Regression.
Continuous predictors, continuous response, linear model	Set of models from ridge, lasso, or elastic net regression	`lasso` or `ridge` See Lasso and Elastic Net or Ridge Regression.
Correlated continuous predictors, continuous response, linear model	Fitted model and fitted coefficients	`plsregress` See Partial Least Squares.
Continuous or categorical predictors, continuous response, unknown model	Nonparametric model	`fitrtree` or `fitensemble` See Decision Trees or Ensemble Methods.
Categorical predictors only	ANOVA	`anova`, `anova1`, `anova2`, `anovan`
Continuous predictors, multivariable response, linear model	Fitted multivariate regression model coefficients	`mvregress`
Continuous predictors, continuous response, mixed-effects model	Fitted mixed-effects model coefficients	`nlmefit` or `nlmefitsa` See Mixed-Effects Models.

Source: http://www.mathworks.com/help/stats/introduction-to-parametric-regression-analysis.html, http://www.mathworks.com/help/stats/introduction-to-parametric-classification.html

Statistics Toolbox – Parametric Regression Analysis

Example: Fitting a linear model using **`fitlm`**

- **Goal**: Fit a linear model to the following data (only the first 20 of 100 lines illustrated):
- In the sample data set “carsmall”, the following data are given
  - Weight
  - Acceleration
  - MPG (Miles per Gallon)
- Considered model structure:

$$\text{MPG} = C_1 + C_2 * \text{Weight} + C_3 * \text{Acceleration}$$

- Unknown parameters: C_1, C_2, C_3

```
load carsmall

tbl = table(Weight, Acceleration, MPG, 'VariableNames',...
          {'Weight', 'Acceleration', 'MPG'});

lm = fitlm(tbl, 'MPG~Weight+Acceleration')
```

ans =

Weight	Acceleration	MPG
3504	12	18
3693	11.5	15
3436	11	18
3433	12	16
3449	10.5	17
4341	10	15
4354	9	14
4312	8.5	14
4425	10	14
3850	8.5	15
3090	17.5	NaN
4142	11.5	NaN
4034	11	NaN
4166	10.5	NaN
3850	11	NaN
3563	10	15
3609	8	14
3353	8	NaN
3761	9.5	15
3086	10	14

Statistics Toolbox – Parametric Regression Analysis

- Output of **fitlm**:

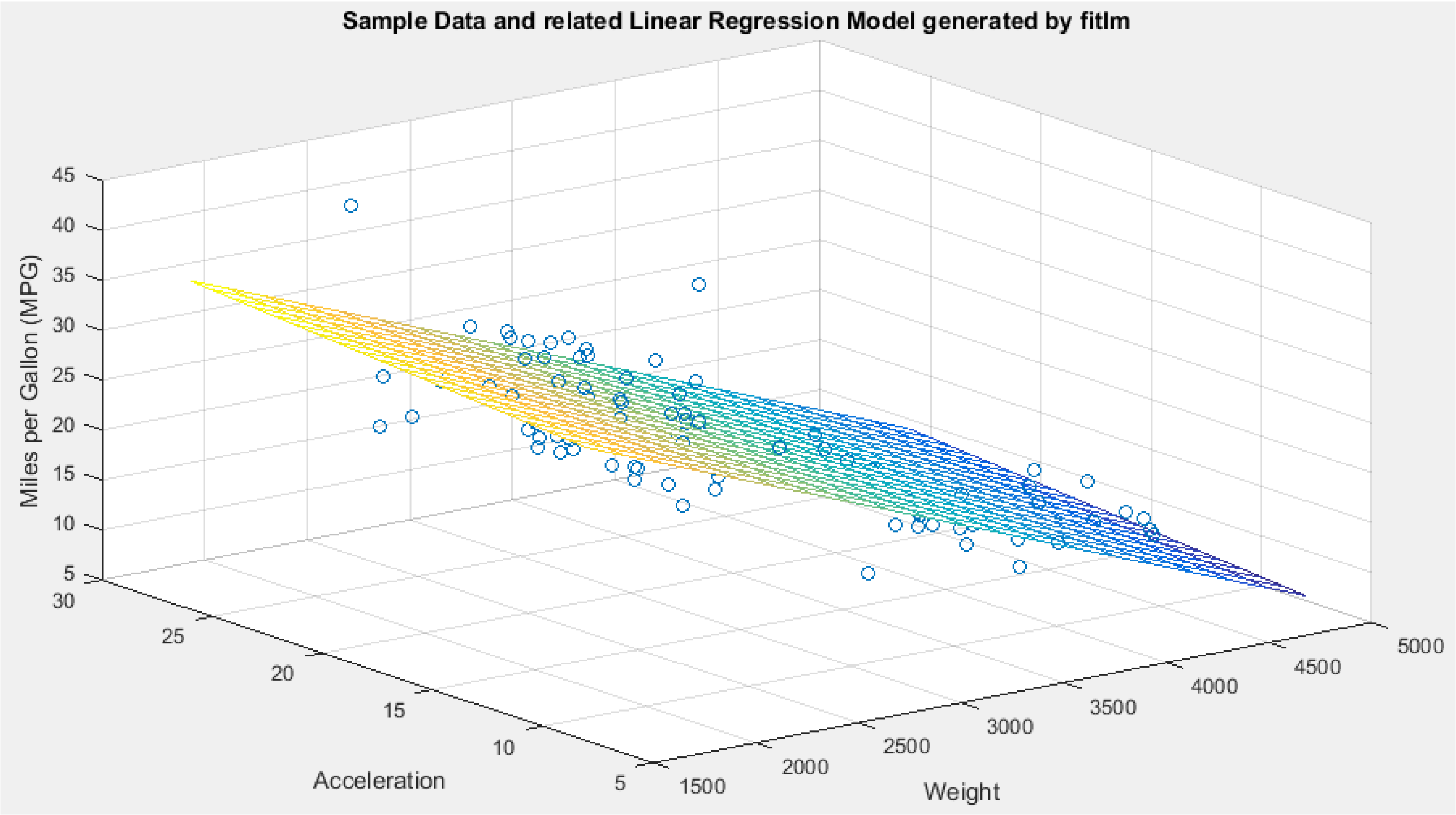
```
lm =

Linear regression model:
    MPG ~ 1 + Weight + Acceleration

Estimated Coefficients:
                    Estimate        SE          tStat       pValue
                   __________    __________    _______    __________

    (Intercept)        45.155        3.4659     13.028    1.6266e-22
    Weight         -0.0082475    0.00059836    -13.783    5.3165e-24
    Acceleration      0.19694       0.14743     1.3359       0.18493

Number of observations: 94, Error degrees of freedom: 91
Root Mean Squared Error: 4.12
R-squared: 0.743,  Adjusted R-Squared 0.738
F-statistic vs. constant model: 132, p-value = 1.38e-27
```

C_1 C_2 C_3

Source: http://www.mathworks.com/help/stats/fitlm.html

Statistics Toolbox – Parametric Regression Analysis

- Illustration of **fitlm**-output:



Source: http://www.mathworks.com/help/stats/fitlm.html

List of Commands

Command	Explanation
`linprog`	Solve linear programming problems
`intlinprog`	Mixed-integer linear programming
`quadprog`	Solve quadratic programming problems
`fminunc`	Minimization of unconstrained nonlinear multivariable function
`optimoptions`	Create optimization options
`fmincon`	Minimization of constrained nonlinear multivariable function
`lsqnonlin`	Solve nonlinear least-squares problems
`scatter`	Generate Scatter plot
`linspace`	Generate linearly spaced vector
`rand`	Returns uniformly distributed random number
`mean`	Returns average value of array

Command	Explanation
`median`	Returns median of an array
`mode`	Returns most frequent value of an array
`std`	Returns the standard deviation of an array
`var`	Returns the variance
`cov`	Returns the covariance
`corrcoef`	Returns the correlation coefficients
`boxplot`	Creates a box plot
`grpstats`	Summary statistics organized by group
`gscatter`	Scatter plot by group
`makedist`	Create probability distribution object
`fitdist`	Fit probability distribution object to data

List of Commands continued

Command	Explanation
`pdf`	Probability density function
`cdf`	Cumulative distribution function
`random`	Generate random numbers
`dfittool`	Open Distribution Fitting app
`kstest`	One-sample Kolmogorov-Smirnov test
`fitlm`	Create a linear regression model
`table`	Create a table from workspace variables