7 Spatial Data

7.1 Types of Spatial Data

Most data in earth sciences are spatially distributed, either as vector data, (points, lines, polygons) or as raster data (gridded topography). Vector data are generated by digitizing map objects such as drainage networks or outlines of lithologic units. Raster data can be obtained directly from a satellite sensor output, but in most cases grid data can be interpolated from irregularly-distributed samples from the field (*gridding*).

The following chapter introduces the use of vector data by using coastline data as an example (Chapter 7.2). Subsequently, the acquisition and handling of raster data are illustrated with help of digital topography data (Chapters 7.3 to 7.5). The availability and use of digital elevation data has increased considerably since the early 90's. With 5 arc minutes resolution, the ETOPO5 was one of the first data sets for topography and bathymetry. In October 2001, it was replaced by the ETOPO2 that has a resolution of 2 arc minutes. In addition, there is a data set for topography called GTOPO30 completed in 1996 that has a horizontal grid spacing of 30 arc seconds (approximately 1 km). Most recently, the 30 and 90 m resolution data from the Shuttle Radar Topography Mission (SRTM) have replaced the older data sets in most scientific studies.

The second part of the chapter deals with surface estimates from irregular-spaced data and statistics on spatial data (Chapters 7.6 to 7.8). In earth sciences, most data are collected in an irregular pattern. Access to rock samples is often restricted to natural outcrops such as shoreline cliffs and the walls of a gorge, or anthropogenic outcrops such as road cuts and quarries. Clustered and traversed data are a challenge for all gridding techniques. The corresponding chapters illustrate the use of the most important gridding routines and outline the potential pitfalls while using these methods. Chapters 7.9 to 7.11 introduce various methods to analyse spatial data, including the application of statistical tests to point distributions (Chapter 7.9), the spatial analysis of digital elevation models (Chapter 7.10) and an overview of geostatistics and kriging (Chapter 7.10).

This chapter requires the Mapping Toolbox although most graphics routines used in our examples can be easily replaced by standard MATLAB functions. An alternative and useful mapping toolbox by Rich Pawlowicz (Earth and Ocean Sciences at the Unversity of British Columbia) is available from

```
http://www2.ocgy.ubc.ca/~rich
```

The handling and processing of large spatial data sets also requires a powerful computing system with at least 1 GB physical memory.

7.2 The GSHHS Shoreline Data Set

The global self-consistent, hierarchical, high-resolution shoreline data base GSHHS is amalgamated from two public domain data bases by Paul Wessel (SOEST, University of Hawaii, Honolulu, HI) and Walter Smith (NOAA Laboratory for Satellite Altimetry, Silver Spring, MD). On the web page of the US National Geophysical Data Center (NGDC)

```
http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html
```

the coastline vector data can be downloaded as MATLAB vector data. First, we define the geographic range of interest as decimal degrees with West and South denoted by a negative sign. For example, the East African coast would be displayed on the latitude between 0 and +15 degrees and longitude of +35 to +55 degrees. Subsequently, it is important to choose the coastline data base from which the data is to be extracted. As an example, the *World Data Bank II* provides maps at the scale 1 : 2,000,000. Finally, the compression method is set to *None* for the ASCII data that have been extracted. The data format is set to be *MATLAB* and *GMT Preview* is enabled. The resulting GMT map and a link to the raw text data can be displayed by pressing the *Submit-Extract* button at the end of the web page. By opening the 430 KB large text file on a browser, the data can be saved onto a new file called *coastline.txt*. The two columns in this file represent the *longitude/latitude* coordinates of NaN-separated polygons or coastline segments.

```
NaN        NaN
42.892067  0.000000
42.893692  0.001760
NaN        NaN
42.891052  0.001467
42.898093  0.007921
```

```
42.904546  0.013201
42.907480  0.016721
42.910414  0.020828
42.913054  0.024642
42.915987  0.028749
42.918921  0.032562
42.922441  0.035789
(cont'd)
```

The `NaN`'s perform two functions: they provide a means for identifying break points in the data. They also serve as pen-up commands when the Mapping Toolbox plots vector maps. The shorelines can be displayed by using

```
data = load('coastline.txt');

plot(data(:,1),data(:,2),'k'), axis equal
xlabel('Longitude'), ylabel('Latitude')
```

More advanced plotting functions are contained in the Mapping Toolbox, which allow to generate an alternative version of this plot (Fig. 7.1):

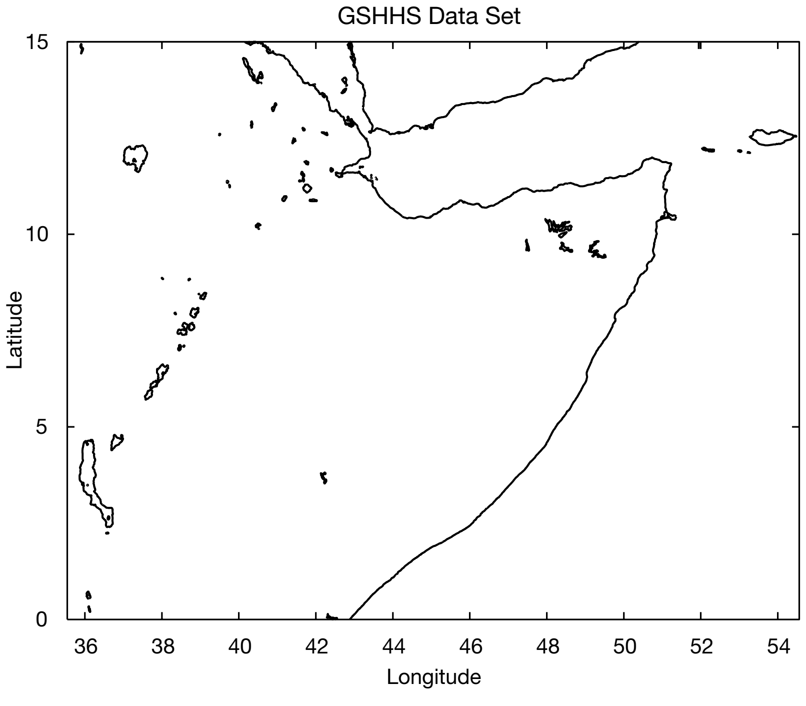


Fig. 7.1 Display of the GSHHS shoreline data set. The map shows an area between 0° and 15° northern latitude, 40° and 50° eastern longitude. Simple map using the function `plot` and equal axis aspect ratios.

```
axesm('MapProjection','mercator', ...
      'MapLatLimit',[0 15], ...
      'MapLonLimit',[35 55], ...
      'Frame','on', ...
      'MeridianLabel','on', ...
      'ParallelLabel','on');
plotm(data(:,2),data(:,1),'k');
```

Note that the input for `plotm` is given in the order *longitude*, followed by the *latitude*. The second column of the data matrix is entered first. In contrast, the function `plot` requires an *xy* input. The first column is entered first. The function `axesm` defines the map axis and sets various map properties such as the map projection, the map limits and the axis labels.

7.3 The 2-Minute Gridded Global Elevation Data ETOPO2

ETOPO2 is a global data base of topography and bathymetry on a regular 2-minute grid. It is a compilation of data from a variety of sources. It can be downloaded from the US National Geophysical Data Center (NGDC) web page

```
http://www.ngdc.noaa.gov/mgg/fliers/01mgg04.html
```

From the menu bar *Free online* we select *Make custom grids* which is linked to the *GEODAS Grid Translator*. First, we choose a Grid ID (e.g., *grid01*), the Grid Data Base (e.g., *ETOPO2 2-minute Global Relief*), our computer system (e.g., *Macintosh*) and the Grid Format (e.g., *ASCII* for both the data and the header). Next we define the *longitude* and *latitude* bounds. For example, the latitude (*lat*) from –20 to +20 degrees and a longitude (*lon*) between +30 and +60 degrees corresponds to the East African coast. The selected area can be transformed into a digital elevation matrix by pressing *Design–a–grid*. this matrix may be downloaded from the web page by pressing *Download your Grid Data*, *Compress and Retrieve* and *Retrieve compressed file* in the subsequent windows. Decompressing the file *grid01.tgz* creates a directory *grid01_data*. This directory contains various data and help files. The subdirectory *grid01* contains the ASCII raster grid file *grid01.asc* that has the following content:

```
NCOLS   901
NROWS  1201
XLLCORNER  30.00000
YLLCORNER -20.00000
CELLSIZE 0.03333333
NODATA_VALUE  -32768
270   294   278   273   262   248   251   236   228   223 ...
```

```
280   278   278   264   254   253   240   234   225   205 ...
256   266   267   283   257   273   248   228   215   220 ...
272   273   258   258   254   264   232   218   229   210 ...
259   263   268   275   242   246   237   219   211   209 ...
(cont'd)
```

The header documents the size of the data matrix (e.g., 901 columns and 1201 rows in our example), the coordinates of the lower-left corner (e.g., x=30 and y=–20), the cell size (e.g., 0.033333 = 1/30 degree latitude and longitude) and the *–32768* flag for data voids. We comment the header by typing `%` at the beginning of the first six lines

```
%NCOLS   901
%NROWS  1201
%XLLCORNER  30.00000
%YLLCORNER -20.00000
%CELLSIZE 0.03333333
%NODATA_VALUE  -32768
270   294   278   273   262   248   251   236   228   223 ...
280   278   278   264   254   253   240   234   225   205 ...
256   266   267   283   257   273   248   228   215   220 ...
272   273   258   258   254   264   232   218   229   210 ...
259   263   268   275   242   246   237   219   211   209 ...
(cont'd)
```

and load the data into the workspace.

```
ETOPO2 = load('grid01.asc');
```

We flip the matrix up and down. Then, the *–32768* flag for data voids has to be replaced by the MATLAB representation for Not-a-Number `NaN`.

```
ETOPO2 = flipud(ETOPO2);
ETOPO2(find(ETOPO2 == -32768)) = NaN;
```

Finally, we check whether the data are now correctly stored in the workspace by printing the minimum and maximum elevations of the area.

```
max(ETOPO2(:))
min(ETOPO2(:))
```

In this example, the maximum elevation of the area is 5199 m and the minimum elevation is –5612 m. The reference level is the sea level at 0 m. We now define a coordinate system using the information that the lower-left corner is *s20e30*, i.e., 20° southern latitude and 30° eastern longitude. The resolution is 2 arc minutes corresponding to 1/30 degrees.

```
[LON,LAT] = meshgrid(30:1/30:60,-20:1/30:20);
```

Now we generate a colored surface from the elevation data using the function `surf`.

```
surf(LON,LAT,ETOPO2)
shading interp
axis equal, view(0,90)
colorbar
```

This script opens a new figure window and generates a colored surface. The surface is highlighted by a set of color shades on an overhead view (Fig. 7.2). More display methods will be described in the chapter on SRTM elevation data.

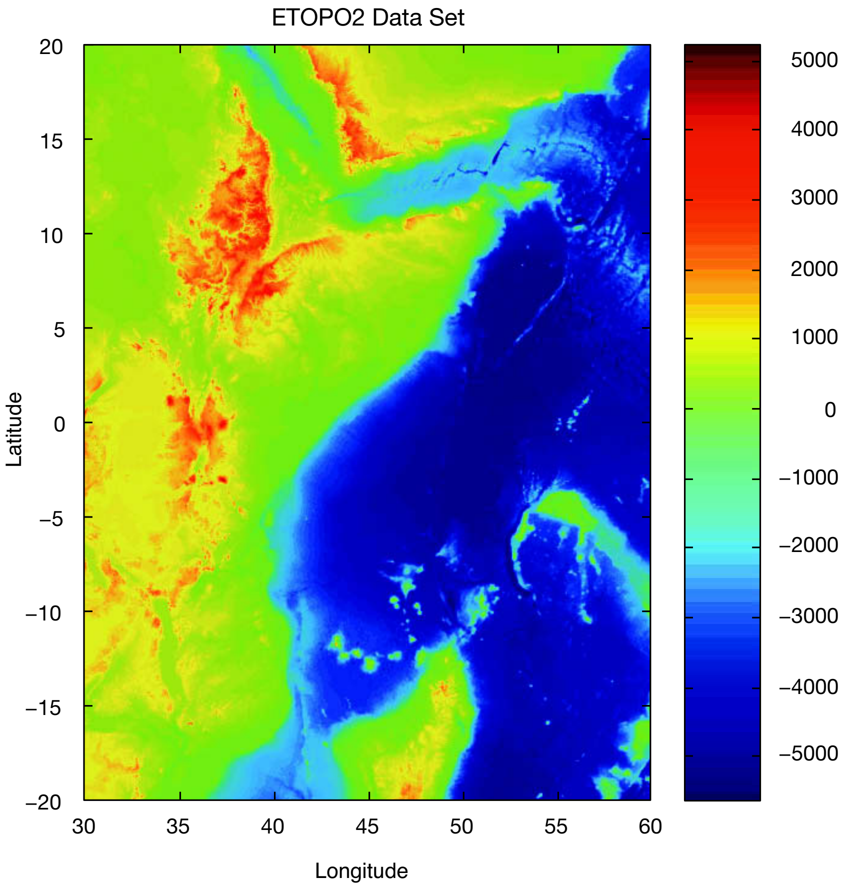


Fig. 7.2 Display of the ETOPO2 elevation data set. The map uses the function `surf` for generating a colored surface. The colorbar provides an information on the colormap used to visualize topography and bathymetry.

7.4 The 30-Arc Seconds Elevation Model GTOPO30

The 30 arc second (approximately 1 km) global digital elevation data set GTOPO30 only contains elevation data, not bathymetry. The data set has been developed by the Earth Resources Observation System Data Center and is available from the web page

```
http://edcdaac.usgs.gov/gtopo30/gtopo30.html
```

The model uses a variety of international data sources. However, it is mainly based on raster data from the Digital Terrain Elevation Model (DTEM) and vector data from the Digital Chart of the World (DCW). The GTOPO30 data set has been divided into 33 pieces or tiles. The tile names refer to the longitude and latitude of the upper-left (northwest) corner of the tile. The tile name *e020n40* refers to the upper-left corner of the tile. In our example, the coordinates of the upper-left corner are 20 degrees eastern longitude and 40 degrees northern latitude. As example, we select and download the tile *e020n40* provided as a 24.9 MB compressed *tar* file. After decompressing the *tar* file, we obtain eight files containing the raw data and header files in various formats. Moreover, the file provides a GIF image of a shaded relief display of the data.

Importing the GTOPO30 data into the workspace is simple. The Mapping Toolbox provides an import routine `gtopo30` that reads the data and stores it onto a regular data grid. We import only a subset of the original matrix:

```
latlim = [-5 5]; lonlim = [30 40];
GTOPO30 = gtopo30('E020N40',1,latlim,lonlim);
```

This script reads the data from the tile *e020n40* (without file extension) in full resolution (scale factor = 1) into the matrix `GTOPO30` of the dimension 1200x1200 cells. The coordinate system is defined by using the *lon/lat* limits as listed above. The resolution is 30 arc seconds corresponding to 1/120 degrees.

```
[LON,LAT] = meshgrid(30:1/120:40-1/120,-5:1/120:5-1/120);
```

We have to reduce the limits by 1/120 to obtain a matrix of similar dimension as the matrix GTOPO30. A grayscale image can be generated from the elevation data by using the function `surf`. The fourth power of the colormap `gray` is used to darken the map at higher levels of elevation. Subsequently, the colormap is flipped vertically in order to obtain dark colors for high elevations and light colors for low elevations.

```
figure
surf(LON,LAT,GTOPO30)
shading interp
colormap(flipud(gray.^4))
axis equal, view(0,90)
colorbar
```

This script opens a new figure window, generates the gray surface using interpolated shading in an overhead view (Fig. 7.3).

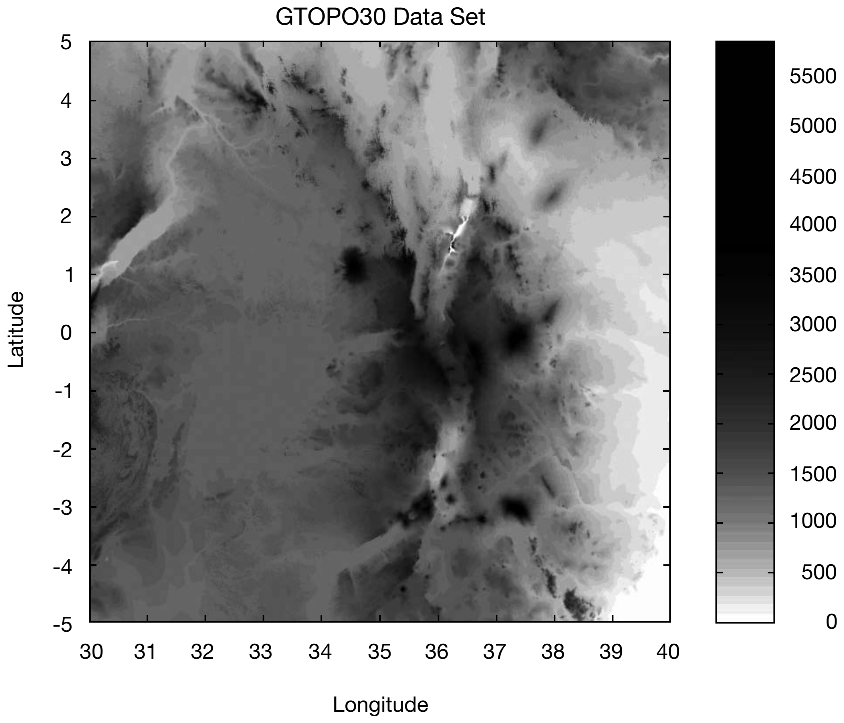


Fig. 7.3 Display of the GTOPO30 elevation data set. The map uses the function `surf` for generating a gray surface. We use the colormap `gray` to power of four in order to darken the colormap with respect to the higher elevation. In addition, we flip the colormap in up/down direction using `flipud` to obtain dark colors for high elevations and light colors for low elevations.

7.5 The Shuttle Radar Topography Mission SRTM

The Shuttle Radar Topography Mission (SRTM) incorporates a radar system that flew onboard the Space Shuttle *Endeavour* during an 11-day mission in February 2000. SRTM is an international project spearheaded by the National Geospatial-Intelligence Agency (NGA) and the National Aeronautics and Space Administration (NASA). Detailed info on the SRTM project including a gallery of images and a users forum can be accessed on the NASA web page:

```
http://www2.jpl.nasa.gov/srtm/
```

The data were processed at the Jet Propulsion Laboratory. They are being distributed through the United States Geological Survey‘s (USGS) EROS Data Center by using the Seamless Data Distribution System.

```
http://seamless.usgs.gov/
```

Alternatively, the raw data files can be downloaded via FTP from

```
ftp://e0srp01u.ecs.nasa.gov/srtm
```

This directory contains zipped files of SRTM-3 DEM’s from various areas of the world, processed by the SRTM global processor and sampled at 3 arc seconds or 90 meters. As an example, we download the 1.7 MB large file *s01e036.hgt.zip* containing the SRTM data. All elevations are in meters referenced to the WGS84 EGM96 geoid as documented at

```
http://earth-info.nga.mil/GandG/wgs84/index.html
```

The name of this file refers to the longitude and latitude of the lower-left (southwest) pixel of the tile, i.e., one degree southern latitude and 36 degrees eastern longitude. SRTM-3 data contain 1201 lines and 1201 samples with similar overlapping rows and columns. After having downloaded and unzipped the file, we save *s01e036.hgt* in our working directory. The digital elevation model is provided as 16-bit signed integer data in a simple binary raster. Bit order is Motorola (*big-endian*) standard with the most significant bit first. The data are imported into the workspace using

```
fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);
```

This script opens the file *s01e036.hgt* for read access using `fopen`, defines the file identifier `fid`, which is then used for reading the binaries from the file using `fread`, and writing it into the matrix `SRTM`. Function `fclose` closes the file defined by `fid`. First, the matrix needs to be transposed and flipped vertically.

```
SRTM = SRTM'; SRTM = flipud(SRTM);
```

The *–32768* flag for data voids can be replaced by `NaN`, which is the MATLAB representation for Not-a-Number.

```
SRTM(find(SRTM == -32768)) = NaN;
```

Finally, we check whether the data are now correctly stored in the workspace by printing the minimum and maximum elevations of the area.

```
max(SRTM(:))

ans =
   3992

min(SRTM(:))

ans =
   1504
```

In our example, the maximum elevation of the area is 3992 m, the minimum altitude is 1504 m above sea level. A coordinate system can be defined by using the information that the lower-left corner is *s01e036*. The resolution is 3 arc seconds corresponding to 1/1200 degrees.

```
[LON,LAT] = meshgrid(36:1/1200:37,-1:1/1200:0);
```

A shaded grayscale map can be generated from the elevation data using the function `surfl`. This function displays a shaded surface with simulated lighting.

```
figure
surfl(LON,LAT,SRTM)
shading interp
colormap gray
view(0,90)
colorbar
```

This script opens a new figure window, generates the shaded-relief map using interpolated shading and a gray colormap in an overhead view. Since SRTM data contain much noise, we first smooth the data using an arbitrary

9×9 pixel moving average filter. The new matrix is stored in the matrix SRTM_FILTERED.

```
B = 1/81 * ones(9,9);
SRTM_FILTERED = filter2(B,SRTM);
```

The corresponding shaded-relief map is generated by

```
figure
surfl(LON,LAT,SRTM_FILTERED)
shading interp
colormap gray
view(0,90)
colorbar
```

After having generated the shaded-relief map (Fig. 7.4), the graph has to be exported onto a graphics file. For instance, the figure may be written onto a JPEG format with 70% quality level and a 300 dpi resolution.

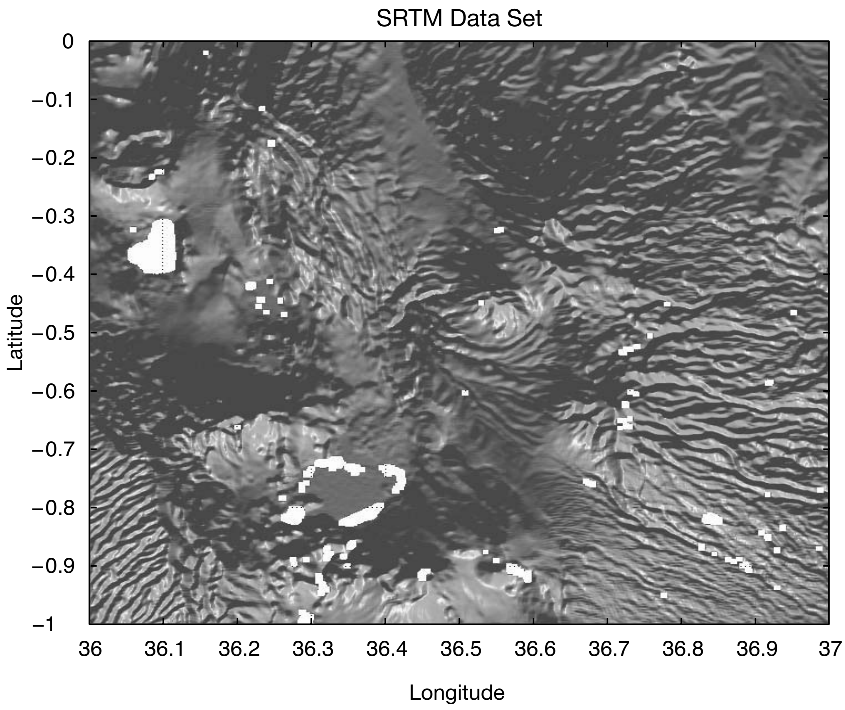


Fig. 7.4 Display of the filtered SRTM elevation data set. The map uses the function surfl for generating a shaded-relief map with simulated lighting using interpolated shading and a gray colormap in an overhead view. Note that the SRTM data set contains a lot of gaps, in particular in the lake areas.

```
print -djpeg70 -r300 srtmimage
```

The new file *srtmimage.jpg* has a size of 300 KB. The decompressed image has a size of 16.5 MB. This file can now be imported to another software package such as Adobe® Photoshop®.

7.6 Gridding and Contouring Background

The previous data sets were all stored in evenly-spaced two-dimensional arrays. Most data in earth sciences, however, are obtained on an irregular sampling pattern. Therefore, irregular-spaced data have to be interpolated, i.e., we compute a smooth and continuous surface from our measurements in the field. *Surface estimation* is typically carried out in two major steps. Firstly, the number of *control points* needs to be selected. Secondly, the *grid points* have to be estimated. Control points are irregularly-spaced field measurements, such as the thicknesses of sandstone units at different outcrops or the concentrations of a chemical tracer in water wells. The data are generally represented as *xyz* triplets, where *x* and *y* are spatial coordinates, and *z* is the variable of interest. In such cases, most gridding methods require continuous and unique data. However, the spatial variables in earth sciences are often discontinuous and spatially nonunique. As an example, the sandstone unit may be faulted or folded. Furthermore, gridding requires spatial autocorrelation. In other words, the neighboring data points should be correlated with each other by a certain relationship. It is not sensible to use random *z* variable for the surface estimation if the data are not autocorrelated. Having selected the control points, the calculation of the *z* values at the evenly-spaced grid points varies from method to method.

Various techniques exist for selecting the control points (Fig. 7.5a). Most methods make arbitrary assumptions on the autocorrelation of the *z* variable. The *nearest-neighbor criterion* includes all control points within a circular neighborhood of the grid point, where the radius of the circle is specified by the user. Since the spatial autocorrelation is likely to decrease with increasing distance from the grid point, considering too many distant control points is likely to lead to erroneous results while computing the grid points. On the other hand, small circular areas limit the calculation of the grid points to a very small number of control points. Such an approach leads to a noisy estimate of the modeled surface.

It is perhaps due to these difficulties that *triangulation* is often used as an alternative method for selecting the control points (Fig. 7.5b). In this tech-

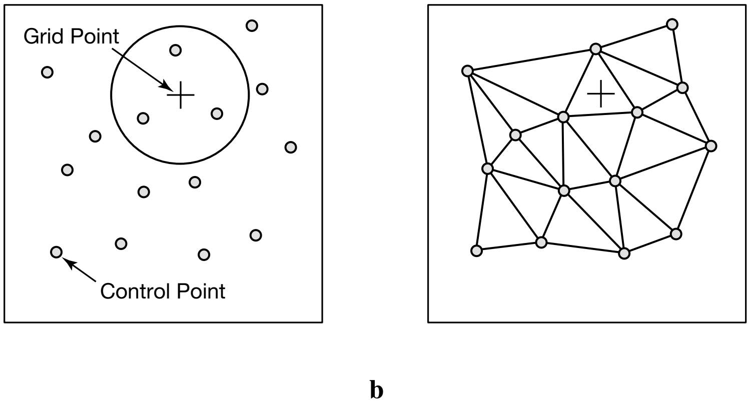


Fig. 7.5 Methods to select the control points for estimating the grid points. **a** Construction of a circle around the grid point (plus sign) with a radius defined by the spatial autocorrelation of the z-values at the control points (circles). **b** Triangulation. The control points are selected from the apices of the triangles surrounding the grid point and optional also the apices of the adjoining triangles.

nique, all control points are connected to a triangular net. Every grid point is located in a triangular area of three control points. The z value of the grid point is computed from the z values of the grid points. In a modification of such gridding, the three points at the apices of the three adjoining triangles are also used. The *Delauney triangulation* uses the triangular net where the acuteness of the triangles is minimized, i.e., the triangles are as close as possible to equilateral.

Kriging introduced in Chapter 7.9 is an alternative approach of selecting control points. It is often regarded as *the* method of gridding. Some people even use the term *geostatistics* synonymous with kriging. Kriging is a method for determining the spatial autocorrelation and hence the circle dimension. More sophisticated versions of kriging use an elliptical area which includes the control points.

The second step of surface estimation is the actual computation of the z values of the grid points. The *arithmetic mean* of the z values at the control points

$$\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i$$

provides the easiest way of computing the grid points. This is a particularly useful method if there are only a limited number of control points. If the study area is well covered by control points and the distance between these points is highly variable, the z values of the grid points should be computed by a *weighted mean*. The z values at the control points are weighted by the inverse distance d_i from the grid points.

$$\bar{z} = \frac{\sum_{i=1}^{N} (z_i / d_i)}{\sum_{i=1}^{N} (1 / d_i)}$$

Depending on the spatial scaling relationship of the parameter z, the inverse square or the root of distance may also be used instead of weighing the z values by the inverse of distance. The fitting of 3D *splines* to the control points provides another method for computing the grid points that is commonly used in the earth sciences. Most routines used in surface estimation involve *cubic polynomial splines*, i.e., a third-degree 3D polynomial is fitted to at least six adjacent control points. The final surface consists of a composite of pieces of these splines. MATLAB also provides interpolation with biharmonic splines generating very smooth surfaces (Sandwell, 1987).

7.7 Gridding Example

MATLAB provides a biharmonic spline interpolation since the beginnings. This interpolation method was developed by Sandwell (1987). This specific gridding method produces smooth surfaces that are particularly suited for noisy data sets with irregular distribution of control points.

As an example, we use synthetic *xyz* data representing the vertical distance of an imaginary surface of a stratigraphic horizon from a reference surface. This lithologic unit was displaced by a normal fault. The foot wall of the fault shows roughly horizontal strata, whereas the hanging wall is characterized by the development of two large sedimentary basins. The *xyz* data are irregularly distributed and have to be interpolated onto a regular grid. Assume that the *xyz* data are stored as a three-column table in a file named *normalfault.txt*.

```
4.32e+02   7.46e+01   0.00e+00
4.46e+02   7.21e+01   0.00e+00
4.51e+02   7.87e+01   0.00e+00
4.66e+02   8.71e+01   0.00e+00
4.65e+02   9.73e+01   0.00e+00
4.55e+02   1.14e+02   0.00e+00
4.29e+02   7.31e+01   5.00e+00
(cont'd)
```

The first and second column contains the coordinates x (between 420 and 470 of an arbitrary spatial coordinate system) and y (between 70 and 120), whereas the third column contains the vertical z values. The data are loaded using

```
data = load('normalfault.txt');
```

Initially, we wish to create an overview plot of the spatial distribution of the control points. In order to label the points in the plot, numerical z values of the third column are converted into string representation with maximum two digits.

```
labels = num2str(data(:,3),2);
```

The 2D plot of our data is generated in two steps. Firstly, the data are displayed as empty circles by using the plot command. Secondly, the data are labeled by using the function text(x,y,'string') which adds text contained in string to the xy location. The value 1 is added to all x coordinates as a small offset between the circles and the text.

```
plot(data(:,1),data(:,2),'o')
hold on
text(data(:,1)+1,data(:,2),labels);
hold off
```

This plot helps us to define the axis limits for gridding and contouring, xlim = [420 470] and ylim = [70 120]. The function meshgrid transforms the domain specified by vectors x and y into arrays XI and YI. The rows of the output array XI are copies of the vector x and the columns of the output array YI are copies of the vector y. We choose 1.0 as grid intervals.

```
x = 420:1:470; y = 70:1:120;
[XI,YI] = meshgrid(x,y);
```

The biharmonic spline interpolation is used to interpolate the irregular-spaced data at the grid points specified by XI and YI.

```
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');
```

The option v4 depicts the biharmonic spline interpolation, which was the sole gridding algorithm until MATLAB4 was replaced by MATLAB5. MATLAB provides various tools for the visualization of the results. The simplest way to display the gridding results is a contour plot using contour. By default, the number of contour levels and the values of the contour levels are chosen automatically. The choice of the contour levels depends on the minimum and maximum values of `z`.

```
contour(XI,YI,ZI)
```

Alternatively, the number of contours can be chosen manually, e.g., ten contour levels.

```
contour(XI,YI,ZI,10)
```

Contouring can also be performed at values specified in a vector `v`. Since the maximum and minimum values of `z` is

```
min(data(:,3))

ans =
   -25

max(data(:,3))

ans =
   20
```

we choose

```
v = -30 : 10 : 20;
```

The command

```
[c,h] = contour(XI,YI,ZI,v);
```

returns contour matrix `c` and a handle `h` that can be used as input to the function `clabel`, which labels contours automatically.

```
clabel(c,h)
```

Alternatively, the graph is labeled manually by selecting the `manual` option in the function `clabel`. This function places labels onto locations that have been selected with the mouse. Labeling is terminated by pressing the `return` key.

```
[c,h] = contour(XI,YI,ZI,v);
clabel(c,h,'manual')
```

Filled contours are an alternative to the empty contours used above. This function is used together with `colorbar` displaying a legend for the graph. In addition, we plot the locations and `z` values of the true data points (black empty circles, text labels) (Fig. 7.6).

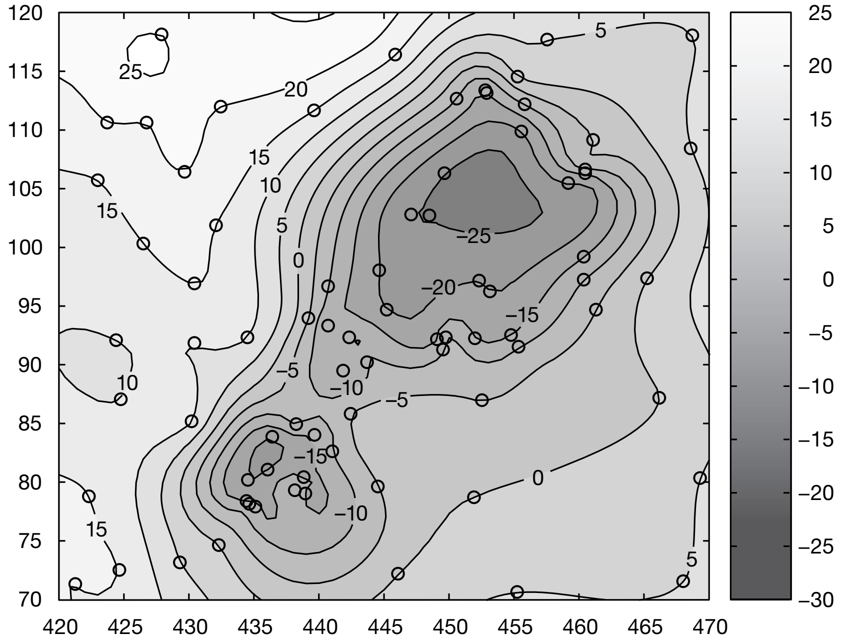


Fig. 7.6 Contour plot of the locations and *z*-values of the true data points (black empty circles, text labels).

```
contourf(XI,YI,ZI,v), colorbar
hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels);
hold off
```

A pseudocolor plot is generated by using the function `pcolor`. Black contours are also added at the same levels as in the above example.

```
pcolor(XI,YI,ZI), shading flat
hold on
contour(XI,YI,ZI,v,'k')
hold off
```

The third dimension is added to the plot by using the `mesh` command. We use this example also to introduce the function `view(az,el)` for a viewpoint specification. Herein, `az` is the azimuth or horizontal rotation and `el` is the vertical elevation (both in degrees). The values `az` = –37.5 and `el` = 30 define the default view of all 3D plots,

```
mesh(XI,YI,ZI), view(-37.5,30)
```

whereas `az` = 0 and `el` = 90 is directly overhead and the default 2D view

```
mesh(XI,YI,ZI), view(0,90)
```

The function `mesh` represents only one of the many 3D visualization methods. Another commonly used command is the function `surf`. Furthermore, the figure may be rotated by selecting the *Rotate 3D* option on the *Edit Tools* menu. We also introduce the function `colormap`, which uses predefined pseudo colormaps for 3D graphs. Typing `help graph3d` lists a number of builtin colormaps, although colormaps can be arbitrarily modified and generated by the user. As an example, we use the colormap *hot*, which is a *black-red-yellow-white* colormap.

```
surf(XI,YI,ZI), colormap('hot'), colorbar
```

Here, *Rotate 3D* only rotates the 3D plot, not the colorbar. The function `surfc` combines both a surface and a 2D contour plot in one graph.

```
surfc(XI,YI,ZI)
```

The function `surfl` can be used to illustrate an advanced application of 3D visualization. It generates a 3D colored surface with interpolated shading and lighting. The axis labeling, ticks and background can be turned off by typing `axis off`. In addition, black 3D contours may be added to the surface plot. The grid resolution is increased prior to data plotting to obtain smooth surfaces (Fig. 7.7).

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

surf(XI,YI,ZI), shading interp, light, axis off
hold on
contour3(XI,YI,ZI,v,'k');
hold off
```

The biharmonic spline interpolation described in this chapter provides a solution to most gridding problems. Therefore, it was the only gridding method that came with MATLAB for quite a long time. However, different applications in earth sciences require different methods for interpolation, but there is no method without problems. The next chapter compares biharmonic splines with other gridding methods and summarizes their strengths and weaknesses.

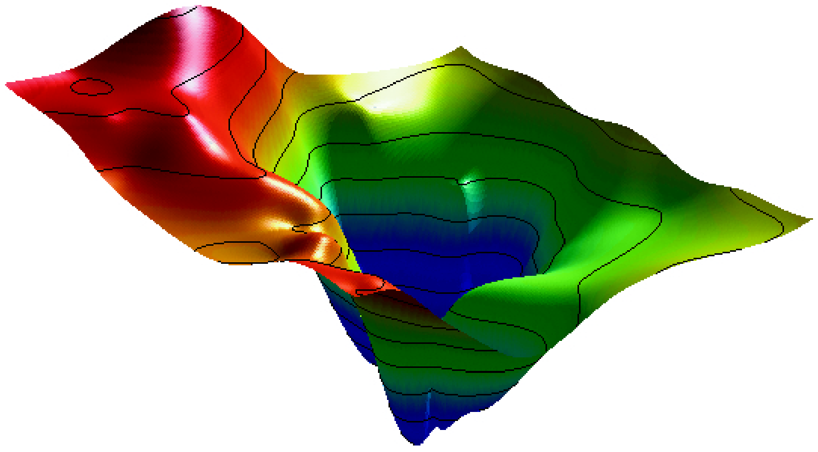


Fig. 7.7 Three-dimensional colored surface with interpolated shading and simulated lighting. The axis labeling, ticks and background are turned off. In addition, the graph contains black 3D contours.

7.8 Comparison of Methods and Potential Artifacts

The first example illustrates the use of the *bilinear interpolation* technique for gridding irregular-spaced data. Bilinear interpolation is an extension of the one-dimensional linear interpolation. In the two-dimensional case, linear interpolation is performed in one direction first, then in the other direction. Intuitively, the bilinear method is one of the simplest interpolation techniques. One would not expect serious artifacts and distortions of the data. On the contrary, this method has a number of disadvantages and therefore other methods are used in many applications.

The sample data used in the previous chapter can be loaded to study the performance of a bilinear interpolation.

```
data = load('normalfault.txt');
labels = num2str(data(:,3),2);
```

We now choose the option `linear` while using the function `griddata` to interpolate the data.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'linear');
```

The results are plotted as contours. The plot also includes the location of the control points.

```
contourf(XI,YI,ZI), colorbar, hold on
plot(data(:,1),data(:,2),'o'), hold off
```

The new surface is restricted to the area that contains control points. By default, bilinear interpolation does not extrapolate beyond this region. Furthermore, the contours are rather angular compared to the smooth outline of the contours of the biharmonic spline interpolation. The most important character of the bilinear gridding technique, however, is illustrated by a projection of the data in a vertical plane.

```
plot(XI,ZI,'k'), hold on
plot(data(:,1),data(:,3),'ro')
text(data(:,1)+1,data(:,3),labels)
title('Linear Interpolation'), hold off
```

This plot shows the projection of the estimated surface (vertical lines) and the labeled control points. The *z*-values at the grid points never exceed the *z*-values of the control points. Similar to the linear interpolation of time series (Chapter 5), bilinear interpolation causes significant smoothing of the data and a reduction of the high-frequency variation.

Biharmonic splines are sort of the other extreme in many ways. They are often used for extremely irregular-spaced and noisy data.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

contourf(XI,YI,ZI), colorbar, hold on
plot(data(:,1),data(:,2),'o'), hold off
```

The contours suggest an extremely smooth surface. In many applications, this solution is very useful, but the method also produces a number of artifacts. As we can see from the next plot, the estimated values at the grid points are often out of the range of the measured *z*-values.

```
plot(XI,ZI,'k'), hold on
plot(data(:,1),data(:,3),'o')
text(data(:,1)+1,data(:,3),labels);
title('Biharmonic Spline Interpolation'), hold off
```

This sometimes makes much sense and does not smooth the data in the way bilinear gridding does. However, introducing very close control points with different *z*-values can cause serious artifacts.

```
data(79,:) = [450 105 5];
data(80,:) = [450 104.5 -5];
labels = num2str(data(:,3),2);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

contourf(XI,YI,ZI), colorbar, hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels)
```

The extreme gradient at the location (450,105) results in a paired *low* and *high* (Fig. 7.8). In such cases, it is recommended to delete one of the two control points and replace the z-value of the remaining control point by the arithmetic mean of both z-values.

Extrapolation beyond the area supported by control points is a common feature of splines (see also Chapter 5). Extreme local trends combined with large areas with no data often cause unrealistic estimates. To illustrate these edge effects we eliminate all control points in the upper-left corner.

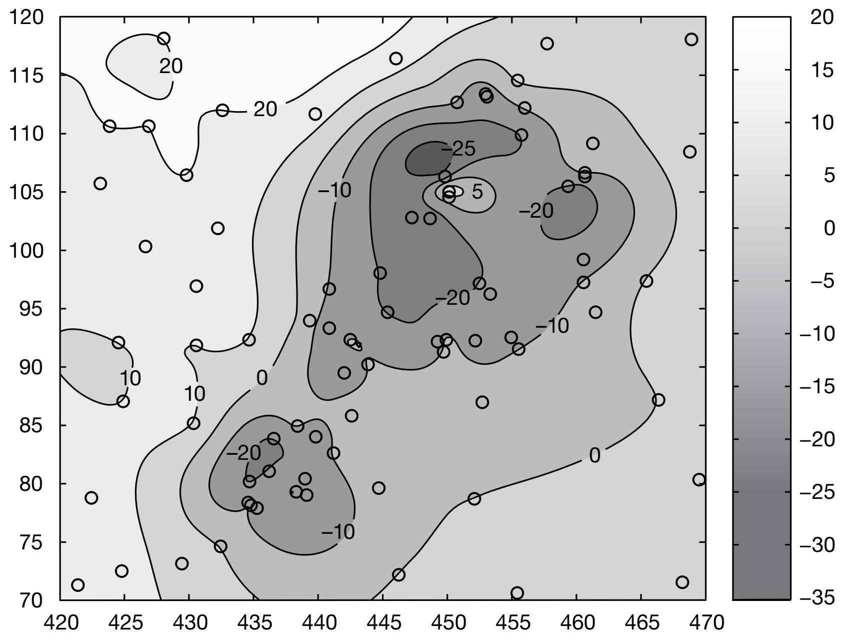


Fig. 7.8 Contour plot of a data set gridded using a biharmonic spline interpolation. At the location (450,105), very close control points with different z-values have been introduced. Interpolation causes a paired low and high, which is a common artefact of spline interpolation of noisy data.

```
[i,j] = find(data(:,1)<435 & data(:,2)>105);
data(i,:) = [];

labels = num2str(data(:,3),2);

plot(data(:,1),data(:,2),'ko')
hold on
text(data(:,1)+1,data(:,2),labels);
hold off
```

We again employ the biharmonic spline interpolation technique.

```
[XI,YI] = meshgrid(420:0.25:470,70:0.25:120);
ZI = griddata(data(:,1),data(:,2),data(:,3),XI,YI,'v4');

v = -40 : 10 : 40;
contourf(XI,YI,ZI,v)
caxis([-40 40])
colorbar
hold on
plot(data(:,1),data(:,2),'ko')
```

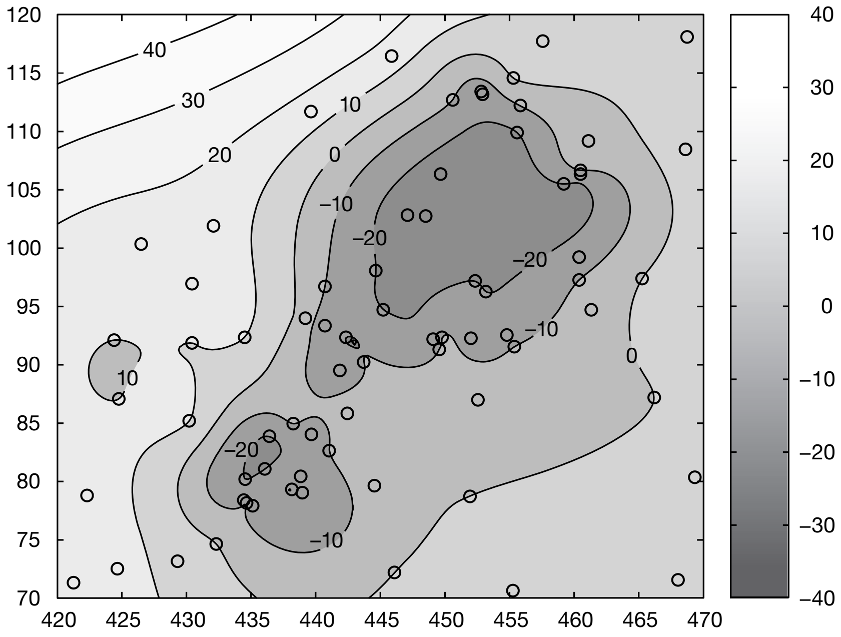


Fig. 7.9 Contour plot of a data set gridded using a biharmonic spline interpolation. No control points are available in the upper left corner. The spline interpolation then extrapolates beyond the area with control points using gradients at the map edges causing unrealistic *z* estimates at the grid points.

```
text(data(:,1)+1,data(:,2),labels)
hold off
```

As we can see from the plot, this method extrapolates beyond the area with control points using gradients at the map edges (Fig. 7.9). Such effect is particular undesired in the case of gridded closed data, such as percentages, or data that have only positive values. In such cases, it is recommended to replace the estimated `z` values by `NaN`. For instance, we erase the areas with `z` values larger than 20, which is regarded as an unrealistic value. The corresponding plot now contains a sector with no data.

```
ZID = ZI;
ZID(find(ZID > 20)) = NaN;

contourf(XI,YI,ZID,v)
caxis([-40 40])
colorbar
hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels)
hold off
```

Alternatively, we can eliminate a rectangular area with no data.

```
ZID = ZI;
ZID(131:201,1:71) = NaN;

contourf(XI,YI,ZID,v)
caxis([-40 40])
colorbar
hold on
plot(data(:,1),data(:,2),'ko')
text(data(:,1)+1,data(:,2),labels)
hold off
```

In some examples, the area with no control points is simply eliminated by putting a legend on this part of the map.

Another very useful MATLAB gridding method are *splines with tension* by Wessel and Bercovici (1998). The *tsplines* use biharmonic splines in tension t, where the parameter t can vary between 0 and 1. A value of $t=0$ corresponds to a standard cubic spline interpolation. Increasing t reduces undesirable oscillations between data points, e.g., the paired *lows* and *highs* observed in one of the above examples. The limiting situation $t \rightarrow 1$ corresponds to linear interpolation.

7.9 Statistics of Point Distributions

This chapter is about the statistical distribution of points in an area, which may help understand the relationship between these objects and properties of the area. For instance, the spatial concentration of handaxes in an archaeological site suggests that a larger population of hominins lived in that part of the area. The clustered occurrence of fossils may document environmental conditions that are favourable to the corresponding organisms. Volcano alignments often help to map tectonic structures in the deeper and shallower subsurface.

The following text introduces methods for the statistical analysis of point distributions. First, the spatial distribution of objects is tested for uniform and random distribution. Then, a simple test for clustered distributions of objects is presented.

Test for Uniform Distribution

We compute synthetic data to illustrate the test for uniform distributions. The function `rand` computes uniformly-distributed pseudo-random numbers drawn from a uniform distribution on the unit interval. We compute *xy* data using `rand` and multiply the data by ten to obtain data on the interval [0,10].

```
rand('seed',0)
data = 10 * rand(100,2);
```

We use the χ^2–test introduced in Chapter 3.8 to test the hypothesis that the data have a uniform distribution. The *xy* data are now organized in 25 classes that are square subareas of the size 2-by-2. We display the data as blue points in a plot *y* versus *x*. The square areas are outlined by red lines (Fig. 7.10).

```
plot(data(:,1),data(:,2),'o')
hold on
x = 0:10; y = ones(size(x));
for i = 1:4, plot(x,2*i*y,'r-'), end
for i = 1:4, plot(2*i*y,x,'r-'), end
hold off
```

The three-dimensional version of histogram `hist3` is used to display the spatial data organized in classes (Fig. 7.11).

```
hist3(data,[5 5]), view(30,70)
```

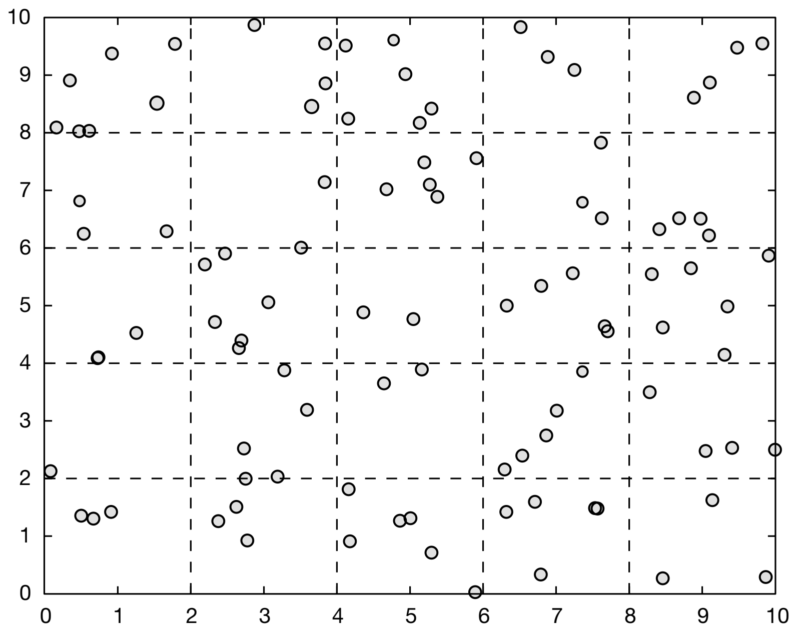


Fig. 7.10 Two-dimensional plot of a point distribution. The distribution of objects in the field are tested for uniform distribution using the χ^2-test. The *xy* data are organized in 25 classes that are square subareas of the size 2-by-2.

Equivalent to the two-dimensional function, the function `hist3` can be used to compute the frequency distribution `n_exp` of the data.

```
n_exp = hist3(data,[5 5]);
n_exp = n_exp(:);
```

For a uniform distribution, the theoretical frequencies for the classes are identical. The expected number of objects in each square area is the size of the total area $10 \times 10 = 100$ divided by the 25 subareas or classes, which comes to be four. To compare the theoretical frequency distribution with the actual distribution of objects, we generate an 5-by-5 array with identical elements four.

```
n_syn = 4 * ones(25,1);
```

The χ^2-test explores the squared differences between the observed and expected frequencies (Chapter 3.8). The quantity χ^2 is defined as the sum of the squared differences divided by the expected frequencies.

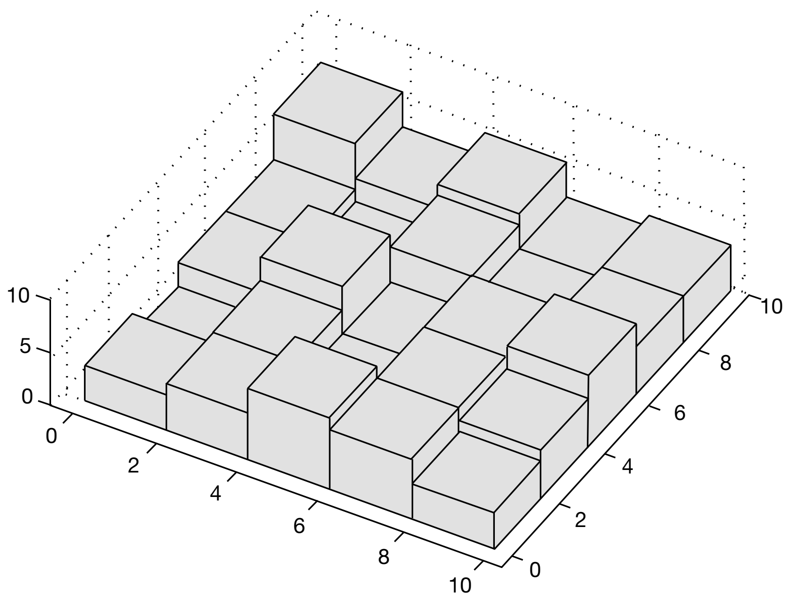


Fig. 7.11 Three-dimensional histogram displaying the numbers of objects for each subarea. The histogram was created using `hist3`.

```
chi2_data = sum((n_exp - n_syn).^2 ./n_syn)

chi2 =
    14
```

The critical χ^2 can be calculated by using `chi2inv`. The χ^2-test requires the degrees of freedom Φ. In our example, we test the hypothesis that the data are uniformly distributed, i.e., we estimate only one parameter (Chapter 3.4). Therefore, the number of degrees of freedom is $\Phi = 25 - (1 + 1) = 23$. We test the hypothesis on a $p = 95\%$ significance level. The function `chi2inv` computes the inverse of the χ^2 CDF with parameters specified by Φ for the corresponding probabilities in p.

```
chi2_theo = chi2inv(0.95,25-1-1)

ans =
    35.1725
```

The critical χ^2 of 35.1725 is well above the measured χ^2 of 14. Therefore, we cannot reject the null hypothesis and conclude that our data follow a uniform distribution.

Test for Random Distribution

The following example illustrates the test for randomly-distributed objects in an area. We use the uniformly-distributed data generated in the previous example and display the point distribution.

```
clear
rand('seed',0)
data = 10 * rand(100,2);
plot(data(:,1),data(:,2),'o')
hold on
x = 0:10; y = ones(size(x));
for i = 1:9, plot(x,i*y,'r-'), end
for i = 1:9, plot(i*y,x,'r-'), end
hold off
```

We generate the three-dimensional histogram and use the function `hist3` to count the objects per class. In contrast to the previous test, we now count the subareas containing a certain number of observations. The number of subareas is usually larger than it would be used for the previous test. In our example, we use 49 subareas or classes.

```
hist3(data,[7 7])
view(30,70)

counts = hist3(data,[7 7]);
counts = counts(:);
```

The frequency distribution of subareas with a certain number of objects follows a Poisson distribution (Chapter 3.4) if the objects are randomly distributed. First, we compute a frequency distribution of subareas with *N* objects. In our example, we count the subareas with 0, …, 5 objects. We also display the histogram of the frequency distribution as a two-dimensional histogram using `hist` (Fig. 7.12).

```
N = 0 : 5;

[n_exp,v] = hist(counts,N);

hist(counts,N)
title('Histogram')
xlabel('Number of observations N')
ylabel('Subareas with N observations')
```

The expected number of subareas E_j with a certain number of objects j can be computed using

$$E_j = Te^{-n/T}\frac{(n/T)^j}{j!}$$

where n is the total number of objects and T is the number of subareas. For $j=0$, $j!$ is taken to be 1. We compute the theoretical frequency distribution using the equation shown above,

```
for i = 1 : 6
    n_syn(i) = 49*exp(-100/49)*(100/49)^N(i)/factorial(N(i));
end
n_syn = sum(n_exp)*n_syn/sum(n_syn);
```

and display both the empirical and theoretical frequency distributions in one plot.

```
h1 = bar(v,n_exp);
hold on
h2 = bar(v,n_syn);
hold off

set(h1,'FaceColor','none','EdgeColor','r')
set(h2,'FaceColor','none','EdgeColor','b')
```

The χ^2-test is again employed to compare the empirical and theoretical distributions. The test is performed on a $p=95\%$ significance level. The Poisson distribution is defined by only one parameter (Chapter 3.4). Therefore, the number of degrees of freedom is $\Phi=6-(1+1)=4$. The measured χ^2 of

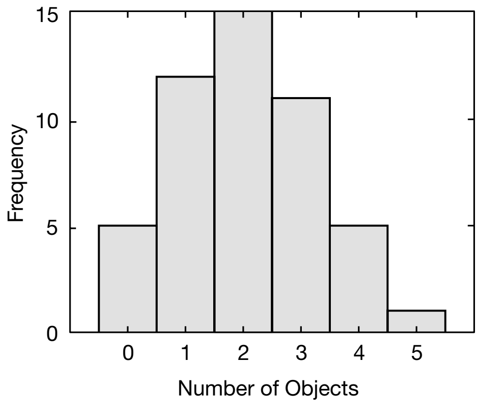


Fig. 7.12 Frequency distribution of subareas with N objects. In our example, we count the subareas with 0, …, 5 objects. We display the histogram of the frequency distribution as a two-dimensional histogram using `hist`.

```
chi2 = sum((n_exp - n_syn).^2 ./n_syn)

chi2 =
    1.4357
```

is well below the critical χ^2, which is

```
chi2inv(0.95,6-1-1)

ans =
    9.4877
```

Therefore, we cannot reject the null hypothesis and conclude that our data follow a Poission distribution. Therfore, the point distribution is random.

Test for Clustering

Point distributions in geosciences are often clustered. We use a *nearest-neighbor criterion* to test a spatial distribution for clustering. Davis (2002) published an excellent summary of the nearest-neighbor analysis, summarizing the work of a number of other authors. Swan and Sandilands (1996) presented a simplified description of this analysis. The test for clustering computes the distances d_i of all possible pairs of nearest points in the field. The *observed mean nearest-neighbor distance* is

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$$

where n is the total number of points or objects in the field. The arithmetic mean of all distances is related to the area of the map. This relationship is expressed by the *expected mean nearest-neighbor distance*, which is

$$\bar{\delta} = \frac{1}{2}\sqrt{A/n}$$

where A is the map area. Small values of this ratio then suggest significant clustering, whereas larger values indicate regularity or uniformity. The test uses a Z statistic (Chapter 3.4), which is

$$Z = \frac{\bar{d} - \bar{\delta}}{s_e}$$

where s_e is the standard error of the mean nearest-neighbor distance, which is defined as

$$s_e = \frac{0.26136}{\sqrt{n^2 / A}}$$

The null hypothesis *randomness* is tested against two alternative hypotheses, *clustering* and *uniformity or regularity*. The Z statistic has critical values of 1.96 and –1.96 at a significance level of 95%. If $-1.96<Z<+1.96$, we accept the null hypothesis that the data are randomly distributed. If $Z<-1.96$, we reject the null hypothesis and accept the first alternative hypothesis of clustering. If $Z>+1.96$, we also reject the null hypothesis, but accept the alternative hypothesis of uniformity or regularity.

As an example, we use the synthetic data analyzed in the previous examples again.

```
clear
rand('seed',0)
data = 10 * rand(100,2);
plot(data(:,1),data(:,2),'o')
```

We first compute the pairwise Euclidian distance between all pairs of observations using the function `pdist` (Chapter 9.4). The resulting distance matrix is then reformatted between upper triangular and square form using `squareform`.

```
distances = pdist(data,'Euclidean');
distmatrix = squareform(distances);
```

The following `for` loop finds the nearest neighbors, stores the corresponding distances and computes the mean distance.

```
for i = 1 : 5
    distmatrix(i,i) = NaN;
    k = find(distmatrix(i,:) == min(distmatrix(i,:)));
    nearest(i) = distmatrix(i,k(1));
end
observednearest = mean(nearest)

observednearest =
    0.5471
```

In our example, the mean nearest distance `observednearest` comes to be 0.5471. Next, we calculate the area of the map. The expected mean nearest-neighbor distance is half the squareroot of the map area divided by the number of observations.

```
maparea = (max(data(:,1)-min(data(:,1)))) ...
         *(max(data(:,2)-min(data(:,2))));
expectednearest = 0.5 * sqrt(maparea/length(data))

expectednearest =
    0.4940
```

In our example, the expected mean nearest distance `expectednearest` is 0.4940. Finally, we compute the standard error of the mean nearest-neighbor distance `se`

```
se = 0.26136/sqrt((length(data).^2/maparea))

se =
    0.0258
```

and the test statistic `Z`.

```
Z = (observednearest - expectednearest)/se

Z =
    2.0561
```

In our example, `Z` is 2.0561. Since `Z`>+1.96, we reject the null hypothesis and conclude that the data are uniformly or regularly distributed, but not clustered.

7.10 Analysis of Digital Elevation Models (by R. Gebbers)

Digital elevation models (DEM) and their derivatives (e.g., slope and aspect) can indicate surface processes like lateral water flow, solar irradiation or erosion. The simplest derivatives of a DEM are the slope and the aspect. The *slope* (or *gradient*) describes the measurement of the steepness, the incline or the grade of a surface measured in percentages or degrees. The *aspect* (or *exposure*) generally refers to the direction to which a mountain slope faces.

We use the SRTM data set introduced in Chapter 7.5 to illustrate the analysis of a digital elevation model for slopes, aspects and other derivatives. The data are loaded by

```
fid = fopen('S01E036.hgt','r');
SRTM = fread(fid,[1201,inf],'int16','b');
fclose(fid);

SRTM = SRTM';
SRTM = flipud(SRTM);
SRTM(find(SRTM==-32768)) = NaN;
```

These data are elevation values in meters above sea level sampled at a 3-arc-second or 90 meter grid. The SRTM data contain small-scale spatial disturbances and noise that could cause problems when computing a consistent drainage pattern. Therefore, we lowpass-filter the data using a two-dimensional moving-average filter using the function `filter2`. The filter used here is a spatial running mean of 3×3 elements. We use only the subset `SRTM(400:600,650:850)` of the original data set to reduce computation time. We also remove the data at the edges of the DEM to eliminate filter artifacts.

```
F = 1/9 * ones(3,3);
SRTM = filter2(F, SRTM(750:850,700:800));
SRTM = SRTM(2:99,2:99);
```

The DEM is displayed as a pseudocolor plot using `pcolor` and the colormap `demcmap` included in the Mapping Toolbox. This colormap creates and assigns a colormap appropriate for elevation data since it provides land and sea colors in proportion to topography and bathymetry.

```
h = pcolor(SRTM);
demcmap(SRTM), colorbar
set(h,'LineStyle','none')
axis equal
title('Elevation [m]')
[r c] = size(SRTM);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The DEM is characterized by a horseshoe-shaped mountain range surrounding a valley descending towards the Southeast (Fig. 7.15a).

The SRTM subset is now analyzed for slopes and aspects. While we are working with DEMs on a regular grid, slope and aspect can be estimated as local derivatives by using centered finite differences in a local 3×3 neighborhood. Figure 7.13 shows the local neighborhood using the cell indexing convention of MATLAB. For calculating slope and aspect, we need two finite differences of the DEM elements z in x and y direction:

$$z_x = \frac{z_{r,c-1} - z_{r,c+1}}{2h}$$

and

$$z_y = \frac{z_{r-1,c} - z_{r+1,c}}{2h}$$

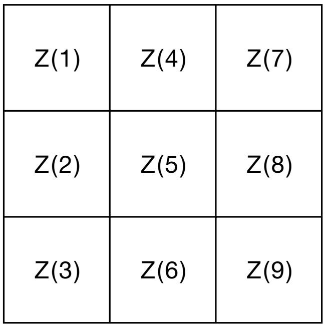


Fig. 7.13 Local neighborhood showing cell number convention of MATLAB.

where h is the cell size, which has the same unit as the elevation. Using the finite differences, the dimensionless slope is then calculated by

$$SLP_{DF} = \sqrt{z_x^{\ 2} + z_y^{\ 2}}$$

Other primary relief attributes such as the *aspect*, the *plan*, the *profile* and the *tangential curvature* can be derived in a similar way using finite differences (Wilson and Galant 2000). The function `gradientm` contained in the Mapping Toolbox calculates slope and aspect of a data grid `z` in units of degrees clockwise from North and up from the horizontal. Function `gradientm(z,refvec)` requires a three-element referencing vector `refvec`. The reference vector contains the number of cells per degree as well as the latitude and longitude of the upper-left (northwest) element of the data array. Since the SRTM digital elevation model is sampled at a 3-arc-second grid, $60\times 60/3=1200$ elements of the DEM correspond to one degree longitude or latitude. For simplicity, we ignore the actual coordinates of the SRTM subset in this example and use the indices of the DEM elements instead.

```
refvec = [1200 0 0];
[asp, slp] = gradientm(SRTM, refvec);
```

We display a pseudocolor map of the slope (in degrees) of the DEM (Fig 7.15b).

```
h = pcolor(slp);
colormap(jet), colorbar
set(h,'LineStyle','none')
```

```
axis equal
title('Slope [°]')
[r c] = size(slp);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

Flat areas can be found everywhere on the summits and the valley bottoms. The southeastern and south-southwestern sectors are relatively flat. Steeper slopes are concentrated in the center and the southwestern sector. Next, a pseudocolor map of the aspect is generated (Fig. 7.15c).

```
h = pcolor(asp);
colormap(hsv), colorbar
set(h,'LineStyle','none')
axis equal
title('Aspect')
[r c] = size(asp);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

This plot displays the aspect in units of degrees clockwise from North. For instance, mountain slopes facing North are displayed in red colors, whereas green areas depict East-facing slopes.

The aspect changes abruptly along the ridges of the mountain ranges where neighboring drainage basins are divided by *watersheds*. The Image Processing Toolbox includes the function `watershed` to detect the drainage divides and to label individual watershed regions or catchments by integer values, where the first watershed region is labeled 1, the elements labeled 2 belong to the second catchment, and so on.

```
watersh = watershed(SRTM);
```

The watershed regions are displayed by a pseudocolor plot where the labels of the regions are assigned by colors given in the color table `hsv` (Fig 7.15d).

```
h = pcolor(watersh);
colormap(hsv), colorbar
set(h,'LineStyle','none')
axis equal
title('Watershed')
[r c] = size(watersh);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The watersheds are displayed as series of red pixels. The largest catchment corresponds to the medium blue region in the center of the map. To the Northwest, this large catchment seems to be neighbored by three catchments (represented by green colors) without an outlet. As in this example, watershed often generates unrealistic results as watershed algorithms are sensitive to local minima that act as spurious sinks. We can detect such sinks in the SRTM data using the function imregionalmin. The output of this function is a binary image that has the value 1 corresponding to the elements of the DEM that belong to regional minima and the value of 0 otherwise.

```
sinks = 1*imregionalmin(SRTM);

h = pcolor(sinks);
colormap(gray)
set(h,'LineStyle','none')
axis equal
title('Sinks')
[r c] = size(sinks);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

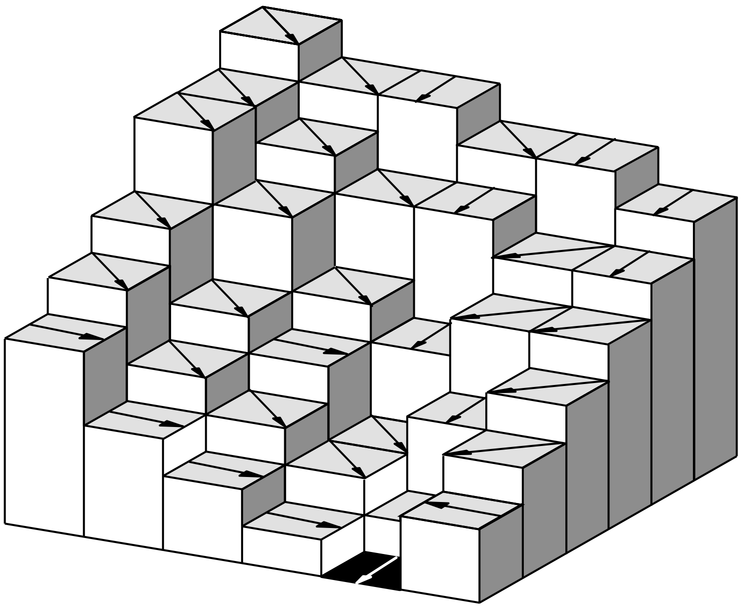
The pseudocolor plot of the binary image exhibits twelve local sinks represented by white pixels that are potentially the locations of non-outlet catchments and should be kept in mind while computing the following hydrological DEM attributes.

Flow accumulation (specific catchment area, upslope contributing area) is defined as the number of cells, or area, which contribute to runoff of a given cell (Fig. 7.14). In contrast to the local parameters slope and aspect, flow accumulation can only be determined from the global neighborhood. The principal operation is to add cell outflows iteratively to lower neighbors. Before cascading the cell outflows, we have to determine the individual gradients to each neighbor indexed by N. The array N contains indices for the eight neighboring cells according to the MATLAB convention as shown in Figure 17.3. We make use of the circshift function to access the neighboring cells. In the case of a two-dimensional matrix Z, the function circshift(Z,[r c]) circularly shifts the values in the matrix Z by an amount of rows and columns given by r and c, respectively. For example, circshift(Z,[1 1]) will circularly shift Z one row down and one column to the right. The individual gradients are calculated by

$$grad = \frac{z_{r+y,c+x} - z_{r,c}}{h}$$

for the eastern, southern, western, and northern neighbors (the so-called *rook's case*) and by

Elevation Mode



a

Flow Direction

↘	↘	↓	↘	↓	↓
↘	↘	↘	↓	↙	↓
↘	↘	↘	↓	↙	↙
↘	↘	→	↘	↓	↙
↘	↘	↘	↘	↓	↙
→	→	→	→	↓	←

Flow Accumulation

1	1	1	1	1	1
1	2	3	1	3	2
1	2	3	8	1	3
1	2	3	16	4	1
1	2	3	1	22	1
1	3	6	10	36	1

b c

Fig. 7.14 Schematic of calculation of flow accumulation by the D8 method

$$grad = \frac{z_{r+y,c+x} - z_{r,c}}{\sqrt{2h}}$$

for the diagonal neighbors (*bishop's case*). Herein, h is the cell size, $z_{r,c}$ is the elevation of the center cell and $z_{r+y,c+x}$ is the elevation of a neighbor. The cell indices x and y are obtained from the matrix `N`. The gradients are stored in a three-dimensional matrix `grads`, where `grads(:,:,1)` contains the gradients towards the neighbors in the East, `grads(:,:,2)` contains the gradients towards the neighbors in the Southeast, and so on. Negative gradients indicate outflow from the center to the respective neighbor. To obtain relative surface flow gradients are transformed by inverse tangent divided by 0.5π.

```
N = [0 -1;-1 -1;-1 0;+1 -1;0 +1;+1 +1;+1 0;-1 +1];
[a b] = size(SRTM);
grads = zeros(a,b,8);
for c = 2 : 2 : 8
   grads(:,:,c) = (circshift(SRTM,[N(c,1) N(c,2)]) ...
      -SRTM)/sqrt(2*90);
end
for c = 1 : 2 : 7
   grads(:,:,c) = (circshift(SRTM,[N(c,1) N(c,2)]) ...
      -SRTM)/90;
end
grads = atan(grads)/pi*2;
```

Since a center cell can have several downslope neighbors, water can flow in several directions. This phenomenon is called *divergent flow*. Early flow accumulation algorithms were based on the single-flow-direction method (D8 method, Fig. 7.14), which allows flow to only one of the cell's eight neighbors. This method cannot model divergence in ridge areas and tends to produce parallel flow lines in some examples. Here, we are illustrating the use of a multiple-flow-direction method, which allows flow from a cell to multiple neighbors. The flow to another neighbor corresponds to the individual gradient and is a fraction of the total outflow. Even though multiple-flow methods reveal more realistic results in most examples, they tend to cause dispersion in valleys where the flow should be more concentrated. Thus, a weighting factor w is introduced, which controls the relation of the outflows.

$$flow_i = \frac{grad_i^w}{\sum_{i=1}^{8} grad_i^w} \quad \text{for} \quad grad_i^w < 0$$

A recommended value for w is 1.1. Higher values will concentrate the flow in the direction of the steepest slope, while $w = 0$ would cause an extreme dispersion. In the following sequence of commands, we first select the gradients less than zero and multiply the result with the weight.

```
w = 1.1;
flow = (grads.*(-1*grads<0)).^w;
```

Then we are summing up the upslope gradients, i.e., the third dimension of `flow`. We replace values of 0 by the value of 1 that avoids the problems with division by zero.

```
upssum = sum(flow,3);
upssum(upssum==0) = 1;
```

We divide the flows by `upssum` to obtain fractional weights summing up to one. In our code, this is done separately for each layer of the 3D `flow` array by a `for` loop:

```
for i=1:8
   flow(:,:,i) = flow(:,:,i).*(flow(:,:,i)>0)./upssum;
end
```

The 2D matrix `inflowsum` will store the intermediate sums of inflows for each step of the iteration. The inflows are summed up to the total flow accumulation `flowac` at the end of each iteration. Initial values of `inflowsum` and `flowac` are provided by `upssum`.

```
inflowsum = upssum;
flowac = upssum;
```

Another 3D matrix `inflow` is now needed to store the intermediate inflow achieved by all neighbors:

```
inflow = grads*0;
```

Flow accumulation is terminated when there is no inflow, or translated into MATLAB code, we use a conditional `while` loop that terminates if `sum(inflowsum(:)) == 0`. The number of non-zero entries in `inflowsum` will decrease during each loop iteration. This is achieved by alternately updating `inflow` and `inflowsum`. Here, `inflowsum` is updated with the intermediate `inflow` of the neighbor(s) weighted by `flow` under the condition that the neighbors are contributing cells, i.e., where `grads` are positive. Since not all neighbors are contributing cells, the intermediate `inflowsum`, and also `inflow` is reduced. Flow accumulation `flowac` is increasing through the consecutive summation of the intermediate `inflowsum`.

```
while sum(inflowsum(:))>0
   for i = 1:8
      inflow(:,:,i) = circshift(inflowsum,[N(i,1) N(i,2)]);
   end
   inflowsum = sum(inflow.*flow.*grads>0,3);
   flowac = flowac + inflowsum;
end
```

We display the result as a pseudocolor map with log-scaled values (Fig 7.15 e).

```
h = pcolor(log(1+flowac));
colormap(flipud(jet)), colorbar
set(h,'LineStyle','none')
axis equal
title('Flow accumulation')
[r c] = size(flowac);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The plot displays areas with high flow accumulation in blue colors, whereas areas with low flow accumulation are displayed in red colors usually corresponding to ridges. We used a logarithmic scaling for mapping the flow accumulation to obtain a better representation of the results. The simplified algorithm to calculate flow accumulation introduced here can be used to analyze DEMs representing a sloping terrain. In flat terrains, where the slope becomes zero, no flow direction can be generated by our algorithm and thus flow accumulation stops. Such examples require more sophisticated algorithms to perform the analysis of DEMs. Furthermore, more advanced algorithms also include sink-filling routines to avoid spurious sinks that interrupt flow accumulation. Small depressions can be filled by smoothing as we have done it at the beginning of this chapter.

The first part of this chapter was about primary relief attributes. Secondary attributes of a DEM are functions of two or more primary attributes. Examples for secondary attributes are the wetness index and the stream power index. The *wetness index* is the log of the ratio of the specific catchment area and tangent of slope:

$$weti = \log\left(\frac{1 + flowac}{\tan(slp)}\right)$$

The term 1+*flowac* avoids the problems with calculating the logarithm of zero when `flowac=0`. The wetness index is used to predict the soil water content (*saturation*) due to the lateral water movement. The potential for water logging is usually high at lower elevations of a catchment with small slopes. Flat areas having a large upslope area have a high wetness index as compared with steep areas with small catchments. The wetness index `weti` is computed and displayed by

```
weti = log((1+flowac)./tand(slp));

h = pcolor(weti);
colormap(flipud(jet)), colorbar
set(h,'LineStyle','none')
axis equal
title('Wetness index')
[r c] = size(weti);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

In this graph, blue colors indicate high values of the wetness index, whereas red colors display low values (Fig. 7.15f). In our example, soils in the Southeast most likely have high water content due to the runoff from the large central valley and the terrain flatness.

The *stream power index* is another important secondary relief attribute which is frequently used in hillslope hydrology, geomorphology, soil science and related disciplines. As a measure of stream power it indicates sediment transport and erosion by water. It is defined as the product of the specific catchment area and tangent of the slope:

$$spi = flowac \cdot \tan(slp)$$

The potential for erosion is high when large quantities of water (calculated by the flow accumulation) are fast flowing due to an extreme slope. The following series of commands compute and display the stream power index:

```
spi = flowac.*tand(slp);

h = pcolor(log(1+spi));
colormap(jet), colorbar
set(h,'LineStyle','none')
axis equal
title('Stream power index')
[r c] = size(spi);
axis([1 c 1 r])
set(gca,'TickDir','out');
```

The wetness and stream power indices are particularly useful in large-scale terrain analysis, i.e., digital elevation models sampled on intervals of less

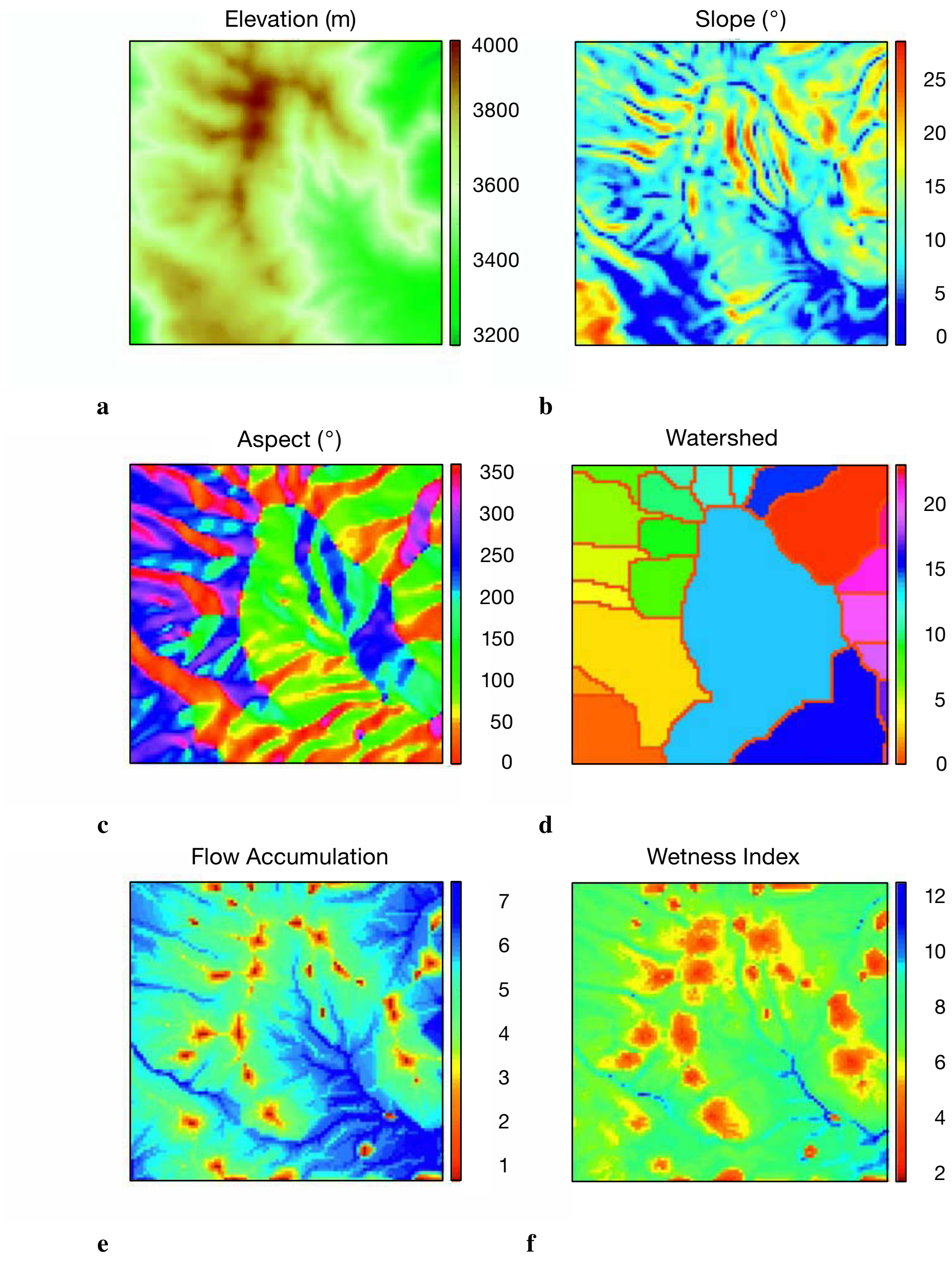


Fig. 7.15 Display of a subset of the SRTM data set used in Chapter 7.5 and primary and secondary attributes of the digital elevation model; **a** elevation, **b** slope, **c** aspect, **d** watershed, **e** flow accumulation and **f** wetness index.

than 30 meters. Though we have calculated `weti` and `spi` from a medium-scale DEM, we have to expect scale dependency of these attributes in our terrain analysis example.

This chapter has illustrated the use of basic tools to analyze digital elevation models. More detailed introductions to digital terrain modelling are given by the book by Wilson & Galant (2002). Furthermore, the article by Freeman (1991) provides a comprehensive summary and introduction to advanced algorithms for flow accumulation.

7.11 Geostatistics and Kriging (by R. Gebbers)

Geostatistics describes the autocorrelation of one or more variables in the 1D, 2D, and 3D space or even in 4D space-time, to make predictions at unobserved locations, to give information about the accuracy of prediction and to reproduce spatial variability and uncertainty. The shape, the range, and the direction of the spatial autocorrelation are described by the *variogram*, which is the main tool in linear geostatistics. The origins of geostatistics can be dated back to the early 50's when the South African mining engineer Daniel G. Krige first published an interpolation method based on spatial dependency of samples. In the 60's and 70's, the French mathematician George Matheron developed the *theory of regionalized variables* which provides the theoretical foundations of Kriges's more practical methods. This theory forms the basis of several procedures for the analysis and estimation of spatially dependent variables, which Matheron called *geostatistics*. Matheron as well coined the term *kriging* for spatial interpolation by geostatistical methods.

Theorical Background

A basic assumption in geostatistics is that a spatiotemporal process is composed of deterministic and stochastic components (Fig. 7.16). The deterministic components can be global and local trends (sometimes called *drifts*). The stochastic component is formed by a purely random and an autocorrelated part. An autocorrelated component implies that on average, closer observations are more similar than more distant observations. This behavior is described by the *variogram* where squared differences between observations are plotted against their separation distances. The fundamental idea of D. Krige was to use the variogram for interpolation as means to determine the magnitude of influence of neighboring observations when predicting

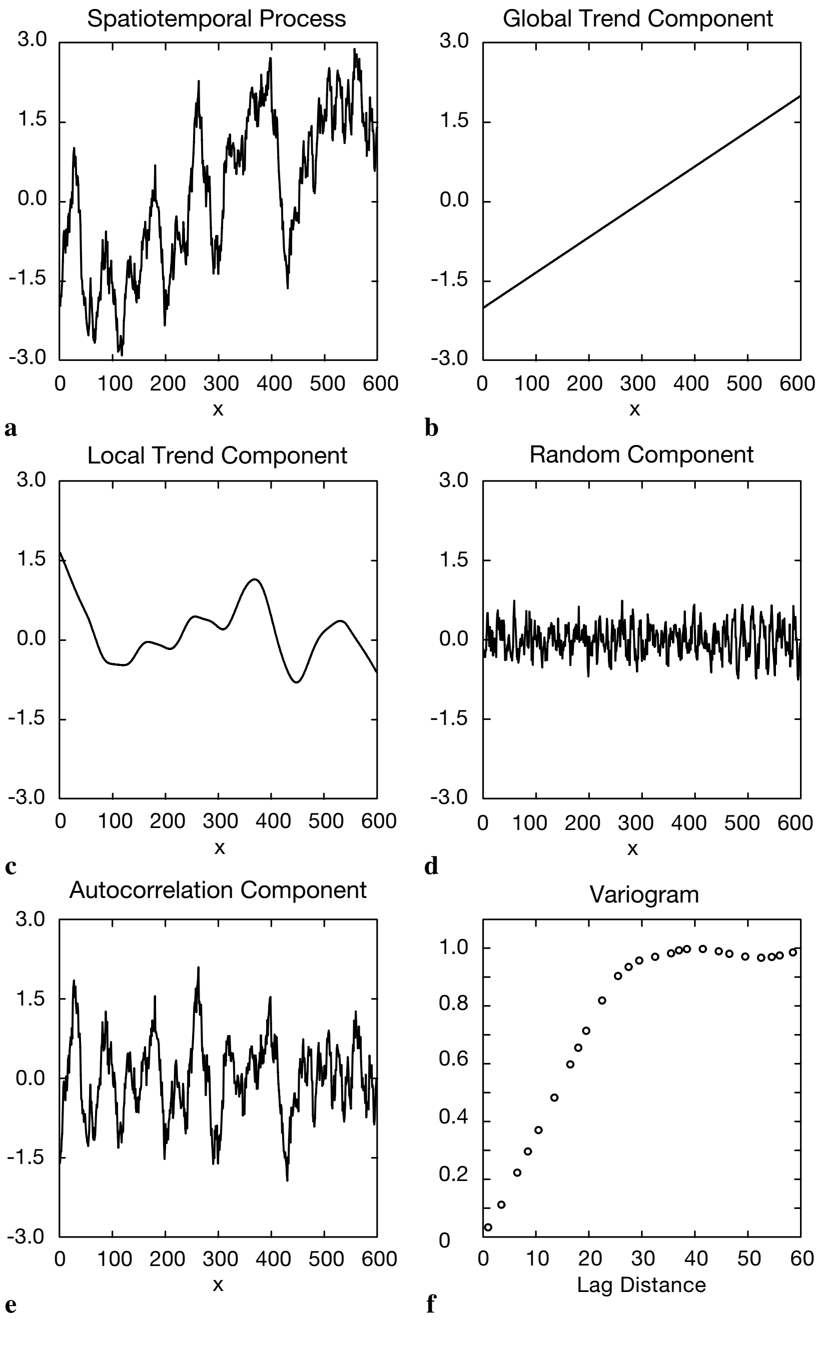


Fig. 7.16 Components of a spatiotemporal process and the variogram. The variogram (**f**) should only be derived from the autocorrelated component.

values at unobserved locations. Basic linear geostatistics includes two main procedures: variography for modeling the variogram and kriging for interpolation.

Preceding Analysis

Because linear geostatistics as presented here is a parametric method, the underlying assumptions have to be checked by a preceding analysis. As other parametric methods, linear geostatistics is sensitive to outliers and deviations from normal distribution. First, after opening the data file *geost_dat.mat* containing *xyz* data triplets we plot the sampling locations. Doing this, we can check point distribution and detect gross errors on the data coordinates *x* and *y*.

```
load geost_dat.mat

plot(x,y,'.')
```

Checking of the limits of the observations *z* can be done by

```
min(z)

ans =
    3.7199

max(z)

ans =
    7.8460
```

For linear geostatistics, the observations *z* should be gaussian distributed. In most cases, this is only tested by visual inspection of the histogram because statistical tests are often too sensitive if the number of samples exceed ca. 100. In addition, one can calculate skewness and kurtosis of the data.

```
hist(z)

skewness(z)

ans =
    0.2568

kurtosis(z)

ans =
    2.5220
```

A flat-topped or multiple peaks distribution suggests that there is more than one population in your data set. If these populations can be related to continuous areas they should be treated separately. Another reason for multiple peaks can be preferential sampling of areas with high and/or low values. This happens usually due to some a priori knowledge and is called cluster effect. Handling of the cluster effect is described in Deutsch and Journel (1998) and Isaaks and Srivastava (1998).

Most problems arise from positive skewness (long upper tail). According to Webster and Oliver (2001), one should consider root transformation if skewness is between 0.5 and 1, and logarithmic transformation if skewness exceeds 1. A general formula of transformation is:

$$z^* = \begin{cases} \dfrac{(z+m)^k - 1}{k} & \text{for } k \neq 0 \\ \log(z+m) & \text{for } k = 0 \end{cases}$$

for $\min(z)+m>0$. This is the so called Box-Cox transform with the special case $k=0$ when a logarithm transformation is used. In the logarithm transformation, m should be added when z values are zero or negative. Interpolation results of power-transformed values can be backtransformed directly after kriging. The backtransformation of log-transformed values is slightly more complicated and will be explained later. The procedure is known as *lognormal kriging*. It can be important because lognormal distributions are not unusual in geology.

Variography with the Classical Variogram

The variogram describes the spatial dependency of referenced observations in a one or multidimensional space. While usually we do not know the true variogram of the spatial process we have to estimate it from observations. This procedure is called variography. Variography starts with calculating the *experimental variogram* from the raw data. In the next step, the experimental variogram is summarized by the variogram estimator. Variography finishes with fitting a variogram model to the *variogram estimator*. The *experimental variogram* is calculated as the difference between pairs of the observed values depending on the *separation vector h* (Fig. 7.17). The classical experimental variogram is given by the *semivariance*,

$$\gamma(h) = 0.5 \cdot (z_x - z_{x+h})^2$$

where z_x is he observed value at location x and z_{x+h} is he observed value at another point within a distance h. The length of the separation vector h is

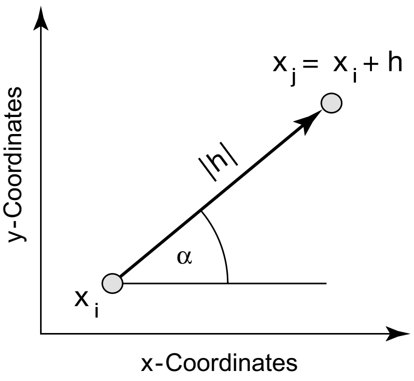


Fig. 7.17 Separation vector *h* between two points.

called *lag distance* or simply *lag*. The correct term for $\gamma(h)$ is *semivariogram* (or *semivariance*), where *semi* refers to the fact that it is half of the variance of the difference between z_x and z_{x+h}. It is, nevertheless, the variance per point when points are considered as in pairs (Webster and Oliver, 2001). Conventionally, $\gamma(h)$ is termed *variogram* instead of semivariogram and so we do at the end of this chapter. To calculate the experimental variogram we first have to build pairs of observations. This is done by typing

```
[X1,X2] = meshgrid(x);
[Y1,Y2] = meshgrid(y);
[Z1,Z2] = meshgrid(z);
```

The matrix of separation distances `D` between the observation points is

```
D = sqrt((X1 - X2).^2 + (Y1 - Y2).^2);
```

where `srqt` is the square root of the data. Then we get the experimental variogram `G` as half the squared differences between the observed values:

```
G = 0.5*(Z1 - Z2).^2;
```

We used the MATLAB capability to vectorize commands instead of using *for* loops to run faster. However, we have computed n_2 pairs of observations although only $n(n-1)/2$ pairs are required. For large data sets, e.g., more than 3000 data points, the software and physical memory of the computer may become a limiting factor. For such cases, a more efficient way of programming is described in the user manual of the software SURFER (2002). The plot of the experimental variogram is called the *variogram cloud* (Fig. 7.18). We get this after extracting the lower triangular portions of the `D` and `G` arrays.

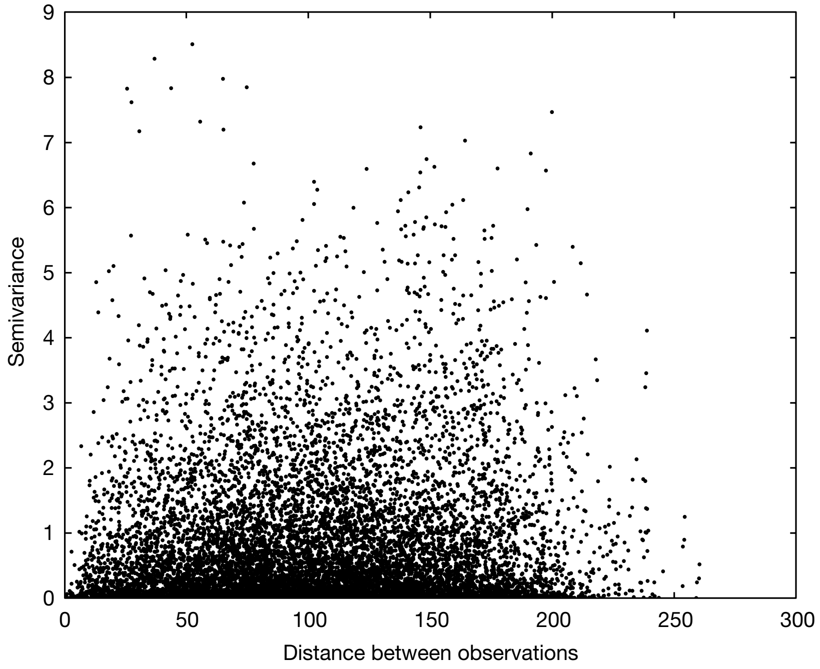


Fig. 7.18 Variogram cloud: Plot of the experimental variogram (half squared difference between pairs of observations) versus the lag distance (separation distance of the pairs).

```
indx = 1:length(z);
[C,R] = meshgrid(indx);
I = R > C;

plot(D(I),G(I),'.' )
xlabel('lag distance')
ylabel('variogram')
```

The variogram cloud gives you an impression of the dispersion of values at the different lags. It might be useful to detect outliers or anomalies, but it is hard to judge from it whether there is any spatial correlation, what form it might have, and how we could model it (Webster and Oliver, 2001). To obtain a clearer view and to prepare variogram modeling the experimental variogram is replaced by the variogram estimator in the next section.

The *variogram estimator* is derived from the experimental variograms to summarize their central tendency (similar to the descriptive statistics derived from univariate observations, Chapter 3.2). The classical variogram estimator is the averaged empirical variogram within certain distance classes or bins defined by multiples of the lag interval. The classification of separation distances is visualized in Figure 7.19.

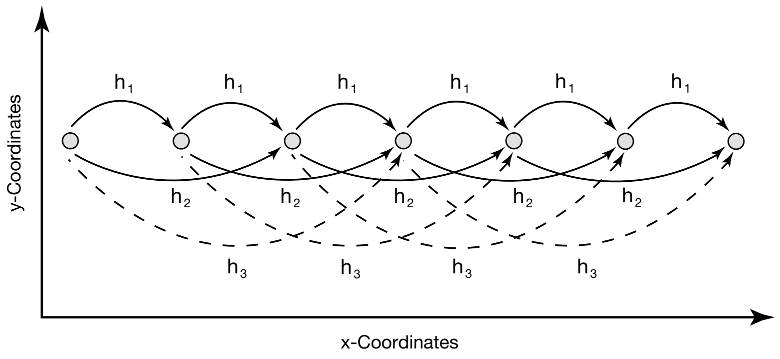


Fig. 7.19 Classification of separation distances in the case of equally spaced observations along a line. The lag interval is h_1 and h_2, h_3 etc. are multiples of the lag interval.

The variogram estimator is calculated by:

$$\gamma_E(h) = \frac{1}{2 * N(h)} \cdot \sum_{i=1}^{N(h)} (z_{xi} - z_{xi+h})^2$$

where $N(h)$ is the number of pairs within the lag interval h.

First, we need an idea about a suitable lag interval h. If you have sampled on a regular grid, you can use the length of a grid cell. If the samples have irregular spacings, as in our case, the mean minimum distance of pairs is a good starting point for the lag interval (Webster and Oliver 2001). To calculate the mean minimum distance of pairs we have to replace the diagonal of the lag matrix `D` zeros with `NaN`'s, otherwise the minimum distance will be zero:

```
D2 = D.*(diag(x*NaN)+1);
lag = mean(min(D2))

lag =
    8.0107
```

While the estimated variogram values tend to become more erratic with increasing distances, it is important to define a maximum distance which limits the calculation. As a rule of thumb, the half maximum distance is suitable range for variogram analysis. We obtain the half maximum distance and the maximum number of lags by:

```
hmd = max(D(:))/2

hmd =
  130.1901
```

```
max_lags = floor(hmd/lag)

max_lags =
    16
```

Then the separation distances are classified and the classical variogram estimator is calculated:

```
LAGS = ceil(D/lag);

for i = 1 : max_lags
    SEL = (LAGS == i);
    DE(i) = mean(mean(D(SEL)));
    PN(i) = sum(sum(SEL == 1))/2;
    GE(i) = mean(mean(G(SEL)));
end
```

where `SEL` is the selection matrix defined by the lag classes in `LAG`, `DE` is the mean lag, `PN` is the number of pairs and `GE` is the variogram estimator. Now we can plot the classical variogram estimator (variogram versus mean separation distance) together with the population variance:

```
plot(DE,GE,'.' )
var_z = var(z);
b = [0 max(DE)];
c = [var_z var_z];

hold on

plot(b,c, '--r')
yl = 1.1 * max(GE);
ylim([0 yl])
xlabel('Averaged distance between observations')
ylabel('Averaged semivariance')

hold off
```

The variogram in Figure 7.20 shows a typical behavior. Values are low at small separation distances (near the origin), they are increasing with increasing distances, than reaching a plateau (*sill*) which is close to the population variance. This indicates that the spatial process is correlated over short distances while there is no spatial dependency over longer distances. The length of the spatial dependency is called the *range* and is defined by the separation distance where the variogram reaches the sill.

The *variogram model* is a parametric curve fitted to the variogram estimator. This is similar to frequency distribution fitting (see Chapter 3.5), where the frequency distribution is modeled by a distribution type and its parameters (e.g., a normal distribution with its mean and variance). Due to

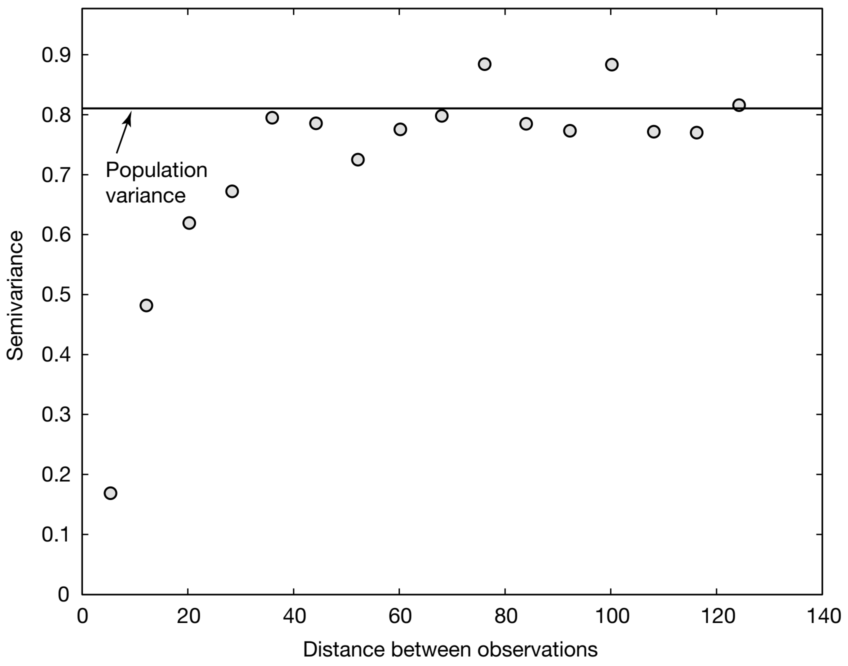


Fig. 7.20 The classical variogram estimator (gray circles) and the population variance (solid line).

theoretical reasons, only functions with certain properties should be used as variogram models. Common *authorized models* are the spherical, the exponential and the linear model (more models can be found in the literature).

Spherical model:

$$\gamma_{\text{sph}}(h) = \begin{cases} c \cdot \left(1.5 \cdot \dfrac{h}{a} - 0.5\left(\dfrac{h}{a}\right)^3\right) & \text{for } 0 \le h \le a \\ c & \text{for } h > a \end{cases}$$

Exponential model:

$$\gamma_{\text{exp}}(h) = c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a}\right)\right)$$

Linear model:

$$\gamma_{\text{lin}}(h) = b \cdot h$$

where c is the sill, a is the range, and b is the slope (in the case of the linear model). The parameters c and a or b have to be modified when a variogram model is fitted to the variogram estimator. The so called *nugget effect* is a special type of variogram model. In practice, when extrapolating the variogram towards separation distance zero, we often observe a positive intercept on the ordinate. This is called the nugget effect and it is explained by measurement errors and by small scale fluctuations (*nuggets*), which are not captured due to too large sampling intervals. Thus, we sometimes have expectations about the minimum nugget effect from the variance of repeated measurements in the laboratory or other previous knowledge. More details about the nugget effect can be found in Cressie (1993) and Kitanidis (1997). If there is a nugget effect, it can be added to the variogram model. An exponential model with a nugget effect looks like this:

$$\gamma_{\text{exp+nug}}(h) = c_0 + c \cdot \left(1 - \exp\left(-3 \cdot \frac{h}{a}\right)\right)$$

where c_0 is the nugget effect.

We can even combine more variogram models, e.g., two spherical models with different ranges and sills. These combinations are called *nested models*. During variogram modeling the components of a nested model are regarded as spatial structures which should be interpreted as the results of geological processes. Before we discuss further aspects of variogram modeling let us just fit some models to our data. We are beginning with a spherical model without nugget, than adding an exponential and a linear model, both with nugget variance:

```
plot(DE,GE,'o','MarkerFaceColor',[.6 .6 .6])
var_z = var(z);
b = [0 max(DE)];
c = [var_z var_z];
hold on
plot(b,c,'--r')
xlim(b)
yl = 1.1*max(GE);
ylim([0 yl])

% Spherical model with nugget
nugget = 0;
sill = 0.803;
```

```
range = 45.9;
lags = 0:max(DE);
Gsph = nugget + (sill*(1.5*lags/range - 0.5*(lags/...
   range).^3).*(lags<=range) + sill*(lags>range));
plot(lags,Gsph,':g')

% Exponential model with nugget
nugget = 0.0239;
sill = 0.78;
range = 45;
Gexp = nugget + sill*(1 - exp(-3*lags/range));
plot(lags,Gexp,'-.b')

% Linear model with nugget
nugget = 0.153;
slope = 0.0203;
Glin = nugget + slope*lags;
plot(lags,Glin,'-m')
xlabel('Distance between observations')
ylabel('Semivariance')
legend('Variogram estimator','Population variance',...
'Sperical model','Exponential model','Linear model')
hold off
```

Variogram modeling is very much a point of discussion. Some advocate *objective* variogram modeling by automated curve fitting, using a weighted least squares, maximum likelihood or maximum entropy method. Contrary to this it is often argued that the geological knowledge should be included in the modeling process and thus, fitting by eye is recommended. In many cases the problem in variogram modeling is much less the question of the appropriate procedure but a question of the quality of the experimental variogram. If the experimental variogram is good, both procedures will yield similar results.

Another question important for variogram modeling is the intended use of the model. In our case, the linear model seems not to be appropriate (Fig. 7.21). At a closer look we can see that the linear model fits reasonably well over the first three lags. This can be sufficient when we use the variogram model only for kriging, because in kriging the nearby points are the most important for the estimate (see discussion of kriging below). Thus, different variogram models with similar fits near the origin will yield similar kriging results when sampling points are regularly distributed. If you are interested in describing the spatial structures it is another case. Then it is important to find a suitable model over all lags and to determine the sill and the range accurately. A collection of geologic case studies in Rendu and Readdy (1982) show how process knowledge and variography can be linked. Good guidelines to variogram modeling are given by Gringarten

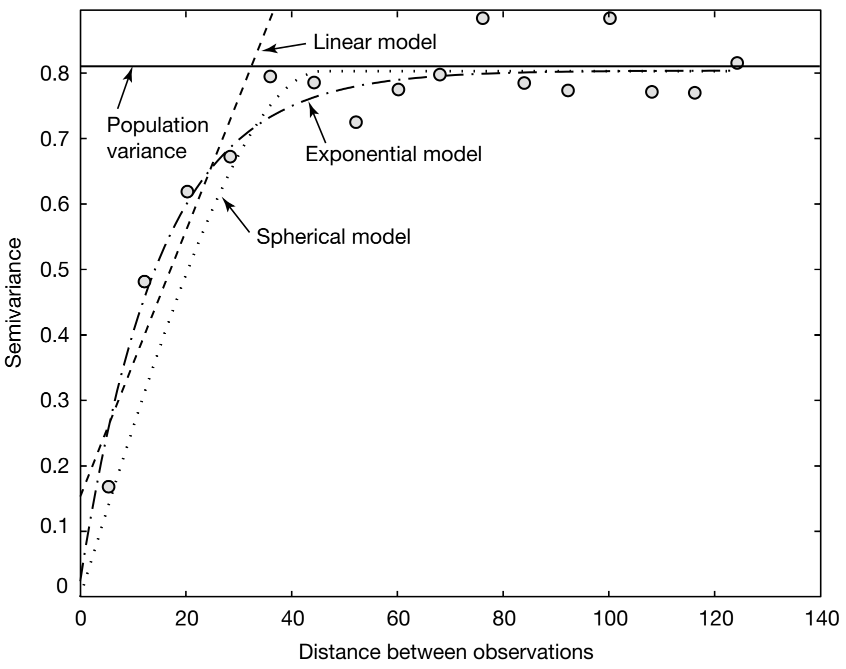


Fig. 7.21 Variogram estimator (gray circles), population variance (solid line), spherical, exponential, and linear models (dashed lines).

and Deutsch (2001) and Webster and Oliver (2001). We will now briefly discuss some more aspects of variography.

- *Sample size* – As in any statistical procedure you need as large a sample as possible to get a reliable estimate. For variography it is recommended to have more than 100 to 150 samples (Webster and Oliver 2001). If you have less, you should consider computing a maximum likelihood variogram (Pardo-Igúzquiza and Dowd 1997).

- *Sampling design* – To get a good estimation at the origin of the variogram sampling design should include observations over small distances. This can be done by a nested design (Webster and Oliver 2001). Other designs were evaluated by Olea (1984).

- *Anisotropy* – Thus far now we have assumed that the structure of spatial correlation is independent of direction. We have calculated *omnidirectional variograms* ignoring the direction of the separation vector *h*. In a more thorough analysis, the variogram should not only be discretized in distance but also in direction (directional bins). Plotting *directional variograms*, usually in four directions, we sometimes can observe different ranges (*geometric anisotropy*), different scales (*zonal anisotropy*), and different shapes (indicating a trend). The treatment of anisotropy needs a highly interactive graphical user interface, e.g., VarioWin by Panatier (1996) which is beyond the scope of this book.

- *Number of pairs and the lag interval* – In the calculation of the classical variogram estimator it is recommended to use more than 30 to 50 pairs of points per lag interval (Webster and Oliver 2001). This is due to the sensitivity to outliers. If there are fewer pairs, the lag interval should be enlarged. The lag spacing has not necessarily to be uniform, it can be chosen individually for each distance class. It is also an option to work with overlapping classes, in this case the *lag width* (*lag tolerance*) has to be defined. On the other hand, increasing the lag width can cause unnecessary smoothing and detail is lost. Thus, the separation distance and the lag width have to be chosen with care. Another option is to use a more robust variogram estimator (Cressie 1993, Deutsch and Journel 1998).

- *Calculation of separation distance* – If your observations are covering a large area, let us say more than 1000 km^2, spherical distances should be calculated instead of the Pythagorean distances from a plane cartesian coordinate system.

Kriging

Now we will interpolate the observations on a regular grid by *ordinary point kriging* which is the most popular kriging method. Ordinary point kriging uses a weighted average of the neighboring points to estimate the value of an unobserved point:

$$\hat{z}_{x0} = \sum_{i}^{N} \lambda_i \cdot z_{xi}$$

where λ_i are the weights which have to be estimated. The sum of the weights should be one to guarantee that the estimates are unbiased:

$$\sum_{i}^{N} \lambda_i = 1$$

The expected (average) error of the estimation has to be zero. That is:

$$E(\hat{z}_{x0} - z_{x0}) = 0$$

where z_{x0} is the true, but unknown value. After some algebra, using the preceding equations, we can compute the mean-squared error in terms of the variogram:

$$E\left((\hat{z}_{x0} - z_{x0})^2\right) = 2\sum_{i=1}^{N} \lambda_i \gamma(x_i, x_0) - \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j \gamma(x_i, x_j)$$

where E is the estimation or *kriging variance*, which has to be minimized, $\gamma(x_i, x_0)$ is the variogram (semivariance) between the data point and the unobserved, $\gamma(x_i, x_j)$ is the variogram between the data points x_i and x_j, and λ_i and λ_j are the weights of the ith and jth data point.

For kriging we have to minimize this equation (quadratic objective function) satisfying the condition that the sum of weights should be one (linear constraint). This optimization problem can be solved using a Lagrange multiplier ν resulting in the *linear kriging system* of N+1 equations and N+1 unknowns:

$$\sum_{i=1}^{N} \lambda_i \gamma(x_i, x_j) + \nu = \gamma(x_i, x_0)$$

After obtaining the weights λ_i, the kriging variance is given by

$$\sigma^2(x_0) = \sum_{i=1}^{N} \lambda_i \gamma(x_i, x_0) + \nu(x_0)$$

The kriging system can be presented in a matrix notation:

$$G\_\mathrm{mod} \cdot E = G\_R$$

where

$$G\_\mathrm{mod} = \begin{bmatrix} 0 & \gamma(x_1,x_2) & \dots & \gamma(x_1,x_N) & 1 \\ \gamma(x_2,x_1) & 0 & \dots & \gamma(x_2,x_N) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \gamma(x_N,x_1) & \gamma(x_N,x_2) & \dots & 0 & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix}$$

is the matrix of the coefficients, these are the modeled variogram values for the pairs of observations. Note that on the diagonal of the matrix, where separation distance is zero, the value of γ vanishes.

$$E = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ \nu \end{bmatrix}$$

is the vector of the unknown weights and the Lagrange multiplier.

$$G\_R = \begin{bmatrix} \gamma(x_1, x_0) \\ \gamma(x_2, x_0) \\ \vdots \\ \gamma(x_N, x_0) \\ 1 \end{bmatrix}$$

is the right-hand-side vector. To obtain the weights and the Lagrange multiplier the matrix *G_mod* is inverted:

$$E = G\_\mathrm{mod}^{-1} \cdot G\_R$$

The kriging variance is given by

$$\sigma^2 = G\_R^{-1} \cdot E$$

For our calculations with MATLAB we need the matrix of coefficients derived from the distance matrix *D* and a variogram model. *D* was calculated in the variography section above and we use the exponential variogram model with a nugget, sill and range from the previous section:

```
G_mod = (nugget + sill*(1 - exp(-3*D/range))).*(D>0);
```

Then we get the number of observations and add a column and row vector of all ones to the `G_mod` matrix and a zero at the lower left corner:

```
n = length(x);
G_mod(:,n+1) = 1;
G_mod(n+1,:) = 1;
G_mod(n+1,n+1) = 0;
```

Now the G_mod matrix has to be inverted:

```
G_inv = inv(G_mod);
```

A grid with the locations of the unknown values is needed. Here we use a grid cell size of five within a quadratic area ranging from 0 to 200 in *x* and *y* direction. The coordinates are created in matrix form by:

```
R = 0 : 5 : 200;
[Xg1,Xg2] = meshgrid(R,R);
```

and converted to vectors by:

```
Xg = reshape(Xg1,[],1);
Yg = reshape(Xg2,[],1);
```

Then we allocate memory for the kriging estimates Zg and the kriging variance s2_k by:

```
Zg = Xg * NaN;
s2_k = Xg * NaN;
```

Now we are kriging the unknown at each grid point:

```
for k = 1 : length(Xg)
    DOR = ((x - Xg(k)).^2 + (y - Yg(k)).^2).^0.5;
    G_R = (nugget + sill*(1 - exp(-3*DOR/range))).*(DOR>0);
    G_R(n+1) = 1;
    E = G_inv * G_R;
    Zg(k) = sum(E(1:n,1).*z);
    s2_k(k) = sum(E(1:n,1).*G_R(1:n,1))+E(n+1,1);
end
```

Here, the first command computes the distance between the grid points (Xg,Yg) and the observation points (x,y). Then we build the right-hand-side vector of the kriging system by using the variogram model G_R and add one to the last row. We next obtain the matrix E with the weights and the lagrange multiplier. The estimate Zg at each point k is the weighted sum of the observations z. Finally, the kriging variance s2_k of the grid point is computed. We plot the results. First, we create a grid of the kriging estimate and the kriging variance:

```
r = length(R);
Z = reshape(Zg,r,r);
SK = reshape(s2_k,r,r);
```

A subplot on the left presents the kriged values:

```
subplot(1,2,1)
h = pcolor(Xg1,Xg2,Z);
set(h,'LineStyle','none')
```

```
axis equal
ylim([0 200])
title('Kriging Estimate')
xlabel('x-Coordinates')
ylabel('y-Coordinates')
colorbar
```

The left subplot presents the kriging variance:

```
subplot(1,2,2)
h = pcolor(Xg1,Xg2,SK);
set(h,'LineStyle','none')
axis equal
ylim([0 200])
title('Kriging Variance')
xlabel('x-Coordinates')
ylabel('y-Coordinates')
colorbar
hold on
```

and we are overlaying the sampling positions:

```
plot(x,y,'ok')
hold off
```

The kriged values are shown in Figure 7.22a. The kriging variance depends only on the distance from the observations and not on the observed values (Fig. 7.22b). Kriging reproduces the population mean when observations are beyond the range of the variogram, at the same time kriging variance increases (lower right corner of the maps in Figure 7.22). The kriging variance can be used as a criterion to improve sampling design and it is needed for backtransformation in lognormal kriging. Back-transformation for lognormal kriging is done by:

$$y(x_0) = \exp(z(x_0) + 0.5 \cdot \sigma^2(x_0) - \nu)$$

Discussion of Kriging

Point kriging as presented here is an exact interpolator. It reproduces exactly the values at an observation point, even though a variogram with a nugget effect is used. Smoothing can be caused by including the variance of the measurement errors (see Kitanidis 1997) and by *block kriging* which averages the observations within a certain neighborhood (block). While kriging variance depends only on the distance between the observed and the unobserved locations it is primary a measure of density of information (Wackernagel 2003). The accuracy of kriging is better evaluated by cross-validation using

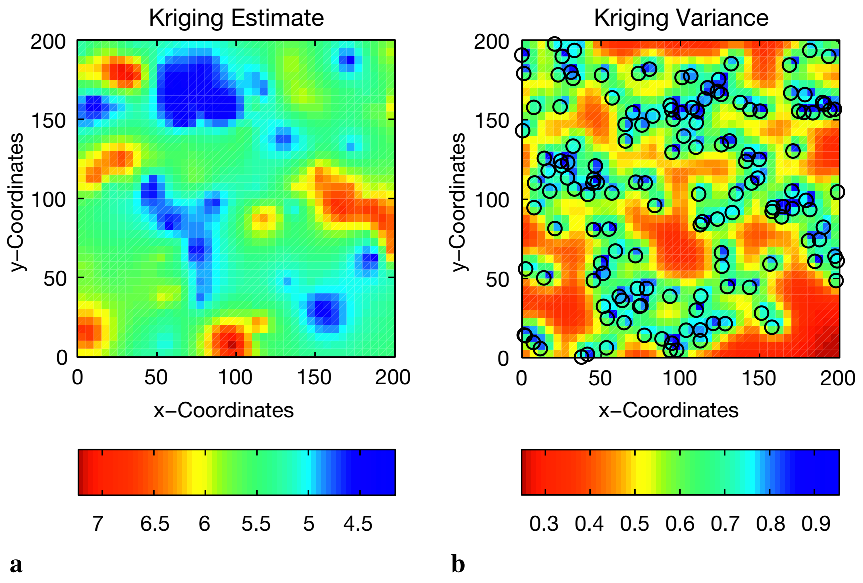


Fig. 7.22 Interpolated values on a regular grid by ordinary point kriging using **a** an exponential variogram model; **b** kriging variance as a function of the distance from the observations (empty circles).

a resampling method or surrogate test (Chapter 4.6 and 4.7). The influence of the neighboring observations on the estimation depends on their configuration. Webster and Oliver (2001) summarize: Near points carry more weight than more distant ones; the relative weight of a point decreases when the number of points in the neighborhood increases; clustered points carry less weight individually than isolated ones at the same distance; data points can be screened by ones lying between them and the target. Sampling design for kriging is different from the design which might be optimal for variography. A regular grid, triangular or quadratic, can be regarded as optimum.

The MATLAB code presented here is a straightforward implementation of the kriging system presented in the formulas above. In professional programs the number of data points entering the *G_mod* matrix are restricted as well as the inversion of *G_mod* is avoided by working with the covariances instead of the variograms (Webster and Oliver 2001, Kitanidis 1997). For those who are interested in programming and in a deeper understanding of algorithms, Deutsch and Journel (1992) is a must. The best internet source is the homepage of AI-GEOSTATISTICS:

```
http://www.ai-geostats.org
```

Recommended Reading

Cressie N (1993) Statistics for Spatial Data, Revised Edition. John Wiley & Sons, New York

Davis JC (2002) Statistics and Data Analysis in Geology, third edition. John Wiley and Sons, New York

Deutsch CV, Journel AG (1998) GSLIB – Geostatistical Software Library and User's Guide, Second edition. Oxford University Press, Oxford

Freeman TG (1991) Calculating Catchment Area with Divergent Flow Based on a Regular Grid. Computers and Geosciences 17:413–422

Gringarten E, Deutsch CV (2001) Teacher's Aide Variogram Interpretation and Modeling. Mathematical Geology 33:507–534

Isaaks E, Srivastava M (1989) An Introduction to Applied Geostatistics. Oxford University Press, Oxford

Gringarten E, Deutsch CV (2001) Teacher's Aide Variogram Interpretation and Modeling. Mathematical Geology 33:507–534

Kitanidis P (1997) Introduction to Geostatistics – Applications in Hydrogeology. Cambridge University Press, Cambridge

Olea RA (1984) Systematic Sampling of Spatial Functions. Kansas Series on Spatial Analysis 7, Kansas Geological Survey, Lawrence, KS

Pannatier Y (1996) VarioWin – Software for Spatial Data Analysis in 2D, Springer, Berlin Heidelberg New York

Pardo-Igúzquiza E, Dowd PA (1997) AMLE3D: A Computer Program for the Interference of Spatial Covariance Parameters by Approximate Maximum Likelihood Estimation. Computers and Geosciences 23:793–805

Rendu JM, Readdy L (1982) Geology and Semivariogram – A Critical Relationship. In: Johnson TB, Barns RJ (eds) Application of Computer & Operation Research in the Mineral Industry. 17th Intern. Symp. American Institute of Mining. Metallurgical and Petroleum Engineers, New York, pp. 771–783

Sandwell DT (1987) Biharmonic Spline Interpolation of GEOS-3 and SEASAT Altimeter data. Geophysical Research Letters 2:139–142

Swan ARH, Sandilands M (1995) Introduction to Geological Data Analysis. Blackwell Sciences, Oxford

The Mathworks (2006) Mapping Toolbox User's Guide – For the Use with MATLAB®. The MathWorks, Natick, MA

Golden Software, Inc. (2002) Surfer 8 (Surface Mapping System). Golden, Colorado

Wackernagel H. (2003) Multivariate Geostatistics: An Introduction with Applications. Third, completely revised edition. Springer, Berlin Heidelberg New York

Webster R, Oliver MA (2001) Geostatistics for Environmental Scientists. John Wiley & Sons, New York

Wessel P, Bercovici D (1998) Gridding with Splines in Tension: A Green Function Approach. Mathematical Geology 30:77–93

Wilson JP, Gallant JC (2000) Terrain Analysis, Principles and Applications. John Wiley and Sons, New York