Parameter Estimation - Computer Laboratory 1

Table of Contents

MATLAB basics

MATLAB is the abbreviation for MATrix LABoratory. It is a numerical computing environment, that offers many opportunities for data analysis, visualization, design, etc. It has its own programming language, which can be used to write scripts, functions and different applications.

The MATLAB language is

- **matrix-based**: variables are multidimensional *arrays*, no matter what type of data. MATLAB operates primarily on whole matrices and arrays, instead of one number.
- **weakly typed**: the variables have their own types, but they are not strict. Expressions between various different types are allowed.

Commands

The commands can be entered and executed directly in the Command Window:

- Create variables:

```
a=1;
b=2
```

```
b = 2
```

The variables a and b appeared in the workspace. Omitting the ; will immediately display the result in the Command Window.

- Perfrom some operations

```
c=a+b
```

```
c = 3
```

```
d=cos(c)
```

```
d = -0.9900
```

```
d^2
```

```
ans = 0.9801
```

```
sqrt(d)
```

```
ans = 0.0000 + 0.9950i
```

When you do not specify an output variable, MATLAB uses the variable `ans`, short for *answer*, to store the results of your calculation. `ans` stores only the current result, it will be overwritten by the next result.

Matrices and arrays

In MATLAB all variables are arrays, and the operations are matrix operations. A matrix is a two dimensional array and a vector is a matrix with only one row or column.

Creating vectors/matrices/arrays

- Create a row vector

```
x=[1 2 3 4]
```

```
x = 1×4
     1     2     3     4
```

- Create a column vector. The end of a row is denoted by;

```
y=[5;6;7;8]
```

```
y = 4×1
     5
     6
     7
     8
```

- Create a 2x3 matrix

```
A=[1 2 3; 4 5 6]
```

```
A = 2×3
     1     2     3
```

```
     4     5     6
```

- Cell arrays: the elements of the array can be of different types, e.g. vector, string, matrix. The elements of cell arrays can be listed between {}

```
C={'text' [0 1]; [] A}
```

```
C = 2×2 cell
```

	1	2
1	'text'	[0,1]
2	[ ]	[1,2,3;4...

- Indexing: the elements of a vector/matrix can be a accessed by specifying the row/column indices between ( ). The cell array indices can be given between { }. **Important:** in MATLAB the indexing starts from 1! The last element can be accessed by the `end` index. The full row/column can be selected by :

```
x(2)    %2nd element of x
```

```
ans = 2
```

```
x(end)  %last element of x
```

```
ans = 4
```

```
y(3)    %3rd element of y
```

```
ans = 7
```

```
A(1,2)  %2nd element in the 1st row of A
```

```
ans = 2
```

```
A(:,1)  %first column of A
```

```
ans = 2×1
     1
     4
```

```
C{2,2}  %2nd element in the 2nd row of C cell array
```

```
ans = 2×3
     1     2     3
     4     5     6
```

- Creating vectors with linear increment

```
v1=0:100  %vector of elements from 0 to 100 with increment 1 between the elements
```

```
v1 = 1×101
     0     1     2     3     4     5     6     7     8     9    10    11    12 ···
```

```
v2=0:5:100  %vector of elements from 0 to 100 with increment 5 between the elements
```

```
v2 = 1×21
     0     5    10    15    20    25    30    35    40    45    50    55    60 ···
```

- Creating linearly spaced vectors

```
v3=linspace(0,100) %linearly spaced vector between 0 and 100 with 100 points
```

```
v3 = 1×100
         0    1.0101    2.0202    3.0303    4.0404    5.0505    6.0606    7.0707 ···
```

```
v4=linspace(0,100,20) %linearly spaced vector between 0 and 100 with 20 points
```

```
v4 = 1×20
         0    5.2632   10.5263   15.7895   21.0526   26.3158   31.5789   36.8421 ···
```

- Creating matrices with functions (`ones`, `zeros`, `eye`, `rand`).

```
O1=ones(3)
```

```
O1 = 3×3
     1     1     1
     1     1     1
     1     1     1
```

```
O2=ones(2,4)
```

```
O2 = 2×4
     1     1     1     1
     1     1     1     1
```

```
Z=zeros(2,3)
```

```
Z = 2×3
     0     0     0
     0     0     0
```

```
E=eye(3)
```

```
E = 3×3
     1     0     0
     0     1     0
     0     0     1
```

Operations

- Scalar addition

```
B=A+0.5
```

```
B = 2×3
    1.5000    2.5000    3.5000
    4.5000    5.5000    6.5000
```

- Scalar multiplication

```
C=2*B
```

```
C = 2×3
     3     5     7
     9    11    13
```

- Transpose

```
A'
```

```
ans = 3×2
     1     4
     2     5
     3     6
```

- Addition: elemet-wise operation - matrices have to be the same size

```
A+B
```

```
ans = 2×3
    2.5000    4.5000    6.5000
    8.5000   10.5000   12.5000
```

- Multiplication: matrix operation - inner matrix dimensions must agree

```
A'*B
```

```
ans = 3×3
   19.5000   24.5000   29.5000
   25.5000   32.5000   39.5000
   31.5000   40.5000   49.5000
```

- Power: for square matrices only!

```
D=[1 2; 3 4]
```

```
D = 2×2
     1     2
     3     4
```

```
D^2
```

```
ans = 2×2
     7    10
    15    22
```

- Element-wise operations: using the .*, ./, .^ you can perform elementwise multiplication, division and power on the matrices.

```
A.*B    %A(1,1) is multiplied by B(1,1), and so on
```

```
ans = 2×3
    1.5000    5.0000   10.5000
   18.0000   27.5000   39.0000
```

```
A./C    %A(1,1) is divided by C(1,1), and so on
```

```
ans = 2×3
    0.3333    0.4000    0.4286
    0.4444    0.4545    0.4615
```

```
B.^3    %B(1,1)^, B(1,2)^3, ...
```

```
ans = 2×3
    3.3750   15.6250   42.8750
   91.1250  166.3750  274.6250
```

- Inverse: for square, invertable matrices only!

```
inv(D)
```

```
ans = 2×2
   -2.0000    1.0000
    1.5000   -0.5000
```

- size of a vector/matrix: `size(v)` returns the number of rows and columns.

```
size(D)
```

```
ans = 1×2
     2     2
```

```
size(v1)
```

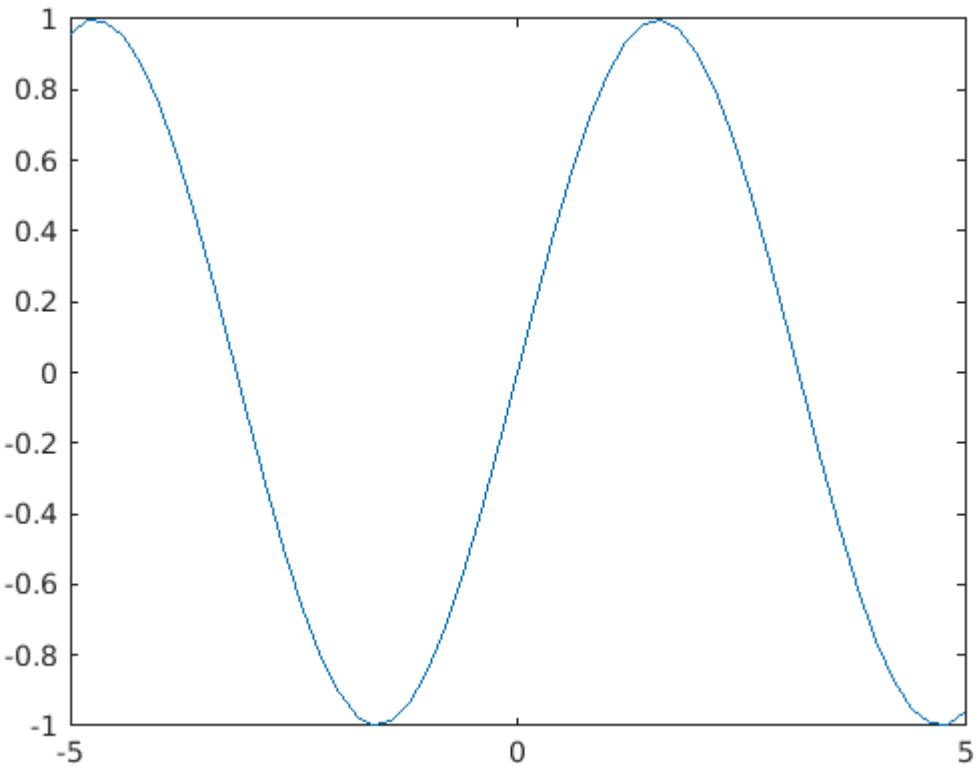
```
ans = 1×2
     1   101
```

Data visualization

Different kinds of plots can be used to display the data, e.g. line scatter, surface and contour plots. You display the selected variable by choosing a plot type from the PLOTS menubar, or entering plot functions in the Command Window.

- 2-D line plot: the `plot` function is the most often used data visualization function. It can display vector variables in a 2D coordinate system.

```
x1=-5:0.2:5;
y1=sin(x1);
plot(x1,y1)
```



- Line style: dashed: -- dotted: :, dash-dotted: -.
- Marker symbols: * (asterix), o (circle), x (cross), + (plus sign), ^ (triangle), . (dot), s (square), d (diamond) etc.
- Multiple vectors with the same data length can be plotted together: `plot(x1,y1,x2,y2,...)`
- Vectors with different lengths can be plotted on the same figure, using the `hold on` command

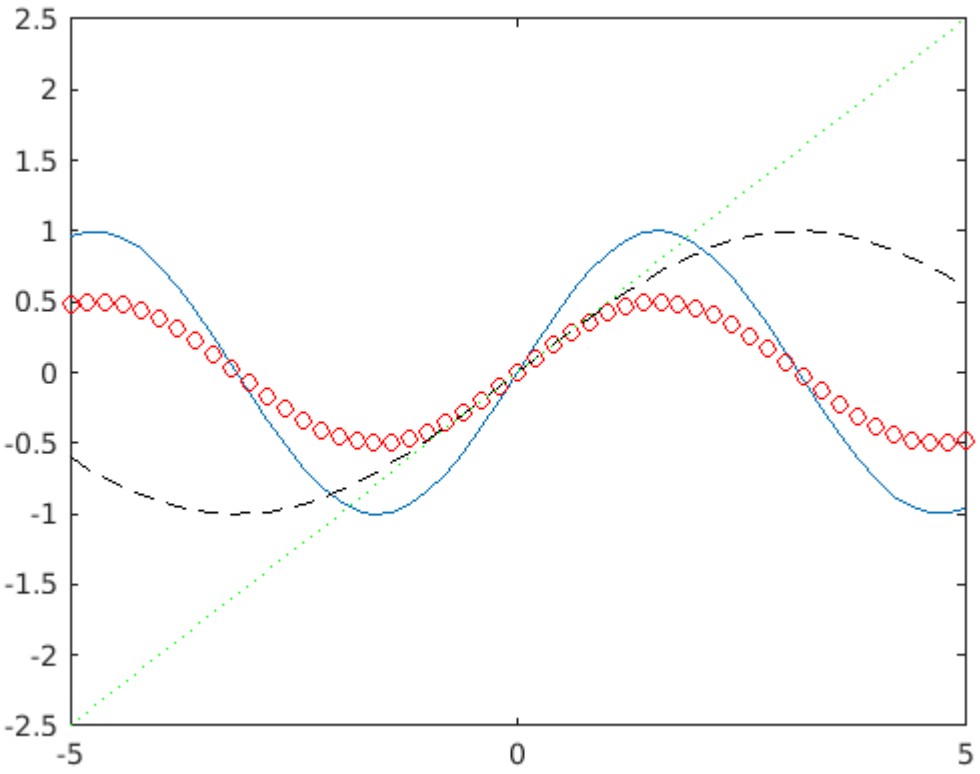
```
y2=0.5*sin(x1);
y3=sin(0.5*x1);
x2=-5:5
```

```
x2 = 1×11
    -5    -4    -3    -2    -1     0     1     2     3     4     5
```

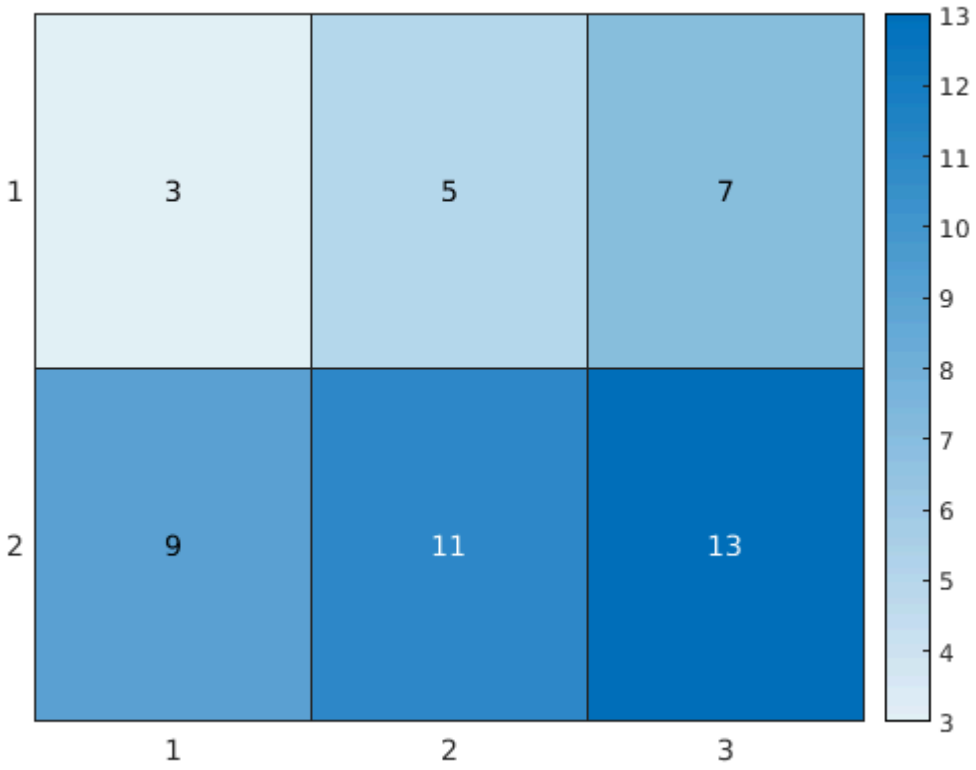
```
plot(x1,y1,x1,y2,'ro',x1,y3,'k--')
hold on
plot(x2,0.5*x2,'g:')
```

- Matrix heatmap: the entries of the matrix are displayed with different colors. Greater elements have stronger colors.

```
hold off
```




```
heatmap(C)
```



Individual tasks

1. Create a row vector v with elements from 1 to 5 with 0.5 increment!
2. Create a matrix `M` whose first row is v and the second row is the square of elements in v !
3. Multiply v with `M'`! Save the result in a new variable `M2`!
4. Is the matrix `M` invertable? Why?
5. Multiply `M` with `M'`! Save the result in a new variable `M3`!
6. Is `M3` invertable? If yes, compute the inverse!
7. Plot the second row of `M` in the function of the first row, using red x markers!

Random variables

True random numbers are generated by natural processes or physical phenomenon, for example atmospheric noise, termal noise, rolling a dice. The outcomes of these processes are totally random, they cannot be predicted and cannot be affected. The implementation of true random number "generators" is very expensive, therefore we use so-called pseudorandom generators in most of the cases.

The pseudorandom generators use different kinds of algorithms to create numbers, that seem to be random. However, unlike the true random numbers, the sequence of pseudorandom numbers can be reproduced, knowing the initialization of the generator (seed).

In MATLAB different pseudorandom number generators are available, that generate numbers with specified probability distributions. The sequence generated by these functions can be used as a sample from the underlying random variable with specified distribution. The generated numbers also compose a white noise process.

Uniformly distributed random numbers

The `rand` function generates uniformly distributed random numbers from the interval (0,1). Syntax: `rand(dim1,dim2,...,dimn)`

```
N=270   %number of samples
```

```
N = 270
```

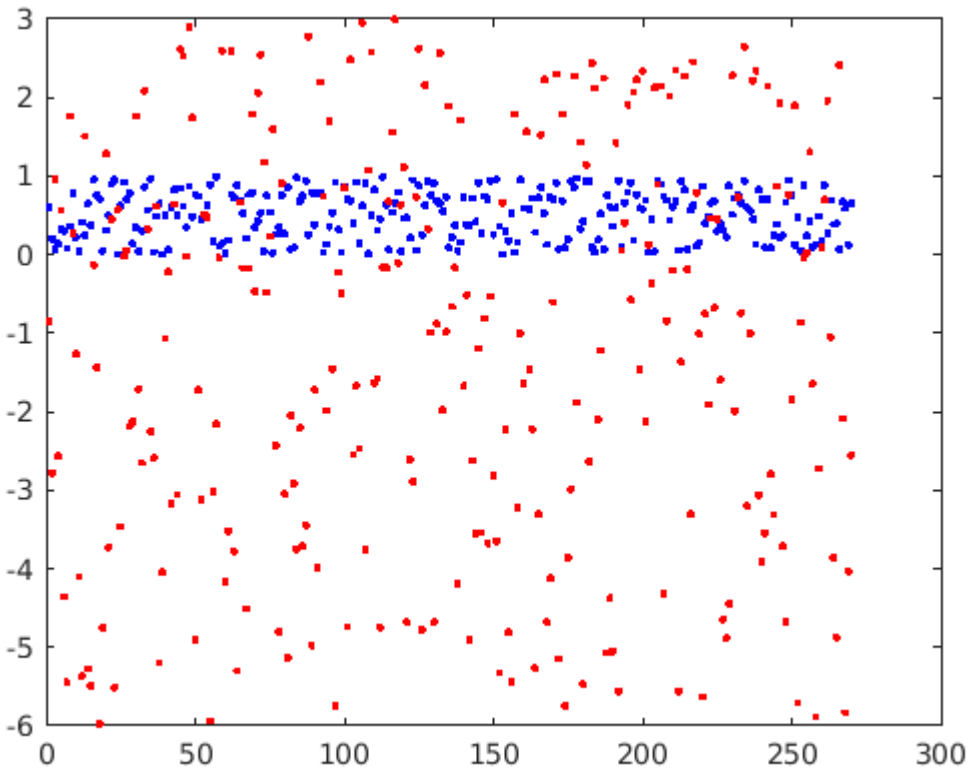
```
a=-6    %left side of the interval
```

```
a = -6
```

```
b=3    %right side of the interval
```

```
b = 3
```

```
r1=rand(1,N);
r2=a+(b-a)*rand(1,N);
clf
plot(r1,'b.')
hold on
plot(r2,'r.')
hold off
```



Normally distributed random numbers

The `randn` function generates random numbers from the standard normal distribution (mean=0, variance=1).

Syntax: `randn(dim1,dim2,...,dimN)`

```
N=1000%number of samples
```

```
N = 1000
```

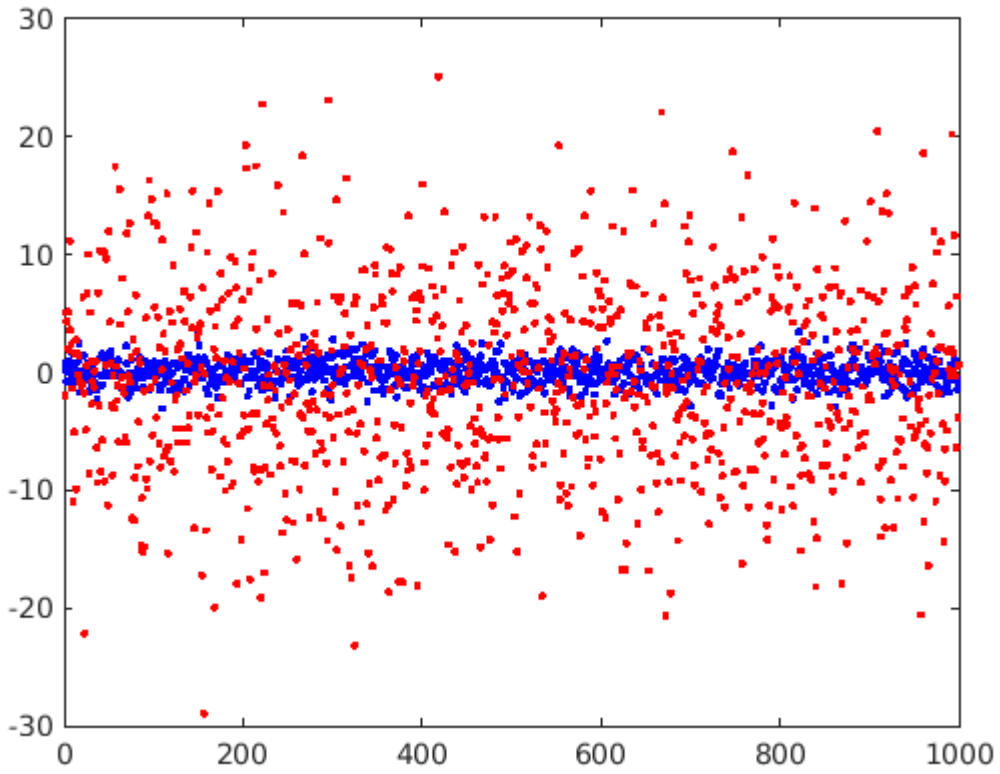
```
m=-1%mean
```

```
m = -1
```

```
s=8%standard deviation
```

```
s = 8
```

```
rn1=randn(1,N);
rn2=m+s*randn(1,N);
clf
plot(rn1,'b.')
hold on
plot(rn2,'r.')
hold off
```



To generate random numbers from a normal distribution with given mean `m` and standard devition `s`, you need to transform the `randn` function.

Normally distributed random numbers (Statistics and Machine Learning Toolbox)

The Statistics and Machine Learning Toolbox offers additional opportunities to create random variables.

The `normrnd` function generates random numbers from the normal distribution with given mean and standard deviation.

- `normrnd(mu,sigma)` generates a single random number from the normal distribution with mean `mu` and standard deviation `sigma`.
- `normrnd(mu,sigma,dim1,...,dimN)` generates an array of random numbers with dimensions `dim1,...,dimN`

```
mu=3.5
```

```
mu = 3.5000
```

```
sigma=2.5
```

```
sigma = 2.5000
```

```
dim1=2
```

```
dim1 = 2
```

```
dim2=5
```

```
dim2 = 5
```

```
normrnd(mu,sigma,dim1,dim2)
```

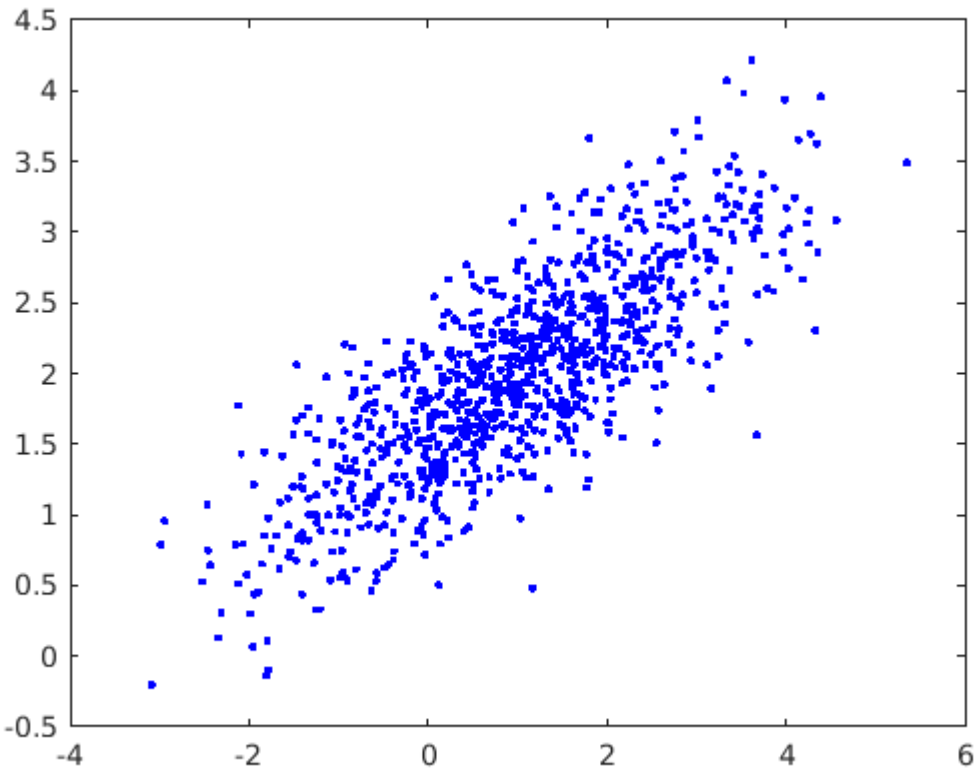
```
ans = 2×5
    5.6906   -0.5083    1.7457    4.0877    3.1103
    5.5199   -2.4051    7.6298    3.1206    6.0954
```

Multivariate normal distribution (Statistics and Machine Learning Toolbox)

The `mvnrnd` function can be used to generate random numbers from multivariate normal distributions. To define the multivariate normal distribution, you need to specify the mean vector `mu` and the covariance matrix `sigma`. If `mu` is a `mxd` matrix, then each row of `mu` refers to one mean vector of a single multivariate normal distribution.

- `mvnrnd(mu,sigma)` returns one random vector from the multivariate normal distribution.
- `mvnrnd(mu,sigma,n)` returns `n` random vectors from the multivariate normal distribution.

```
mu=[1 2];
sigma=[2 0.8; 0.8 0.5];
N=1000;
R=mvnrnd(mu,sigma,N);
plot(R(:,1),R(:,2),'b.');
```



Control random number generation

The generated random numbers depend on the initial state of the generator. The generator resets its state when MATLAB is restarted, that is why you get the same numbers after the start of MATLAB. The random number generation can be controlled by specifying the type of the generator and the seed, and the state of the generator can be saved and restored later to reproduce the experiment.

- `rng(seed)` sets the initial state of the generator, the seed can be 0, a positive integer, `'default'`, `'shuffle'` (current time)
- `rng('generator')` sets the type of the random number algorithm, e.g `'twister','philox'`
- `S=rng` saves the state of the generator
- `rng(S)` restore the state of the generator

```
rng(2); %seed=2
S=rng
```

```
S = struct with fields:
     Type: 'twister'
     Seed: 2
    State: [625×1 uint32]
```

```
rv1=rand(1,10)
```

```
rv1 = 1×10
    0.4360    0.0259    0.5497    0.4353    0.4204    0.3303    0.2046    0.6193 ···
```

```
rng('shuffle','philox')
rv2=rand(1,10)
```

```
rv2 = 1×10
    0.1127    0.2588    0.0173    0.7158    0.5740    0.5595    0.3695    0.6679 ···
```

```
rng(S)
rv3=rand(1,10)
```

```
rv3 = 1×10
    0.4360    0.0259    0.5497    0.4353    0.4204    0.3303    0.2046    0.6193 ···
```

Individual tasks

1. Create 1000x1 vector (u1) of random numbers with uniform distribution on the interval (-1,5)!
2. Create a 1000x1 (u2) vector of random numbers from a normal distribution with mean 6 and variance 4!
3. Plot the first two vectors on the same figure! Use dot markers!
4. Let us assume that we have two random variables: $\xi_1 = N(-2,3)$ and $\xi_2 = N(3,5)$. Create 1000 random vectors (u3) from the vector valued normal distribution, that is composed of ξ_1 and ξ_2: $\xi = [\xi_1,\xi_2]^T$, assuming that ξ_1 and ξ_2 are independent.
5. Consider another vector valued random variable $\eta = [\xi_1, \xi_2]^T$, but now the covariance between ξ_1 and ξ_2 is 3.2. Generate 1000 random vectors (u4) from η.
6. Plot u3 and u4 in the same 2D coordinate system!

Basic statistics

A sample is a collection of n independent random variables. The sample corresponds to a set of measurements about the random variable. Statistics is a deterministic function of the sample elements. It can be used to construct an estimate. The measured data set is a realization of the sample. In MATLAB the following basic statistics functions are available.

Mean value

$$m = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$$

The `mean` function computes the mean value of the elements in a vector or a matrix. In case of matrices the mean value is computed along the columns by default.

- `mean(x)` returns the mean of the vector x
- `mean(A)` returns the mean of the matrix `A` along the columns
- `mean(A,dim)` returns the mean of the matrix `A` along the specified dimension (1 - columns, 2 - rows)

```
mean(rn2) %mean of the rn2 vector (normal distribution with mean m)
```

```
ans = -0.5919
```

```
mean(R) %mean of the R matrix (samples from multivariate normal distribution, 2 columns
```

```
ans = 1×2
    1.0275    2.0029
```

```
mean(A,2) %mean of elements in the rows of A
```

```
ans = 2×1
     2
     5
```

Standard deviation

$$\sigma = \sqrt{\frac{1}{n-1}\left((x_1 - m)^2 + (x_2 - m)^2 + \ldots + (x_n - m)^2\right)}$$

The `std` function can be used to compute the standard deviation of elements in a vector or matrix. In case of matrices the standard deviation is computed along the columns by default.

- `std(x)` returns thestandard deviation of the vector x
- `std(A)` returns the standard deviation of the matrix `A` along the columns
- `std(A,w,dim)` returns the weighted standard deviation of the matrix `A` along the specified dimension (w: 0 - normalized by N-1, 1 normalized by N, dim: 1 - columns, 2 - rows)

```
std(rn2)       %std of the rn2 vector (normal distribution with mean m)
```

```
ans = 7.8073
```

```
std(R)          %std of the R matrix (samples from multivariate normal distribution, 2 c
```

```
ans = 1×2
    1.4091    0.7059
```

```
std(A,0,2)      %std of elements in the rows of A
```

```
ans = 2×1
     1
     1
```

Variance

$$\sigma^2 = \frac{1}{n-1}\left((x_1 - m)^2 + (x_2 - m)^2 + \ldots + (x_n - m)^2\right)$$

The `var` function can be used to compute the variance of elements in a vector or matrix. In case of matrices the variance is computed along the columns by default.

- `var(x)` returns the variance of the vector `x`
- `var(A)` returns the variance of the matrix `A` along the columns
- `var(A,w,dim)` returns the weighted variance of the matrix `A` along the specified dimension (w: 0 - normalized by N-1, 1 normalized by N, dim: 1 - columns, 2 - rows)

```
var(rn2)        %variance of the rn2 vector (normal distribution with mean m)
```

```
ans = 60.9542
```

```
var(R)          %variance of the R matrix (samples from multivariate normal distributior
```

```
ans = 1×2
    1.9855    0.4983
```

```
var(A,0,2)      %variance of elements in the rows of A
```

```
ans = 2×1
     1
     1
```

Covariance

$$r_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}((x_{k,i} - m_i)(x_{k,j} - m_j)) \text{ with } m_i = \frac{1}{n}\sum_{k=1}^{n} x_{k,i}$$

The covariance matrix is a symmetric square matrix, whose diagonal elements are the variances of the corresponding samples.

$$\mathrm{COV} = \begin{bmatrix} \sigma_x^2 & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,x) & \sigma_y^2 & \mathrm{cov}(y,z) \\ \mathrm{cov}(z,x) & \mathrm{cov}(z,y) & \sigma_z^2 \end{bmatrix}$$

The `cov` function can be used to compute the covariance of two random variables.

- If the input is a vector, then `cov(x)=var(x).`
- If the input is two vector, then `cov(x,y)` returns the covariance matrix of `x` and `y.`
- If the input is a matrix, then its columns are corresponding to the separate measurements.

```
cov(rv1,rv2)
```

```
ans = 2×2
    0.0297    0.0032
    0.0032    0.0619
```

```
cov(R)
```

```
ans = 2×2
    1.9855    0.7943
    0.7943    0.4983
```

Autocovariance

$$r_{xx}(s) = \frac{1}{n-s}\sum_{i=1}^{n-s}((x_i - m)(x_{i+s} - m))$$

The `xcov` function computes the cross-covariance between two vectors. If the input is only one vector, then it computes the autocovariance.

- `c=xcov(x)` returns the autocovariance sequence of `x`.
- `[c,lags]=xcov(x)` returns the autocovaraince sequence and the time lags, where the covariances were computed.
- `xcov(_,maxlags)` specifies the maximum lags
- `xcov(_,scaleopt)` specifies the normalization options. By default the computed autocovariances are not normalized (i.e. the sum is not divided by `n-s`). The following options can be specified:
- `'none':` raw, unscaled cross-covariance.
- `'biased'` — Biased estimate of the cross-covariance, the value is normalized by 1/n.
- **`'unbiased'` — Unbiased estimate of the cross-covariance, the value is normalized by 1/(n-s)**
- `'normalized'` or `'coeff'` — Normalizes the sequence so that the autocovariances at zero lag equal 1.

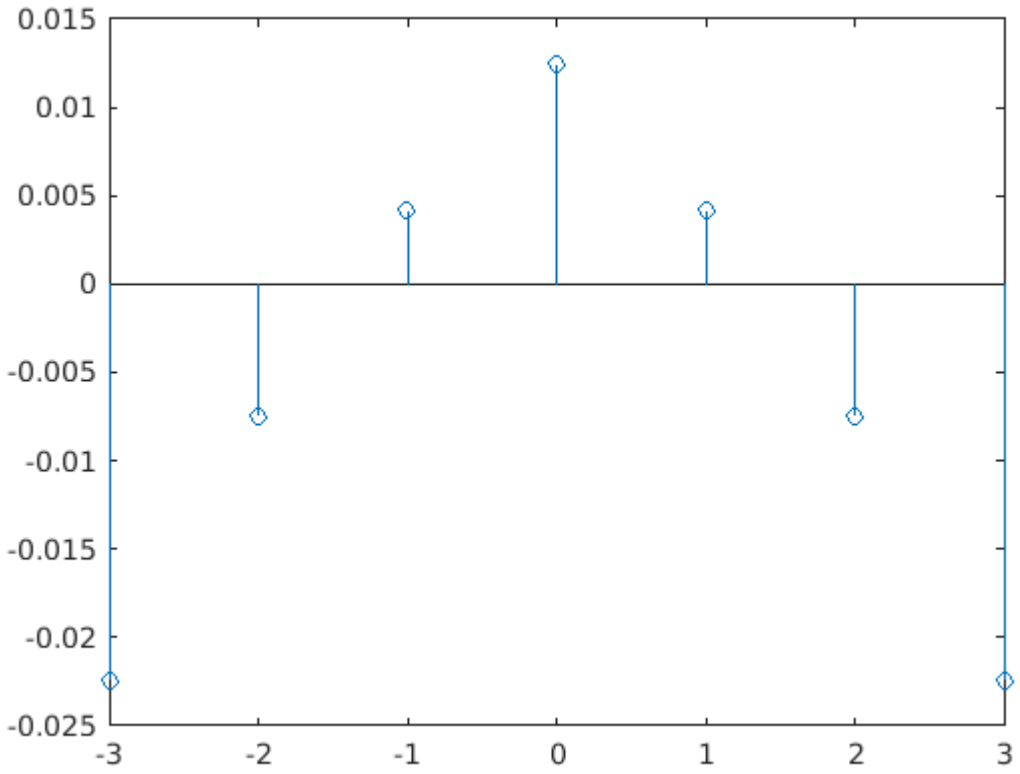
```
x=[0.1 0.2 0.3 0.4]
```

```
x = 1×4
    0.1000    0.2000    0.3000    0.4000
```

```
[c,lags]=xcov(x,'unbiased')
```

```
c = 1×7
   -0.0225   -0.0075    0.0042    0.0125    0.0042   -0.0075   -0.0225
lags = 1×7
    -3    -2    -1     0     1     2     3
```

```
stem(lags,c);
```




```
xcov(x,1)
```

```
ans = 1×3
    0.0125    0.0500    0.0125
```

Correlation

$$\rho(x_i, x_j) = \frac{cov(x_i, x_j)}{\sigma_i \sigma_j}$$

estimate of the correlation coefficient: $\widehat{\rho}_{ij} = \frac{\sum_{k=1}^{n}(x_{i,k} - m_i)(x_{j,k} - m_j)}{(n-1)\sigma_i \sigma_j} = \frac{\sum_{k=1}^{n}(x_{i,k} - m_i)(x_{j,k} - m_j)}{\sqrt{\sum_{k=1}^{n}(x_{i,k} - m_i)^2 \sum_{k=1}^{n}(x_{j,k} - m_j)^2}}$

The correlation between two random variables is the normalized covariance. It informs us about the relationship between the two random variables. If the random variables are independent, then their correlation is 0, however 0 correlation implies independence only in case of Gaussian joint distribution! Non-zero correlation implies that the two random variables are dependent, but it does not mean causation. Spurious correlations

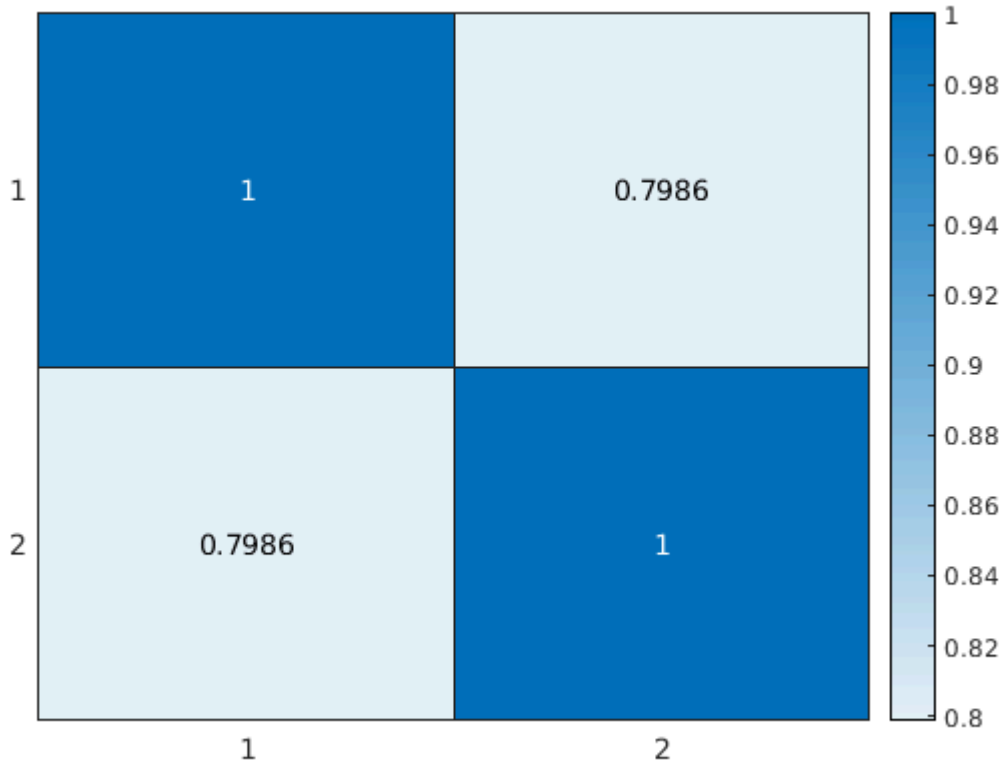
In MATLAB the `corrcoef` function returns the correlation matrix between two (or more) vector of observations of random variables. The correlation matrix is always a symmetrical matrix with 1 in the diagonal. Example:

$$R = \begin{bmatrix} 1 & \rho(x,y) & \rho(x,z) \\ \rho(y,x) & 1 & \rho(y,z) \\ \rho(z,x) & \rho(z,y) & 1 \end{bmatrix}$$

```
corr=corrcoef(R)
```

```
corr = 2×2
    1.0000    0.7986
    0.7986    1.0000
```

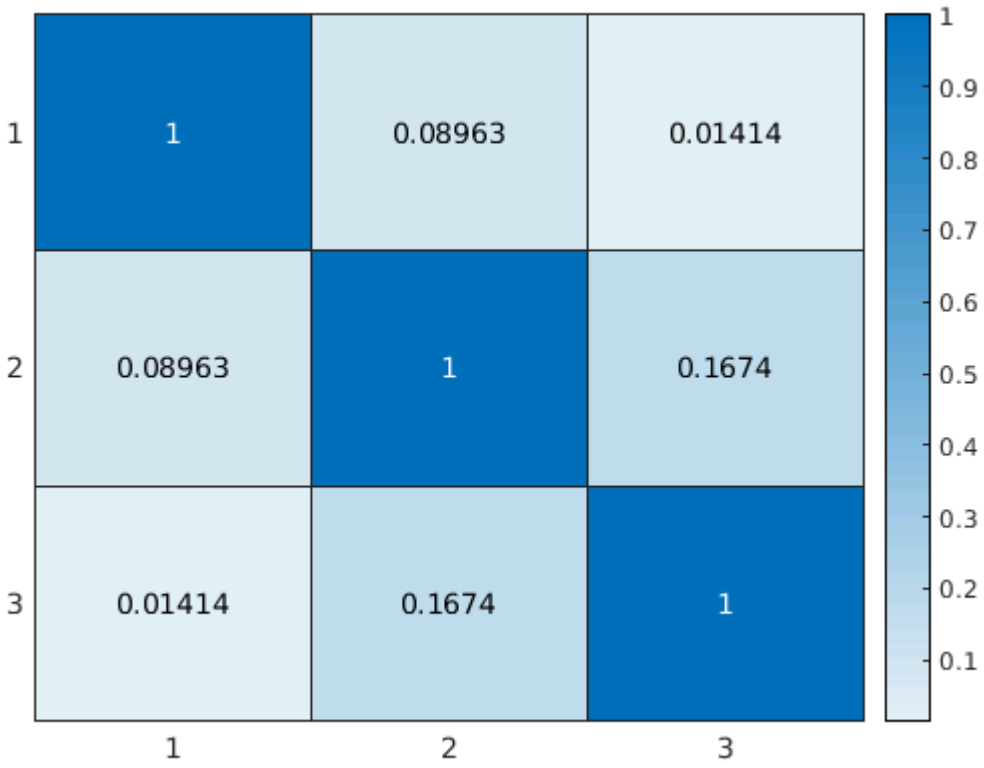
```
heatmap(corr)
```




```
R2=[randn(100,1),2*randn(100,1),3*randn(100,1)];
corrcoef(R2)
```

```
ans = 3×3
    1.0000    0.0896    0.0141
    0.0896    1.0000    0.1674
    0.0141    0.1674    1.0000
```

```
heatmap(corrcoef(R2))
```

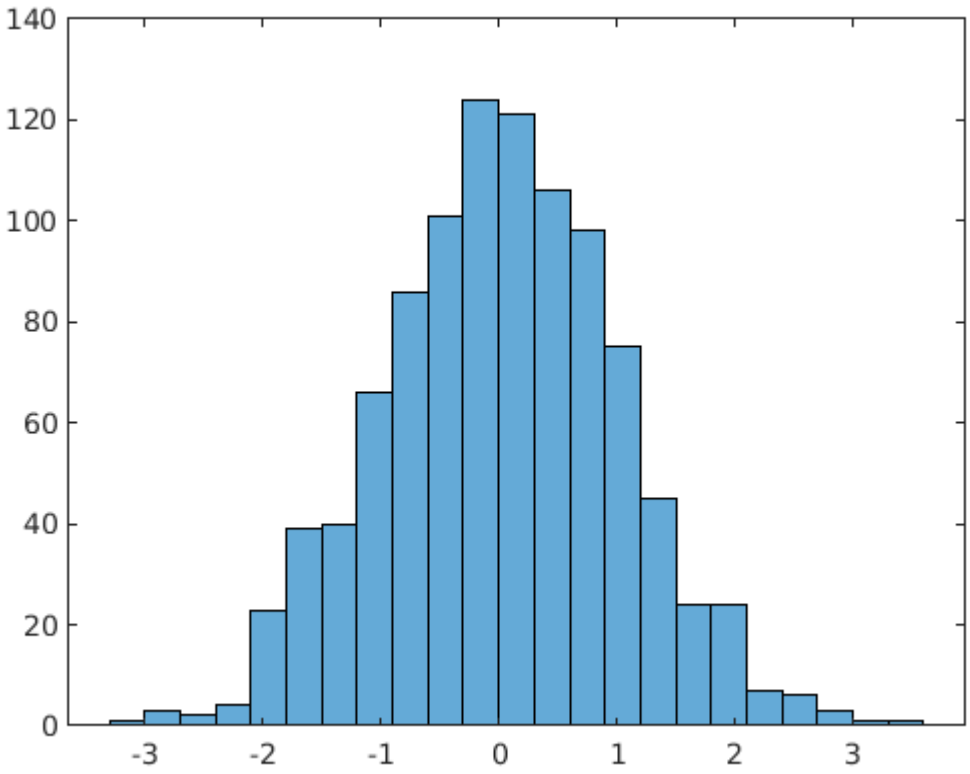


Histogram

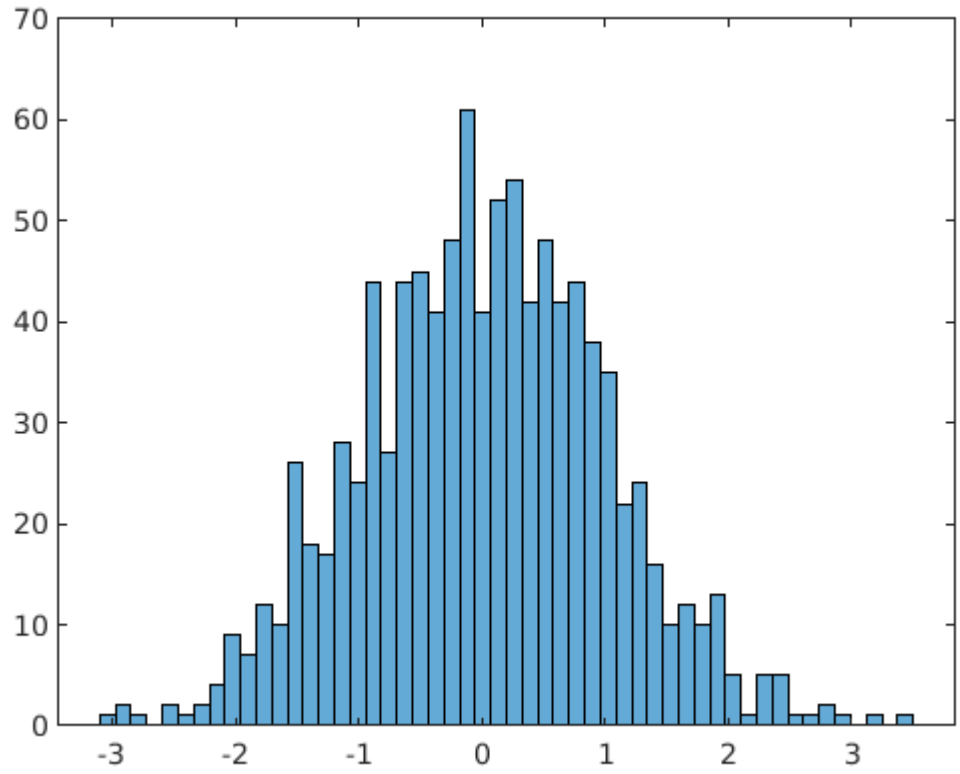
The histogram is approximate representation of the probability density function of a (scalar valued) random variable. In MATLAB the `histogram` function can be used to display the histogram plot.

- `histogram(X)` simply creates a histogram plot with automatically computed number of bins.
- `histogram(X,nbins)` creates a histogram plot with `nbins` number of bins.

```
bins=52;
histogram(rn1);
```

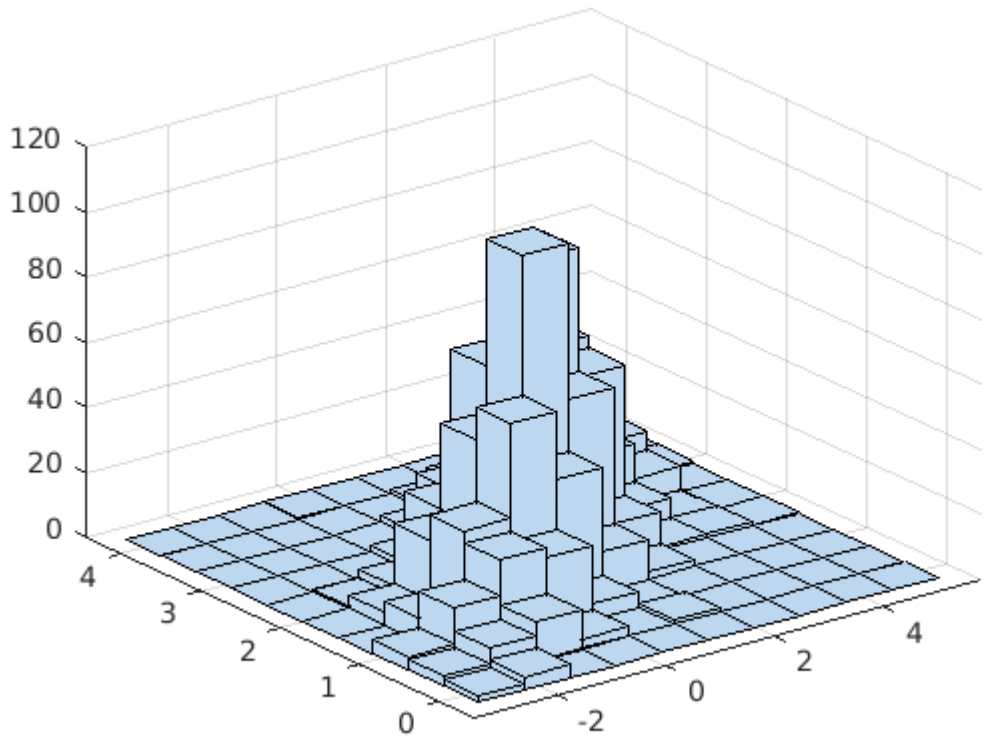



```
histogram(rn1,bins);
```



- `hist3(X)` creates a 3D histogram of the bivariate data in `X` using a 10x10 grid of containers.
- `hist3(X,NBINS)` creates a 3D histogram plot using the custom grid of containers.

```
hist3(R)
```



Individual tasks

1. Compute the mean value, standard deviation and variance of u1 and u2!
2. Compute the covariance and correlation between u1 and u2!
3. Create (separate) histogram plots of u1 and u2! Try different number of bins.
4. Compute the mean value of the vector valued random variable, whose observations can be found in u3!
5. Compute the covariance and the correlation between the $\xi_1$and ξ_2 if the measurement data can be found in u3! What can you say about the independence of $\xi_1$and ξ_2?
6. Compute the mean value of the vector valued random variable, whose observations can be found in u4!
7. Compute the covariance and the correlation between the $\xi_1$and ξ_2 if the measurement data can be found in u4! What can you say about the independence of $\xi_1$and ξ_2?
8. Create (separate) 3D histogram plots of u3 and u4!

HOMEWORK (Deadline: 2020 September 30, 10:30)

1. Create a 100x1 vector (x1) of random numbers with normal distribution with mean 1 and standard deviation 2.
2. Compute the mean (m1) and the a variance (v1) of x1 along the rows (normalize the variance with N-1)
3. Compute the unbiased autocovariance sequence (ac) of x1 from lags -2 to 2.

4. Create another vector (`x2`) with the same size as `x1`, with normal distribution with mean -2 and standard deviation 2.
5. Compute the covariance and the correlation matrices of `x1` and `x2`. Are they independent?

Send the created script file (NEPTUNKOD-HW1.m) to pozna.anna@virt.uni-pannon.hu!