MATLAB Tutorial: Part II.

Conchi Ausín Olivera
Econometrics
Ph.D. Program in Business Administration and Quantitative Methods.

April 4, 2005

Contents

1 Statistical analysis and simulation using MATLAB

The Statistics Toolbox for MATLAB provides a large number of function for analyzing data. However, in practical situations, we are often faced with the difficulty of programming. In these notes, we will consider different problems that will be solved both by programming and by using the Statistical Toolbox for MATLAB. This toolbox includes functions for Probability distributions, Parameter estimations, Descriptive Statistics, Multivariate statistics, Linear and nonlinear modeling, Statistical plotting, Statistical Process Control, Design of Experiments, Hypothesis Tests, Cluster Analysis. See `help stats`.

1.1 Descriptive statistics

1.1.1 Numerical description

We can define this simple function to compute the mean value of the elements of a vector.

```
function xbar = mean1(x)
% For a vector x, mean1(x) returns the mean of x.
n = length(x);
xbar = x(1);
for i = 2:n
    xbar = xbar + x(i);
end
xbar = xbar/n;
```

Now, you can type the following statement to see how this function works:

```
>> x = rand(1000,1);
>> m = mean1(x)
```

The mean of `x` given by `m` should be close to `0.5`.

Alternatively, we could have make use of the `sum` command to avoid including a `for` loop:

```
function xbar = mean2(x)
% For a vector x, mean1(x) returns the mean of x..
n = length(x);
xbar = sum(x);
xbar = xbar/n;
```

Observe that the function `mean2` (unlike `mean1`) can also be applied to matrices where each column is a sample of data from a different variable. For matrices, `mean2(x)` is a row vector containing the mean value of each column.

We can construct a function to compute simultaneously the sample mean, $\bar{x}$, standard deviation, s, skewness, SK, and kurtosis, K, for each column of a matrix where,

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \qquad s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}}, \qquad SK = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{ns^3}, \qquad K = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{ns^4}.$$

```
function [xbar,s,SK,K] = statistics(X)
```

```
% X is a matrix whose columns are sample
% of data from a certain variable.
% xbar = sample mean
% s = standard deviation
% SK = skewness
% K = kurtosis
[n,m] = size(X);
xbar = sum(X)/n;
X = X - ones(n,1)*xbar;
s = sqrt(sum(X.^2)/n);
SK = sum(X.^3)./(n*s.^3);
K = sum(X.^4)./(n*s.^4);
```

An example to make use of the function statistics can be the following:

```
>> X=randn(8);
>> [xbar,s,SK,K] = statistics(X)
```

As we will see in next section, the Statistics toolbox of MATLAB supplies specific functions to compute all these characteristics and many more. For example, you can test your previous results with the following commands:

```
>> mean(X)
>> std(X,1)
>> skewness(X)
>> kurtosis(X)
```

1.1.2 Descriptive statistics with the Statistics Toolbox of MATLAB

Some of the functions to compute the most frequent statistics are the following:

```
mean(x)                 % Mean value of the elements in x.
median(x)               % Median value of the elements in x.
std(x),var(x)           % Standard deviation and variance of x normalized by n-1.
std(x,1),var(x,1)       % Standard deviation and variance of x normalized by n.
range(x)                % Range of x.
iqr(x)                  % Interquartile range of x.
mad(x)                  % Mean absolute deviation of x.
max(x),min(x)           % Maximum and minimum element of x.
skewness(x),kurtosis(x) % Skewness and kurtosis of x.
moment(x,order)         % Central moment of x specified by order.
prctile(x,p)            % pth percentile of x (if p=50, returns the median of x)
```

Observe that if `x` is a matrix, then the result of these functions is a row vector containing the statistic for each column of `x`.

Other two interesting functions are `cov` and `corrcoef`. For vectors, the command `cov` returns the variance:

```
>> x=rand(100,1);cov(x)
```

For matrices, where each row is an observation, and each column a variable, returns the covariance matrix:

```
>> x=rand(100,5); cov(x)
```

For two vectors, `z` and `w`, of equal length, `cov(z,t)` returns a matrix with the variances of `z` and `w` in the diagonal and the covariance of `z` and `w` in the two off-diagonal entries.

```
>> z=rand(100,1);t=rand(100,1);cov(z,t)
```

Observe that `cov(z,t)=cov([z t])`. For two matrices, `cov(X,Y)=cov(X(:),Y(:))`. Finally, `cov(x)` or `cov(x,y)` normalizes by $(n-1)$ and `cov(x,1)` or `cov(x,y,1)` normalizes by n, where n is the number of observations.

The `corrcoef(X)` command calculates a matrix of correlation coefficients for an array `X`, in which each row is an observation and each column is a variable. Observe that `corrcoef(X,Y)`, where `X` and `Y` are column vectors, is the same as `corrcoef([X Y])`.

```
>> corrcoef(x)
```

The Statistics Toolbox and some built-in functions of MATLAB allows to plot a number of useful graphics in descriptive statistics.

```
hist(x)       % Histogram.
boxplot(x)    % Boxplots of a data matrix (one per column).
cdfplot(x)    % Plot of empirical cumulative distribution function.
normplot(x)   % Normal probability plot (one per column).
qqplot(x,y)   % Quantile-Quantile plot.
```

You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files.

For example, imagine we are interested in plotting a variant of the histogram where the counts are replaced by the normalized counts, that is, the relative histogram. By normalized count, we mean the count in a class divided by the total number of observation times the class width. For this normalization, the area (or integral) under the histogram is equal to one. Now, we can look for the file `hist.m` and modify it. This file is usually in the following path (or something similar):

```
c:\MATLAB6p5\toolbox\matlab\datafun
```

Open it and let's try to change it. Observe that the `hist` command produces a histogram bar plot if there is no output arguments, that is, we look for the sentences:

```
if nargout == 0
bar(x,nn,'hist');
...
```

The sentence `bar(x,nn,'hist')` draws the values of the vector `nn` (*frequency*) as a group of vertical bars whose midpoints are the values of `x`, see `help bar`. For example, we can change the previous sentences by the following ones to obtain a white normalized histogram:

```
if nargout == 0
bar(x,nn/(length(y)*(x(2)-x(1))),'hist','w');
...
```

You can also change the help section including for example a sentence like this:

```
% HIST(...)  without output arguments produces a normalized histogram bar
```

```
% plot of the results.
```

and now, save the changed file as `histn.m`, for example. If you want `histn` to be a global function, you can save it in the same folder `hist.m` was. Otherwise, you can save it in a different folder and then `histn` will only work if you are in this directory or if you add it to the MATLAB's search path, (see `path`).

1.2 Probability distributions

1.2.1 Random variables

Obviously, we can construct function files to evaluate the probability distribution of a specific random variable. For example, we can create an M-file to compute the Binomial probability density function given by,

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}, \qquad x = 1, 2, ..., n$$

with the following function:

```
function y = binomialpdf(x, n, p)
y =(gamma(n+1) * (p.^x) .* ((1-p).^(n-x))) ./ (gamma(x+1) .* gamma(n-x+1));
```

Observe that we can apply this function either to vectors or matrices. In these cases, it is recommended to make use of the logarithm of the gamma function to avoid numerical errors. This is defined with the `gammaln` function. Then, an alternative to the previous function could be:

```
function y = binomialpdf(x, n, p)
y = (gammaln(n+1) + x*log(p) + (n-x)*log(1-p) - gammaln(x+1) - gammaln(n-x+1));
y =exp(y);
```

To analyze the performance of these functions, let p a constant parameter and let vary the values for n considering the probability density functions of $Bin(5, 0.2)$, $Bin(10, 0.2)$ and $Bin(20, 0.2)$.

```
>> x5 = 0:5;
>> y5 = binomialpdf(x5, 5, 0.2);
>> x10 = 0:10;
>> y10 = binomialpdf(x10, 10, 0.2);
>> x20 = 0:20;
>> y20 = binomialpdf(x20, 20, 0.2);
>> plot(x5, y5, '.', x10, y10, '+', x20, y20, '*');
>> legend('n = 5 , p = 0.2' , 'n = 10 , p = 0.2' , 'n = 20 , p = 0.2');
```

Now, let's consider different values for p and let constant the value of n considering the probability density functions of $Bin(100, 0.1)$, $Bin(100, 0.5)$ and $Bin(100, 0.8)$.

```
>> x = 0:100;
>> y1 = binomialpdf(x, 100, 0.1);
>> y2 = binomialpdf(x, 100, 0.5);
>> y3 = binomialpdf(x, 100, 0.8);
>> plot(x, y1, '.', x, y2, '+', x, y3, '*');
>> legend('n = 100 , p = 0.1' , 'n = 100 , p = 0.5' , 'n = 100 , p = 0.8');
```

We can also construct a function to generate `m` random numbers from a Binomial distribution with parameters `n` and `p`:

```
function y = binomialrnd(m, n, p)
prob = rand(n, m) ;
successes = prob < p ;
y = sum(successes);
```

and we can try this function with:

```
>> y10 = binomialrnd(10, 100, 0.1);
>> mean(y10);
>> var(y10);
>> y100 = binomialrnd(100, 100, 0.1);
>> mean(y100);
>> var(y100);
```

1.2.2 Probability distributions with the Statistics Toolbox of MATLAB.

For a large number of distributions, the Statistics Toolbox provides five functions to provide the Probability Density Function (`pdf`), the Cumulative Distribution Function (`cdf`), the Inverse Cumulative Distribution Function, a Random Number Generator and the Mean and Variance as a Function of Parameters.

- Probability Density Function (`pdf`)

  The `pdf` function call has the same general format for every distribution in the Statistics Toolbox. The following commands illustrate how to call the pdf for the normal distribution. See `help normpdf`.

  ```
  >> x = [-3:0.1:3];
  ```

  ```
  >> f = normpdf(x,0,1);
  ```

  The variable `f` contains the density of the normal pdf with parameters $\mu = 0$ and $\sigma = 1$ at the values in `x`. The first input argument of every pdf is the set of values for which you want to evaluate the density. Other arguments contain as many parameters as are necessary to define the distribution uniquely. The normal distribution requires two parameters; a location parameter (the mean, μ) and a scale parameter (the standard deviation, σ).

- Cumulative Distribution Function (`cdf`)

  The cdf function call has the same general format for every distribution in the Statistics Toolbox. The following commands illustrate how to call the cdf for the normal distribution. See `help normcdf`.

  ```
  >> x = [-3:0.1:3];
  ```

  ```
  >> p = normcdf(x,0,1);
  ```

  The variable `p` contains the probabilities associated with the normal cdf with parameters $\mu = 0$ and $\sigma = 1$ at the values in `x`. The first input argument of every cdf is the set of values for which you want to evaluate the probability. Other arguments contain as many parameters as are necessary to define the distribution uniquely.

- Inverse Cumulative Distribution Function

  The inverse cumulative distribution function returns critical values for hypothesis testing given significance probabilities. To understand the relationship between a continuous cdf and its inverse function, try the following:

  >> `x = [-3:0.1:3];`

  >> `xnew = norminv(normcdf(x,0,1),0,1);`

  How does `xnew` compare with `x`? Conversely, try this:

  >> `p = [0.1:0.1:0.9];`

  >> `pnew = normcdf(norminv(p,0,1),0,1);`

  How does `pnew` compare with `p`?

  Calculating the cdf of values in the domain of a continuous distribution returns probabilities between zero and one. Applying the inverse cdf to these probabilities yields the original values. See `help norminv`.

- Random Number Generator

  You can generate random numbers from each distribution. This function provides a single random number or a matrix of random numbers, depending on the arguments you specify in the function call. The following command generate a random number from the normal distribution with parameters $\mu = 0$ and $\sigma = 1$.

  >> `x=normrnd(0,1)`

  Alternatively, normrnd(0,1,M,N) returns an M by N matrix of random numbers generated from a $N(0,1)$. See `help normrnd.`

- Mean and Variance as a Function of Parameters.

  The mean and variance of a probability distribution are generally simple functions of the parameters of the distribution. The Statistics Toolbox functions ending in `"stat"` all produce the mean and variance of the desired distribution for the given parameters. For example, the following command,

  >> `[m,v] = normstat(0,1)`
  returns the mean and variance of the normal distribution with parameters $\mu = 0$ and $\sigma = 1$. See `help normstat.`

There are a large number of functions to evaluate the Probability Density Function (`pdf`), the Cumulative Distribution Function (`cdf`), the Inverse Cumulative Distribution Function, a Random Number Generator and the Mean and Variance as a Function of Parameters of many other distributions than the normal distribution. This can be done replacing the term `"norm"` by one of the

following terms. See `lookfor distribution`.

`beta:`	Beta distribution	`logn:`	Lognormal distribution
`bino:`	Binomial distribution	`mvn:`	Multivariate normal distribution
`chi2:`	Chi-Square distribution	`mvt:`	Multivariate t distribution
`exp:`	Exponential distribution	`nbin:`	Negative binomial
`f:`	F distribution	`poiss:`	Poisson cumulative distribution
`gam:`	Gamma distribution	`unid:`	Uniform (discrete) distribution
`geo:`	Geometric distribution	`unif:`	Uniform (continuous) distribution
`hyge:`	Hypergeometric distribution	`weib:`	Weibull distribution

Type `randtool` for a random sample generation demo.

1.3 Multiple linear regression

1.3.1 Programming functions

We consider the multiple linear regression model,

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon,$$

where $\varepsilon \sim N\left(\mathbf{0}, \sigma^2\mathbf{I}_n\right)$ and $\mathbf{Y}$ is an $n \times 1$ vector of observations, $\mathbf{X}$ is an $n \times k$ matrix of regressors, β is a $n \times 1$ vector of parameters and ε is an $n \times 1$ vector of random disturbances. The least squares estimator of β is given by,

$$\hat{\beta} = \left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{Y},$$

whose variance is,

$$Var\left(\hat{\beta}\right) = \sigma^2\left(\mathbf{X}'\mathbf{X}\right)^{-1}$$

the predicted values are given by,

$$\hat{\mathbf{Y}} = \mathbf{X}\beta,$$

the residuals are,

$$\mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}},$$

and the residual variance is,

$$s_R^2 = \frac{\sum e_i^2}{n-k-1}.$$

We can now define the following function to solve the regression problem:

```
function [beta,Var_beta,ypred,sR2] = regres(X,Y)
% X = matrix of regressors with a column of ones.
% Y = vector of data from the dependent variable.
% beta = vector of regression coefficients.
% Var_beta = The variance covariance matrix of beta.
% ypred = predicted values for Y.
% sR2 = residual variance.
[n,k] = size(X);
k = k - 1;
```

```
beta = (X'*X)\(X'*Y);
ypred = X*beta;
resid = Y - ypred;
sR2 = sum(resid.^2)/(n - k - 1);
Var_beta = sR2*inv(X'*X);
```

We can test this function with the following commands:

```
>> n = 50; k = 4;
>> X = randn(n,k);
>> b = [0;1;0;1];
>> y = X*b + randn(n,1);
>> X = [ones(n,1) X];
>> [beta,Var_beta,ypred,sR2] = regres(X,y);
>> resid = y - ypred;
>> subplot(2,1,1),plot(resid,'o'),title('residuals versus row number')
>> subplot(2,1,2),plot(resid,ypred,'o'),title('residuals versus predicted')
```

1.3.2 Multiple linear regression with the Statistics Toolbox of MATLAB

The Statistics Toolbox provide the `regress` function to address the multiple linear regression problem. `regress` uses QR decomposition of $\mathbf{X}$ followed by the backslash operator to compute $\hat{\beta}$. The QR decomposition is not necessary for computing $\hat{\beta}$, but the matrix R is useful for computing confidence intervals.

`b = regress(y,X)` returns the least squares estimator $\hat{\beta}$.

`[b,bint,r,rint,stats] = regress(y,X)` returns an estimate of β in b, a 95% confidence interval for β in the $k \times 2$ array `bint`. The residuals are returned in `r` and a 95% confidence interval for each residual is returned in the $n \times 2$ array `rint`. The vector stats contains the R2 statistic along with the F and p values for the regression.

`[b,bint,r,rint,stats] = regress(y,X,alpha)` gives 100(1 - alpha)% confidence intervals for `bint` and `rint`. For example, `alpha = 0.2` gives 80% confidence intervals.

Let's see an example. Suppose the true model is,

$$\mathbf{Y} = \mathbf{X} \begin{pmatrix} 10 \\ 1 \end{pmatrix} + \varepsilon, \qquad \varepsilon \sim N\left(0, 0.01\mathbf{I}\right),$$

where $\mathbf{I}$ is the identity matrix. Suppose we have the following data:

```
>> X = [ones(10,1) (1:10)']
>> y = X * [10;1] + normrnd(0,0.1,10,1)
```

Then,

```
>> [b,bint] = regress(y,X,0.05)
```

Compare `b` to `[10 1]'`. Note that `bint` includes the true model values.

Another example comes from Chatterjee and Hadi (1986) in a paper on regression diagnostics. The data set (originally from Moore (1975)) has five predictor variables and one response.

```
>> load moore
```

```
>> X = [ones(size(moore,1),1) moore(:,1:5)];
```

Matrix `X` has a column of ones, and then one column of values for each of the five predictor variables. The column of ones is necessary for estimating the y-intercept of the linear model.

```
>> y = moore(:,6);
>> [b,bint,r,rint,stats] = regress(y,X);
```

The y-intercept is b(1), which corresponds to the column index of the column of ones.

```
>> stats
stats =
0.8107 11.9886 0.0001
```

The elements of the vector stats are the regression R2 statistic, the F statistic (for the hypothesis test that all the regression coefficients are zero), and the p-value associated with this F statistic.

R2 is 0.8107 indicating the model accounts for over 80% of the variability in the observations. The F statistic of about 12 and its p-value of 0.0001 indicate that it is highly unlikely that all of the regression coefficients are zero.

```
>> rcoplot(r,rint)
```

The plot shows the residuals plotted in case order (by row). The 95% confidence intervals about these residuals are plotted as error bars. The first observation is an outlier since its error bar does not cross the zero reference line.

1.4 Stochastic processes

In this section, we will simulate and represent graphically various simple stochastic processes.

1.4.1 Bernoulli process

A Bernoulli process is a discrete-time stochastic process consisting of finite or infinite sequence of independent random variables $X_1, X_2, X_3, ...$ such that,

$$X_i = \begin{cases} 1, & \text{with } prob = p, \\ -1, & \text{with } prob = 1 - p. \end{cases}$$

Random variables associated with the Bernoulli process include:

- The number of successes in the first n trials; this has a binomial distribution;
- The number of trials needed to get r successes; this has a negative binomial distribution.
- The number of trials needed to get one success; this has a geometric distribution, which is a special case of the negative binomial distribution.

We can simulate a realization of size 100 of a Bernoulli process with $p = 0.5$ as follows.

```
>> u=rand(100,1);
>> X(u<0.5)=1;X(u>0.5)=-1;
```

We could also have written:

```
X=1-2*floor(u*2);
```

or:

```
X=2*(u<0.5)-1;
```

We can simulate another realization of a Bernoulli process with $p = 0.25$ and observe the differences.

```
>> u=rand(100,1);
>> Y(u<0.25)=1;Y(u>0.25)=-1;
>> plot(1:100,X,'ro',1:100,Y,'k*')
```

1.4.2 Random walk

By using the `cumsum` command, we can simulate random walks from the Bernoulli processes simulated previously.

```
>> plot(1:100,cumsum(X),'r',1:100,cumsum(Y),'k')
```

1.4.3 Poisson process

Firstly, observe that continuous time processes are only possible to simulate by discretization of the unit time.

A Poisson process, X_t, with rate λ verifies the following property:

$$X_t = \text{ Number of occurrences in } [0,t) \sim Po\left(\lambda t\right).$$

If we want simulate a realization with 10 ocurrences from a Poisson process of rate $\lambda = 2$, we can first simulate 10 exponential times of mean $1/\lambda = 0.5$ between occurrences.

```
>> x=exprnd(0.5,1,10);
```

Then, we can obtain the occurrence times as follows.

```
>> x=cumsum(x);
>> subplot(2,1,1),plot(x,zeros(length(x)),'.')
```

Suppose we want to know the value of the process X_t at the following instant times:

```
>> t=0:0.1:ceil(x(end));
```

Then, we can compute:

```
>> for i=1:length(t);X(i)=sum(x<t(i));end
>> subplot(2,1,2),plot(t,X)
```

1.4.4 Autoregressive process

Suppose we want to simulate $T = 100$ values from an autoregressive model AR(1),

$$x_t = \alpha x_t + e_t,$$

where e_t are i.i.d. $N\left(0,1\right)$ and assume three values for $\alpha \in \{0.8, 0.5, -0.8\}$.

One possibility is to assume $x_1 = e_1$ and then obtain recursively the remaining values.

```
>> e=randn(100,1);
>> x=zeros(100,1);
>> x(1)=e(1);
>> alpha=0.8;
```

```
>> for i=2:100, x(i)=alpha*x(i-1)+e(i); end
```

Alternatively, we could simulate a realization of size $T = 200$ and then, consider the last 100 values.

We can calculate the sample coefficient of the autocorrelation function. For example, the first coefficient is the sample correlation coefficient of x_{t-1} and x_t:

```
>> corrcoef(x(1:99),x(2:100));
>> plot(x(1:99),x(2:100),'.')
```

Observe that after 10 lags, there is almost no relation between x_{t-10} and x_t.

```
>> plot(x(1:90),x(11:100),'.')
```

1.4.5 Moving average process.

Suppose now that we want to simulate $T = 100$ values from a moving average model MA(1),

$$x_t = \theta e_{t-1} + e_t,$$

where e_t are i.i.d. $N(0, 1)$ and assume three values for $\alpha \in \{0.8, 0, -0.8\}$.

This process is easier to initialize because we just have to simulate e_0.

```
>> e=randn(101,1);
>> theta=0.8;
>> x=theta*e(1:100,1)+e(2:101,1);
```

Compute the first two coefficients of the autocorrelation function and observe the following plots:

```
>> plot(x(1:99),x(2:100),'.')
>> plot(x(1:98),x(3:100),'.')
```

1.5 Monte Carlo simulation of known results

Frequently, in statistics, the results are shown by simulation. The idea of Monte Carlo simulation is to generate samples from a distribution by using a computer. This technique is very useful, in special when there is not a closed expression for the results we want to obtain. In this section, we will simulate some known results in statistics.

1.5.1 Central limit theorem

We generate 1000 samples of size 50 from an exponential distribution with mean $\lambda^{-1} = 2$:

```
>> E=exprnd(2,50,1000);
```

Columns represent different samples. We define a row vector containing the mean value of each column. Then, we plot a histogram of this mean values, what do you expect to see?

```
>> m=mean(E);
>> hist(m)
```

Taking into account that the mean of the exponential distribution is $\mu = \lambda^{-1} = 2$ and the variance is $\sigma^2 = \lambda^{-2} = 4$, which values do you expect for the sample mean and variance of `m`?

```
>> mean(m)
```

```
>> var(m)
```

Finally, compute the sample mean of the variances obtained for each sample, what result do you expect?

```
>> v=var(E);
>> mean(v)
```

1.5.2 Some properties of a random sample from a normal population

We will show by simulation the following properties of the normal distribution. Given a random sample $X_1, X_2, ..., X_n$ from a normal population $N(\mu, \sigma)$ and let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ the sample mean, and let $S^2 = \frac{1}{n-1}\left[\frac{1}{n}\sum_{i=1}^{n} X_i^2 - n\bar{X}\right]$ the sample variance, it can be shown that:

1. $\bar{X} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$
2. $\frac{(n-1)S^2}{\sigma^2} \sim \chi_{n-1}$
3. $\bar{X}$ and S^2 are independent.

We generate 100 samples of size $n = 10$ from an normal distribution, $N(\mu = 5, \sigma = 2)$.

```
>> mu=5;sigma=2;n=10;
>> N=normrnd(mu,sigma,n,100);
```

And compute the mean and variance of each column which represent a different sample.

```
>> xbar=mean(N);
>> S2=var(N);
```

Property 1:

```
>> mean(xbar)
>> std(xbar)
>> histn(xbar)
>> hold on;
>> x=2.5:.01:7.5;plot(x,normpdf(x,mu,sigma/sqrt(n)))
```

Property 2:

```
>> mean(S2)
>> chi=(n-1)*S2/sigma^2;
>> histn(chi)
>> hold on;
>> x=0:.1:30;plot(x,chi2pdf(x,n-1))
```

Property 3:

```
>> plot(xbar, S2,'.')
>> corrcoef(xbar, S2)
```

1.6 Optimization: Newton-Raphson method to obtain maximum likelihood estimators

Given a sample of data, $x_1, ..., x_n$, i.i.d. from a common distribution $f(x \mid \theta)$, the maximum likelihood estimator of θ is the value (or vector of values) that maximizes the likelihood function,

$$likelihood = f(\theta \mid x_1, ..., x_n),$$

where f is the joint density function of the sample as a function of the parameters θ.

A necessary condition for $\hat{\theta}$ to maximize f is that,

$$\nabla_\theta f\left(\hat{\theta}\right) = 0,$$

which, in general, is a non linear system of equations where it is difficult to obtain the value of $\hat{\theta}$. Thus, it can be useful to consider numerical methods to approximate its value.

Newton-Raphson method is frequently used to maximize/minimize functions. The Newton-Raphson method uses the following iterative process to maximize the function $f(x)$ defined on $x \in \mathbb{R}^n$,

$$x_{k+1} = x_k - H_k^{-1} g_k,$$

where $g_k$and H_k are the gradient and the Hessian matrix of f evaluated at x_k and x_0 is a value close to the optimum. This iteration process finishes when g_k is approximately equal to 0.

We will consider this algorithm to estimate the parameters of a Weibull distribution. We need first the extreme value distribution. In the following program, the logarithm of the likelihood, the gradient and the Hessian of an extreme value distribution. See `help weibfit`.

```
function [z,grad,H] = LogVerEV(param,X,r)
% Logarithm of the likelihood function of an Extreme
% Value distribution with parameters param=(u,b)
% X = data sample where the first r observations are complete
% z = Logarithm of the likelihood function.
% grad = gradient vector
% H = Hessian matrix
n = length(X);
u = param(1);
b = param(2);
Y = X(1:r);
s1 = sum((Y - u)./b);
s2 = sum(exp((X - u)./b));
z = -r*log(b) + s1 - s2;
grad = zeros(2,1);
grad(1) = -r/b + s2/b;
s3 = sum(((X - u)./b).*(exp((X - u)./b)));
grad(2) = -r/b - s1/b + s3/b;
H = zeros(2,2);
H(1,1) = -s2/(b<2);
s4 = sum((((X - u)./(b<2)).<2).*(exp((X - u)./b)));
```

```
H(2,2) = r/(b<2) + 2*s1/(b<2) - 2*s3/(b<2) - s4;
H(2,1) = r/(b<2) - s2/(b<2) - s3/(b<2);
H(1,2) = H(2,1);
```

Now, we can make use of the Newton-Raphson method to estimate the parameters of an Extreme Value and finally, we apply the corresponding transformation of the parameters to obtain the maximum likelihood estimators of the parameters of a Weibull distribution.

```
clear
clf
% We generate 50 samples of size 300 from a
% Weibull distribution (alpha = 2, beta = 0.5) and
% the last 60 observation are censored
n = 300;
r = 240;
rand('seed',sum(100*clock));
alpha = 2;
beta = 0.5;
X = weibull(alpha,beta,n);
if any(X==0)==1,
I = find(X==0);
X(I) = eps*ones(length(I),1);
end
X = sort(X);
X(r+1:n) = ones(n-r,1)*X(r);
Y = log(X);
param = [0.6,1.5]'; % punto inicial
[z,grad,H] = LogVerEV(param,Y,r);
while grad > 1e-4,
param = param - inv(H)*grad;
[z,grad,H] = logVerEV(param,Y,r);
end
alpha = exp(param(1));
beta = 1/param(2);
```

References

[1] The MathWorks - Statistics Toolbox.
http://www.mathworks.com/access/helpdesk/help/toolbox/stats/stats.shtml

[2] Prácticas de Estadística. Ingeniería de Telecomunicación. Departamento de Estadística. Universidad Carlos III. (*In Spanish).*
http://halweb.uc3m.es/esp/docencia/esteleco/ps-pdf/practica5new.pdf
http://halweb.uc3m.es/esp/docencia/esteleco/ps-pdf/practica6.pdf
http://halweb.uc3m.es/esp/docencia/esteleco/ps-pdf/practica7.pdf