Time Series and Spatial Data Analysis with SCILAB

By

Gilberto E. Urroz, Ph.D., P.E.

Distributed by

infoClearinghouse.com

A "zip" file containing all of the programs in this document (and other SCILAB documents at InfoClearinghouse.com) can be downloaded at the following site:

http://www.engineering.usu.edu/cee/faculty/gurro/Software_Calculators/Scilab_Docs/ScilabBookFunctions.zip

The author's SCILAB web page can be accessed at:

http://www.engineering.usu.edu/cee/faculty/gurro/Scilab.html

Please report any errors in this document to: gurro@cc.usu.edu

Time series and spatial data

In this chapter we present the subjects of time series and spatial data, aspects of statistical analysis of interest in hydrology, geology, and other Earth sciences. The subject of time series is of interest in the analysis of time-dependent phenomenon such as temperature variations in the atmosphere or the ocean, precipitation, economic patterns, transportation volumes in highways and airports, and even TV ratings. Spatial data and geostatistics involve the application of selected statistical techniques to data with spatial distributions, such as vegetation patterns in selected basins, spatial precipitation patterns, aquifer properties, etc.

We will start the chapter with the treatment of time series.

Time series

As the name indicates, a time series is simply a collection of data values that vary with time. One of the main applications of time series analysis is the determination of periodic components in the series (also referred to as a time-dependent signal). To facilitate the analysis, the data included in time series is typically collected at regular intervals. Thus, government agencies such as the National Weather Service or the U.S. Geological Survey maintain data bases of such parameters as daily minimum, maximum, and mean temperature, pressure, precipitation, flow discharges, water surface elevations, etc.

Non-stationary time series of natural variables present a certain gradual trend, i.e., a slowly-varying increase or decrease of the quantity measured that is apparent if the periodic components were removed. Even in non-stationary time series there are some statistical properties that present some degree of stationarity. The trend can be removed out of a non-stationary time series by simply taking the first difference of the signal values, i.e., the increments between consecutive values in the time series. If u_i, $i=1,2,\ldots,n$, represents the time series, with $u_i = u(t_i)$, and t_i increasing at regular intervals, the first difference in the time series is simply the vector of quantities

$$\Delta u_i = u_{i+1} - u_i,$$

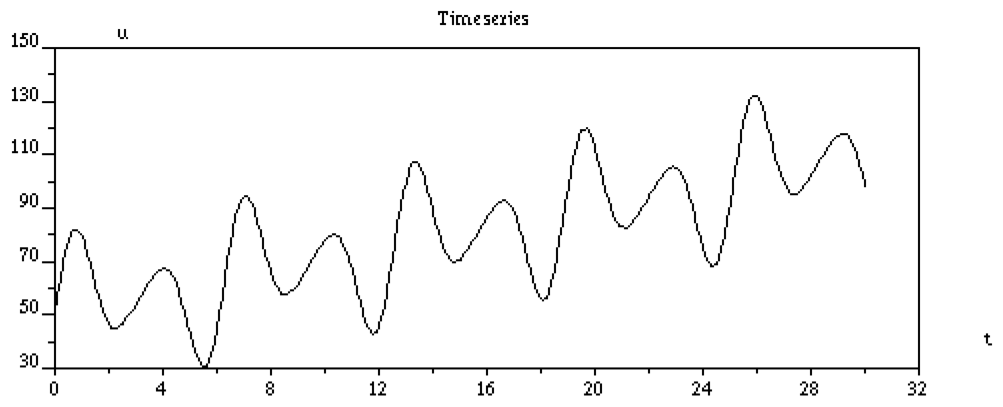
for $i=1,2,3,\ldots,n-1$.

To illustrate the use of first differences for removing the trend in a non-stationary time series, we generate a synthetic time series, $u(t)$, using the following SCILAB statements:

```
-->t = [0:0.1:30]; u = 10*sin(t)+20*sin(2*t)+5*sin(3*t);

-->u = u + 2*t + 50 + 0.5*(rand(1,length(t))-0.5);
```

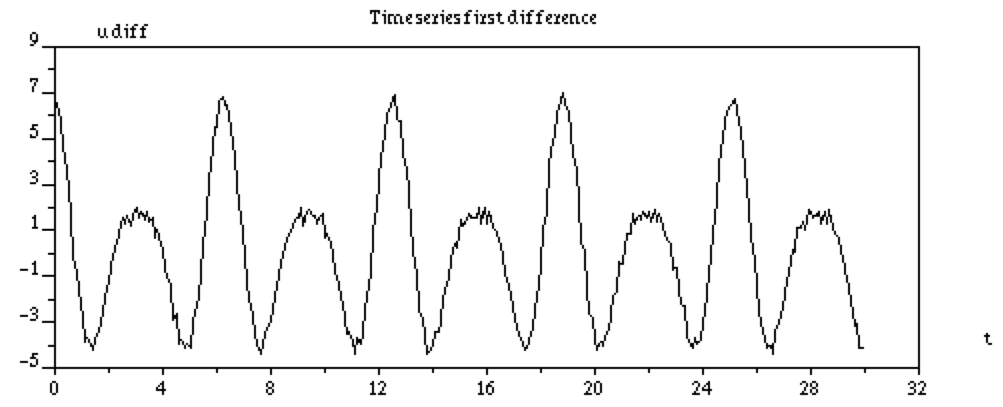
A plot of the time series is shown next. There is obviously an increasing trend apparent from the signal, upon which the periodic components of the signal are superimposed.

```
-->xset('window',0);plot(t,u,'t','u','Time series')
```



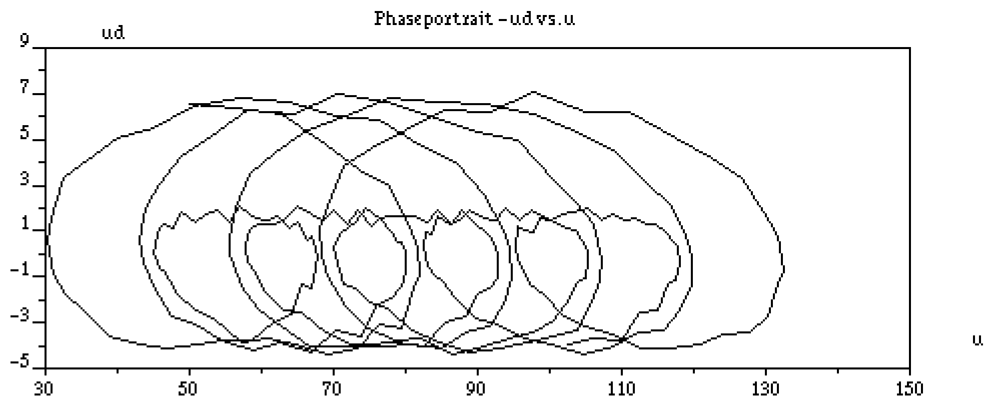
The first difference vector can be calculated using the SCILAB function *mtlb_diff*. The resulting vector, *ud*, has *n-1* components. To plot this first difference vector we use a time vector, *td*, that incorporates components *1* through *n-1* of the original vector *t*. The plot of the first differences is shown next.

```
-->ud=mtlb_diff(u);td=t(1:length(t)-1);

-->xset('window',1);plot(td,ud,'t','u_diff','Time series first difference')
```



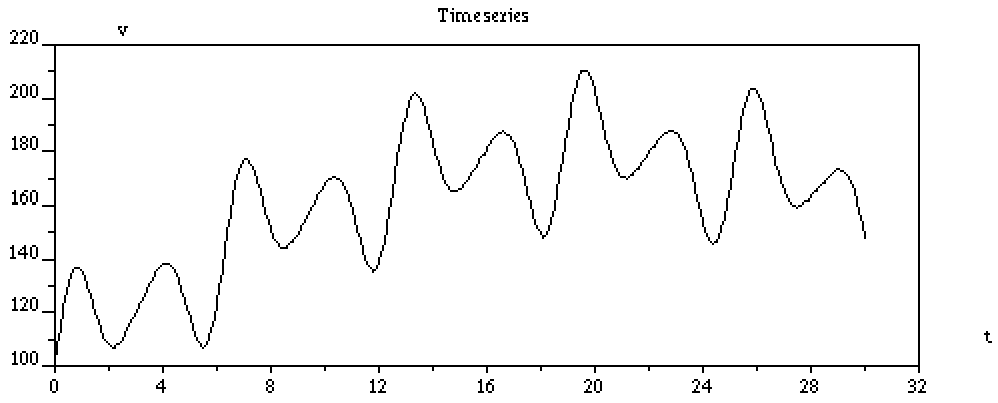
A phase portrait of the first difference *ud* against the original signal *u* is obtained by using:

```
-->plot(ud,u(1:length(ud),’ud’,’u’
```



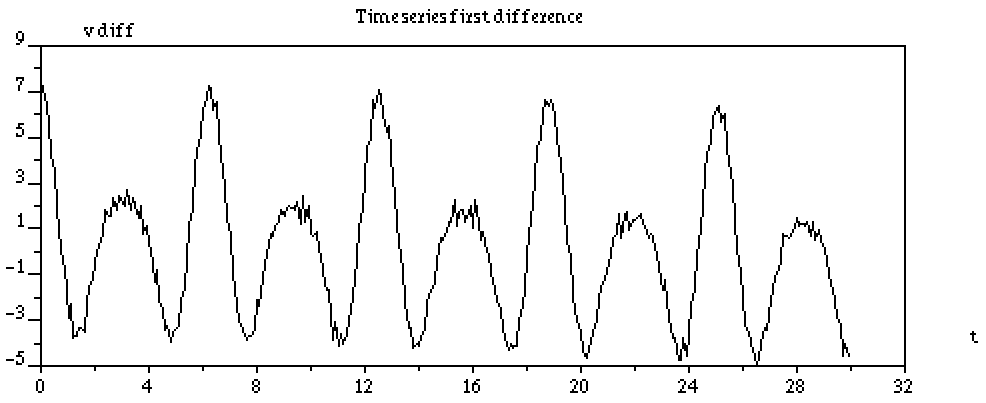
The following SCILAB statements produce a non-stationary time signal with a curvilinear trend.

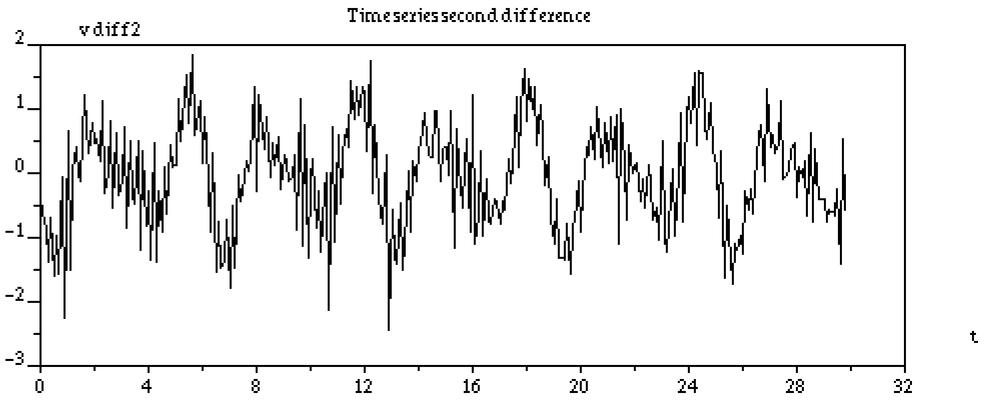
```
-->v = u + 2*t.*(30-t)/10 + 50 + 0.5*(rand(1,length(t))-0.5);
-->xset('window',2);plot(t,v,'t','v','Time series')
```



Plots of the first and second difference for this signal are shown next.

```
-->vd=mtlb_diff(v);td=t(1:length(t)-1);

-->xset('window',3);plot(td,vd,'t','v_diff','Time series first difference')

-->vdd=mtlb_diff(vd);tdd=td(1:length(td)-1);

-->xset('window',4);plot(tdd,vdd,'t','v_diff2','Time series second difference')
```



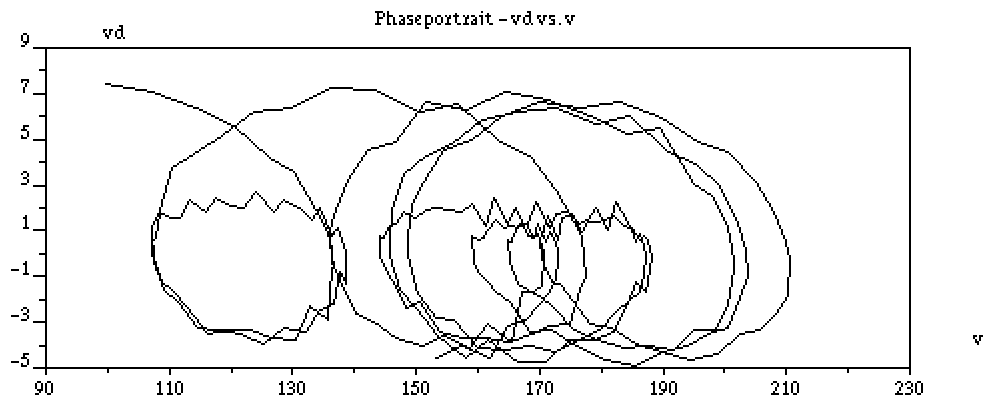


The plot of first differences or the second signal, *v(t)*, is very similar to the plot of first differences for the first signal, *u(t)*, since the stationary (periodic) components were generated using the same function. The second difference plot for the second signal shows also a periodic behavior.

Notice that the last graph (second difference of *v(t)*) shows a periodic behavior modified by some random components that were introduced in the signal by using SCILAB function *rand()*. This is another characteristic of time series, i.e., the presence of a certain random *noise* that many a times needs to be removed from the actual periodic signal.

A phase portrait of the first difference *vd* against the signal *v* is shown next.

```
-->plot(vd,v(1:length(vd),'v','vd','Phase portrait - vd vs. v')
```



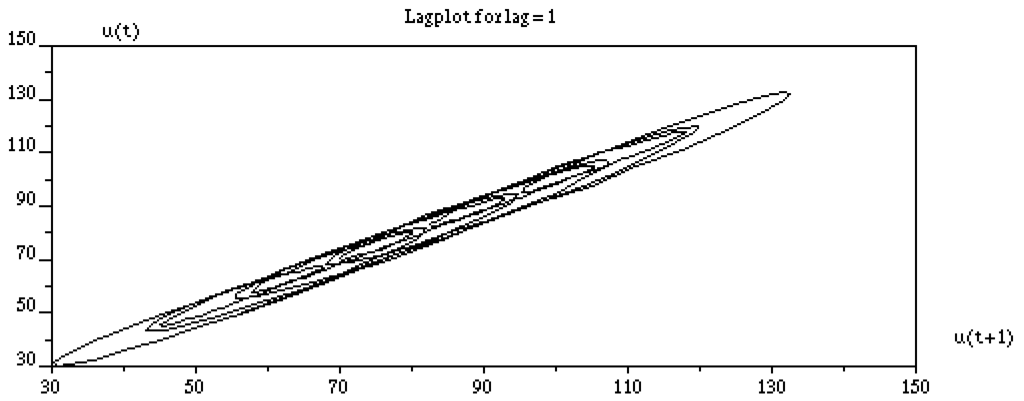
Lag plots

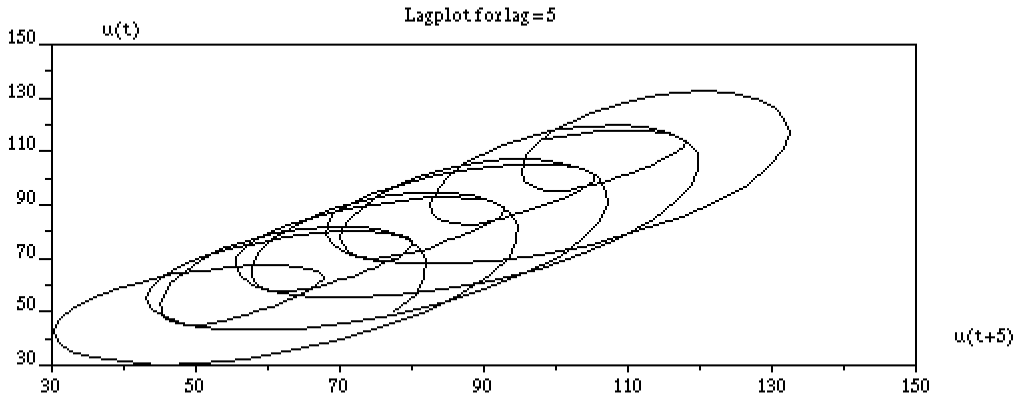
Lag plots are representations of the signal *u(t+τ)* against the signal *u(t)*. The value *τ* is known as the *lag* between the two signals. Lag plots can be used to identify periodicity in the signal. Periodic signals show certain regular *orbits* as illustrated below with a few lag plots for signal *u(t)*. To produce the lag plots we use the following function:

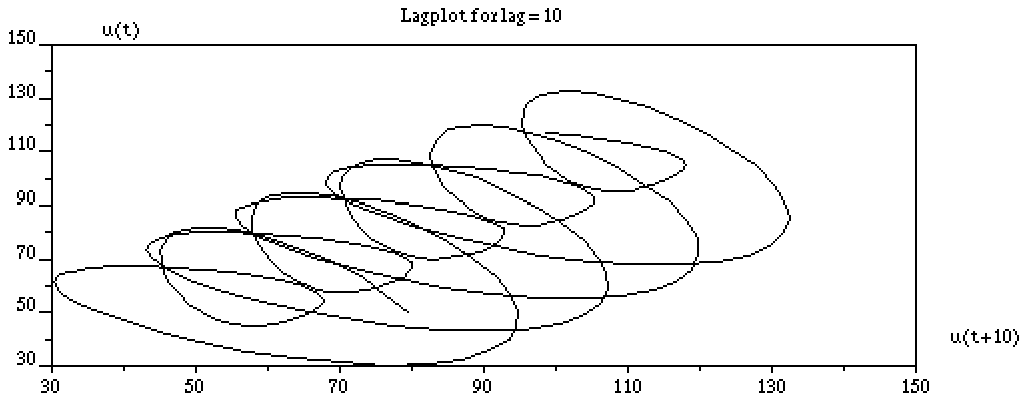
```
function [] = lagplot(u,lag)

//Plot signal u(t+lag) vs u(t) given a vector
//of data u and an integer lag

n = length(u)

uu = u(lag+1:n);
up = u(1:length(uu));

plot(uu,up,'u(t+'+string(lag)+')','u(t)','Lagplot for lag = '+string(lag));
```

The following plots are lag plots for signal *u(t).* The lag is identified in the plot labels.

```
-->lagplot(u,1)

-->lagplot(u,5)

-->lagplot(u,10)
```

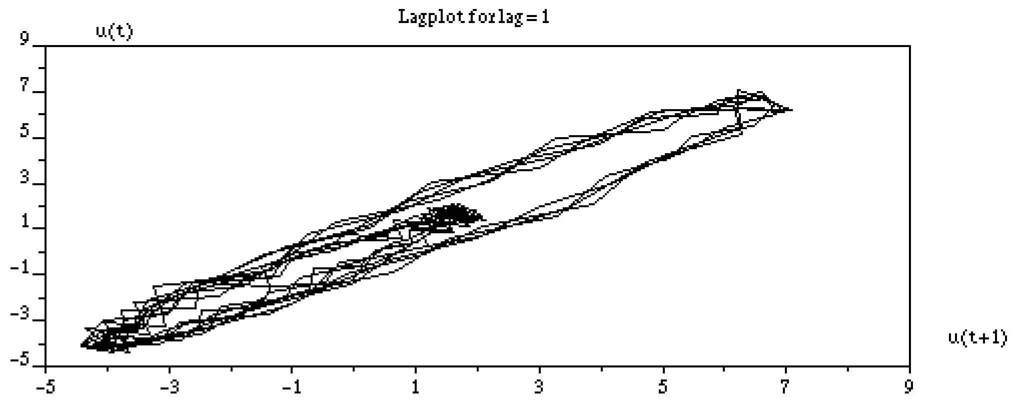


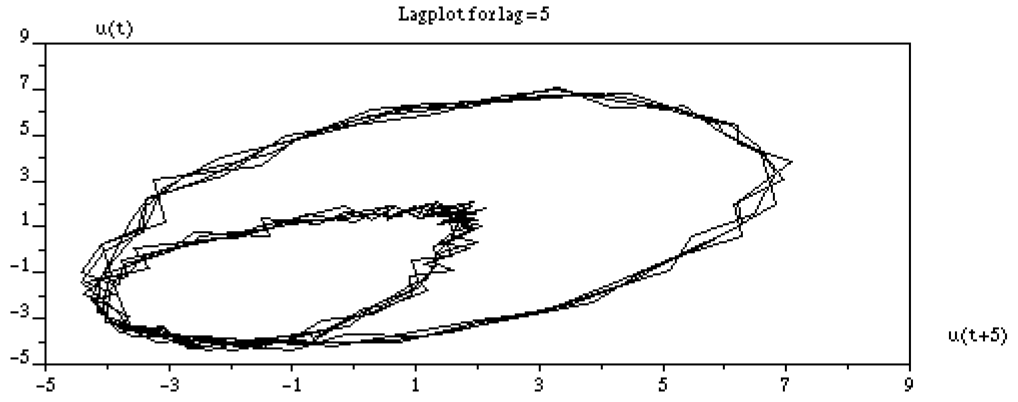


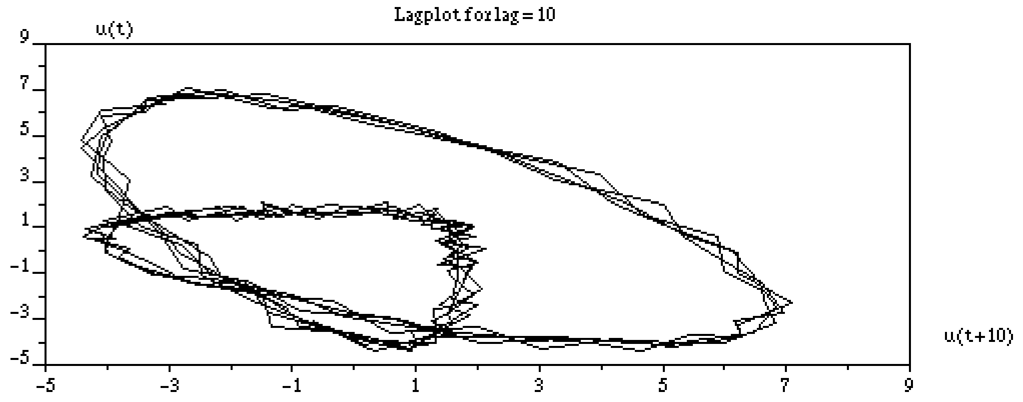


The periodicity of the signal is obvious from the regular orbits that result from the lag plots. Lag plots of the first difference are shown next. Because the trend of the signal has been removed in the first difference vector, resulting in a stationary periodic function, the result is a single orbit, affected, of course, by the random noise the was included in the original signal.

```
-->lagplot(ud,1)

-->lagplot(ud,5)

-->lagplot(ud,10)
```







Periodic stationary components

Once the trend has been removed from a non-stationary time series, the resulting stationary signal may present a periodic behavior as demonstrated earlier. The identification of the period or frequency of each of the periodic components in a stationary time series constitutes an important application of time series analysis. It is often assumed that a stationary time series posses a certain *fundamental period* T_0, i.e., the time interval after which the signal repeats itself. In many instances, such a period is not easily identified and it is taken as the length of the signal itself. The frequency of a periodic component of period T is

$$f = 1/T,$$

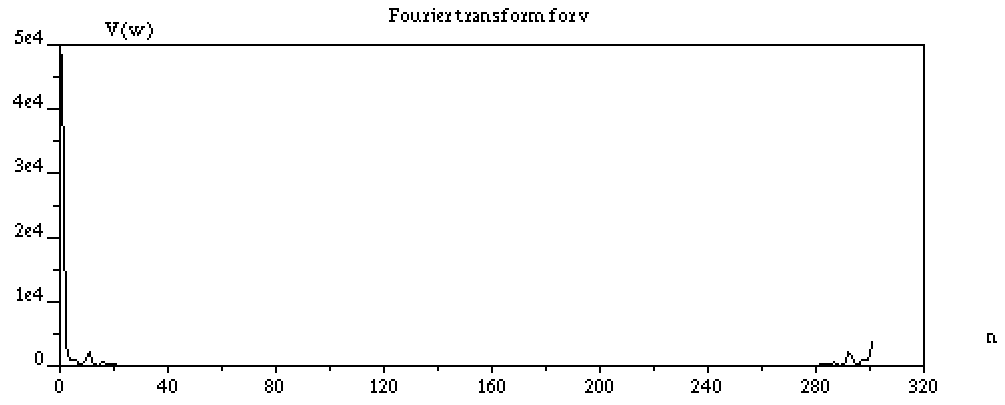
while the angular frequency is given by

$$\omega = 2\pi/T = 2\ \pi\ f.$$

The techniques of Fourier analysis, presented in Chapter 10, are extremely useful in time series analysis. Through the use of Fourier transforms it is possible to convert a signal from the *time domain,* i.e., *f(t)*, to the *frequency domain*, i.e., *F(f)* or *F(ω)*, and vice versa.

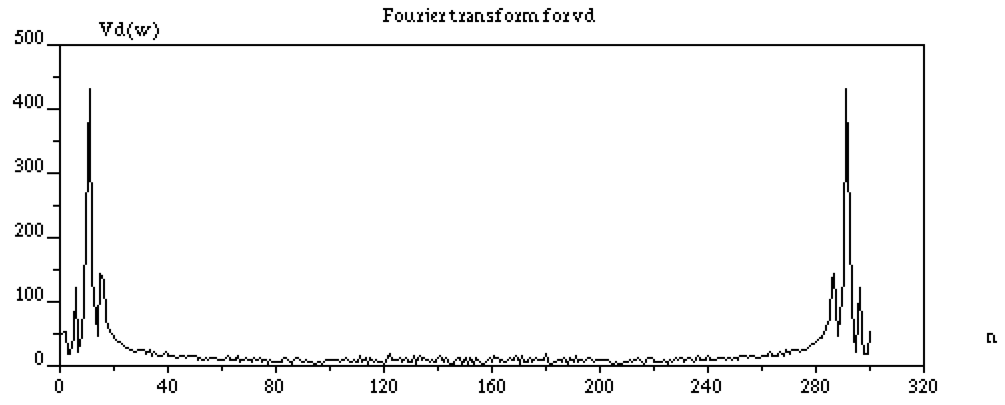
As indicated in Chapter 10, SCILAB provides functions *dft* and *fft* to determine the Fourier transforms from a signal *v* given as a vector of data. For example, for the data *v(t)* generated earlier, *dft* produces the following Fourier transform.

```
-->V = dft(v,-1);VA = abs(V);plot(VA);
-->xtitle('Fourier transform for v','n','V(w)');
```



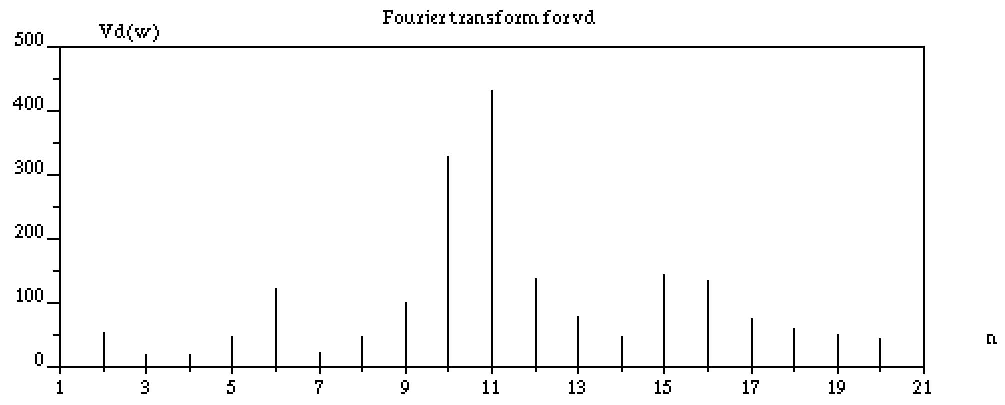
Notice that there is very large component at n = 1, and a tiny peak near 10. The very large component at n = 1 is associated with the length of the signal, i.e., the discrete Fourier transform assumes a fundamental period equal to the length of the signal, or $T_0 = 30$. Eliminating the trend, we can eliminate that large component for *n =1*. A plot of the Fourier transform of the first differences in the signal *v(t)* is shown next:

```
-->Vd = dft(vd,-1);VdA = abs(Vd);

-->plot(VdA);xtitle('Fourier transform for vd','n','Vd(w)');
```



This graph of the absolute value of the Fourier transform of the second differences in signal *v(t)* reveals three main components located at values of *n* close to 10. A plot showing the components *n=1* to *20* reveals that the main components of the Fourier transform correspond to values of *n* close to *6, 11*, and *15*. Recall that the initial time series was generated by adding three sine functions with different frequencies. Thus, the Fourier transform of the first differences reveals also three main components.

```
-->plot2d3(‘onn’,[1:1:20]’,VdA(1:20));

-->xtitle(‘Fourier transform for vd’,’n’,’Vd(w)’)
```



Autocovariance and autocorrelation

We will define *autocovariance* as the covariance between the signal *u(t)* and the lagged signal *u(t+τ)*, i.e.,

$$R(\tau) = E[u(t)\cdot u(t+\tau)].$$

Some references (e.g., Newland, D.E., 1993, “*An Introduction to Random Vibrations, Spectral & Wavelet Analysis*,” Longman Scientific & Technical, London) use the term *autocorrelation function* for *R(τ)*. On the other hand, Middleton (Middleton, G.V., 2000, “*Data Analysis in the Earth Sciences Using MATLAB®,”* Prentice Hall, Upper Saddle River, New Jersey) indicates that the term autocorrelation refers to *R(τ)* divided by the variance of the signal *u(t)*.

SCILAB provides function *corr* to calculate the autocovariance function out of a vector signal *u*. The call to the function with this purpose is

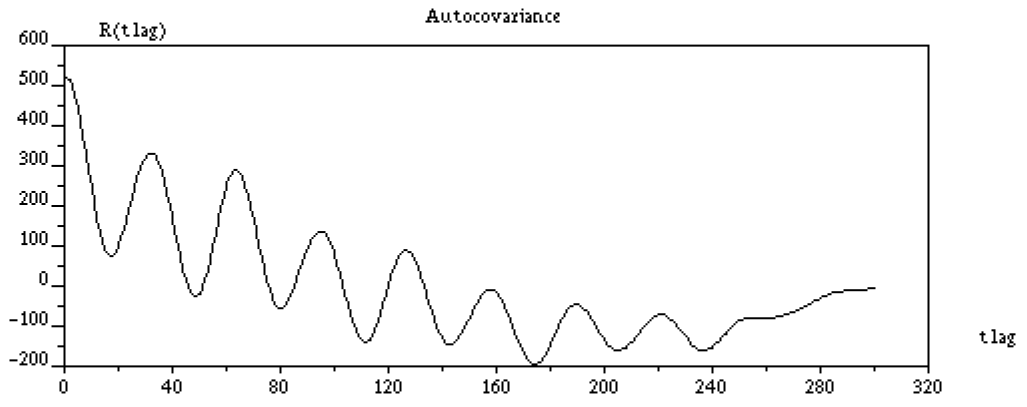
[cov,ubar] = corr(u,nlags)

where *cov* is the vector of length *nlags* containing the values of *R(τ),* where *τ* represents (integer) lags from *1* to *nlags*, and *ubar* is the mean value of the signal *u*.

The following plot shows the autocovariance function *R(τ)* for the signal *u(t)* defined earlier.

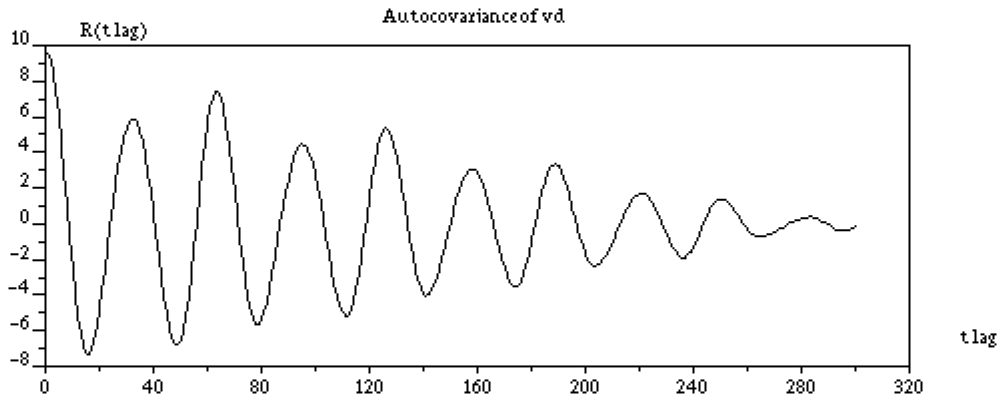
```
-->[cov,xbar]=corr(u,300);
```

```
-->xbar
 xbar  = 80.994022


-->plot(cov,'t lag','R(t lag)','Autocovariance')
```

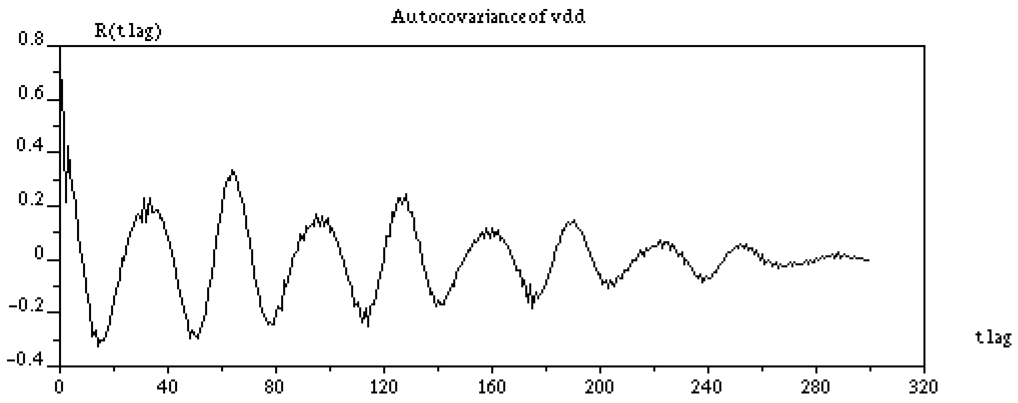


The autocovariance of the first and second differences of signal *v(t)* is shown next.

```
-->[covvd,meanvd]=corr(vd,300);

-->plot(covvd,'t lag','R(t lag)','Autocovariance of vd')
```



```
-->[covvdd,meanvdd]=corr(vdd,299);

-->plot(covvdd,'t lag','R(t lag)','Autocovariance of vdd')
```



Cross-covariance and cross-correlation

The c*ross-covariance* (also referred as *cross-correlation function*) of two time series *u(t)* and *v(t)* is defined as

$$R_{uv} = E[u(t)\cdot v(t+\tau)].$$

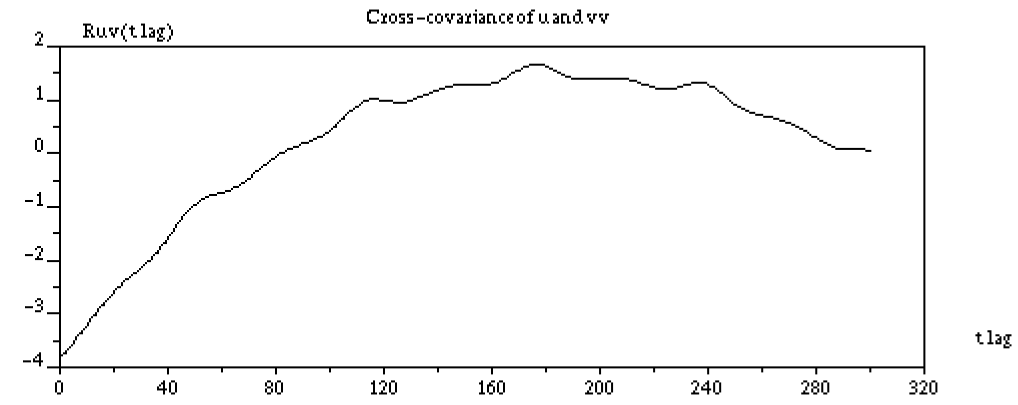
Cross-covariances can be calculated using function *corr* with the call

[cov,means] = corr(u,v,nlags)

where *cov* is the vector of length *nlags* containing the values of $R_{uv}(\tau)$, where τ represents (integer) lags from *1* to *nlags*, and *means* is a vector containing the mean values of the signals *u* and *v*.

To illustrate the use of cross-covariance calculations, we first generate a vector of values of the function *vv(t) = exp(-0.1t)*, for the values of *t* defined earlier, and calculate the cross-covariance of signal *u(t)* with signal *vv(t)* as follows:

```
-->vv = exp(-0.1*t);
-->[couvv,means]=corr(u,vv,300);
-->plot(couvv,'t lag','Ruv(t lag)','Cross-covariance of u and vv')
```



The cross-covariance of signal *vv(t)* and *u(t)* is calculated as follows:

```
-->[couvv,means]=corr(u,vv,300);
-->plot(couvv,'t lag','Ruv(t lag)','Cross-covariance of u and vv')
```

You can check that the cross-covariance of *u(t)* with *vv(t)* is the same as that of *vv(t)* with *u(t)*.

Convolution

The operation of convolution is defined as

$$y(t) = (u * v)(t) = \int_{-\infty}^{t} u(t-\tau)v(\tau)d\tau$$

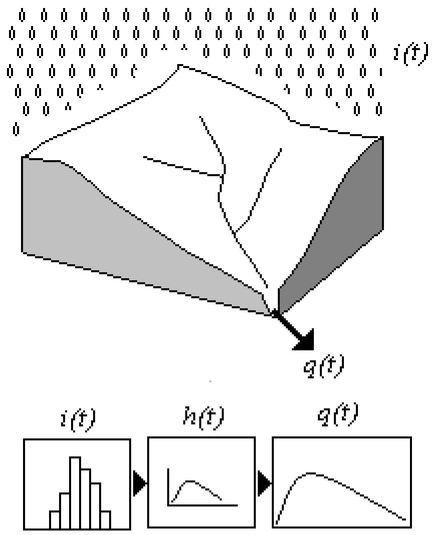
If *U(ω), V(ω),* and *Y(ω)*, represent the Fourier transforms of the signals *u(t), v(t),* and *y(t)*, respectively, then

$$Y(\omega) = U(\omega)\cdot V(\omega).$$

A physical interpretation of the convolution operation is found in the calculation of a hydrograph in a hydrologic basin. A hydrologic basin can be considered as a (linear) system. The input to the system is a set of values of precipitation *i(t)* whose graph is known as a hyetograph. The basin is characterized by a function *h(t)*, known as the unit hydrograph, which represents the discharge output corresponding to a unit precipitation on the basin. The output from the system is the water discharge out of the basin *q(t)*,which is calculated as

$$q(t) = (h * i)(t) = \int_{-\infty}^{t} h(t-\tau)i(\tau)d\tau.$$

These ideas are illustrated in the figure shown below.



For a couple of discrete signals, $u = [u_1\ u_2\ \dots\ u_n]$, $v = [v_1\ v_2\ \dots\ v_m]$, the convolution $y(t) = (u*v)(t)$ is achieved by calculating the elements of signal y as

$$y_i = \sum_{j=1}^{i} u_j v_{i-j}.$$

This operation is implemented in SCILAB using function *convol*, which is described below.

SCILAB function *convol*

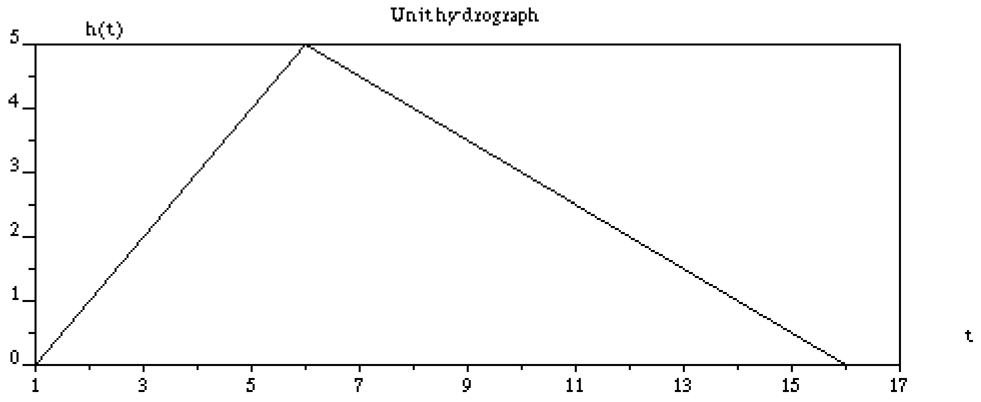
SCILAB provides function *convol* to calculate the convolution of two functions. The simplest call to the function is

[y] = convol(u,v)

where y is the vector of data representing the convolution function *y(t) = (u*v)(t).*

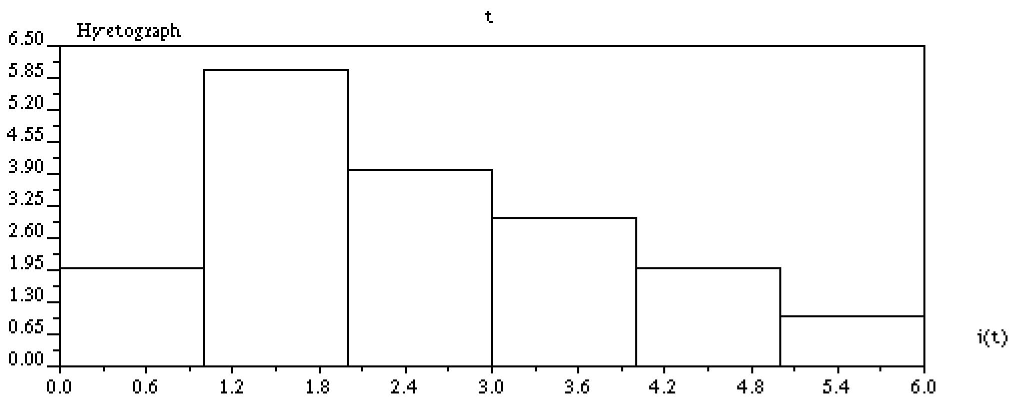
For example, consider the convolution of the following two functions *h(t)*, representing a unit hydrograph, and *i(t)*, representing a hyetograph. The unit hydrograph is shaped as a triangle:

```
-->h=[0 1 2 3 4 5 4.5 4 3.5 3 2.5 2 1.5 1 0.5 0];

-->plot(h,'t','h(t)','Unit hydrograph')
```



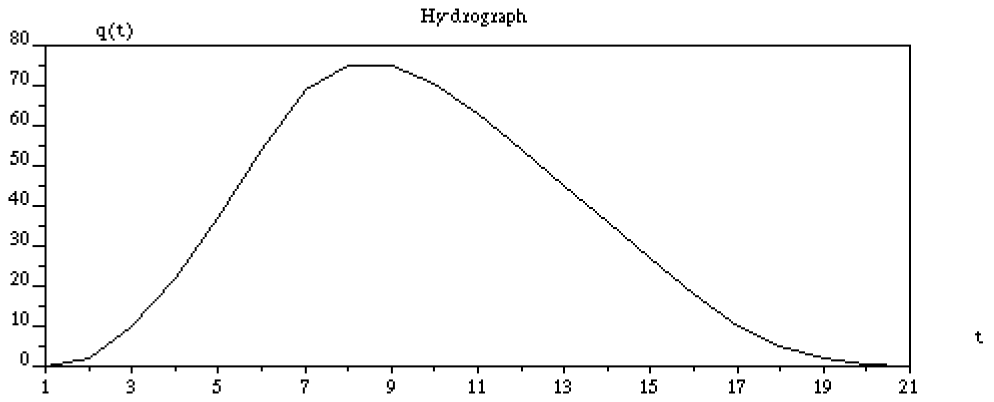
The hyetograph is given by six values of precipitation:

```
-->i=[2 6 4 3 2 1]; ii = [i i(6)];
-->plot2d3('onn',[1:6]', i',1,'011',' ',[0 0 6 6.5]);
-->plot2d2('onn',[0:6]',ii',1,'011',' ',[0 0 6 6.5]);
-->xtitle('t','i(t)','Hyetograph')
```



The convolution q(t) = (h*i)(t) produces the hydrograph:

```
-->q = convol(h,i);
-->plot(q,'t','q(t)','Hydrograph')
```



Moving averages

A moving average consists in replacing a signal *v(t)*, represented by a vector *v* of length *n*, by a signal $v_k(t)$ where *k* is a positive integer, such that

$$(v_k)_j = \frac{1}{k}\sum_{i=j}^{j+k-1} v_i,$$

j=1,2,…,n-k+1. The resulting signal is represented by a vector *vk* with *n-k+1* elements. This approach will produce a reduced-size smoothed signal.

An alternative definition is to use the convolution of a vector of *n* ones, *u=[1 1 … 1]*, with the signal *v* divided by *n*, i.e.,

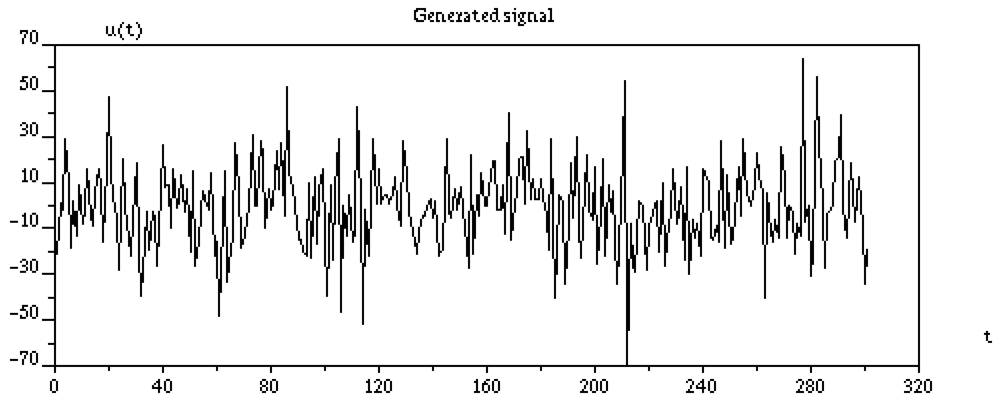
*vk = (u*v)(t)/n.*

To illustrate the use of moving averages, we present the following function, *ranf*, that generates a signal consisting of random components combined in a Fourier series. A listing of the function follows:

```
function [y] = ranf(x)
//Random Fourier series
y=[];
for j = 1:length(x)
    y = [y ff(x(j))];
end;

function [y] = ff(x)

n=100*rand();
aa = int(10*(rand(1,n)-0.5));
bb = int(10*(rand(1,n)-0.5));
y = 0.0;
for j = 1:n
    y = y + aa(j)*sin(2*%pi*x)+bb(j)*cos(2*%pi*x);
end;
```

The function requires as input a vector of values of x. We will generate a signal of 301 points by using the following SCILAB commands:

```
-->getf('ranf')

-->x=[0:1:300];u=ranf(x);plot(u,'t','u(t)','Generated signal')
```



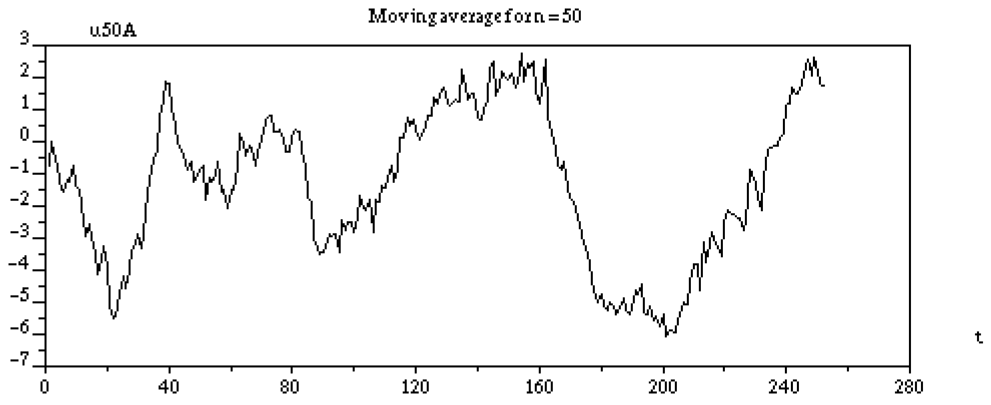
Next, we define a function to calculate the moving averages using the two approaches outlined above. The function is listed next.

```
function [uk,ukp] = moving(u,k)

//Calculation of moving averages
//Averaging reduces vector size to n-k+1 elements
n = length(u);
m = n-k+1;
uk = zeros(1,m);
for j = 1:k
  uk = uk + u(j:m+j-1);
end;
```

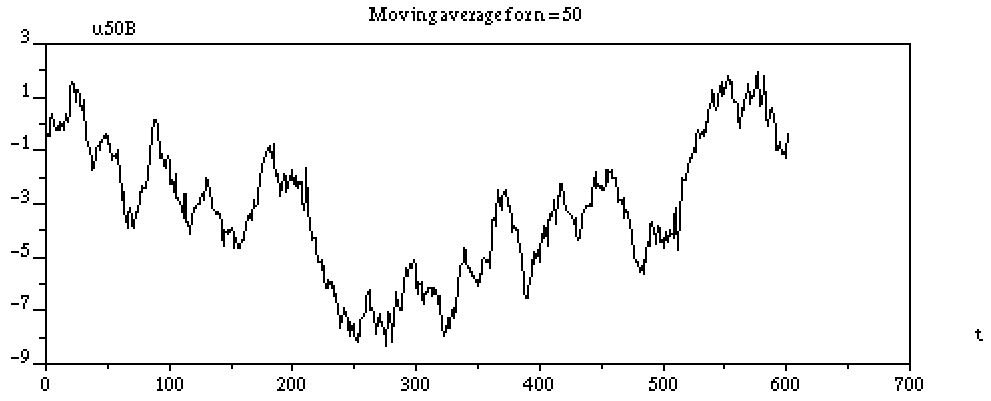
```
uk = uk/k;

//Using convolution
uu  = ones(1,n);
ukp = convol(uu,u)/k;
```

To obtain the moving averages for *n=50*, we first load the function *moving*. The moving averages using the approach that produces a shorter signal are stored in variable *u50A*, while those calculated through convolution are stored in variable *u50B*. Plots of the functions are shown below.

```
-->getf('moving')

-->[u50A,u50B] = moving(u,50);

-->xset('window',1);plot(u50A,'t','u50A','Moving average for n = 50')
```



```
-->xset('window',2);plot(u50B,'t','u50B','Moving average for n = 50')
```

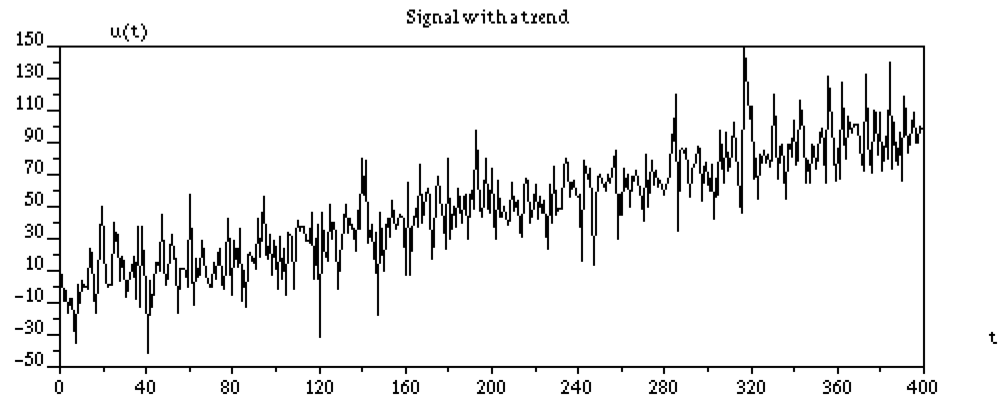


Notice that both approaches produce smoother signals. The first approach, in which a shorter signal is produced, reduced the size of the moving averages to about 250 from the original signal length of 300. The second approach, in which convolution is used, actually increases the size of the moving averages to about twice that of the original signal. This is a characteristic of the discrete convolution operation involved in function *convol*. To produce the first *n* moving averages (in this case *n=50*), the convolution approach “pads” the original signal with zeros thus producing values of doubtful validity. The first approach, in which the actual averages are calculated, will produce more reliable results.

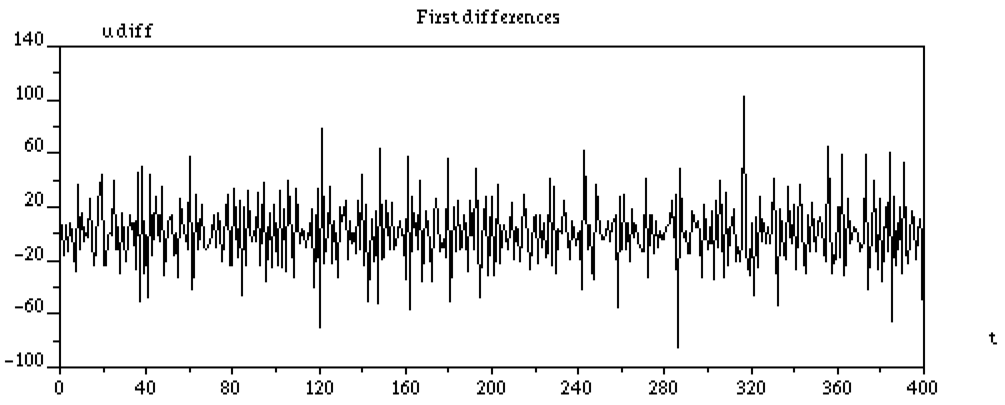
Differencing as convolution

Differencing or taking the first difference of a signal can be obtained as the convolution of the original signal *u* and the vector *v = [1 -1]*. As an example we generate a time series using the function *randf* and calculate the convolution for differencing as follows:

```
-->x=[0:0.1:39.9];u=2.5*x+ranf(x);

-->xset('window',0);plot(u,'t','u(t)','Signal with a trend')
```



```
-->ud=convol(u,[1 -1]);

-->xset('window',1);plot(ud,'t','u_diff','First differences')
```



Removal of seasonal components

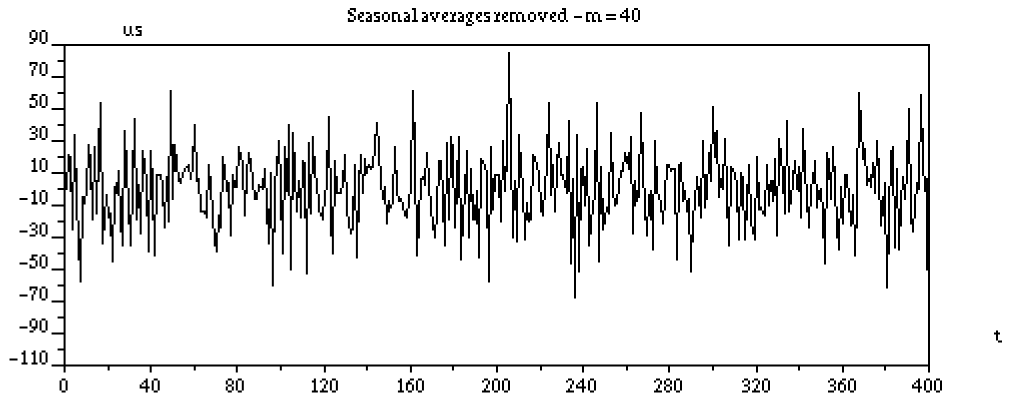
Seasonal components are removed by subtracting from the data the seasonal averages. To facilitate calculations, we use data whose length is a multiple of the seasonal length. For example, if a month represents a season, the number of terms in a season may be 4 if weekly data is used. The "monthly" averages will be calculated for every four data values. If we call *m* the size of the season (e.g., *m=4*), and *n* the length of the signal *(e.g., n=48)*, then the number of seasons will be *s = n/m* (e.g., *s=48/4=12*).

As an example consider the data for the stationary signal *ud* calculated earlier. We will produce the signal *us* after removing seasonal averages assuming that a season has 40 elements. To produce the removal of seasonal components we propose the following function, *removeseason*:

```
function [us] = removeseason(u,m)

n  = length(u);              //length of signal
s  = int(n/m);               //number of seasons in signal
if modulo(n,m)<>0 then       //modify signal if the signal length
   k  = s*m+m-n;             //   is not a multiple of m
   lmean = mean(u(s*m:n))    //mean value for last season
   uu = [u lmean*ones(1,k)]; //extend last season
   s  = s+1;
else
   uu = u;
end;
ur = matrix(uu,m,s)';       //rearrange matrix with seasons in each row
um = mean(ur,'c');          //calculate seasonal mean (by rows)
umean=[]                    //put together matrix with seasonal means
for j = 1:m
    umean = [umean um];
end;
us = ur-umean;                   //subtract seasonal mean from signal
us = matrix(us,1,length(us));    //convert modified signal to a row vector
if modulo(n,m)<>0 then           //reduce resulting signal length if needed
   us = us(1:n);
end;
```

The SCILAB commands required to produced the adjusted signal are as shown below. A plot of the adjusted signal follows.

```
-->us  = removeseason(ud,40);
-->plot(us,'t','us','Seasonal averages removed - m = 40');
```



Spatial data and geostatistics

Of interest to Earth scientists, geologists, hydrologists, etc., is the analysis of spatially distributed data. In this chapter we present the practical operation of contouring for irregularly spaced data. Geostatistics refers to the application of statistical techniques to the analysis of spatially distributed data. Some geostatistics techniques are also presented in this chapter.

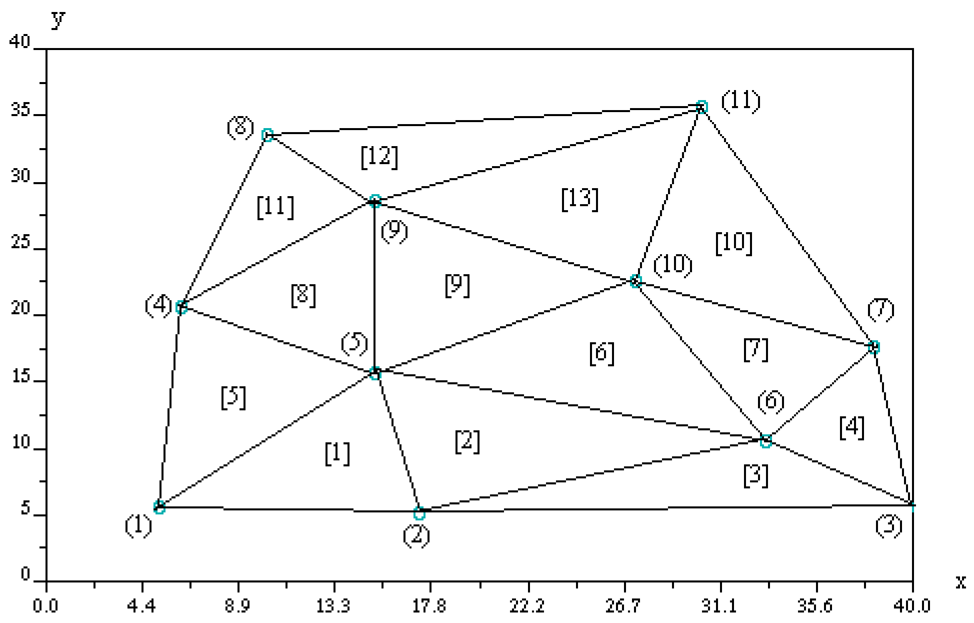
Contouring

Contouring refers to the development of contour plots in a two-dimensional area given an irregularly located set of data points. SCILAB provides function *fac* to produce a contour plot for such type of data distribution. Details for the operation of function *fac* can be obtained by using the help facility in SCILAB, i.e., `-->help fac`

The following function, *contouring*, uses function *fac* but it includes a contouring scale and provides additional information on the graph. The function can be used to generate contour plots of irregularly spaced data. The input to the function consists of vectors *x, y,* and *z,* all of length *n* representing the coordinates of points in space. The matrix *triang* has *m* rows and 4 columns. The first column of *triang* represents an identification number for the triangles into which the contouring area is divided. The three remaining columns of *triang* contain the indices of the vertices included in each of the triangles. The vector *zinfo* has the form *[zmin,Dz,zmax]* representing the minimum, increment, and maximum values of the contours to be plotted. The function returns a contour plot.

```
function [] = contouring(x,y,z,triang,zinfo)

zmin=zinfo(1);
Dz=zinfo(2);
zmax=zinfo(3);
N=(zmax-zmin)/Dz+1;
[n m] = size(triang);
triang = [triang ones(n,1)];
xbasc()


// use the upper part of graphic window for contour plot
xsetech([0,0,1.0,3/4])
xset('foreground',1);
xset("colormap",hotcolormap(N));

fec(x,y,T,z);                  //SCILAB contouring function
plot2d(x,y,-1);

for j = 1:length(z), xstring(x(j),y(j),string(z(j))); end;
xtitle('Contouring','x','y');

// second picture to show the contouring scale
xsetech([0,3/4,1,1/5]) ; // use the lower part of graphic window
Matplot((1:xget("lastpattern")),"010",[0,0.5,N+2,1.5],[1,N+2,1,1]);

for j=0:N+2
       xstring(j,0.1,string(zmin+j*Dz));
end;

xtitle('Contouring scale');
```

As an example of contouring, consider the figure below including 11 vertices and 13 triangles for contouring.



* vertices numbers within parentheses *triangle numbers within square brackets

The following vertex data giving the coordinates of the vertices of a region to be contoured:

x	*y*	*z*
5.	*5.*	*20.5*
17.	*4.5*	*21.2*
40.	*5.*	*15.5*
6.	*20.*	*10.6*
15.	*15.*	*22.3*
33.	*10.*	*20.3*
38.	*17.*	*19.2*
10.	*33.*	*17.5*
15.	*28.*	*23.5*
27.	*22.*	*24.8*
30.	*35.*	*20.2*

These values will be entered into SCILAB as follows:

```
-->x=[5,17,40,6,15,33,38,10,15,27,30];

-->y=[5,4.5,5,20,15,0,17,33,28,22,35];

-->z=[20.5,21.2,15.5,10.6,22.3,20.3,19.2,17.5,23.5,24.8,20.2]
```

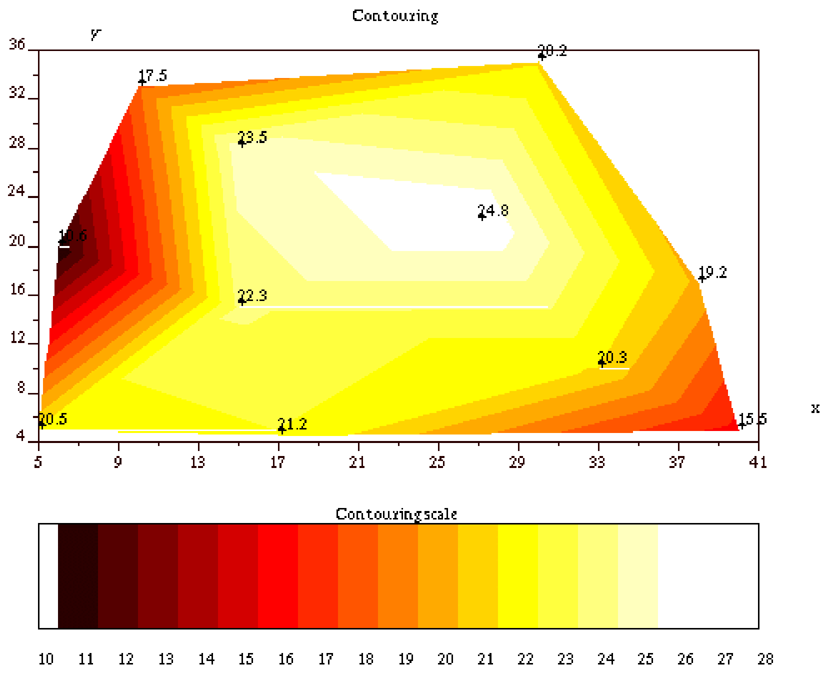
The following matrix, *T*, includes the information regarding the triangles (i.e., matrix *triang*). The first column represents the triangle number, and the remaining three columns represent the vertices of each triangle.

```
-->T  =

  [1.     1.     5.     2...
   2.     2.     5.     6...
```

```
    3.      2.      6.      3...
    4.      6.      7.      3...
    5.      1.      4.      5...
    6.      5.      10.     6...
    7.      6.      10.     7...
    8.      4.      9.      5...
    9.      5.      9.      10...
    10.     10.     11.     7...
    11.     4.      8.      9...
    12.     9.      8.      11...
    13.     10.     9.      11];
```

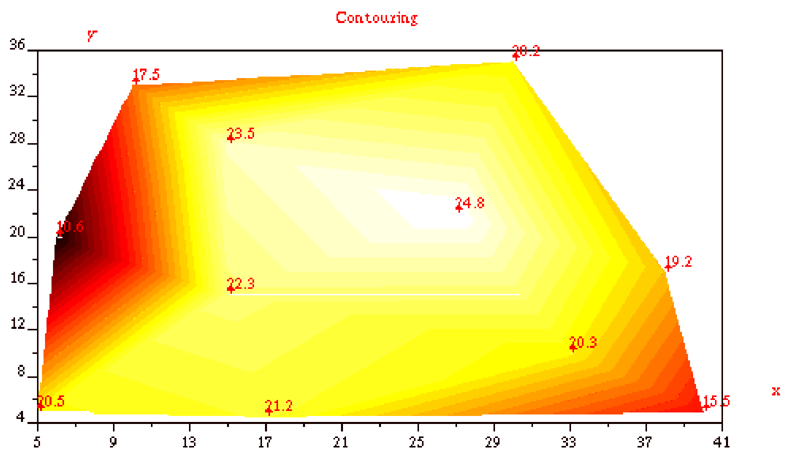
Next, we load function *contouring* and call it using the data values *x, y, z,* and *T*. We also include in the call to the function the vector indicating the minimum value of *z* for the contours, the increment in the contour values, and the final value of *z* for the contours. For example, in the first cased illustrated below $z_{min} = 10$, $\Delta z = 1$, and $z_{max} = 25$.

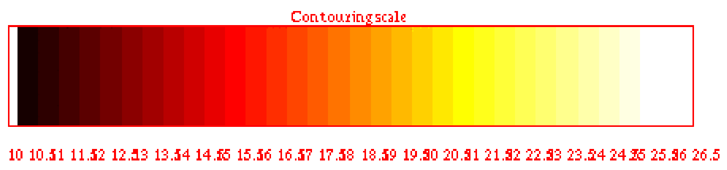
```
-->getf('contouring')

-->contouring(x,y,z,T,[10,1,25])
```



In the next example we use the same *x, y, z,* and *T* data, but change the description of the contours to $z_{min} = 10$, $\Delta z = 0.5$, and $z_{max} = 25$.

```
-->contouring(x,y,z,T,[10,0.5,25])
```





Trend identification

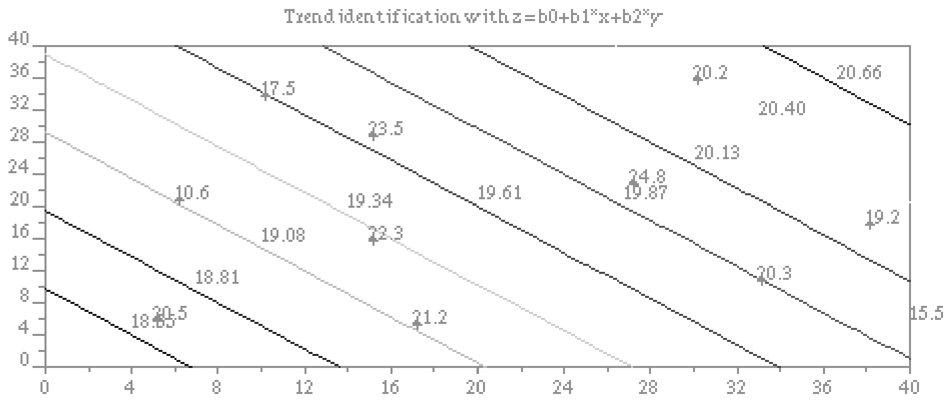
Trend identification, or trend analysis, in spatial data distribution is nothing but data fitting using a variety of functions of positions (x,y). We can obtain data fitting to almost any function by using SCILAB's own *datafit* function. (Function *datafit* was described in detail in Chapters 8 and 17.) In this section we use it to fit functions of the forms:

$f(x,y) = b_0 + b_1x + b_2y$
$f(x,y) = b_0 + b_1x + b_2y + b_3xy$
$f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$

to the data used in the previous example for contouring. The general procedure used consists on defining function *G(b,z)* as required by function *datafit*, an initial value of the coefficients, *b0*, and using function *datafit* to determine the coefficients of the fitting. After that step is completed, we proceed to define the appropriate function *f(x,y)*, and to evaluate it an a regular grid, $0 < x < 40$, $0 < y < 40$. The x- and y-grids are stored in variables *xx* and *yy*, respectively. Function *feval* is used to calculate the fitted values of *z* into matrix *zz*. SCILAB's function *contour* is used to produce contour plots of the fitted data.

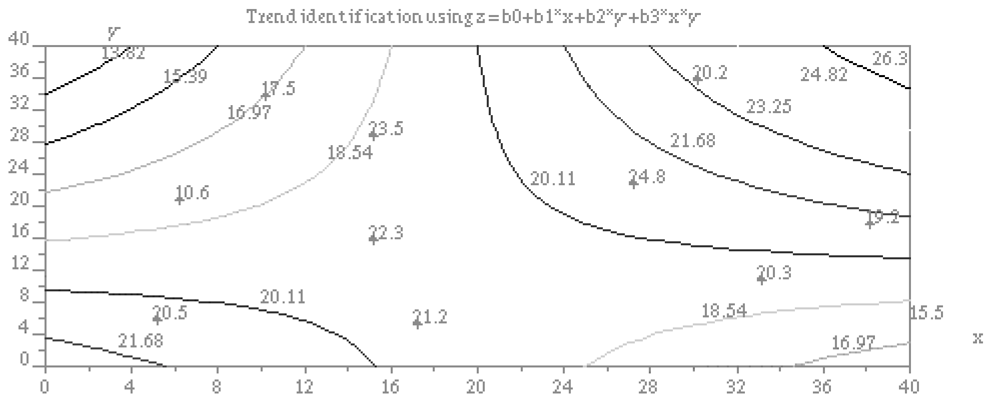
Case1. Linear function $f(x,y) = b_0 + b_1x + b_2y$

```
-->Z = [x;y;z];     //This definition applies to the three cases
-->deff('[e]=G(b,z)','e=z(3)-b(1)-b(2)*z(1)-b(3)*z(2)')
-->b0=[1;1;1];
-->[b,err]=datafit(G,Z,b0)
 err  =  153.14402


 b  =

!   18.285047 !
!    .0389328 !
!    .0271277 !

-->deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y')

-->xx=[0:2.5:40];yy=xx;zz=feval(xx,yy,f);
-->contour(xx,yy,zz,10)
-->plot2d(x,y,-1,'010',' ',[0 0 40 40])
-->for j=1:11, xstring(x(j),y(j),string(z(j))); end;
-->xtitle('Trend identification with z = b0+b1*x+b2*y')
```



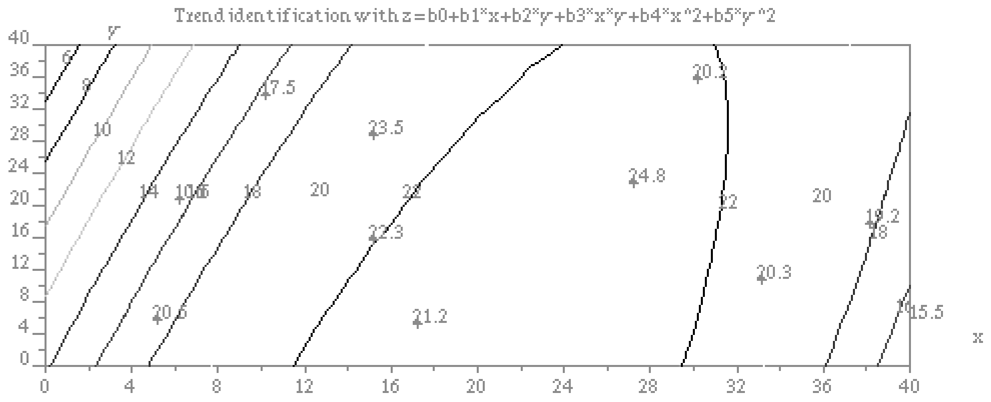
Case 2. Including one non-linear term, $f(x,y) = b_0 + b_1x + b_2y + b_3xy$

```
-->deff('[e]=G(b,z)','e=z(3)-b(1)-b(2)*z(1)-b(3)*z(2)-b(4)*z(1)*z(2)')

-->b0=[1;1;1;1];

-->[b,err]=datafit(G,Z,b0)
 err  =

    124.19016
 b  =

!   22.567309 !
! -  .1616097 !
! -  .2578275 !
!    .0138546 !

-->deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y+b(4)*x*y')
-->xx=[0:2.5:40];yy=xx;zz=feval(xx,yy,f);
-->contour(xx,yy,zz,10);
-->for j=1:11, xstring(x(j),y(j),string(z(j))); end;
-->plot2d(x,y,-1,'011',' ',[0 0 40 40])
-->xtitle('Trend identification using z = b0+b1*x+b2*y+b3*x*y','x','y')
```



Case 3. Quadratic relationship, $f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$

```
-->deff('[e]=G(b,z)','e=z(3)-b(1)-b(2)*z(1)-b(3)*z(2)-b(4)*z(1)*z(2)-
b(5)*z(1)^2-b(6)*z(2)^2')

-->b0=[1;1;1;1;1;1];

-->[b,err]=datafit(G,Z,b0)
 err  =
    58.673471
 b  =
!   13.775238 !
!   1.0137698 !
! -  .2015857 !
!    .0086525 !
! -  .0247882 !
! -  .0011964 !

-->deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y+b(4)*x*y+b(5)*x^2+b(6)*y^2')

-->xx=[0:2.5:40];yy=xx;zz=feval(xx,yy,f);
-->contour(xx,yy,zz,10)
-->plot2d(x,y,-1,'010',' ',[0 0 40 40])
-->for j=1:11, xstring(x(j),y(j),string(z(j))); end;
-->xtitle('Trend identification with z = b0+b1*x+b2*y+b3*x*y+b4*x^2+b5*y^2','x','y')
```



Based purely on the size of the error incurred in the three models, it seems that the quadratic fitting, namely, $f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$, produces the best result. Notice also that the quadratic function is the closest to the contour pattern found earlier using function *contouring*. The user must realize that the identification of trends is only an approximation to the actual distribution of the functions analyzed; therefore, the resulting data fittings must be used very carefully.

A function for trend identification and removal

The following function, *trend2d*, provides a way to select one out of the three models used earlier, namely, $f(x,y) = b_0 + b_1x + b_2y$, $f(x,y) = b_0 + b_1x + b_2y + b_3xy$, or $f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$, to produce a surface fitting. The function takes an input the row vectors *x*, *y*, and *u*, all of the same length. The function is interactive requesting input from the user in terms of selecting the fitting to use. It also requests input from the user regarding whether or not the values of the signal, *u(x,y)*, should be printed in the contour plot that the function produces. The function returns as output the signal *ud* equal to the original signal minus the corresponding fitted values. The general call to the function is

[ud,er] = trend2d(x,y,u)

A listing of the function follows:

```
function [ud,er] = trend2d(x,y,u)

//Check model to use
printf(' ');
printf('Models available for trend identification:')
printf('1 - u(x,y) = b(1)+b(2)*x+b(3)*y')
printf('2 - u(x,y) = b(1)+b(2)*x+b(3)*y+b(4)*x*y')
printf('3 - u(x,y) = b(1)+b(2)*x+b(3)*y+b(4)*x*y+b(5)*x^2+b(6)*y^2')
idmod = input('Select the model to fit:')
printf(' ');

if idmod <> 1 & idmod <> 2 & idmod <> 3 then
   error('trend2d - model id number must be 1, 2, or 3');
   abort;
end;

//Prepare data for data fitting
X = [x;y;u];
xmin = min(x); xmax = max(x);xrange = xmax-xmin;
xmin = xmin-xrange/10;xmax = xmax+xrange/10;
xrange = xmax-xmin; xx = [xmin:xrange/20:xmax];
ymin = min(y); ymax = max(y);yrange = ymax-ymin;
ymin = ymin-yrange/10;ymax = ymax+yrange/10;
yrange = ymax-ymin; yy = [ymin:yrange/20:ymax];
rect = [xmin ymin xmax ymax];

//Obtain parameters for selected data fitting
if idmod == 1 then
   deff('[e]=G(b,z)',...
   'e=b(1)+b(2)*z(1)+b(3)*z(2)-z(3)');
   b0 = [1;1;1];
   [b,er] = datafit(G,X,b0);
   title = 'Linear model, u = b1+b2*x+b3*y';
   deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y');
elseif idmod == 2 then
   deff('[e]=G(b,z)',...
   'e=b(1)+b(2)*z(1)+b(3)*z(2)+b(4)*z(1)*z(2)-z(3)');
   b0 = [1;1;1;1];
   [b,er] = datafit(G,X,b0);
   title = 'Model with one non-linear term, u = b1+b2*x+b3*y+b4*x*y';
   deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y+b(4)*x*y');
```

```
else
   deff('[e]=G(b,z)',...
   'e=b(1)+b(2)*z(1)+b(3)*z(2)+b(4)*z(1)*z(2)+...
      b(5)*z(1)^2+b(6)*z(2)^2-z(3)');
   b0 = [1;1;1;1;1;1];
   [b,er] = datafit(G,X,b0);
   title = 'Quadratic model, u = b1+b2*x+b3*y+b4*x*y+b5*x^2+b6*y^2';
   deff('[w]=f(x,y)','w=b(1)+b(2)*x+b(3)*y+b(4)*x*y+b(5)*x^2+b(6)*y^2');
end;

//Plot contour plots of signal
uu = feval(xx,yy,f);
xset('window',1);
xbasc();
xset('mark',-1,1);
plot2d(x,y,-1,'011',' ',rect);
xtitle(title,'x','y');
contour(xx,yy,uu,10);
printf(' ');
answer = input(...
'Plot original points in the original contour plot? y/n:',...
'string');
if answer == 'y' then
   for j=1:length(x)
    xstring(x(j),y(j),' '+string(u(j)),0,1);
   end;
end;

//Remove trend
ud = zeros(1,length(u));
for j =1:length(ud)
    ud(j) = u(j)-f(x(j),y(j));
end;
```

As an example of application of function *trend2d*, consider the following spatial signal *u* = *u(x,y)*:

x	*y*	*u*
20	*120*	*25.7*
10	*80*	*37.8*
35	*90*	*39.0*
50	*120*	*23.0*
72	*122*	*27.0*
67	*88*	*42.0*
40	*75*	*36.0*
70	*60*	*47.5*
42	*46*	*42.5*
20	*60*	*37.0*

First, we load the data into SCILAB:

```
-->x=[20,10,35,50,72,67,40,70,42,20];
-->y=[120,80,90,120,122,88,75,60,46,60];
-->u=[25.7,37.8,39.0,23.0,27.0,42.0,36.0,47.5,42.5,37.0];
```

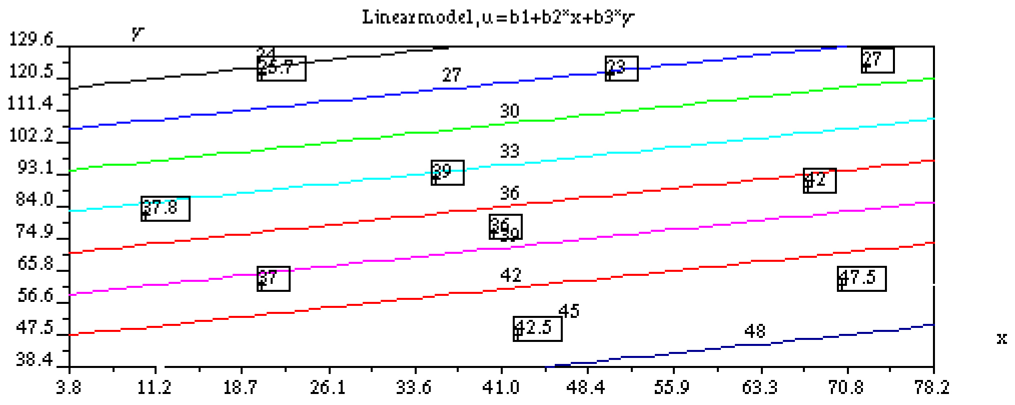
Next, we use function *trend2d* to produce the fitting using each of the three models:

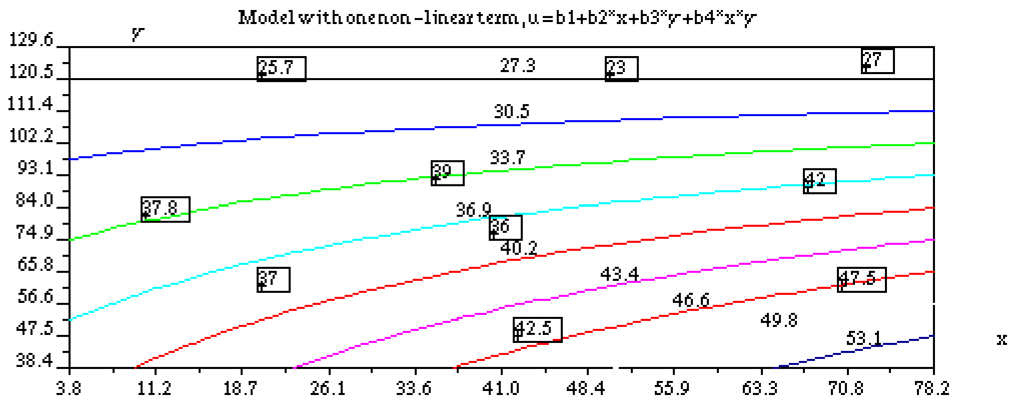
```
-->[ud,er]=trend2d(x,y,u)
```

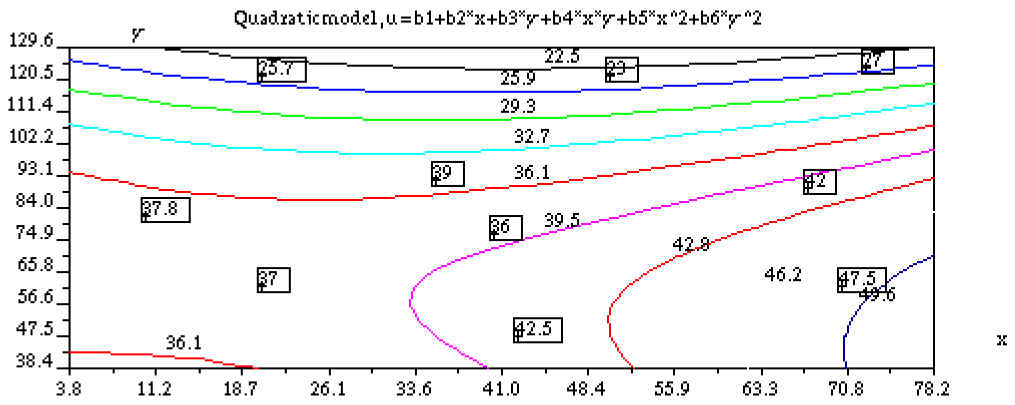
The fitting mean square errors (MSE) for each of the fitting models for this case are:

```
Model 1 - error  = 120.1648
Model 2 - error  =  99.508868
Model 3 - error  =  31.133571
```

Contour plots for the three fitting functions are shown next:







Spatial autocorrelation - the semivariogram

In the analysis of the dependency of two variables x,y, given a data set $\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$, the *semivariogram* of the data set is defined as

$$\gamma_{XY} = \frac{1}{2n}\sum_{i=1}^{n}(x_i - y_i)^2.$$

Let $\overline{x}$ and $\overline{y}$ be the mean values of the vectors x and y, s_X and s_Y, the standard deviation of the vectors x and y, and s_{XY} the covariance of the data. You can prove that the semivariogram is related to these statistics by

$$2\gamma_{XY} = s_X^2 + s_Y^2 + (\overline{x} - \overline{y})^2 - 2s_{XY}.$$

Similar to the case of time-dependent signals, spatial data tend to be autocorrelated. A measure of spatial autocorrelation is a function called the *semivariogram*. Consider a spatial series u_i, $i=1,2,\ldots,n$, so that $u_i = u(\mathbf{x}_i)$, where $\mathbf{x}$ represents a spatial variable, with values of $\mathbf{x}$ equally spaced by an amount Δx. The semivariogram of the signal $u(\mathbf{x})$ for a spatial lag $\xi = k\cdot\Delta x$ is given by

$$\gamma(\xi) = \frac{1}{2A(\xi)}\int_{A\cap A_{-\xi}}[u(\mathbf{x}) - u(\mathbf{x}+\boldsymbol{\xi})]^2\,dx,$$

where $A\cap A_{-\xi}$ is the intersection of a region A and the translated region $A_{-\xi}$, so that if $\mathbf{x}$ is contained in the intersection, $A\cap A_{-\xi}$, then both $\mathbf{x}$ and $\mathbf{x}+\boldsymbol{\xi}$ are in A, and $A(\xi)$ is the measure of the intersection (i.e., an area or volume). Implied in this definition is the fact that $\mathbf{x}$ and $\boldsymbol{\xi}$ are vectors of one, two, or three dimensions.

For a discrete signal, along a single direction, the semivariogram corresponding to a spatial lag $\xi = k\cdot\Delta x$, i.e., a lag of k positions, is given by

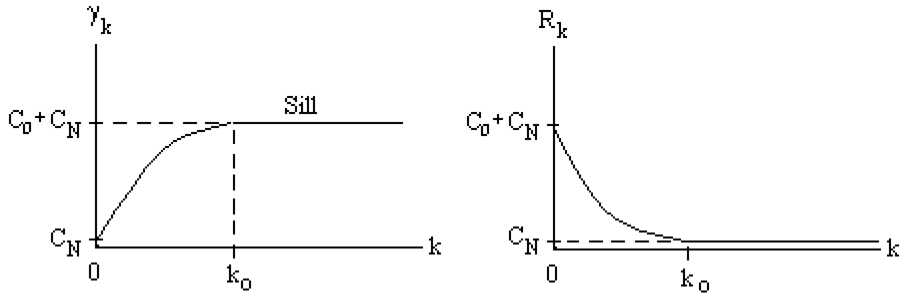
$$\gamma_k = \frac{1}{2n}\sum_{i=1}^{n-k}(u_i - u_{i+k})^2.$$

The same expression applies to an *isotropic* (independent of direction) signal distributed in a plane or in space.

The semivariogram will be related to the (spatial) autocovariance R_k by

$$\gamma_k = (C_0 + C_N) - R_k,$$

where C_0 and C_N are constant values. The figure below shows a typical *semivariogram.* The semivariogram reaches the constant value $\gamma_k = C_0 + C_N$ for very large values of the lag k, starting at a value $k=k_0$. The value $C_0 + C_N$ is referred to as *the sill* of the variogram. For a given variogram, thus, the autocovariance $R_k = C_0 + C_N - \gamma_k$ is a decreasing value of the lag, k. The variation of the autocovariance is also illustrated in the figure below.



In theory, the semivariogram should be zero at zero lag, i.e., $\gamma_0 = 0$, or $C_N = 0$. In practice, however, we allow for the existence of a small value, C_N, referred to as the *nugget value*.

Calculation of the experimental semivariogram

The practical calculation of the semivariogram in an isotropic field is outlined by Kitanidis(op. cit.) as shown next. It is assumed that we have a series of *n* measurements $\{u_1(\mathbf{x}_1), u_2(\mathbf{x}_2), \dots, u_n(\mathbf{x}_n)\}$, where $\mathbf{x}$ represents a vector or array of coordinates where the data was collected, e.g., $\mathbf{x}_1 = (x_1,y_1)$, etc. The steps in the determination of the semivariogram are the following:

1. Plot the $n(n-1)/2$ squared differences, $\Delta u_{ij} = (u(\mathbf{x}_i)-u(\mathbf{x}_j))^2/2$, against the distances $k_{ij} = ||\mathbf{x}_i-\mathbf{x}_j||$ (i≠j). The resulting figure is known as the *raw variogram*.

2. Determine the minimum and maximum value of the distances, *k*, and produce a series of classes of values of *k*, similar to the classes used in the development of frequency distributions (Chapter 13). Say, that you use *N* classes whose class boundaries are given by:

$$kb = [kb_1, kb_2, \dots, kb_{N+1}].$$

3. Compute a value of the variogram, $\hat{\gamma}(k_m)$, for class k_m, as

$$\hat{\gamma}(k_m) = \frac{1}{2M_m}\sum_{i=1}^{M_m}[u(\mathbf{x}_i) - u(\mathbf{x}'_i)]^2,$$

where M_k is the number of distances $k_i = ||\mathbf{x}_i-\mathbf{x}'_j||$ contained within class number *m*, i.e., $kb_i \le ||\mathbf{x}_i-\mathbf{x}'_j|| < kb_{i+1}$. The value of k_m is taken as the average of the M_m distances $k_i = ||\mathbf{x}_i-\mathbf{x}'_j||$ contained within class number *m*:

$$k_m = \frac{1}{M_m}\sum_{i=1}^{M_m}\|\mathbf{x}_i - \mathbf{x}'_i\|.$$

4. Plot the values $(k_m, \hat{\gamma}(k_m))$, $m=1,2,\dots,N$, and join these points to produce what is called the *experimental semivariogram*.

Notes:

(1) The value of k_m can be taken also to be the median of the M_m distances k_i = $||\mathbf{x}_i\text{-}\mathbf{x'}_j||$ contained within class number m, or simply as the class mark for class number m, i.e., $k_m = (kb_m+kb_{m+1})/2$.

(2) The algorithm outlined above is for an isotropic medium. Anisotropic applications require to take into account the direction in which the distances are measured.

A function to calculate the experimental semivariogram

The following function, *semivariogram*, implements Kitanidis' algorithm to calculate the semivariogram for data distributed in a plane. The function takes as input vectors of values of *x, y,* and *u*, and returns the set of values of *k* and *Du* computed in the first step of the algorithm, as well as the semivariogram data $(kk,gam) = (k_m, \hat{\gamma}(k_m))$. The function is interactive, in the sense that the user needs to provide a vector with the class boundaries for *k* after the function has determined the minimum and maximum values of *k* from step 1 in the algorithm. The function also produces a plot showing the values (*k,Du*) as discrete points, and the values (*kk,gam*) as a continuous line.

```
function [k,Du,kk,gam]=semivariogram(x,y,u)
//First step: calculate distances and differences of u squared
n = length(x);
X = [x' y'];
P = n*(n-1)/2;
Du= zeros(1,P);
k = zeros(1,P);
l = 0
for i = 1:n
    for j = i+1:n
        l = l+1;
        Du(l) = (u(i)-u(j))^2;
        k(l)  = norm(X(i,:)-X(j,:));
    end;
end;

//Request class boundary information from user:
kmin = min(k);kmax = max(k);
disp(' ');
printf('Values of minimum and maximum distances = %g %g',kmin,kmax);
kb = input('Enter vector of class boundaries for k:');

//Obtain frequency count and semivariogram information
N  = length(kb)-1;
M   = zeros(1,N);
kk  = zeros(1,N);
gam = zeros(1,N);

for i = 1:P
    for j = 1:N
        if k(i)>=kb(j) & k(i)<kb(j+1) then
           M(j)  = M(j)+1;
           kk(j) = kk(j) + k(i);
           gam(j)= gam(j) + Du(i);
        end;
    end;
end;

//Calculate semivariogram
NN=0;
for j = 1:N
    if M(j) <> 0
       NN = NN + 1;
       kk(j) = kk(j)/M(j);
       gam(j)= gam(j)/M(j);
    end
end;

//Modify semivariogram data eliminating empty classes
if NN<>N then
   gamm = [];kkt = [];
   for j = 1:N
       if M(j)<>0 then
          gamm = [gamm gam(j)];
          kkt  = [kkt  kk(j)];
       end;
   end;
   gam = gamm;kk = kkt;
end;

//Plot semivariogram data
gmin = min([gam Du]); gmax = max([gam Du]);
rect=[0 0 kmax gmax];
xset('window',1);xset('mark',-1,1);
plot2d(k,Du,-1,'011',' ',rect);
plot2d(kk,gam,1,'011',' ',rect);
xtitle('Semivariogram','k','gamma(k)');
```

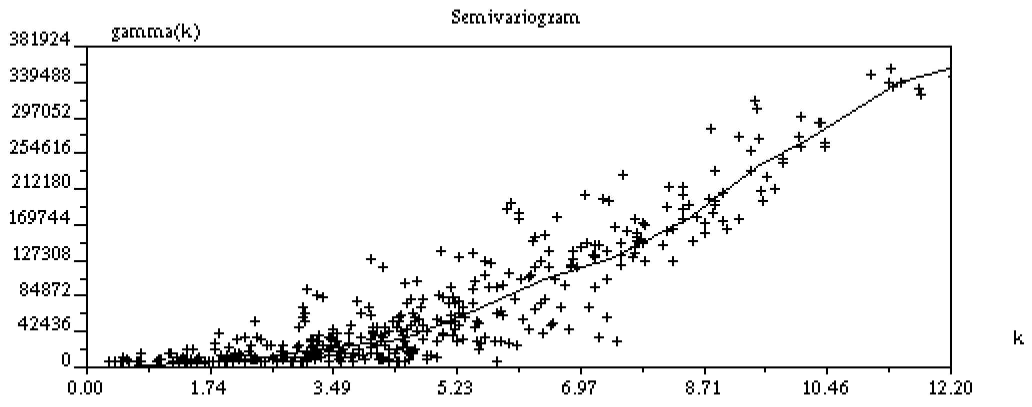
The following script loads data taken from Kitanidis, P.K.,1997, "*Introduction to Geostatistics - Applications in Hydrogeology*," Cambridge University Press, Cambridge CB2 1RP, UK. The first set of data corresponds to position (x,y) and water table elevation (u), all in *ft*, in an aquifer. The second set of data corresponds to position (X,Y), in *km*, and aquifer transmissivity, in m^2/day. The data is loaded using the following SCILAB script:

```
x=[6.86,4.32,5.98,11.61,5.52,10.87,8.61,12.64,14.70,13.91,9.47,...
  14.36,8.99,11.93,11.75,13.23,13.56,8.06,10.95,14.71,16.27,...
  12.33,13.01,13.56,13.76,12.54,8.97,9.22,9.64];

y=[6.41,5.02,6.01,4.99,3.79,8.27,3.92,6.77,10.43,10.91,5.62,...
  11.03,7.31,6.78,10.8,10.18,9.74,5.76,3.72,11.41,7.27,6.87,...
   7.05,7.42,8.35,9.04,8.6,8.55,3.38];

u = [1061,1194,1117, 880,1202, 757,1038, 817, 630, 617, 986,...
      625, 840, 847, 645, 662, 685,1023, 998, 584, 611, 847, 745,...
      725, 688, 676, 768, 782,1022];


X=[0.876 0.188 2.716 2.717 3.739 1.534 2.078 3.324];
Y=[0.138 0.214 2.119 2.685 0.031 1.534 0.267 1.670];
U=[2.9   2.5   4.7   4.2   4.2   2.1   2.4   5.8  ];
```

A plot of the semi-variogram for the first data set is obtained by using:

```
-->exec('c:\myScilab\LoadXYU')

(output removed)
```

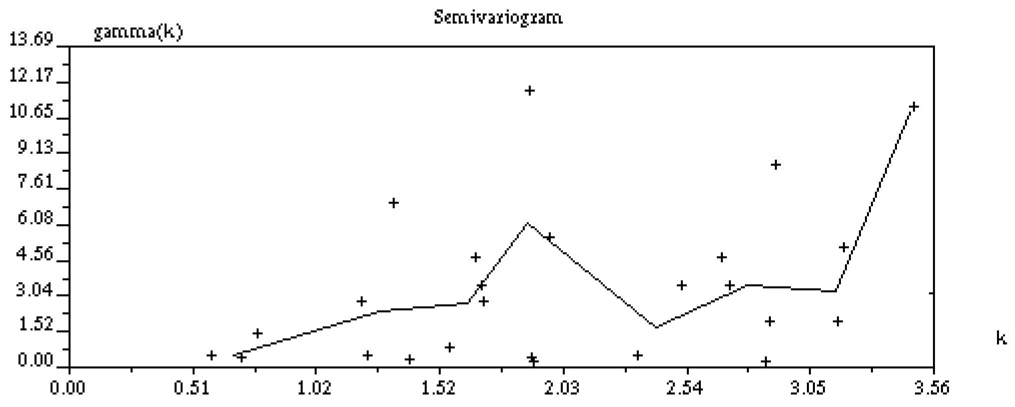
```
-->[k,Du,kk,gam]=semivariogram(x,y,u);

Values of minimum and maximum distances =  .25495 12.1977

 Enter vector of class boundaries for k:
```
[0:1:13];



A plot of the semi-variogram for the second data set is obtained by using:

```
-->[k,Du,kk,gam]=semivariogram(X,Y,U);

Values of minimum and maximum distances =  .56600 3.55571

 Enter vector of class boundaries for k:
```

[0.5:0.3:3.7];



Semivariogram models

Semivariograms obtained from real data can be written as linear combinations of a number of basic semivariogram models. For a stationary function, i.e., one whose mean value is constant and whose two-point covariance function depends only on the distance between the two points, we can use the following semivariogram models [see Middleton, G.V., 2000, "*Data Analysis in the Earth Sciences Using Matlab*", Prentice Hall, Upper Saddle River, New Jersey, or

Kitanidis, P.K.,1997, "*Introduction to Geostatistics - Applications in Hydrogeology*," Cambridge University Press, Cambridge CB2 1RP, UK]:

The spherical model

$$\gamma_k = \begin{cases} \left(\frac{3k}{2k_0} - \frac{k^3}{2k_0^3}\right) \cdot C_0, 0 \le k \le k_0 \\ C_0, \qquad k > k_0 \end{cases}$$

Here, $C_0 = \sigma^2$, is a variance, and k_0 is referred to as the *range*.

The exponential model

$$\gamma_k = \begin{cases} \left(1 - \exp(-\frac{|k|}{\mu}\right) \cdot C_0, k > 0 \\ 0, \qquad otherwise \end{cases}$$

Here, μ is an integral scale parameter, $C_0 = \sigma^2$, and the range is $k_0 \approx 3\mu$.

The Gaussian model

$$\gamma_k = \begin{cases} \left(1 - \exp\left(-\left(\frac{k}{L}\right)^2\right)\right) \cdot C_0, k > 0 \\ 0, \qquad otherwise \end{cases}$$

Here, again, $C_0 = \sigma^2$, and the range is defined as the distance at which the correlation is 0.05, i.e., $k_0 \approx 7L/4$.

The hole-effect model

$$\gamma_k = \begin{cases} \left(1 - \frac{k}{L}\right) \exp\left(-\frac{k}{L}\right) \cdot C_0, k > 0 \\ 0, \qquad otherwise \end{cases}$$

Once more, $C_0 = \sigma^2$. This model is commonly used for one-dimensional functions in hydrology.

The nugget-effect model

$$\gamma_k = \begin{cases} (1 - \delta(k)) \cdot C_0, k > 0 \\ 0, \qquad otherwise \end{cases}$$

where C_0 is the nugget variance, and $\delta(k)$ is the Kronecker delta function. The nugget effect is named after mining applications where the distribution of mineral tends to be in the form of discrete nuggets rather than a continuous distribution.

For non-stationary functions, i.e., those whose semivariogram grows unbounded as the distance *k* grows unbounded, the following semivariogram models are available:

The power model

$$\gamma_k = \theta \cdot k^r,$$

where $\theta > 0$ and $0 < r < 2$. This model is used to describe self-similar functions.

The linear model

$$\gamma_k = \theta \cdot k,$$

a special case of the power model ($r=1$), containing only one parameter, θ.

The logarithmic model

$$\gamma_k = A \ln(k),$$

where $A>0$. This model is typically used on finite volumes.

Linear combinations of the semivariogram models are typically used to fit data from the experimental semivariogram. For example, a combination of the nugget-effect and exponential models will produce a semivariogram defined by

$$\gamma_k = \begin{cases} C_N + \left(1 - \frac{k}{L}\right)\exp\left(-\frac{k}{L}\right) \cdot C_0, k > 0 \\ 0, \quad otherwise \end{cases}$$

Once we have selected a particular model, or combination of models, to fit a particular experimental semivariogram, we can use function *datafit* (see Chapters 8 and 18) to obtain the unknown parameters in the model. For details on the procedures used in practice to obtain a semivariogram function out of the experimental semivariogram the reader is referred to Kitanidis (op. cit., Chapter 4). The following example shows a simple application using the spatial signal *u(x,y)* described in the following table (this data set was used earlier in applications of function *trend2d* for trend identification and trend removal):

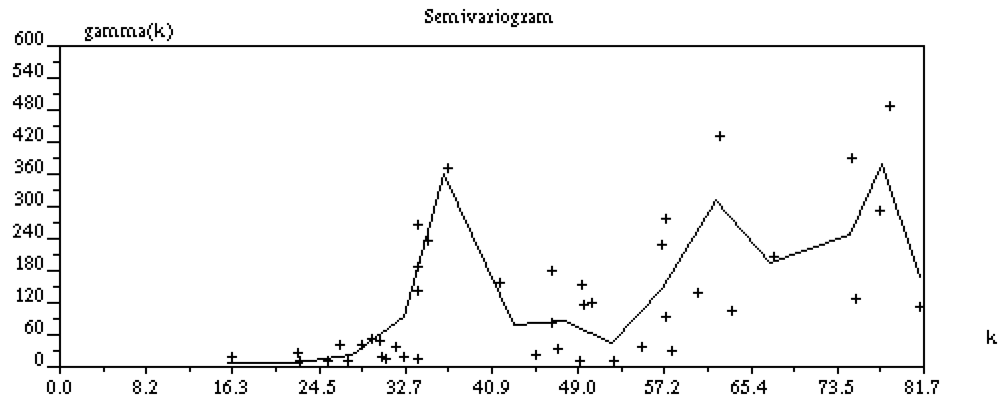
x	*y*	*u*
20	120	25.7
10	80	37.8
35	90	39.0
50	120	23.0
72	122	27.0
67	88	42.0
40	75	36.0
70	60	47.5
42	46	42.5
20	60	37.0

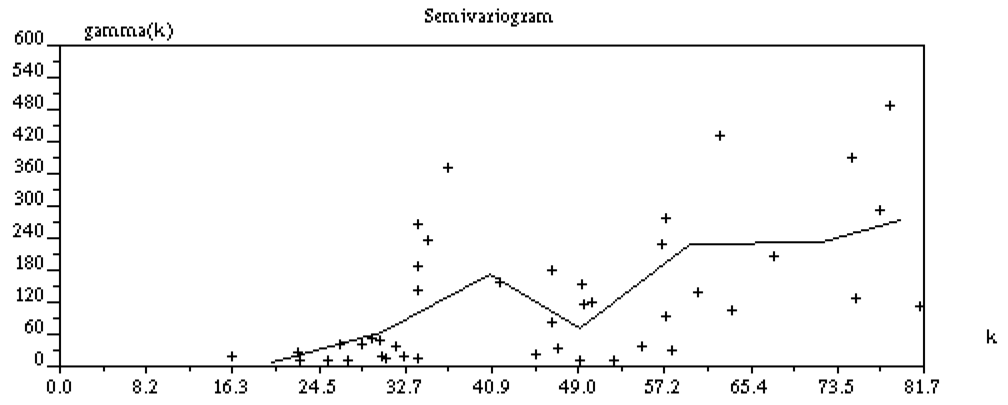
The following four figures correspond to the experimental semivariogram for distance classes *k* identified as follows:

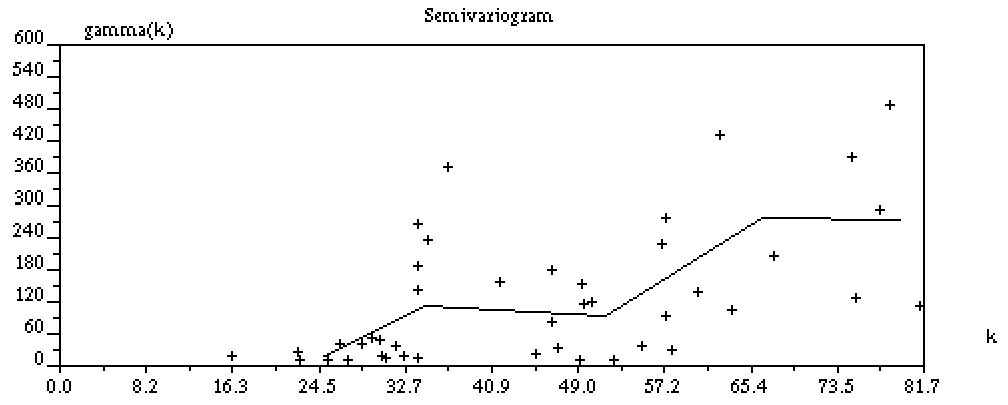
Case 1: [15,20,25,…,85]
Case 2: [15,25,35,…,85]
Case 3: [15,30,45,…,90]
Case 4: [15,35,55,…,95]

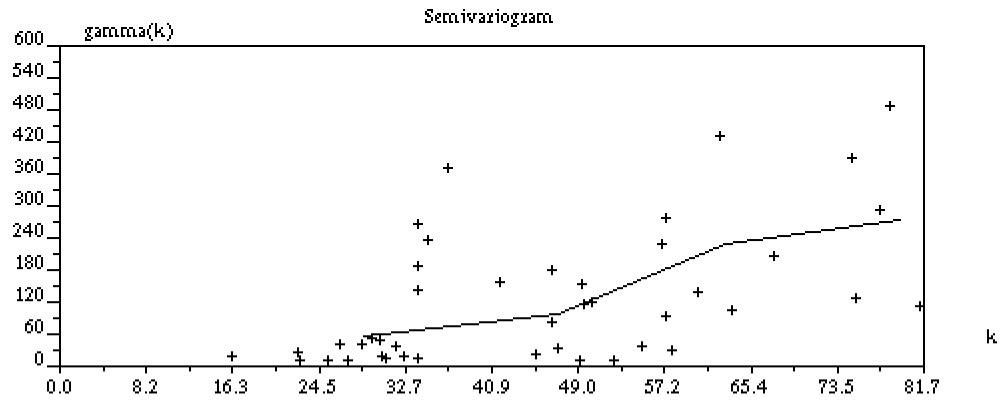
These figures are the result of using function *semivariogram*, i.e.,

```
-->[k,Du,kk,gam] = semivariogram(x,y,u);
```









The figures suggest a linear model for the variogram. To determine the linear model corresponding to each of the four figures above, we use function *linreg*, developed in Chapter 17. The following table summarizes the results of using function *linreg*:

Case	m	b	$\overline{k}$	$\overline{\gamma}$	s_k	s_γ	$s_{k\gamma}$	$r_{k\gamma}$	s_e
1	3.80	-35.24	49.78	153.95	21.20	128.26	1708.20	0.63	103.86
2	4.24	-62.74	50.23	150.42	21.65	101.58	1989.24	0.90	47.45
3	4.75	-88.72	51.41	155.50	22.32	115.10	2367.04	0.92	51.69
4	4.66	-90.09	54.57	164.19	21.71	104.13	2195.76	0.97	30.27

The columns represent m = slope of linear fitting, b = intercept of linear fitting, $\overline{k}$ = mean value of distances, $\overline{\gamma}$ = mean value of semivariograms, s_k = standard deviation of distances, s_γ = standard deviation of semivariogram, $s_{k\gamma}$ = covariance of k and γ, $r_{k\gamma}$ = correlation coefficient, s_e = standard error of estimate. The correlation coefficient for the first last three cases is in the range 0.90-0.97 indicating good correlation for a linear fitting. The main drawback of the fittings is that the intercept is negative. We may just keep the slope and force the intercept to zero, i.e., $\gamma(k) = m\cdot k$. We notice that the slopes are very similar. We could select, for example, the average of the three last cases, i.e., $m = 4.55$, and use the semivariogram model

$$\gamma(k) = 4.55\cdot k.$$

Kriging

Kriging is a method of interpolation of spatial data using the semivariogram. The method was developed first by a South African mining engineer by the name of D. G. Krige, thus the name *kriging*. Predictions by using kriging are of two types: *point kriging*, which consists of interpolating point values, and *block kriging*, consisting of estimating an average value on a spatial region or block. Point kriging uses the semivariogram to predict the best values of certain *weigth factors*, w_i, so that the value of the function of interest, $u(\mathbf{x})$, at a point P, i.e., $u(\mathbf{x}_P)$, is estimated as

$$u_P = \sum_{i=1}^{n} w_i u_i .$$

The kriging system is a linear system of equations that results from applying the method of Lagrange multipliers (see Chapter 12) to the problem of minimizing the mean square error between u_P and $u(\mathbf{x}_P)$, i.e., minimizing the objective function $E[(u_P - u(\mathbf{x}_P))^2]$. The kriging system consists of $n+1$ equations with $n+1$ unknowns:

$$-\sum_{i=1}^{n} w_i \gamma(k_{ij}) + \lambda = -\gamma(d_0), \quad i = 1,2,\dots,n$$

and

$$\sum_{i=1}^{n} w_i = 1.$$

The value λ in the system is a Lagrange multiplier, and the values k_{ij} represent the distances $k_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$, between points *i* and *j*.

Using matrix notation, the kriging system can be rewritten as **A**·**x**=**b**, where

$$A = \begin{bmatrix} 0 & -\gamma(k_{12}) & \cdots & -\gamma(k_{1n}) & 1 \\ -\gamma(k_{21}) & 0 & \cdots & -\gamma(k_{2n}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\gamma(k_{n1}) & -\gamma(k_{n2}) & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix}, \quad x = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \lambda \end{bmatrix}, \quad b = \begin{bmatrix} -\gamma(k_{10}) \\ -\gamma(k_{20}) \\ \vdots \\ -\gamma(k_{n0}) \\ 1 \end{bmatrix}.$$

The solution to the kriging system includes the *n* weigth coefficients used in the calculation of u_P, as well as the Lagrange multiplier λ, which is used to calculate the variance of the error,

$$s_e^2 = \sum_{i=1}^{n} w_i \gamma(k_{i0}) + \lambda.$$

<u>A function that calculates a point estimate using kriging</u>

The following function, *kriging*, can be used to calculate a point estimate using the technique of kriging. The general call to the function is

[uP,w,lambda,MSE] = kriging(x,y,u,x0,gam)

where *uP* is the point estimate being calculated, *w* is a vector containing the weight coefficients w_i, *lambda* is the Lagrange multiplier λ, and *MSE* is the mean square error or variance of the error. The function requires an input vectors *x, y, u,* and *x0*. Vectors *x, y,* and *u* are the data vectors describing the signal *u(x,y)*, and *x0* is a vector containing the coordinates of the point where the estimate is to be evaluated. The input parameter *gam* is a SCILAB external function representing the semivariogram model to be used in the kriging procedure.

A listing of the function follows:

```
function [uP,w,lambda,MSE] = kriging(x,y,u,x0,gam)

//Calculates the estimate uP = u(x0)at point x0 (a vector)
//using the kriging system.  The variables x, y, and u are row vectors
//of the same length with u(j) = u(x(j),y(j)), and x0 is a row vector
```

```
//representing the point where the value of u is to be estimated.
//gam is a SCILAB external function representing the semivariogram of
//the signal u(x).  MSE = means square error.

n          = length(x);
A          = zeros(n+1,n+1);
A(1:n,n+1) = ones(n,1);
A(n+1,1:n) = ones(1,n);
b          = zeros(n,1);
b(n+1)     = 1.0;
for i = 1:n
    b(i) = -gam(norm([x(i),y(i)]-x0));
    for j = 1:n
       if i==j then
           A(i,j) = 0.0;
        else
           A(i,j) = -gam(norm([x(i),y(i)]-[x(j),y(j)]));
        end;
    end;
end;

xsol   = A\b;
w      = xsol(1:n);
lambda = xsol(n+1);
uP     = u*w;
MSE    = sqrt(-b'*xsol);
```

As an example, we will use the data vectors *x*, *y*, and *u* loaded for the example on semivariogram model determination, namely,

```
-->x=[20,10,35,50,72,67,40,70,42,20];

-->y=[120,80,90,120,122,88,75,60,46,60];

-->u=[25.7,37.8,39.0,23.0,27.0,42.0,36.0,47.5,42.5,37.0];
```

We will use the semivariogram model $\gamma(k) = 4.55\cdot k$, developed in an earlier section, to produce the kriging estimates. As an example, suppose that we wan to estimate the value of the signal *u* and point $x_0 = [20,80]$, we would use the following SCILAB commands:

```
-->deff('[g]=gam(k)','g=4.55*k')

-->getf('kriging')

-->[uP,w,lambda,MSE] = kriging(x,y,u,[20,80],gam)

 MSE     = 7.3340059

 lambda  = 5.9538866



 w  =

!    .0172807 !
!    .5206510 !
!    .2341965 !
! -  .0085810 !
! -  .0034533 !
! -  .0192623 !
!    .1448531 !
! -  .0097721 !
! -  .0172351 !
!    .1413225 !

 uP  = 37.405745
```

Kriging model verification

Model verification in the context of kriging applications consists in testing a couple of statistics based on the model residuals. The residuals are calculated as

$\delta_k = u_k - (u_P)_k$,

for $k=2,3,\ldots,n$, where u_k is the k-th value in the signal of interest, while $(u_P)_k$ is the estimated value found through the kriging procedure. Kriging for the *k-th* estimate uses the first $k-1$ values of the signal u_k. The normalized residuals are defined as

$$\varepsilon_k = \delta_k/\sigma_k,$$

where σ_k is the standard error

$$\sigma_k = (MSE_k)^{1/2}.$$

The value of σ_k is also obtained through the kriging procedure.

The test statistics that can be used for model verification are

$$M_1 = \frac{1}{n-1}\sum_{i=2}^{n}\varepsilon_k,$$

and

$$M_2 = \frac{1}{n-1}\sum_{i=2}^{n}\varepsilon_k^2.$$

Kitanidis (op. cit) indicates that M_1 follows a normal distribution with mean zero and standard deviation $1/(n-1)^{1/2}$, thus, with a level of significance of $\alpha = 0.05$, the model is rejected if

$$|M_1| > \frac{2}{\sqrt{n-1}}.$$

Alternatively, we can use the statistic M_2, which follows the chi-square (χ^2) distribution with $(n-1)$ degrees of freedom, to verify the model validity. For a level of significance $\alpha = 0.05$, we need to obtain an upper limit $U = \chi^2_{\alpha}$, and a lower limit $L = \chi^2_{1-\alpha}$, and reject the kriging model if

$$M_2 > U \quad \text{or} \quad M_2 < L.$$

A function for kriging model verification

The following function, *kriging_all*, is used to calculate the estimates of the signal $(u_P)_i$, residuals δ_i, standard errors σ_i, and normalized residuals ε_i, for $i=2,3,\dots,n$. The function provides a table listing all these values. The function also calculates the statistics M_1 and M_2, and test these values against the normal or chi-square variates corresponding to a level of significance $\alpha = 0.05$. The function returns a vector with the values of the estimates u_P. The general call to the function is

[uP] = kriging_all(x,y,u,gam)

where *uP* is the vector of estimates u_P; *x, y,* and *u* are vectors describing the original spatial signal *u(x,y)*, and *gam* is a SCILAB external function representing the semivariogram model for the kriging application. A listing of the function follows:

```
function [uP] = kriging_all(x,y,u,gam)
n = length(u);
delta    = zeros(1,n);
uP       = zeros(1,n);
sigma    = zeros(1,n);
epsilon  = zeros(1,n);
uA(1)    = u(1);

for j = 2:n
    xx = x(1:j-1); yy = y(1:j-1); uu = u(1:j-1);
    [uA,w,lambda,MSE]=kriging(xx,yy,uu,[x(j),y(j)],gam);
    uP(j)=uA;sigma(j)=sqrt(MSE);delta(j)=u(j)-uP(j);
    epsilon(j) = delta(j)/sigma(j);
end;
printf(' ');
printf('Kriging model verification');
printf('--------------------------------------------------------------');
printf('   i       u(i)       uP(i)   sigma(i)  delta(i)  epsilon(j)');
printf('--------------------------------------------------------------');
for i = 2:n
    printf('%4.0f %10.6f %10.6f %10.6f %10.6f %10.6f',...
            i,u(i),uP(i),sigma(i),delta(i),epsilon(i));
end;
printf('--------------------------------------------------------------');

M1  = sum(epsilon)/(n-1);
if abs(M1) > 2/sqrt(n-1) then
   printf('Reject kriging model based on the statistic M1 = %10.6f',M1);
else
   printf('Do not reject kriging model based on the statistic M1 = %10.6f',M1);
end;
printf('--------------------------------------------------------------');
M2     = sum(epsilon.^2)/(n-1);
nu     = n-1;
alpha  = 0.05;
Uchisq = cdfchi('X',nu,1-alpha,alpha);
Lchisq = cdfchi('X',nu,alpha,1-alpha);
if M2 > Uchisq | M2 < Lchisq then
   printf('Reject kriging model based on the statistic M2 = %10.6f',M2);
else
```

```
   printf('Do not reject kriging model based on the statistic M2 = %10.6f',M2);
end;
printf('--------------------------------------------------------------');
printf('  Degrees of freedom       = %8.4f',nu);
printf('  Chi-square lower limit   = %10.6f',Lchisq);
printf('  Chi-square upper limit   = %10.6f',Uchisq);
printf('  Significance level       =   0.05');
printf('--------------------------------------------------------------');
```

The application of function *kriging_all* to the data used above with function *kriging* produces the following table of results:

```
-->getf('kriging_all')

-->uP = kriging_all(x,y,u,gam);

Kriging model verification
-------------------------------------------------------------
   i       u(i)       uP(i)   sigma(i)  delta(i)  epsilon(j)
-------------------------------------------------------------
   2  37.800000  25.700000   4.401153  12.100000   2.749280
   3  39.000000  32.720674   3.657278   6.279326   1.716940
   4  23.000000  31.283744   3.796221  -8.283744  -2.182102
   5  27.000000  24.841117   3.745999   2.158883    .576317
   6  42.000000  33.769924   3.693156   8.230076   2.228467
   7  36.000000  41.661315   3.202054  -5.661315  -1.768026
   8  47.500000  40.825938   3.775930   6.674062   1.767528
   9  42.500000  42.984844   3.648739  - .484844  - .132880
  10  37.000000  39.345939   3.215456  -2.345939  - .729582
-------------------------------------------------------------
Do not reject kriging model based on the statistic M1 =    .469549
-------------------------------------------------------------
Reject kriging model based on the statistic M2 =   3.040690
-------------------------------------------------------------
  Degrees of freedom       =   9.0000
  Chi-square lower limit   =   3.325113
  Chi-square upper limit   =  16.918978
  Significance level       =   0.05
-------------------------------------------------------------
```

The test on statistic M_1 indicates not rejecting the kriging model, while the test on the statistic M_2 suggest we should reject the model. The chi-square testing on M_2 bases the rejection of the kriging model on the fact that $M_2 < L$. However, the values M_2 and L are relatively close, and not rejecting the kriging model may be the correct decision.

Generating synthetic signals

Synthetic signals can be generated to produce, for example, input data or parameter distributions for simulation of certain geophysical (or other type of) models. Synthetic signals can be generated in both the time and space domains. Synthetic time signals can be used, for example, to simulate precipitation series in hydrologic models, traffic pattern in transportation simulation, seasonal energy demands, etc. On the other hand, synthetic spatial signals can be used to simulate, for example, spatial distribution of aquifer properties, or to simulate the distribution of geological properties.

Generating one-dimensional signals

The procedure to generate a one-dimensional signal, given a particular autocovariance or autocorrelation function $R(\tau)$, for time signals, or $R(\xi)$, for spatial signals, is to generate a vector of values of R that serve as the basis from which the Fourier transform of the series will be generated. For details on the justification of using the autocovariance to generate synthetic signals the reader is referred to Chapter 6 in Bras, R.L. and I. Rodriguez-Iturbe, ,"*Random Functions and Hydrology,"* Addison-Wesley Publishing Company, Reading, Massachussetts.

Recall from the applications of Fourier transforms in Chapter 10 that, to generate a real signal *x* using SCILAB function *dft*, the *n* Fourier transform coefficients *X* must have a real number in the first position X_1, while the remaining *n-1* values must be complex numbers such that $X_{n-j+2} = \overline{X}_j$, for $j=2,3,\dots,(n+1)/2$. ($\overline{X}$ stands for the complex conjugate of *X*). In this case, the value of *n* must be an odd number.

Based on a vector of *m* components representing values of the autocovariance *R*, we can construct a vector of Fourier transforms *Xf* with *n=2m-1* elements satisfying the conditions of symmetry outlined above. Afterwards, we will use function *dft* (discrete Fourier transform) to generate the signal. The following function, *synthetic*, takes as input the vector of values of *R*, and returns the signal *x*. The function also produces plots of the absolute values of the Fourier transform coefficients and of the signal itself. The plots are produced without labels to allow the user to label them at their will. A list of the function follows:

```
function [x] = synthetic(X)

//Given a number of values of the spectrum X(w)
//this function generates a signal x(t)

m=length(X);n=2*m-1;            //lengths of vectors
rand('uniform');                //uniform random numbers
th = [0 2*%pi*rand(1,m-1)];     //generate angles in [0,2*%pi]
XX = X.*(cos(th)+%i*sin(th));   //generate 1/2 of spectrum
Xf = [XX zeros(1:m-1)];         //extend spectrum vector
for j=2:(n+1)/2                 //make spectrum vector symmetric
    Xf(n-j+2)=conj(Xf(j));
end;
XfA = abs(Xf);                  //spectrum magnitude
xset('window',1);plot(XfA);     //plot spectrum magnitude
x = real(fft(Xf,1));            //generate signal
xset('window',2);plot(x);       //plot signal
```

Notice that function *synthetic* generates a number of random phase angles, *th*, to calculate the synthetic signal. This is necessary in order to obtain the complex numbers that represent the Fourier transform coefficients. The values of *th* are randomly generated from a uniform distribution so that $0 < th < 2\pi$. You can add randomness to the magnitude of the Fourier transform coefficients by multiplying the values in vector *X* by random numbers generated out of a uniform or a normal distribution. An example of randomness in the magnitude of the coefficients will be shown later in this section.

The correlation models, R_k, for applications in geostatistics, can be obtained from the semivariogram models since

$$R_k = C_0 - \gamma_k.$$

Thus, for the exponential model, in which

$$\gamma_k = \left(1 - \exp(-\frac{|k|}{\mu}\right) \cdot C_0,$$

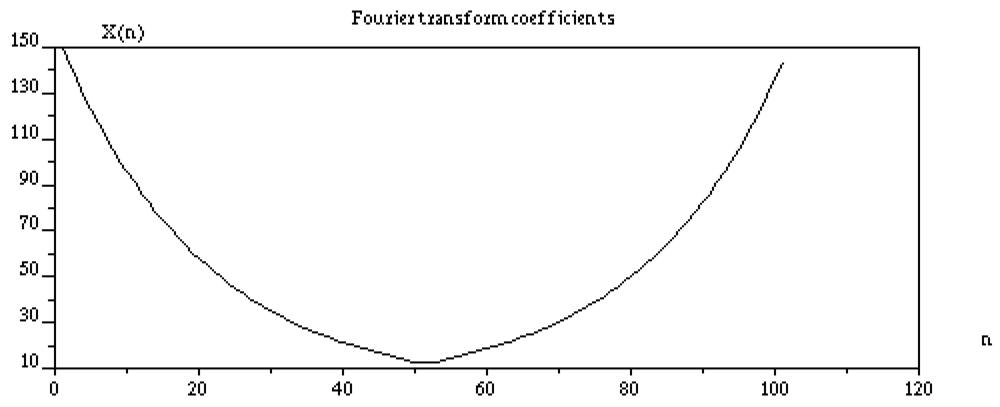
we get the following autocovariance function

$$R_k = C_0 \exp(-|k|/\mu).$$

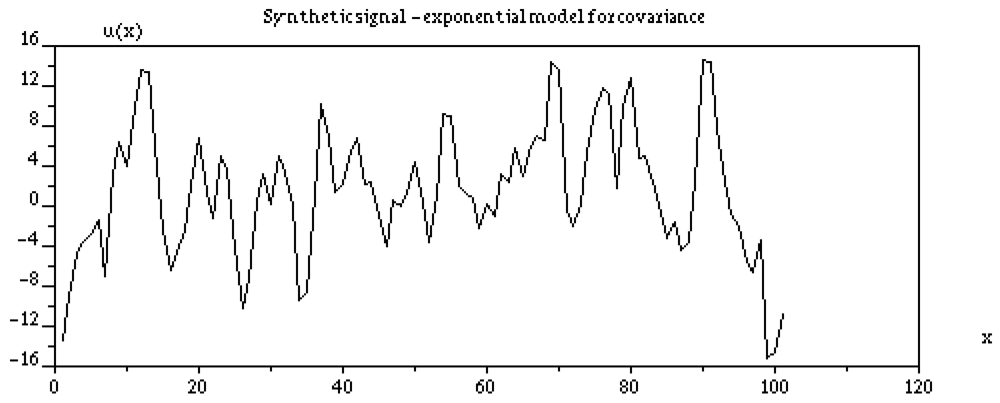
Choosing arbitrary values of the parameters C_o and μ, we can generate the vector of values of R necessary to generate a synthetic signal x, and then use function *synthetic* to generate the signal itself. For example, for $C_0 = 150$, and $\mu = 20$, we will generate a vector of values of R with the distance k varying between 0 and 50 with k increasing by 1, and use function *synthetic* to produced the corresponding signal:

```
-->k=[0:1:50];R=150*exp(-abs(k)/20);x=synthetic(R);

-->xtitle('Synthetic signal - exponential model for covariance','x','u(x)')

-->xtitle('Fourier transform coefficients','n','X(n)')
```

A plot of the Fourier transform coefficients is shown next:



The synthetic signal generated out of the Fourier transform coefficients shown above is presented in the graphic below:



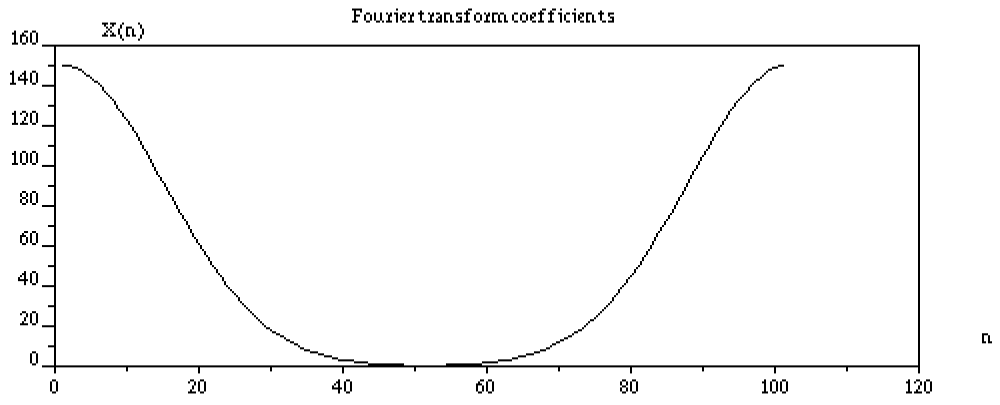
An example using the autocovariance function for the Gaussian model, i.e.,

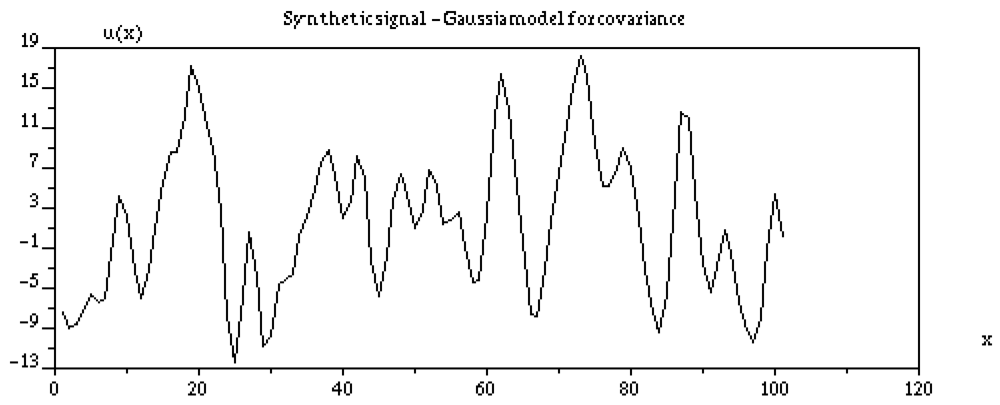
$$R_k = C_0 \exp\left(-\frac{k^2}{L^2}\right),$$

with arbitrary values C_0 = *120, L = 50*, we can generate a signal using:

```
-->k=[0:1:50];R=150*exp(-k^2/400);x=synthetic(R);

-->xtitle('Synthetic signal - Gaussian model for covariance','x','u(x)')

-->xtitle('Fourier transform coefficients','n','X(n)')
```

Plots of the Fourier transform coefficients and of the synthetic signal are shown next:



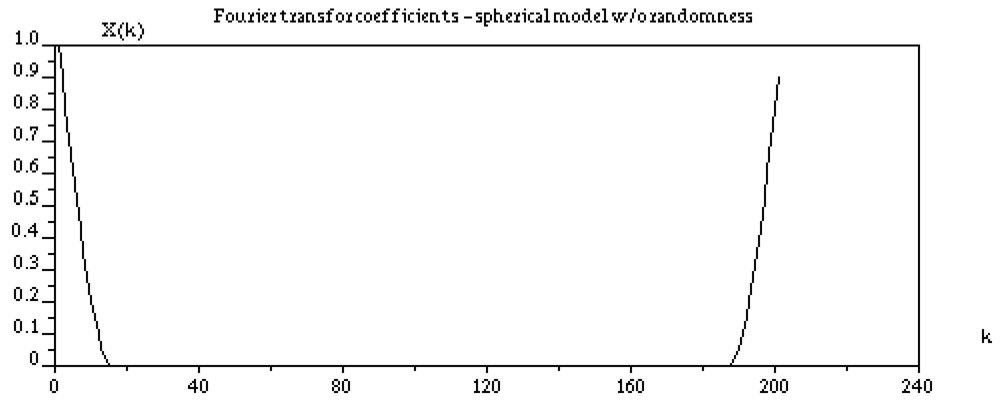


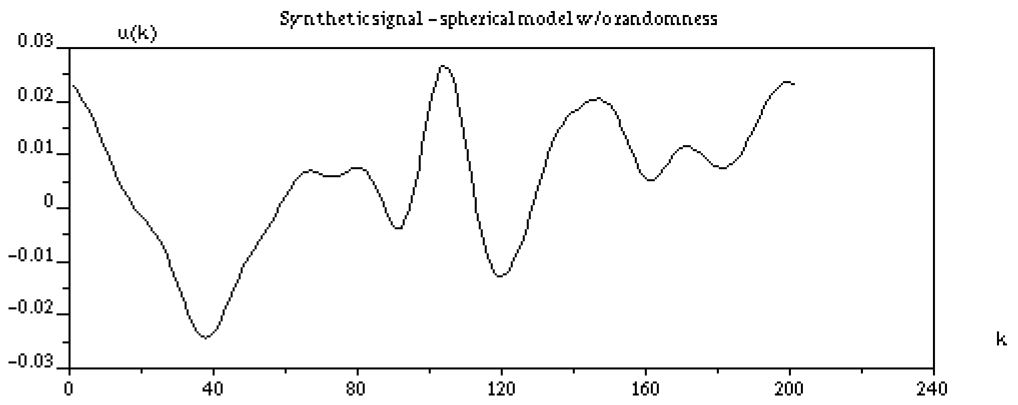
In the following example we use the spherical model for the autocovariance, namely,

$$R_k = C_0 \cdot (1-1.5(k/k_0)+0.5(k/k_0)^3).$$

We will use the values $C_0 = 1.0$ and $k_0 = 0.15$ for the parameters in the spherical distribution. The following figures show the Fourier transform coefficients and the signal generated using this spherical model.

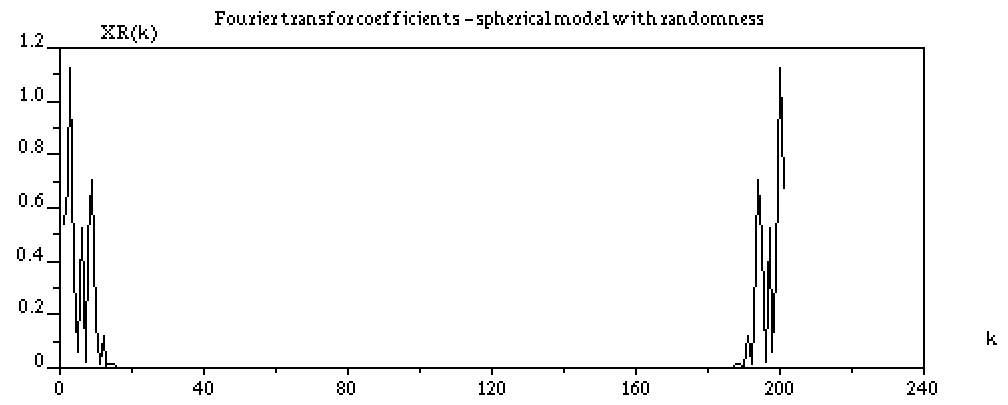
```
-->k=[0:0.01:1];X=[];for j=1:length(k),X=[X f(k(j))]; end;

-->u=synthetic(X);

-->xtitle('Synthetic signal - spherical model w/o randomness','k','u(k)')


-->xtitle('Fourier transform coefficients - spherical model w/o
randomness','k','X(k)')
```

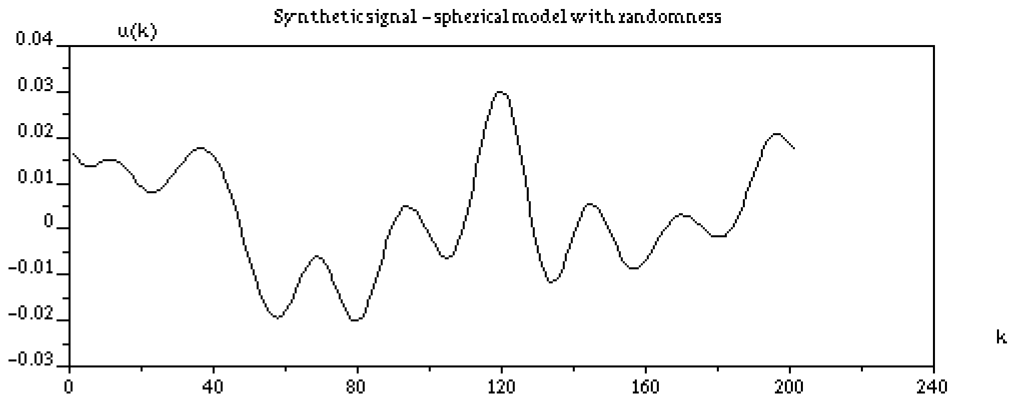




Next, we incorporate randomness into the magnitude of the Fourier transform coefficients by multiplying the vector of values of R_k times a vector of normally distributed random numbers. The following SCILAB commands will be used to generate the randomized vector of Fourier transform coefficients and the corresponding signal and graphics:

```
-->rand('normal');XR = X.*rand(1,length(k));u=synthetic(XR);

-->xtitle('Synthetic signal - spherical model with randomness','k','u(k)')

-->xtitle('Fourier transform coefficients - spherical model with randomness','k','XR(k)')
```





The general shapes of the signals generated with or without randomness incorporated in the magnitude of the Fourier transform coefficients are very similar. To generate synthetic data that simulates real measurements, however, the inclusion of randomness into the magnitude of the Fourier series coefficients is recommended.

Generating two-dimensional signals

The generation of two-dimensional signals using Fourier transforms was first introduced in Chapter 10. The Fourier transform coefficients must be provided as a matrix of (mostly) complex numbers that satisfies certain conditions of symmetry. For the case in which the $n_1 \times n_2$ matrix of coefficients for the Fourier transform is such that both n_1 and n_2 are even, the *values* $Z(1,1)$, $Z(n_1/2+1,1)$, $Z(1,n_2/+1)$, and $Z(n_1/2+1,n_2/2+1)$ must always be real. The remaining elements of matrix Z are such that $Z(n_1-k+1,1) = \bar{Z}(k,1)$, for $k = 1,2,\ldots,n_1/2$, $Z(1,n_2-m+1) = \bar{Z}(1,m+1)$, for $m = 1,2,\ldots,n_2$, and $Z(n_1-k+1,n_2-m+1) = \bar{Z}(k,m)$, for $k = 1,2,\ldots,n_1$, $m = 1,2,\ldots,n_2$, where $\bar{Z}$ represents the complex conjugate of *Z*.

The magnitude of the coefficients, for the case of spatially distributed data based on autocovariance models, is generated by using expressions of the form $R(k) = R((x^2+y^2)^{1/2})$. The following function, *synthetic2d*, can be used to generate a two-dimensional signal *z(x,y)* given vectors of values of x and y, i.e., *xr = [x0:Dx:xn]* and *yr = [y0:Dy:yn]*, and the function *f(x,y)* that represents the autocovariance. The general call to the function is

[z,yy,xx] = synthetic2d(xr,yr,f)

where *z* is a matrix representing the two dimensional signal *z(x,y)*, and *xx* and *yy* are vectors corresponding to the x and y values over which the signal was calculated. If the input vectors *xr* and *yr* contained an even number of elements, the vectors *xx* and *yy* returned by the function are exactly the same as *xr* and *yr*, respectively. If the number of elements of either *xr* or *yr* is odd, the corresponding vector (or vectors) gets recalculated so that the number of elements is even. The function produces plots of the magnitudes of the Fourier series coefficients and of the signal, and returns vectors *xx*, and *yy*, and matrix *z*. A listing of the function is shown next.

```
function [z,yy,xx] = synthetic2d(xr,yr,f)
```

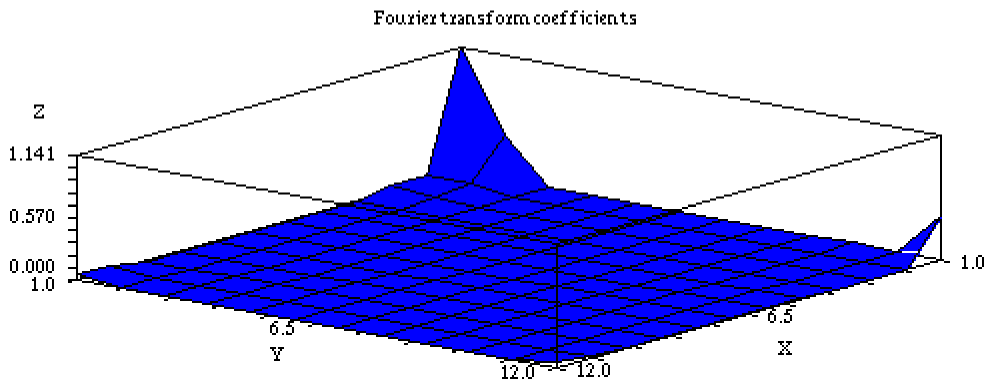
```
//Generates two-dimensional signal using Fourier transforms

//Check x-vector, modify it to make it of even length if needed
x0 = xr(1);Dx = xr(2)-xr(1);
n1 = length(xr);xn1 = xr(n1);
if modulo(n1,2) <> 0 then
   n1 = n1+1;
   Dx = (xn1-x0)/(n1-1);
   xx = [xr(1):Dx:xn1];
else
   xx = xr;
end;

//Check y-vector, modify it to make it of even length if needed
y0 = yr(1);Dy = yr(2)-yr(1);
n2 = length(yr);yn2 = yr(n2);
if modulo(n2,2) <> 0 then
   n2 = n2+1;
   Dy = (yn2-y0)/(n2-1);
   yy = [yr(1):Dy:yn2];
else
   yy = yr;
end;

//Generate basic values of 2-D Fourier transform
Z  =  feval(xx,yy,f);
rand('normal');             //Remove this and next line if no
Z  =  Z.*rand(n1,n2);       //random effects desired in magnitude of Z
rand('uniform');
th =  2*%pi*rand(n1,n2);
Z  =  Z.*exp(%i*th);

//Symmetry conditions for Fourier transform coefficient
for i = 2:n1/2
    Z(n1-i+2,1) = conj(Z(i,1));
end;


for j = 2:n2/2
    Z(1,n2-j+2) = conj(Z(1,j));
end;

for i = 2:n1
    for j = 2:n2
        Z(n1-i+2,n2-j+2) = conj(Z(i,j));
    end;
end;


Z(1,1)           = abs(Z(1,1));
Z(n1/2+1,1)      = abs(Z(n1/2+1,1));
Z(1,n2/2+1)      = abs(Z(1,n2/2+1));
Z(n1/2+1,n2/2+1) = abs(Z(n1/2+1,n2/2+1));

//Plot magnitude of Fourier transform coefficients and signal
x=[1:1:n1]; y=[1:1:n2];
xset('window',1);xbasc();plot3d(x,y,abs(Z));
z=real(fft(Z,1));
xset('window',2);xbasc();plot3d(x,y,z);


printf(' ');
printf('==============================================================================
');
printf('Notes:')
printf('1. Original xr and yr vectors may have been modified in xx and yy.');
printf('2. Magnitude of Fourier transform coefficients shown in graph window
1.');
printf('3. Signal shown in SCILAB graph window 2.');
printf('==============================================================================
');
printf(' ');
```

As an example consider the two-dimensional exponential model for the covariance $R(x,y) = C_0 \exp(-(x^2+y^2)^{1/2}/\mu)$, with values of $C_0 = 1.0$ and $\mu = 0.05$. We define the function *R(x,y)*, and the vectors *xr* and *yr* as follows:

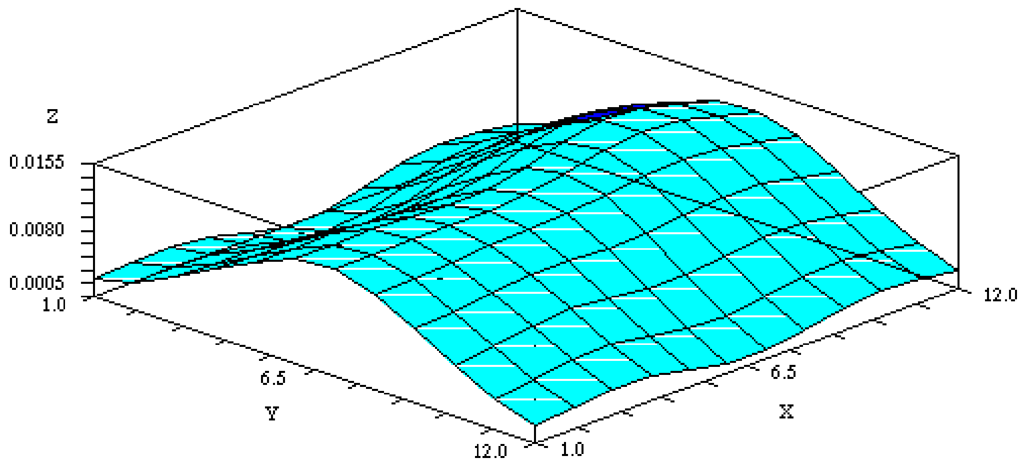
```
-->deff('[w]=f(x,y)','w=exp(-norm([x,y])/0.05)')

-->xr = [0:0.1:1]; yr = [0:0.1:1];
```

The call to function *synthetic2d* produce the following plot for the Fourier transform coefficients:

```
-->[u,yu,xu] = synthetic2d(xr,yr,f);
```



The plot of the signal is shown next:

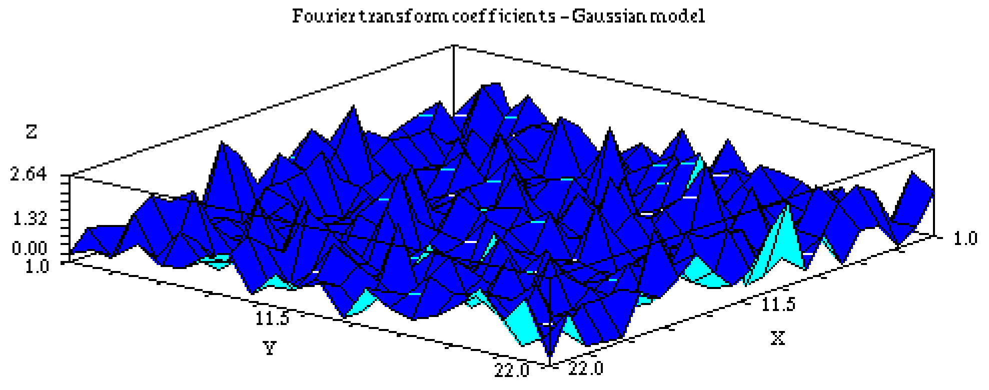


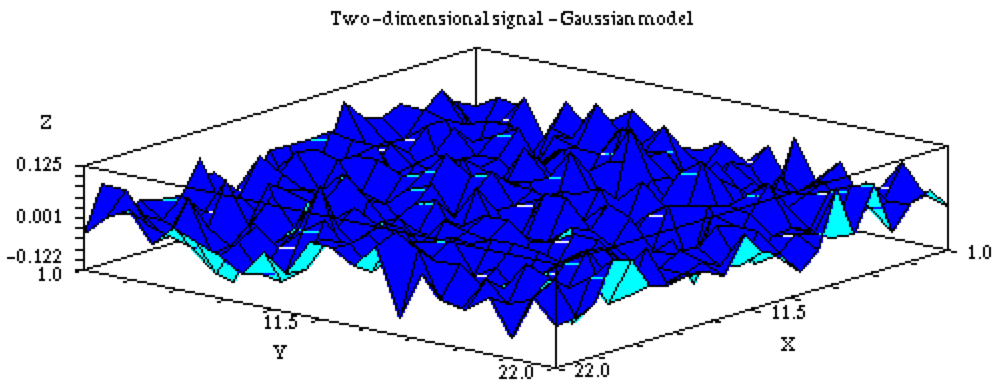
A Gaussian model for the two-dimensional covariance has the form

$$R(x,y) = C_0 \cdot \exp\left(-\frac{x^2+y^2}{L^2}\right)$$

We use the Gaussian model to generate a two-dimensional signal with C_0 = *1.0* and *L* = 5 using function *synthetic2d*:

```
-->deff('[w]=f(x,y)','w=exp(-norm([x,y])^2/5^2)')

-->xr = [-0.5:0.1:0.5]; yr = [-0.5:0.1:0.5];

-->[u,yu,xu] = synthetic2d(xr,yr,f);
```





Exercises

The following function, *rsignal*, can be used to generate a pseudo-random time series. The input is *n*, the number of points in the signal. The signal is returned as a row vector *u*.

```
function [u] = rsignal(n)
m  = grand(1,1,'uin',5,15);
rand('normal');
aa = int(100*(rand(1,m)-0.5));
bb = int(100*(rand(1,m)-0.5));
a0 = int(100*(rand(1,1)-0.5));
rand('uniform');
x  = [0:1/(n-1):1];
u  = zeros(1,n);
for i = 1:n
    u(i) = a0;
    for j = 1:m
        u(i)=u(i)+...
             aa(j)*sin(2*j*%pi*x(i))+bb(j)*cos(2*j*%pi*x(i))+50*rand();
    end;
end;
umin=min(u);umax=max(u);
umin=umin-abs(umax-umin)/10;
u = u-umin;
```

[1]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 300 points.

a) Produce a plot of the first difference of the signal.
b) Produce a plot of the second difference of the signal.
c) Produce a phase portrait of the signal against its first difference.
d) Produce a lag plot of the signal for a lag of 5.
e) Produce a lag plot of the signal for a lag of 10.
f) Produce a lag plot of the signal for a lag of 20.

[2]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 500 points.

a) Produce a plot of the first difference of the signal.
b) Produce a plot of the second difference of the signal.
c) Produce a phase portrait of the signal against its first difference.
d) Produce a lag plot of the signal for a lag of 5.
e) Produce a lag plot of the signal for a lag of 10.
f) Produce a lag plot of the signal for a lag of 20.

[3]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 300 points.

a) Produce a plot of the first difference of the signal.
b) Produce a plot of the second difference of the signal.
c) Produce a phase portrait of the signal against its first difference.
d) Produce a lag plot of the signal for a lag of 5.
e) Produce a lag plot of the signal for a lag of 10.
f) Produce a lag plot of the signal for a lag of 20.

[4]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 300 points.

a) Use SCILAB function *dft* and plot the magnitude of the Fourier series coefficients for the signal.
b) Plot the autocovariance function for this signal.
c) Plot the autocovariance for the first differences of the signal.

[5]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 100 points.

a) Use SCILAB function *dft* and plot the magnitude of the Fourier series coefficients for the signal.
b) Plot the autocovariance function for this signal.
c) Plot the autocovariance for the first differences of the signal.

[6]. Use the user-defined function *rsignal*, shown above, to obtain a time series with 500 points.

a) Use SCILAB function *dft* and plot the magnitude of the Fourier series coefficients for the signal.
b) Plot the autocovariance function for this signal.
c) Plot the autocovariance for the first differences of the signal.

[7]. Use the user-defined function *rsignal*, shown above, to obtain two time series *u* and *v*, both with 300 points.

a) Plot the cross-covariance function for signals *u* and *v*.
b) Plot the convolution of signals *u* and *v*.

[8]. Use the user-defined function *rsignal*, shown above, to obtain two time series *u* and *v*, both with 150 points.

a) Plot the cross-covariance function for signals *u* and *v*.
b) Plot the convolution of signals *u* and *v*.

[8]. Use convolution to obtain the hydrograph produced by the following hyetograph and unit hydrograph.

	hyetograph	**unit hydrograph**
t(hr)	*i(mm/hr)*	*q(m^{3}hr/(mm s))*
0	*1*	*0*
1	*3*	*24.5*
2	*2*	*52*
3	*1*	*35.5*
4		*23.2*
5		*11.2*
6		*1.5*
7		*0*

[9]. Use convolution to obtain the hydrograph produced by the following hyetograph and unit hydrograph.

	hyetograph	**unit hydrograph**
t(hr)	*i(mm/hr)*	*q(m^3hr/(mm s))*
0	*1*	*0*
1	*3*	*112*
2	*5*	*231*
3	*2*	*556*
4	*4*	*423*
5	*1*	*325*
6		*305*
7		*245*
8		*209*
9		*180*
10		*150*
11		*115*
12		*95*
13		*60*
14		*20*
15		*0*

[10]. The following table shows the annual maximum flow for the Ganga River in India measured at specific station.

Year	*Q(m^3/s)*	*Year*	*Q(m^3/s)*	*Year*	*Q(m^3/s)*	*Year*	*Q(m^3/s)*
1885	*7241*	*1907*	*7546*	*1929*	*4545*	*1951*	*4458*
1886	*9164*	*1908*	*11504*	*1930*	*5998*	*1952*	*3919*
1887	*7407*	*1909*	*8335*	*1931*	*3470*	*1953*	*5470*
1888	*6870*	*1910*	*15077*	*1932*	*6155*	*1954*	*5978*
1889	*9855*	*1911*	*6493*	*1933*	*5267*	*1955*	*4644*
1890	*11887*	*1912*	*8335*	*1934*	*6193*	*1956*	*6381*
1891	*8827*	*1913*	*3579*	*1935*	*5289*	*1957*	*4548*
1892	*7546*	*1914*	*9299*	*1936*	*3320*	*1958*	*4056*
1893	*8498*	*1915*	*7407*	*1937*	*3232*	*1959*	*4493*
1894	*16757*	*1916*	*4726*	*1938*	*3525*	*1960*	*3884*
1895	*9680*	*1917*	*8416*	*1939*	*2341*	*1961*	*4855*
1896	*14336*	*1918*	*4668*	*1940*	*2429*	*1962*	*5760*
1897	*8174*	*1919*	*6296*	*1941*	*3154*	*1963*	*9192*
1898	*8953*	*1920*	*8174*	*1942*	*6650*	*1964*	*3024*
1899	*7546*	*1921*	*9079*	*1943*	*4442*	*1965*	*2509*
1900	*6652*	*1922*	*7407*	*1944*	*4229*	*1966*	*4741*
1901	*11409*	*1923*	*5482*	*1945*	*5101*	*1967*	*5919*
1902	*9164*	*1924*	*19136*	*1946*	*4629*	*1968*	*3789*
1903	*7404*	*1925*	*9680*	*1947*	*4345*	*1969*	*4546*
1904	*8579*	*1926*	*3698*	*1948*	*4890*	*1970*	*3842*
1905	*9362*	*1927*	*7241*	*1949*	*3619*	*1971*	*4542*
1906	*7092*	*1928*	*3698*	*1950*	*5899*		

a) Plot the time series for the discharge *Q* and the first difference of the series in separate graphs.
b) Produce lag plots for the discharge *Q* using lags of 1, 5, 10, and 15 years.
c) Plot the magnitude of the Fourier transform coefficients for the discharge time series.
d) Plot the autocovariance of the discharge time series.
e) Produce the moving average plot for an averaging period of 5 years.
f) Produce the moving average plot for an averaging period of 10 years.
g) Produce the moving average plot for an averaging period of 15 years.
h) Produce the moving average plot for an averaging period of 20 years.
i) Remove the 5-year seasonality in the data (assuming such seasonality exists).
j) Remove the 10-year seasonality in the data (assuming such seasonality exists).
k) Remove the 15-year seasonality in the data (assuming such seasonality exists).
l) Remove the 20-year seasonality in the data (assuming such seasonality exists).

[11]. The following table shows the coordinates *(x,y)* and the elevations, above mean sea level, of a particular type of sandstone.

x(mi)	*y(mi)*	*z(ft)*	*x(mi)*	*y(mi)*	*z(ft)*	*x(mi)*	*y(mi)*	*z(ft)*
0.4	3.8	607	2.6	2.5	236	4.6	1.6	255
0.9	3.5	491	2.7	2.1	292	4.7	1.9	285
1.4	3.7	555	2.8	2.4	246	5.2	1.6	286
1.7	2.9	359	2.9	2.2	255	5.6	1.4	326
1.9	3.2	406	3.2	2.7	305	5.8	1.4	354
3.6	4.1	761	3.7	2.2	247	5.5	0.4	513
3.9	3.4	556	3.4	1.8	243	5.6	0.7	676
2.2	2.0	238	3.6	1.9	249			
2.4	2.4	231	4.2	3.1	594			

a) Produce contours of the sandstone elevation using the user-defined function *contouring.*
b) Produce contours that fit the data from the table to a linear function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y$$

c) Produce contours that fit the data from the table to a non-linear function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y + b_3xy$$

d) contours that fit the data from the table to a quadratic function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$$

e) Obtain the experimental variogram of the original elevation data *z*.
f) Obtain the experimental variogram of the elevation data after removing the trend from the linear function identified in part (b).

g) Obtain the experimental variogram of the elevation data after removing the trend from the non-linear function identified in part (c).
h) Obtain the experimental variogram of the elevation data after removing the trend from the quadratic function identified in part (d).

[12]. For the original sandstone elevation data of problem [11] obtain the parameters that fit the semivariogram to the following models:

a) spherical model
b) exponential model
c) Gaussian model
d) hole-effect model

[13]. The following data represents different properties of granite samples taken at the locations indicated by the coordinates *x(mi)* and *y(mi)* on a specific site. The properties listed in the table are as follows: x_1 = percentage of quartz in the sample, x_2 = color index (a percentage), x_3 = percentage of total feldspar , and *w* = specific gravity (water = 1.0)

x_1	x_2	x_3	*w*	*y*	*x*
21.3	5.5	73.0	2.63	0.920	6.090
38.9	2.7	57.4	2.64	1.150	3.625
26.1	11.1	62.6	2.64	1.160	6.750
29.3	6.0	63.6	2.63	1.300	3.010
24.5	6.6	69.1	2.64	1.400	7.405
30.9	3.3	65.1	2.61	1.590	8.630
27.9	1.9	69.1	2.63	1.750	4.220
22.8	1.2	76.0	2.63	1.820	2.420
20.1	5.6	74.1	2.65	1.830	8.840
16.4	21.3	61.7	2.69	1.855	10.920
15.0	18.9	65.6	2.67	2.010	14.225
0.6	35.9	62.5	2.83	2.040	10.605
18.4	16.6	64.9	2.70	2.050	8.320
19.5	14.2	65.4	2.68	2.210	8.060
34.4	4.6	60.7	2.62	2.270	2.730
26.9	8.6	63.6	2.63	2.530	3.500
28.7	5.5	65.8	2.61	2.620	7.445
28.5	3.9	67.8	2.62	3.025	5.060
38.4	3.0	57.6	2.61	3.060	5.420
28.1	12.9	59.0	2.63	3.070	12.550
37.4	3.5	57.6	2.63	3.120	12.130
0.9	22.9	74.4	2.78	3.400	15.400
8.8	34.9	55.4	2.76	3.520	9.910
16.2	5.5	77.6	2.63	3.610	11.520
2.2	28.4	69.3	2.74	4.220	16.400
29.1	5.1	65.7	2.64	4.250	11.430
24.9	6.9	67.8	2.70	4.940	5.910
39.6	3.6	56.6	2.63	5.040	1.840
17.1	11.3	70.9	2.71	5.060	11.760
0.0	47.8	52.2	2.84	5.090	16.430
19.9	11.6	67.2	2.68	5.240	11.330
1.2	34.8	64.0	2.84	5.320	8.780
13.2	18.8	67.4	2.74	5.320	13.730

13.7	21.2	64.0	2.74	5.330	12.450
26.1	2.3	71.2	2.61	5.350	1.430
19.9	4.1	76.0	2.63	5.610	4.150
4.9	18.8	74.30	2.77	5.850	13.840
15.5	12.2	69.70	2.72	6.460	11.660
0.0	39.7	60.20	2.83	6.590	14.640
4.5	30.5	63.90	2.77	7.260	12.810
0.0	63.8	35.20	2.92	7.420	16.610
4.0	24.1	71.80	2.77	7.910	14.650
23.4	12.4	63.10	2.79	8.470	13.330
29.5	9.8	60.40	2.69	8.740	15.770

For each of the variables x_1, x_2, x_3, and w,

a) Produce contours of the variable using the user-defined function *contouring.*
b) Produce contours that fit the data for each variable to a linear function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y$$

c) Produce contours that fit the data for each variable to a non-linear function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y + b_3xy$$

d) contours that fit the data for each variable to a quadratic function of the form

$$z = f(x,y) = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2$$

e) Obtain the experimental variogram of the original values for each variable.
f) Obtain the experimental variogram for each variable after removing the trend from the linear function identified in part (b).
g) Obtain the experimental variogram for each variable after removing the trend from the non-linear function identified in part (c).
h) Obtain the experimental variogram for each variable after removing the trend from the quadratic function identified in part (d).

[14]. For the original x_1 data of problem [13] obtain the parameters that fit the semivariogram to the following models:

a) spherical model
b) exponential model
c) Gaussian model
d) hole-effect model

[15]. For the original x_2 data of problem [13] obtain the parameters that fit the semivariogram to the following models:

a) spherical model
b) exponential model
c) Gaussian model
d) hole-effect model

[16]. For the original x_3 data of problem [13] obtain the parameters that fit the semivariogram to the following models:

a) spherical model
b) exponential model
c) Gaussian model
d) hole-effect model

[17]. For the original *w* data of problem [13] obtain the parameters that fit the semivariogram to the following models:

a) spherical model
b) exponential model
c) Gaussian model
d) hole-effect model

[15]. This problem refers to the sandstone elevation data of problem [11]. Use kriging to obtain the sandstone elevation at point $(x,y) = (1.8,3.0)$. Use each of the semivariogram models of problem [12] to estimate the elevation.

[16]. This problem refers to the data of problem [11]. Use function *kriging_all* to verify the four kriging models developed in problem [12]. Which models should be rejected, if any?

[17]. This problem refers to the x_1 data of problem [13]. Use kriging to obtain x_1 at point $(x,y) = (4.25,2.80)$. Use each of the semivariogram models of problem [14] to estimate x_1.

[18]. This problem refers to the data of problem [13]. Use function *kriging_all* to verify the four kriging models for x_1 developed in problem [14]. Which models should be rejected, if any?

[19]. This problem refers to the x_2 data of problem [13]. Use kriging to obtain x_2 at point $(x,y) = (4.25,2.80)$. Use each of the semivariogram models of problem [15] to x_2.

[20]. This problem refers to the data of problem [13]. Use function *kriging_all* to verify the four kriging models for x_2 developed in problem [15]. Which models should be rejected, if any?

[21]. This problem refers to the x_3 data of problem [13]. Use kriging to obtain x_3 at point $(x,y) = (4.25,2.80)$. Use each of the semivariogram models of problem [16] to estimate x_3.

[22]. This problem refers to the data of problem [13]. Use function *kriging_all* to verify the four kriging models for x_3 developed in problem [16]. Which models should be rejected, if any?

[23]. This problem refers to the *w* data of problem [13]. Use kriging to obtain *w* at point $(x,y) = (4.25,2.80)$. Use each of the semivariogram models of problem [16] to estimate *w*.

[24]. This problem refers to the data of problem [13]. Use function *kriging_all* to verify the four kriging models for *w* developed in problem [17]. Which models should be rejected, if any?

[25]. The one-dimensional signal corresponding to a fractal Brownian motion (*fBm*) has Fourier transform coefficients whose magnitude is given by $|X(k)|^2 = C/k^{5-2D}$, where $1 < D < 2$. The value *D* is known as the *fractal dimension* of the resulting signal. Produce one-dimensional *fBm*'s consisting of *300* values for $C = 1000000$ (this is an arbitrary constant, you can choose a different value if you want) and the following fractal dimensions: (a) $D = 1.2$; (b) $D = 1.4$; (c) $D = 1.6$; (e) $D = 1.8$.

[26]. A two-dimensional fractal signal has Fourier transform coefficients whose magnitude is given by

$$|X(k,m)|^2 = \frac{C}{\left(k^2+m^2\right)^{4-D}},$$

where *k* and *m* represent components of the frequency in two orthogonal directions, *D* is the fractal dimension, and $2 < D < 3$. Produce one-dimensional *fBm*'s based on a *64x64* grid of *k* and *m* values for $C = 1000000$ (this is an arbitrary constant, you can choose a different value if you want) and the following fractal dimensions:(a) $D = 2.2$; (b) $D = 2.4$; (c) $D = 2.6$; (e) $D = 2.8$.

[27]. The simulation of smooth surfaces such as those of sea waves or cloud surfaces utilizes the so-called Pierson-Moskowics spectrum, $|X(k)|^2 = (C/k^5)\ exp(-b/k^4)$, where *C* and *f* are constants. Produce a one-dimensional smooth surface consisting of *300* values based on the Pierson-Moskowics spectrum for $C = 1000000$ and $b = 100000$.

[28]. The two-dimensional Pierson-Moskowics spectrum can be calculated using $|X(f)|^2 = (C/f^5)\ exp(-b/f^4)$, where $f = (k^2+m^2)^{1/2}$. Produce a two-dimensional smooth surface based on the Pierson-Moskowics spectrum for $C = 1000000$ and $b = 100000$.

REFERENCES (for all SCILAB documents at InfoClearinghouse.com)

Abramowitz, M. and I.A. Stegun (editors), 1965,"*Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*," Dover Publications, Inc., New York.

Arora, J.S., 1985, "*Introduction to Optimum Design*," Class notes, The University of Iowa, Iowa City, Iowa.

Asian Institute of Technology, 1969, "*Hydraulic Laboratory Manual*," AIT - Bangkok, Thailand.

Berge, P., Y. Pomeau, and C. Vidal, 1984,"*Order within chaos - Towards a deterministic approach to turbulence*," John Wiley & Sons, New York.

Bras, R.L. and I. Rodriguez-Iturbe, 1985,"*Random Functions and Hydrology*," Addison-Wesley Publishing Company, Reading, Massachussetts.

Brogan, W.L., 1974,"*Modern Control Theory*," QPI series, Quantum Publisher Incorporated, New York.

Browne, M., 1999, "*Schaum's Outline of Theory and Problems of Physics for Engineering and Science*," Schaum's outlines, McGraw-Hill, New York.

Farlow, Stanley J., 1982, "*Partial Differential Equations for Scientists and Engineers*," Dover Publications Inc., New York.

Friedman, B., 1956 (reissued 1990), "*Principles and Techniques of Applied Mathematics*," Dover Publications Inc., New York.

Gomez, C. (editor), 1999, “*Engineering and Scientific Computing with Scilab*,” Birkhäuser, Boston.

Gullberg, J., 1997, "*Mathematics - From the Birth of Numbers*," W. W. Norton & Company, New York.

Harman, T.L., J. Dabney, and N. Richert, 2000, "*Advanced Engineering Mathematics with MATLAB® - Second edition*," Brooks/Cole - Thompson Learning, Australia.

Harris, J.W., and H. Stocker, 1998, "*Handbook of Mathematics and Computational Science*," Springer, New York.

Hsu, H.P., 1984, "*Applied Fourier Analysis*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, San Diego.

Journel, A.G., 1989, "*Fundamentals of Geostatistics in Five Lessons*," Short Course Presented at the 28th International Geological Congress, Washington, D.C., American Geophysical Union, Washington, D.C.

Julien, P.Y., 1998,”*Erosion and Sedimentation*,” Cambridge University Press, Cambridge CB2 2RU, U.K.

Keener, J.P., 1988, "*Principles of Applied Mathematics - Transformation and Approximation*," Addison-Wesley Publishing Company, Redwood City, California.

Kitanidis, P.K., 1997,”*Introduction to Geostatistics - Applications in Hydogeology*,” Cambridge University Press, Cambridge CB2 2RU, U.K.

Koch, G.S., Jr., and R. F. Link, 1971, "*Statistical Analysis of Geological Data - Volumes I and II*," Dover Publications, Inc., New York.

Korn, G.A. and T.M. Korn, 1968, "*Mathematical Handbook for Scientists and Engineers*," Dover Publications, Inc., New York.

Kottegoda, N. T., and R. Rosso, 1997, "*Probability, Statistics, and Reliability for Civil and Environmental Engineers*," The Mc-Graw Hill Companies, Inc., New York.

Kreysig, E., 1983, "*Advanced Engineering Mathematics - Fifth Edition*," John Wiley & Sons, New York.

Lindfield, G. and J. Penny, 2000, "*Numerical Methods Using Matlab®,*" Prentice Hall, Upper Saddle River, New Jersey.

Magrab, E.B., S. Azarm, B. Balachandran, J. Duncan, K. Herold, and G. Walsh, 2000, "*An Engineer's Guide to MATLAB®*", Prentice Hall, Upper Saddle River, N.J., U.S.A.

McCuen, R.H., 1989,”*Hydrologic Analysis and Design - second edition*,” Prentice Hall, Upper Saddle River, New Jersey.

Middleton, G.V., 2000, "*Data Analysis in the Earth Sciences Using Matlab®,"* Prentice Hall, Upper Saddle River, New Jersey.

Montgomery, D.C., G.C. Runger, and N.F. Hubele, 1998, "*Engineering Statistics*," John Wiley & Sons, Inc.

Newland, D.E., 1993, "*An Introduction to Random Vibrations, Spectral & Wavelet Analysis - Third Edition*," Longman Scientific and Technical, New York.

Nicols, G., 1995, "*Introduction to Nonlinear Science*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Parker, T.S. and L.O. Chua, , "*Practical Numerical Algorithms for Chaotic Systems*," 1989, Springer-Verlag, New York.

Peitgen, H-O. and D. Saupe (editors), 1988, "*The Science of Fractal Images*," Springer-Verlag, New York.

Peitgen, H-O., H. Jürgens, and D. Saupe, 1992, "*Chaos and Fractals - New Frontiers of Science*," Springer-Verlag, New York.

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, 1989, "*Numerical Recipes - The Art of Scientific Computing (FORTRAN version)*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Raghunath, H.M., 1985, "*Hydrology - Principles, Analysis and Design*," Wiley Eastern Limited, New Delhi, India.

Recktenwald, G., 2000, "*Numerical Methods with Matlab - Implementation and Application*," Prentice Hall, Upper Saddle River, N.J., U.S.A.

Rothenberg, R.I., 1991, "*Probability and Statistics*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, San Diego, CA.

Sagan, H., 1961,"*Boundary and Eigenvalue Problems in Mathematical Physics*," Dover Publications, Inc., New York.

Spanos, A., 1999,"*Probability Theory and Statistical Inference - Econometric Modeling with Observational Data*," Cambridge University Press, Cambridge CB2 2RU, U.K.

Spiegel, M. R., 1971 (second printing, 1999), "*Schaum's Outline of Theory and Problems of Advanced Mathematics for Engineers and Scientists*," Schaum's Outline Series, McGraw-Hill, New York.

Tanis, E.A., 1987, "*Statistics II - Estimation and Tests of Hypotheses*," Harcourt Brace Jovanovich College Outline Series, Harcourt Brace Jovanovich, Publishers, Fort Worth, TX.

Tinker, M. and R. Lambourne, 2000, "*Further Mathematics for the Physical Sciences*," John Wiley & Sons, LTD., Chichester, U.K.

Tolstov, G.P., 1962, "*Fourier Series*," (Translated from the Russian by R. A. Silverman), Dover Publications, New York.

Tveito, A. and R. Winther, 1998, "*Introduction to Partial Differential Equations - A Computational Approach*," Texts in Applied Mathematics 29, Springer, New York.

Urroz, G., 2000, "*Science and Engineering Mathematics with the HP 49 G - Volumes I & II*", www.greatunpublished.com, Charleston, S.C.

Urroz, G., 2001, "*Applied Engineering Mathematics with Maple*", www.greatunpublished.com, Charleston, S.C.

Winnick, J., , "*Chemical Engineering Thermodynamics - An Introduction to Thermodynamics for Undergraduate Engineering Students*," John Wiley & Sons, Inc., New York.