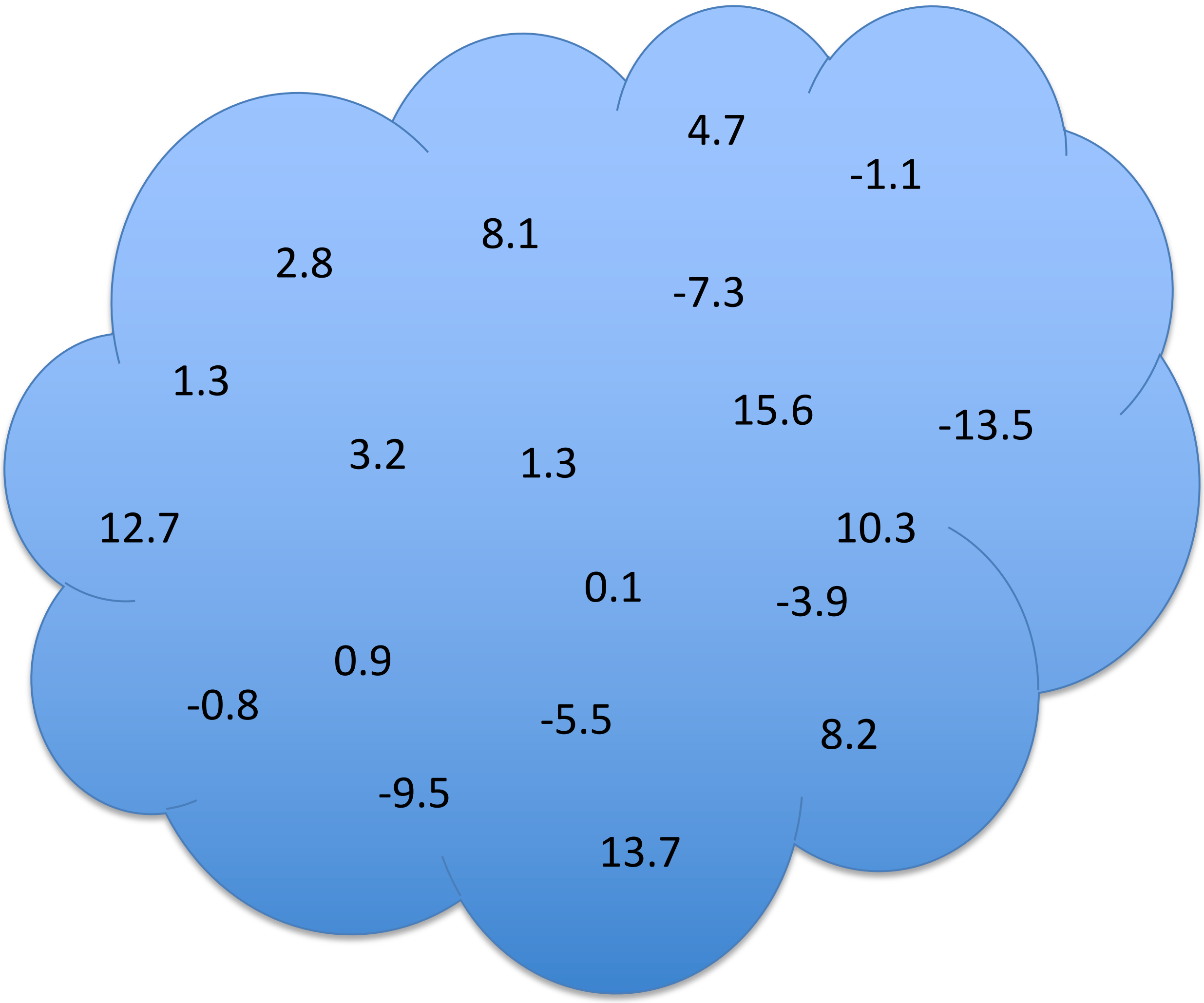
Unit 4

Data Analysis



what are the characteristics of this 'cloud' of numbers?

▪ **describing data sets – single numbers:**

given a data set, e.g. a number of experimentally measured values, one can calculate various (statistical) quantities to describe the data:

data set: for example, monthly average temperature in Madrid

$$T = [8.5, 11.0, 14.9, 18.4, 21.2, 26.9, 30.8, 29.5, 25.0, 18.5, 12.8, 8.8]$$

(the sequence covers the months [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sept, Oct, Nov, Dec])

statistical description:

1. **`mean()`**:

$$\langle T \rangle = \frac{1}{N} \sum_{i=1}^{N} T_i$$

2. **`std()`**:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(T_i - \langle T \rangle\right)^2}$$

Note:

• the standard deviation `std()` measure the scatter of data points about the mean, e.g. consider the following two data sets with the same mean:

A = [9, 5, 10], $\langle A \rangle$=8, σ_A=2.65
B = [7, 8, 9], $\langle B \rangle$=8, σ_B=0.82

➔ it is obvious that each individual value in set B is closer to the mean which is quantified by the substantially lower standard deviation

3. **`mode()`**: the most frequent value

4. **`median()`**: $T_{1/2}$ = data point that separates the higher from the lower half of the sample

data sample: $T = [8.5, 11.0, 14.9, 18.4, 21.2, 26.9, 30.8, 29.5, 25.0, 18.5, 12.8, 8.8]$

sorted data: $T = [\underbrace{8.5, 8.8, 11.0, 12.8, 14.9, 18.4}_{\text{lower half}}, \underbrace{18.5, 21.2, 25.0, 26.9, 29.5, 30.8}_{\text{upper half}}]$

^

mid-point

median: $T_{1/2}$ = (18.4+18.5)/2 = 18.45

Note:

• if the number of data points N is even the median could be calculated as the mean of the two values bracketing the mid-point of the sorted sample

• if the number of data points N is odd the median is that data point with the index `floor(N/2)+1`

▪ **Note:** we are dealing with a 1D data set, i.e. a collection of numbers

Data Analysis *statistics*

➢ **exercise**:

- calculate mean, standard deviation, and median of the monthly average temperature in Madrid
- compare the values for Madrid to the values for London:

$$T_{\text{London}} = [6.4\ ,7.1,\ 10.1,\ 13.3,\ 16.9,\ 20.3,\ 21.8,\ 21.5,\ 18.5,\ 14.2,\ 10.1,\ 7.4]$$

- remarks:
  - do ***not*** use MATLAB's in-built functions `mean()`, `median()`, and `std()`
  - remember the usage of `sum()` and `length()`
  - you could use `sort()` for the median

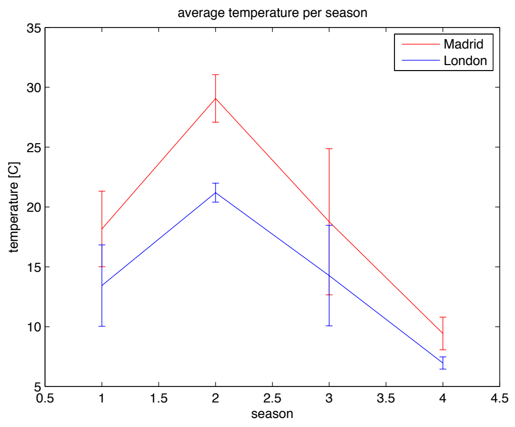
- **describing data sets – "binning"**:
  - most of the times it is useful to combine data points on certain intervals, e.g. *to bin the data*.
  - for the example of the temperatures one can, for instance, bin the data in four intervals:

• spring	= Mar, Apr, May
• summer	= Jun, Jul, Aug
• autumn	= Sep, Oct, Nov
• winter	= Dec, Jan, Feb

➢ **exercise**:

- bin both the temperature in Madrid and London into the four seasons
- plot the mean in each bin as a function of season
- use the standard deviation of the values in each bin as error bars

- ***hints***:
  - now you can use the in-built functions `mean()` and `std()`
  - use the command `errorbar()` (cf. Unit 1) to plot the binned data
  - the resulting plot should look like this:



• ***Note***: you need to number the four seasons, e.g..
1 = spring
2 = summer
3 = autumn
4 = winter

Data Analysis *statistics*

▪ **describing data sets – probability distribution**:

- the *probability distribution $p(x)$* tells you the probability to find the value x within a certain interval $[x_{min},x_{max}]$ when performing an experiment, e.g. when measuring the temperature in Madrid in summer.

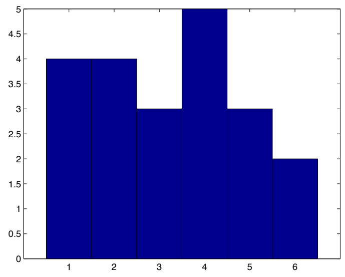
1. to obtain the ***number distribution*** we need to measure how many elements exist within a certain internal, e.g.

▪ **hist() – method A**

```
>> data = [1,5,3,6,4,2,1,4,3,2,1,4,5,3,2,4,6,2,1,4,5];
>> x    = [1 2 3 4 5 6];
>> hist(data,x)
```

➔ the function `hist(data,x)` generates a histogram of the data at the positions `x`, i.e. `hist()` bins the elements of the vector `data()` into `length(x)` containers centred on the values stored in `x()`:

`data()` contains...
4 times the number 1
4 times the number 2
3 times the number 3
5 times the number 4
3 times the number 5
2 times the number 6

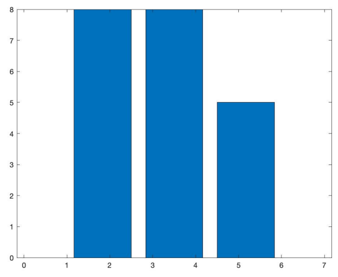


▪ **hist() – method B**

```
>> data  = [1,5,3,6,4,2,1,4,3,2,1,4,5,3,2,4,6,2,1,4,5];
>> Nbins = 3;
>> [N,x]=hist(data,Nbins)
>> bar(x,N)
```

➔ the function `hist(data,Nbins)` generates a histogram of the data using Nbins number of bins, i.e. `hist()` divides the interval [min(data),max(data)] into Nbins equally spaced bins and counts the number of data values in each of those bins. Note, `hist()` returns the number distribution `N()` as well as `x()` that will contain the centre position of the bins.

`data()` contains...
8 numbers in the interval [1.000,2.666]
8 numbers in the interval [2.666,4.333]
5 numbers in the interval [4.333,6.000]



▪ use `help hist` to find out more about `hist()`

Data Analysis *statistics*

▪ **describing data sets – probability distribution**:

- the *probability distribution $p(x)$* tells you the probability to find the value x within a certain interval $[x_{min},x_{max}]$ when performing an experiment, e.g. when measuring the temperature in Madrid in summer.

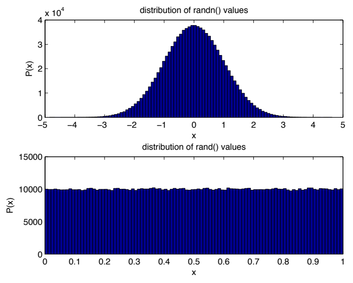
➢ **exercise**:

▪ write a script that shows that...

- the function `randn()` returns Gaussian-distributed random numbers centred on 0
- the function `rand()` returns uniformly-distributed random numbers on the interval [0,1]

▪ ***hints***:

- choose the number of data points large enough (N>100000)
- do **not** provide the positions `x`
- use a sensible number of bins N_{bins} (more bins than data points is obviously meaningless!)
- your figure should look similar to this one:



• ***Notes***:

- the amplitude of the histogram depends on the number of points N
- the amplitude of the histogram depends on the number of bins N_{bins}

→ play with both these numbers and observe/understand what happens!

➔ to obtain the actual ***probability distribution*** we need to normalise the histogram!

Data Analysis *statistics*

▪ **describing data sets – probability distribution**:

• the *probability distribution* $p(x)$ tells you the probability to find the value x within a certain interval $[x_{min},x_{max}]$ when performing an experiment, e.g. when measuring the temperature in Madrid in summer.

2. to obtain the ***probability distribution*** we need to normalise the histogram, i.e. the probability distribution $p(x)$ has to fulfill the criterion

$$1 = \int_{-\infty}^{+\infty} p(x)dx$$

```
>> data = [1.1,5.6,3.2,6.2,4.3,2.5,1.2,4.1,3.9,2.3,1.3,4.4,5.2,3.7,2.3,4.7,6.2,2.1,1.9,4.4,5.6];
>> [N,x] = hist(data,10)
>> p = normalise(N,x)
```

Note the different usage of `hist()` in this example!

➢ **exercise**:

▪ write a script function `normalise(N,x)` that normalises the integral over $N(x)$ to unity.

▪ ***hints***:

▪ the number distribution $N(x)$ obtained with `hist()` is not normalized but gives

$$C = \int_{-\infty}^{+\infty} N(x)dx$$

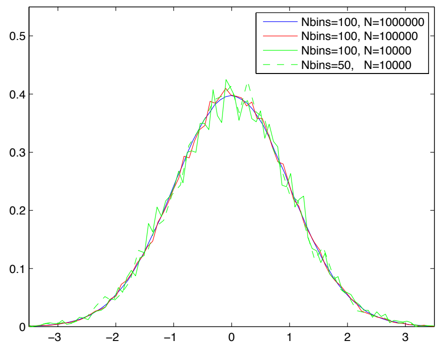
➞ dividing both sides by C leads to a normalized distribution!

▪ the bounds of the integrals should in principle be ±∞, but you only have data for $[x_{\min},x_{\max}]$

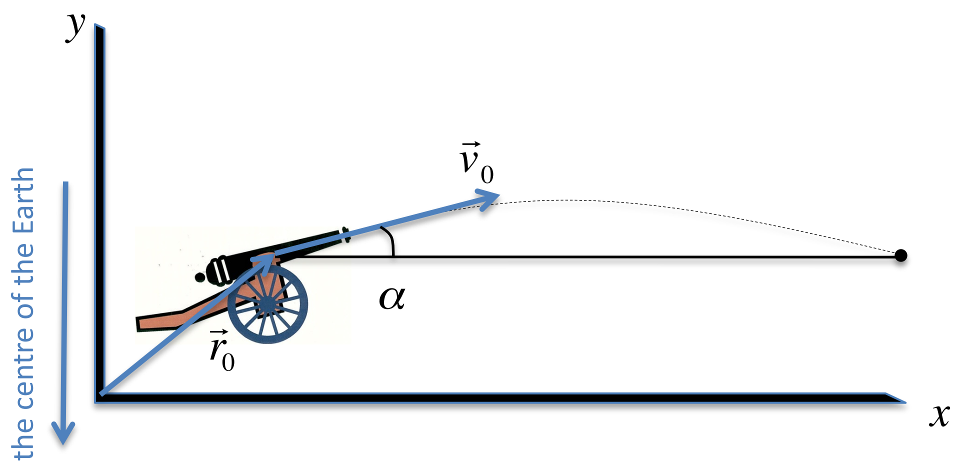
➢ **exercise**:

▪ use `normalise.m` to plot again the distribution of values returned by `randn()`.

▪ show that $p(x)$ now does not depend on the number of points N or bins N_{bins} anymore by generating a figure like this:



Probabilities *cannonball*



➢ **exercise**:

- adjust your **cannonball.m** script from Unit 1 in the following way:
  - let the angle α vary randomly between 5° and 85⁰ keeping the initial velocity constant
  - for each random angle record the distance until reaching the starting level y_0 again
  - plot the distribution of distances $x(T_{\mathrm{end}})$
  - correlate distances with angles (and explain the observed correlation!)

➢ **exercise**:

- compare the distributions of distance and correlations between angles and distances for a cannonball shot on Earth and on the Moon.

Probabilities *random walk*

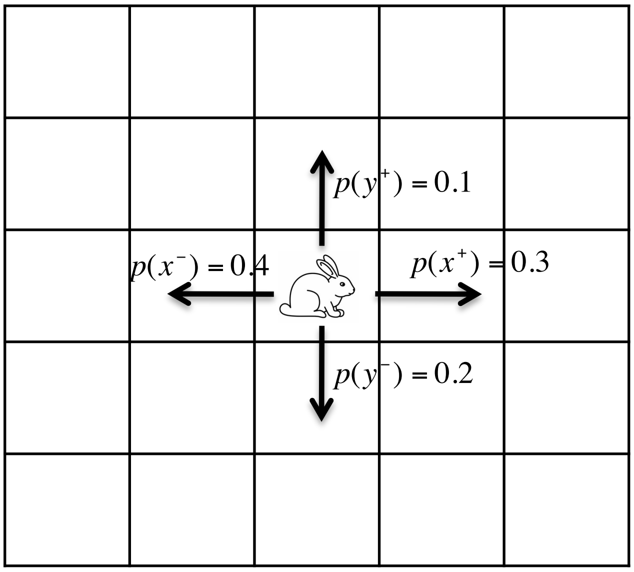
- a rabbit is jumping back and forth as well as left an right with different probabilities in each direction:

$p(x^+)$	= 0.3; probability to jump right
$p(x^-)$	= 0.4; probability to jump left
$p(y^+)$	= 0.1; probability to jump up
$p(y^-)$	= 0.2; probability to jump down

(Note that the probabilities must sum to unity!)

The rabbit jumps the following distances:

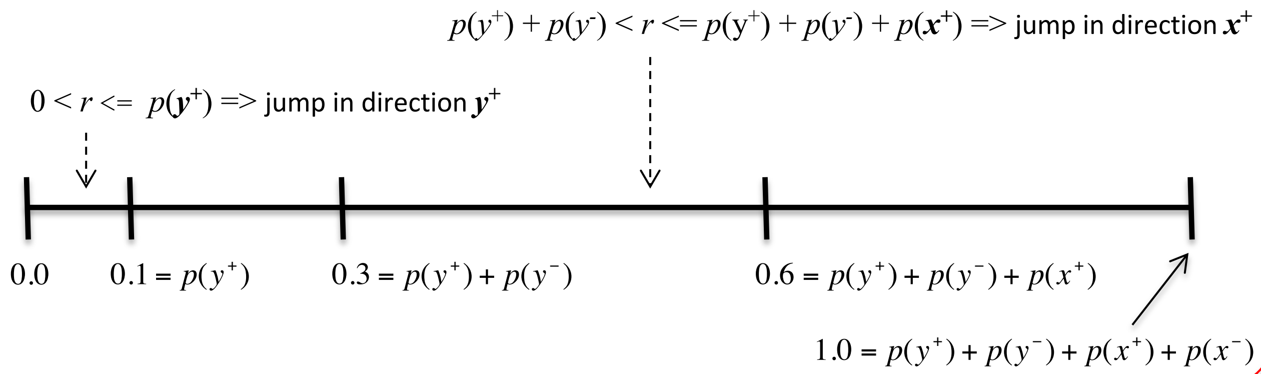
right:	Δx^+ = +5 cm,
left:	Δx^- = -4 cm,
up:	Δy^+ = +20 cm,
down:	Δy^- = -10 cm.



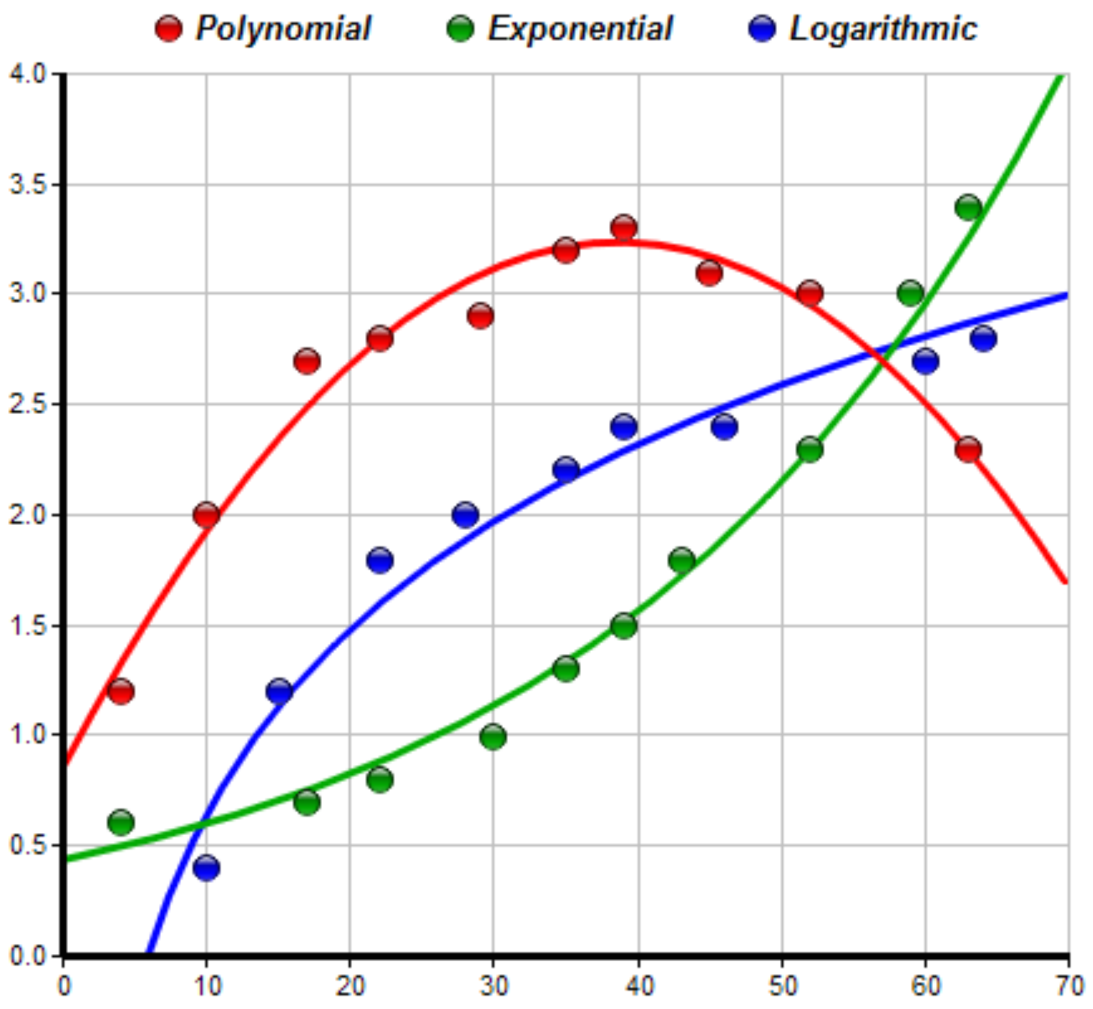
➢ **exercise**:

- calculate the trajectory of the rabbit 1000 jumps when starting at (0,0)
- plot its trajectory marking the starting and end-point with a cross (using different colours for the crosses).
- ***hint***: the direction of each jump can be calculated as follows:
  - pick a (uniform) random number between 0 and 1: r = rand(1)
  - check in which interval r lies:

$0 < r <= p(y^+)$ ⇒ jump distance Δy^+ in direction y^+
$p(y^+) < r <= p(y^+) + p(y^-)$ ⇒ jump distance Δy^- in direction y^-
$p(y^+) + p(y^-) < r <= p(y^+) + p(y^-) + p(x^+)$ ⇒ jump distance Δx^+ in direction x^+
$p(y^+) + p(y^-) + p(x^+) < r <= p(y^+) + p(y^-) + p(x^+) + p(x^-)$ ⇒ jump distance Δx^- in direction x^-


$p(y^+) + p(y^-) < r <= p(y^+) + p(y^-) + p(\boldsymbol{x}^+)$ => jump in direction $\boldsymbol{x}^+$
$0 < r <= p(\boldsymbol{y}^+)$ => jump in direction $\boldsymbol{y}^+$
0.0
$0.1 = p(y^+)$
$0.3 = p(y^+) + p(y^-)$
$0.6 = p(y^+) + p(y^-) + p(x^+)$
$1.0 = p(y^+) + p(y^-) + p(x^+) + p(x^-)$

- Note that this 'random walk' exercise is the reverse to the previous probability exercises:
  - previous exercises = generate $p(x)$ from existing data
  - random walk exercise = generate data from existing $p(x)$



how to describe experimental data with analytical functions?

Data Analysis *fitting*

▪ **modeling data sets**

• after performing an experiment (e.g. measuring the temperature in Madrid every month of the year) we mostly like to describe the results by an analytical function, i.e. our theoretical model, and hence require to fit that model to the data

▪ **fitting in general**

1. we need to express our model as a mathematical formula

$$f(x, a_1, a_2, a_3, \dots, a_{N_{param}})$$

where x is the independent variable and $a_1, \dots, a_{Nparam}$ are the free parameters of the model.

2. we need to fit that formula to the data

▪ **example**: monthly temperature variations in Madrid

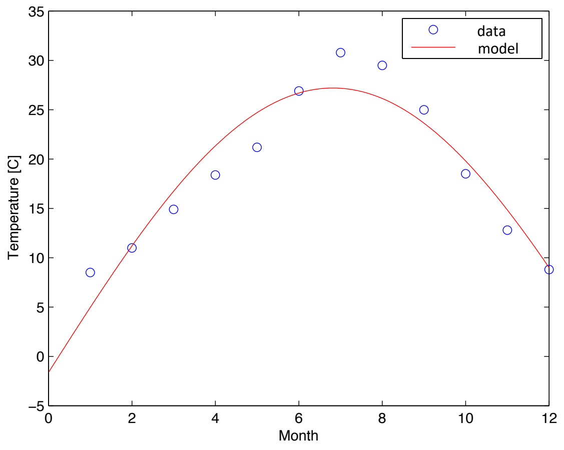
• experimentally obtained data:

T = [8.5, 11.0, 14.9, 18.4, 21.2, 26.9, 30.8, 29.5, 25.0, 18.5, 12.8, 8.8]

• theoretical model (in words):

"the temperature varies like a sine-function"

1. mathematical formula for model: $T(x) = a_1 \sin(a_2 x + a_3)$
2. a_1, a_2, and a_3 are the free parameters of our model to be determined via fitting:



▪ **question**: how can we obtain the best-fit parameters a_1, a_2, a_3 ?

▪ **Note:** we are dealing with a 2D data set, i.e. a set of independent variables and corresponding function values

Data Analysis *fitting*

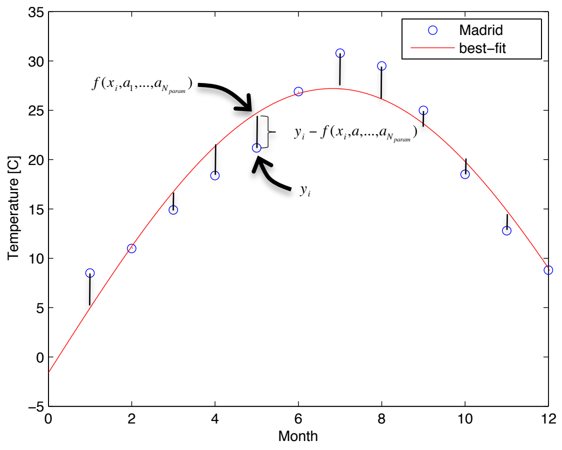
▪ least-square fitting

- "least-square" means that to obtain the best-fit parameters you are minimizing the function

$$F(a_1,\dots,a_{N_{param}}) = \sum_{i=1}^{N}\left(y_i - f(x_i,a_1,\dots,a_{N_{param}})\right)^2$$

where (y_i, x_i) are the data points and $f(x_i,a_1,\dots,a_{Nparam})$ the analytical model.

- the geometrical interpretation of this technique can be visualized as follows



➔ one assumes a model given $a_1,\dots,a_{Nparam}$ and calculates $F(a_1,\dots,a_{Nparam})$ which is the sum of the square of the differences between the model and the data; this sum is minimal for the best-fit parameters!

- minimizing $F(a_1,\dots,a_{Nparam})$ means calculating the solution to

$$\frac{\partial F}{\partial a_1} = 0 = -2\sum_{i=1}^{N}\left(y_i - f(x_i,a_1,\dots,a_{N_{param}})\right)\frac{\partial f(x_i,a_1,\dots,a_{N_{param}})}{\partial a_1}$$

$$\dots$$

$$\frac{\partial F}{\partial a_{N_{param}}} = 0 = -2\sum_{i=1}^{N}\left(y_i - f(x_i,a_1,\dots,a_{N_{param}})\right)\frac{\partial f(x_i,a_1,\dots,a_{N_{param}})}{\partial a_{N_{param}}}$$

▪ *Notes*:

- the more parameters you use, the better your fit will be
- the less parameters you use, the better your model!
- N_{param} is the number of parameters of your model, N is the number of your data points
- "least-square" is just one of many possible techniques to optimize the fit

Data Analysis *fitting*

▪ linear least-square fitting

- in case $f(x,a_1,a_2,a_3,\ldots)$ only contains a linear combination of the N_{param} fit parameters $a_1,a_2,a_3,\ldots$

$$f(x,\vec{a}) = \sum_{j=1}^{N_{param}} a_j\ \phi_j(x)$$

we can find a solution to the least-square minimization by solving a linear system for $\boldsymbol{a}=(a_1,a_2,a_3,\ldots)$

Note: this method will not work for models such as, for instance, $f(x,\vec{a}) = a_1\sin(a_2x + a_3)$

- the optimization is then equivalent to solving

$$0 = \sum_{i=1}^{N}\big(y_i - f(x_i,\vec{a})\big)\ \phi_j(x_i) \qquad \forall j \in N_{param}$$

which is a linear system for $a_1,a_2,a_3,\ldots$

$$\begin{pmatrix} a_1 \\ \ldots \\ a_{N_{param}} \end{pmatrix} = \left(X^TX\right)^{-1}X^T\begin{pmatrix} y_1 \\ \ldots \\ y_N \end{pmatrix} \qquad X_{ij} = \frac{\partial f(x_i,\vec{a})}{\partial a_j} = \phi_j(x_i) \Rightarrow X = \begin{pmatrix} \phi_1(x_1) & \ldots & \phi_{Nparam}(x_1) \\ \ldots & \ldots & \ldots \\ \phi_1(x_N) & \ldots & \phi_{Nparam}(x_N) \end{pmatrix}$$

i: line index
j: column index

$(x_1,y_1),\ldots,(x_N,y_N)$ are your data points

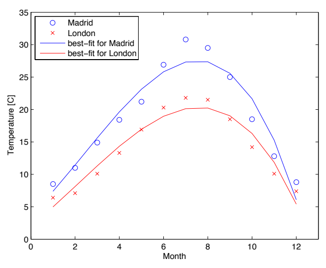
- ***Note***: the $\phi_j(x)$ can be arbitrary (non-linear!) functions of x, e.g. fitting $f(x,\vec{a}) = a_1\sin(x)$ will work!

➢ **exercise**: *monthly temperature variations in Madrid and London*

- experimentally obtained data:

T_{Madrid} = [8.5, 11.0, 14.9, 18.4, 21.2, 26.9, 30.8, 29.5, 25.0, 18.5, 12.8, 8.8]

T_{London} = [6.4 ,7.1, 10.1, 13.3, 16.9, 20.3, 21.8, 21.5, 18.5, 14.2, 10.1, 7.4]



- theoretical model (in words):

"the temperature varies like a 3rd order polynomial"

- write a script that plots the data and determines the best-fit 3rd order polynomial
- what are the best-fit values?
- ***Note***: you require transpose vectors and matrices carefully...

➢ **exercise**: *Gaussian (or "normal") probability distribution*

• a Gaussian probability distribution is characterized by:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\langle x\rangle)^2}{2\sigma^2}}$$

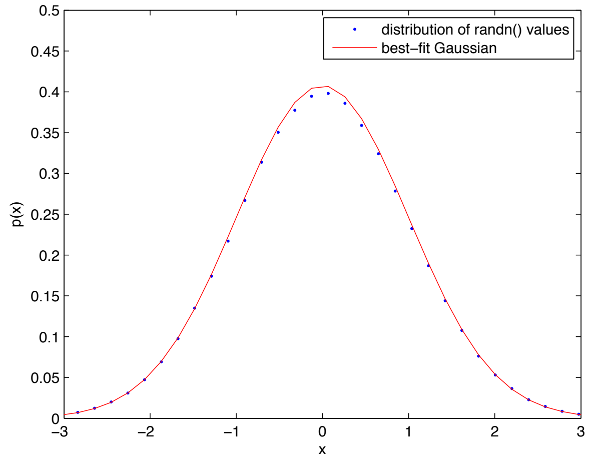
• fit the numerically retrieved probability distribution for the return values of `randn()` to a Gaussian probability distribution of the form (where <x>=0)

$$p(x) = p_0 e^{-ax^2}$$

• use the best-fit p_0 and a to calculate σ? do they agree? what σ do you actually expect?

• do these best-fit values vary with N and N_{bins} used for obtaining the numerical distribution?

• generate a figure similar to the one to the following one:



• ***hints***:

• use `hist()` to generate the distribution of values returned by `randn()`

• do not forget to normalise the distribution obtained with `hist()`

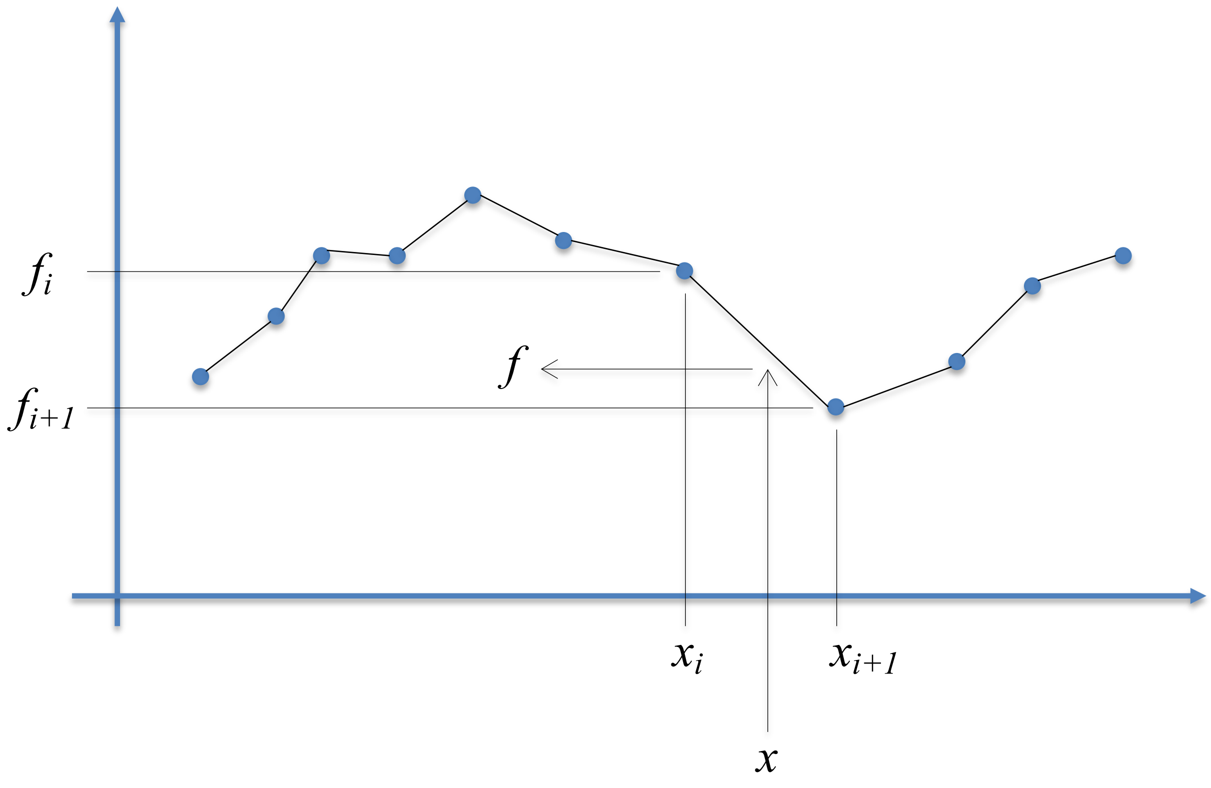
• you can only perform a linear least-square fit of $\ln\big(p(x)\big) = \ln(p_0) - ax^2 = b - ax^2$

• ***Notes***:

• the number of data points used for the fitting is N_{bins} in this exercise

• be careful that none of the values of the distribution returned by `hist()` is zero!

(use `find()` to filter out `p()` values greater than zero; do not forget that you also need to adjust `x()`!)

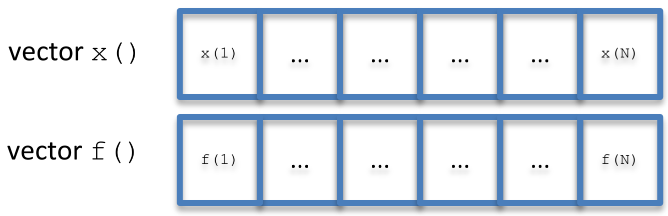


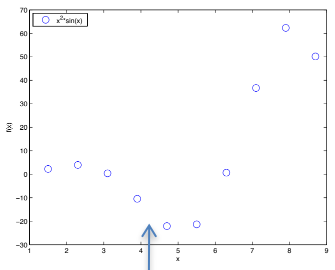
how to obtain function values at non-tabulated positions?

Data Analysis *interpolation*

- **problem**

  - using two vectors `x()` and `f()` to represent a function $f(x)$ the values are only given at discrete points x_i





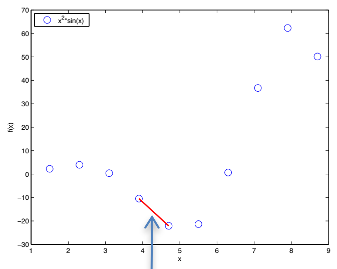
how to calculate $f(x)$ at x=4.1?

  - but how can we calculate $f(x)$ at a point x not stored in `x()` ?

- **solution:**

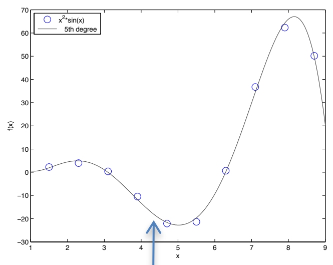
  - interpolate the tabulated data in-between the given data points!

    - ***linear interpolation***: use a straight line to connect x_i with x_{i+1}



use a linear approximation to calculate $f(x)$ at x=4.1

    - ***polynomial interpolation***: use a polynomial that connects M neighbouring points



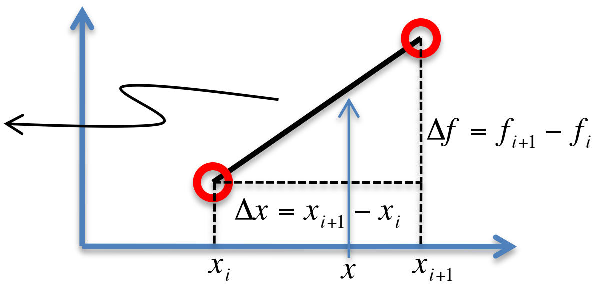
use best-fit polynomial to calculate $f(x)$ at x=4.1

Data Analysis *interpolation*

▪ linear interpolation

• the formula for calculating $f(x)$ at any point x in-between x_i and x_{i+1} (i.e. $x_i<x<x_{i+1}$) is simply:

$$f(x) = f(x_i) + (x - x_i)\left(\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}\right)$$



➢ **exercise**:

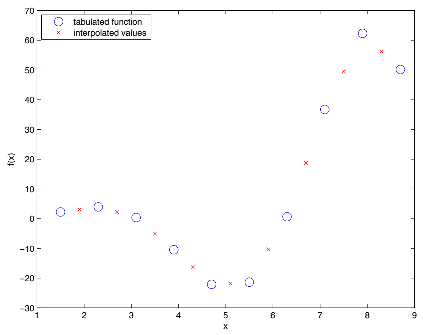
• write a function script linterpol.m that returns $f(x)$ and takes as arguments x_i, x_{i+1}, f_i, f_{i+1}, and x, e.g.

```
function [f] = linterpol(x1,x2,f1,f2,x)
```

• plot $f(x) = x^2 \sin(x)$ on the interval [1.5,8.7] using N=10 points

• use your script function `linterpol()` to overplot the function at the mid-points,

i.e. generate this plot:

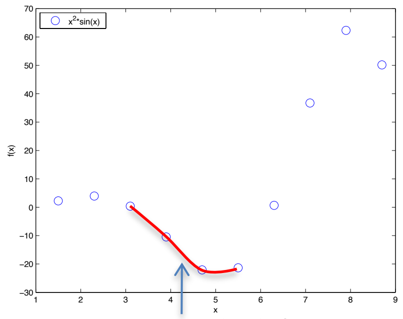


• ***hints***:

• `x1` and `x2` are the points left and right of `x` (and `f1` and `f2` the corresponding function values)

• there are obviously only N-1 mid-points

- **polynomial interpolation**
  - we try to find the best-fit polynomial $p(x) = \sum_{m=0}^{M-1} a_m x^m$ to M consecutive data points
  - those data points should bracket the point x at which we want to calculate $f(x)$



e.g., we are using M=4 points, 2 to the left and 2 to the right of x = 4.1

- **obtaining the unknown coefficients a_n**
  - we require the polynomial ***to go through*** all M points $p(x_i) = f(x_i) = f_i \quad \forall i \in [0, M-1]$
  - this condition can be transformed to the following linear system for the coefficients a_m

$$\begin{bmatrix} x_0^0 & x_0^1 & \dots & x_0^{M-2} & x_0^{M-1} \\ x_1^0 & x_1^1 & \dots & x_1^{M-2} & x_1^{M-1} \\ \dots & \dots & \dots & \dots & \dots \\ x_{M-1}^0 & x_{M-1}^1 & \dots & x_{M-1}^{M-2} & x_{M-1}^{M-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_{M-1} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \dots \\ f_{M-1} \end{bmatrix}$$

  - solving this linear system (cf. Unit 3) will give the coefficients to be used for the polynomial interpolation

  - ***Notes***:
    - the matrix is constructed the same way as done for the fitting exercises

$$X = \begin{bmatrix} \phi_0(x_1) & \dots & \phi_{M-1}(x_1) \\ \dots & \dots & \dots \\ \phi_0(x_N) & \dots & \phi_{M-1}(x_N) \end{bmatrix} \qquad \phi_j(x) = x^j$$

    - the solution is $\boldsymbol{a} = X^{-1}\boldsymbol{f}$ as we force the polynomial to go directly through the points, i.e. $a_0,\dots,a_{M-1}$ will not be best-fit values, but the actual coefficients of the polynomial

Data Analysis *interpolation*

▪ polynomial interpolation

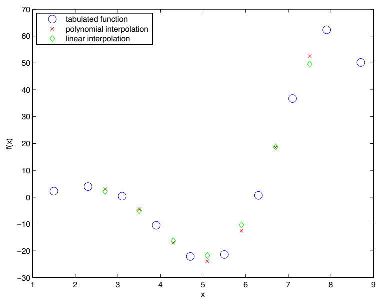
- we try to find the best-fit polynomial $p(x) = \sum_{m=0}^{M-1} a_m x^m$ to M consecutive data points

➢ **exercise**:

- write a function `p4interpol.m` that returns *f(x)* based upon a polynomial interpolation with M=4

```
function [fp] = p4interpol(x,f,xp)
```

  where `x()` and `f()` are the vectors containing x_i and $f(x_i)$, and x_p is the evaluation point for $f(x_p)$
- plot $f(x) = x^2 \sin(x)$ on the interval [1.5,8.7] using N=10 points
- use your script function `p4interpol()` to overplot the function at the mid-points
- plot into the same figure also the result obtained with `linterpol()`
- generate a plot like this one:



- ***hints***:
  - remember MATLAB's function `find()`
  - there are only N-3 mid-points for which this method works
  - note that the mode of operation of `linterpol()` and `p4interpol()` is rather different!

Data Analysis I/O

▪ **reading and writing data sets**

• the data to be analysed is mostly not defined in your script as a vector (or matrix) but instead stored in a file on the hard drive and hence needs to be loaded into memory

• the commands for writing are

write block

```
>> data = [1 2; 3 4; 5 6; 7 8];        define some Nx2 matrix
>> save('file','data');                save data to file.mat  (binary format)
>> save('file','data','-ascii');       save data to file      (ASCII format)
```

• the command for reading is

read block

```
>> load('file');                       reads binary or ASCII file
```

• if `file` is binary , you will recover `data`
• if `file` is ASCII, the data will be written to a variable called `file`

• **Notes**:
- binary files automatically receive an extension `.mat`
- the names of ASCII files are exactly as given in `save()`
- `load()` does not require the extension `.mat`
- you can write multiple variables to file:
  - either list them all: `save('file','var1','var2',...);`
  - or omit any variable name saving everything: `save('file');`
- unless you provide the full path, files will be saved into a MATLAB specific folder

➢ **exercise**:

• split the cannonball.m script from Unit 1 into two separate scripts:

- cannonball-save.m calculates and then saves all variables to a file without plotting anything
- cannonball-load.m reads that files and then does all the plotting

Data Analysis

I/O

advanced formatted input/output

▪ **reading and writing data sets**

• the data to be analysed is mostly not defined in your script as a vector (or matrix) but instead stored in a file on the hard drive and hence needs to be loaded into memory

• the commands for writing are

write block

```
>> data = [1 2; 3 4; 5 6; 7 8];          define some Nx2 matrix
>> fid  = fopen('file.dat','w');         open the file 'file.dat' for writing ('w')
>> fprintf(fid, '%f %f\n', data');       writes data() into 'file.dat'
>> fclose(fid);                          close the file
```

• the commands for reading are

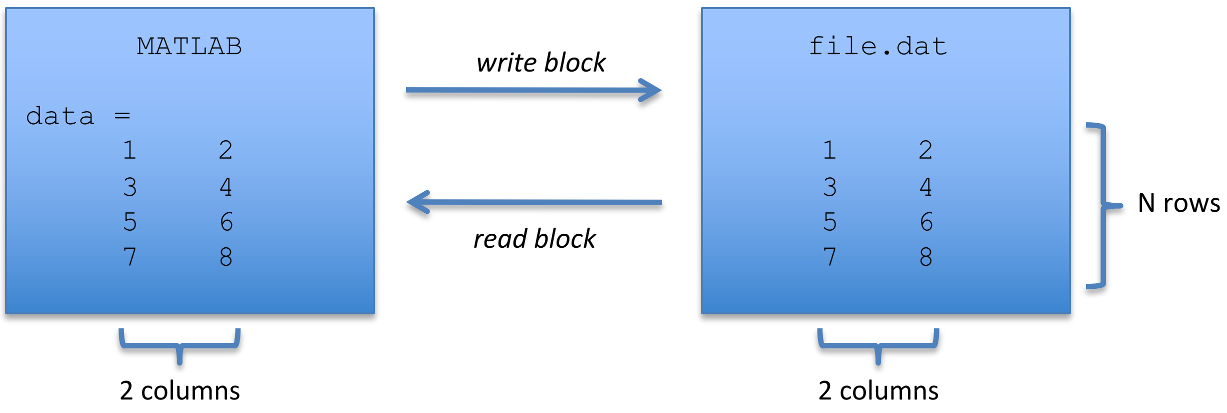
read block

```
>> fid  = fopen('file.dat','r');         open the file 'file.dat' for reading ('r')
>> data = fscanf(fid, '%f %f');          read the first two columns into data(,)
>> N    = fix(length(data)/2);           calculate the number of lines read
>> data = reshape(data,2,N)';            reshape data into a Nx2 matrix
>> fclose(fid);                          close the file
```

• **Notes**:
- `fscanf()` reads the data into a 1D vector
- but in above example we are assuming data() to be a Nx2 matrix, i.e. N rows and 2 columns
- make sure you understand all the transposition operations (i.e. " ' ")
- the format descriptor '%f' denotes a floating-point variable
- the `fprintf()` format argument requires to the trailing "\n" in order to start a new line
- the `fscanf()` format argument does not require a trailing "\n" to start a new line
- use `help fprintf` and `help fscanf` to learn more about the format descriptors

- when omitting `fid` from `fprintf()` the result will be printed to the command window

- above commands (i.e. the *write* and *read* block) map `data(,)` onto a file as follows:



Data Analysis I/O

▪ **reading and writing data sets**

• the data to be analysed is mostly not defined in your script as a vector (or matrix) but instead stored in a file on the hard drive and hence needs to be loaded into memory

➢ **exercise**:

• download the data file `Galaxies.dat` from the class web site

• write a script that reads the file and then writes the data back to a file '`file.dat`' in ASCII format (either using `fprintf()`/`fscanf()` or `save()`/`load()`)

• compare the original `Galaxies.dat` and your `file.dat`

• ***Notes***:

– the file contains 4 columns instead of 2 as in the example on the previous page

– both files should contain the same values in the same format

▪ **`Galaxies.dat`**:

• the file contains information about galaxies in the Universe as derived from a simulation of cosmic structure formation on a supercomputer, i.e. this is not the real distribution of galaxies but a model Universe

• the columns in the file have the following meaning

– 1st column	X position
– 2nd column	Y position
– 3rd column	Z position
– 4th column	Galaxy Mass

• the positions are given in the astronomical length unit 'Mpc' and the mass in solar masses, but do not concern yourself about the units for the exercises...

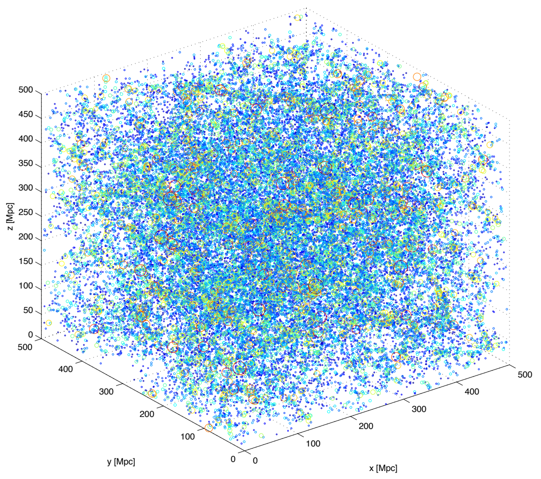
• if you cannot download the file using the link above, here is the full URL:
http://popia.ft.uam.es/Computacion/files/exercises/Galaxies.dat
right-click on it and select 'Download file as...'

Examples

application – Galaxies in the Universe

➢ **exercise**:

• make a 3D visualisation of the galaxies found in `Galaxies.dat`, e.g. produce a plot like the following:



• ***Note***:

• use `scatter3()` to plot the positions (x,y,z)

• make the size of the (filled) circles proportional to the mass of the galaxy

• make the colour of the circle proportional to the mass, too

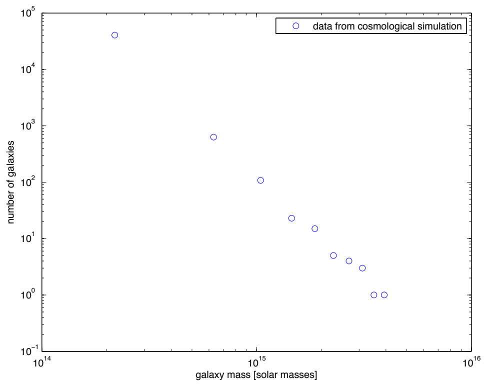
➢ **exercise**:

• calculate the distribution of galaxy mass, i.e. the mass function of galaxies

• ***Note***:

• you need to bin the data on a logarithmic scale!

• the plot should look like this:



Examples *application – Planck Curve*

▪ the "Planck curve" describes the energy emitted at a certain wavelength of a black body at a certain temperature and has the form

$$f(x) = C\, x^3 \frac{1}{e^x - 1}$$

▪ we aim at determining the amplitude of the best-fit Planck curve to the following experimentally determined data points:

```
x=[1.1   1.2   2.4   3.1   4.1   5.5    6.1   7.0    8.4   8.7];
f=[1.65  1.86  3.45  3.55  2.60  1.73   1.19  0.88   0.31  0.27];
```

➢ **exercise:**

• plot the data points using as a symbol a circle of size 10

➢ **exercise:**

• interpolate `f()` to the mid-points of `x()` using 3rd order polynomial interpolation using `p4interpol.m` from the exercises in class.

➢ **exercise:**

• plot the interpolated data points into the same figure as the actual data itself using a different symbol and colour.

➢ **exercise:**

• generate a combined data set that contains the original and the interpolated data

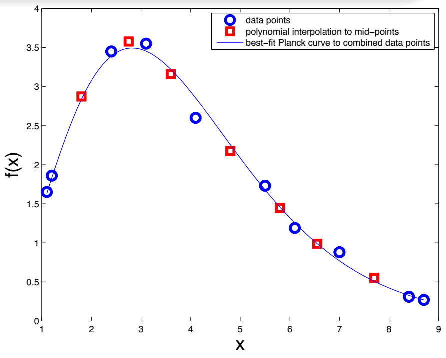
```
xcomb = ?, fcomb = ?
```

➢ **exercise:**

• fit a Planck curve to the (combined) data determining the (unknown) amplitude C using the least-square method as explained in class.

• plot the best-fit curve into the same figure as above

eventually you should end up with a figure like this →



hints:
• store the mid-point data in two new vectors `fmid()` and `xmid()`
• it may help to use an anonymous function for $g(x)=x^3/(e^x-1)$ when generating the matrix X_{ij} used to determine the best-fit parameter C
• there is no need to sort the combined data for the fitting
• do not forget axis labels, the legend, and comments in your script

Examples

application – free-fall in air

- the force acting on a free-falling body in air reads

$$F = mg - cv$$

where m=0.325 kg is the body's mass, g=9.81 m/s^2 the Earth's acceleration, v the body's velocity, and c the air's friction coefficient. Approximating v=gt we obtain

$$\frac{d^2x}{dt^2} = \frac{F}{m} = g - \frac{cg}{m}t \qquad \Rightarrow \qquad x(t) \approx \frac{g}{2}t^2 - \frac{cg}{6m}t^3$$

- we perform an experiment of letting a stone fall from a 20m high tower and obtain this data

```
x=[0.0   -3.2   -6.4   -9.6   -12.8  -16.0];
t=[0.0    0.83   1.16   1.45    1.65  1.88];
```

➢ **exercise:**

• plot the data points using as a symbol a circle of size 10

➢ **exercise:**

• find the best-fit analytical function $x(t)$ using all data points

➢ **exercise:**

• determine the Earth's acceleration g and the friction constant c from you fit values

➢ **exercise:**

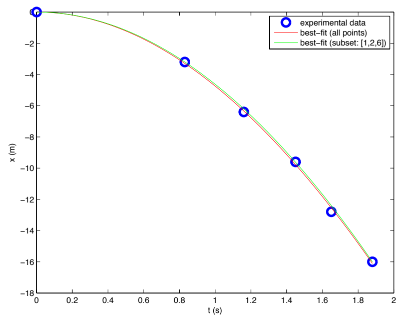
• as the system is over-determined (i.e. there are more equations than fit-parameters) any subset N>1 drawn form the data `x[]` and `t[]` serves equally well to perform the fit.

• add a best-fit curve (and g & c determination) to your script for any subset defined inside the script by the user in an index array, for instance, `isubset = [1, 2, 6]`

➢ **exercise:**

• how long does it take the stone to reach the ground?

eventually you should end up with a figure like this →



Examples

application – jumper

- A particle on a 2D surface can only move in one dimension and within the range $-x_l<x<+x_l$. And each of the jumps to either the right or left are covering a distance of 0.36nm where the probability to move right is the same as moving to the left and is given by the following formula

$$p_r = p_l = \frac{1}{2} e^{-\Delta E/kT}$$

where ΔE is the energy barrier, k the Boltzmann constant, and T the temperature.

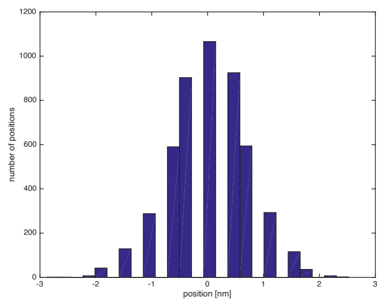
The probability for the particle to not jump is naturally given by $p_{nj}=1-p_r-p_l$.

➢ **exercise:**

- calculate the final positions x_f for 5000 particles where each particle had 20 attempts to jump, all starting at position $x_i=0$. Use $x_l=2.88\text{nm}$, $\Delta E=6.88\text{e-}21\text{J}$, $k=1.38\text{e-}23$ J/K, $T=307\text{K}$.

➢ **exercise:**

- plot a histogram of the final positions:



➢ **exercise:**

- Calculate the mean and standard deviation of the final distances jumped by each particle.

Data Analysis *summary*

- **statistical description of data sets:**
  - mean, standard deviation, mode, median, error bars
  - (probability) distribution function
  - linear least-square fitting of data
  - linear interpolation of data
  - polynomial interpolation of data (limited)

- **reading/writing of data**