University of California, Los Angeles
Department of Statistics

Statistics C173/C273 **Instructor: Nicolas Christou**

Spatial prediction.

(An Introduction to Applied Geostatistics, by Edward Isaaks, Nohan Srivastava: New York, Oxford University Press (1990)

Applied Spatial Data Analysis with R, by Roger Bivand, Edzer Pebesma, Virgilio Gomez-Rubio: Springer (2008))

- **Spatial prediction:**

  a. One of the goals of geostatistics is to predict values at unsampled loacations. This will create a "raster map" of the area under consideration.

  b. Idea: The value at an unsampled location is related to the values at sampled locations. If not, then the predicted value can be simply the average of the sampled locations.

  c. There is a relationship between the point being estimated and the samples around it based on its *location* and the distance between the points.

  d. Different methods for spatial prediction. All methods assume that nearby samples are more important (they have more weight) in predicting the unknown value.

- **Some simple methods of spatial prediction:**

  1. Method of polygons.
  2. Triangulation.
  3. Weighted average based on triangulation.
  4. Inverse distance method.

A. Method of polygons:
Very simple. The point being predicted is equal to the nearest observed sampled data point.

B. Triangulation

Form a triangle around the point being estimated with three observed data points. For these three data points we know their x, y coordinates and the value z of the variable of interest as follows:

x	y	z
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_3	z_3

We use these points to find the equation of a plane by solving the system of equations:

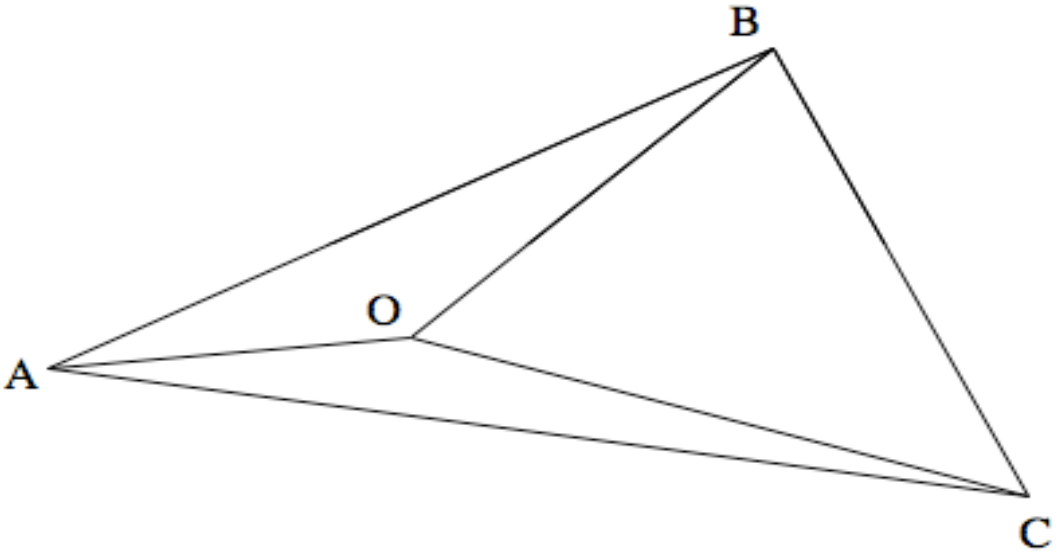
$$\begin{aligned} ax_1 + by_1 + c &= z_1 \\ ax_2 + by_2 + c &= z_2 \\ ax_3 + by_3 + c &= z_3 \end{aligned}$$

Suppose now we want to find the value of z_0 at location given by the coordinates (x_0, y_0) of a point within the triangle.

Example:
Predict the value at (65, 137) if the surrounding three points are: $z_1 = 696$ at (63, 140), $z_2 = 227$ at (64, 129), and $z_3 = 606$ at (71, 140).

C. Weighted average based on triangulation:

Suppose we want to predict the value at point O based on the values z_A, z_B, z_C from the figure below:



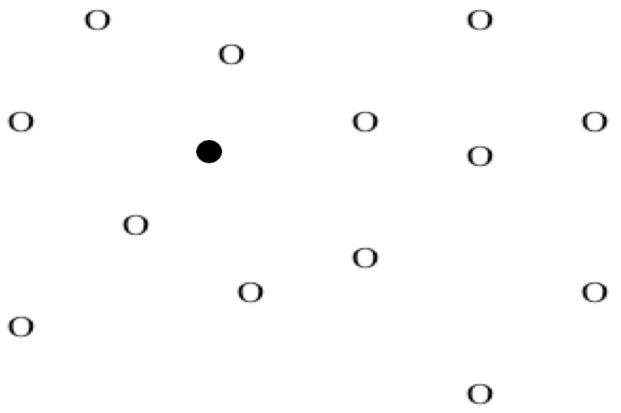
A weighted average is used as follows:

$$\hat{z}_O = z_A \times \frac{\text{Area}(OBC)}{\text{Area}(ABC)} + z_B \times \frac{\text{Area}(OCA)}{\text{Area}(ABC)} + z_C \times \frac{\text{Area}(OAB)}{\text{Area}(ABC)}$$

This method gives more weight to the data point closer to the point being estimated.

D. Inverse distance method:

Suppose we want to predict data point • from the observed points ∘ as shown in the figure below:



Each observed data point has an impact on the point being predicted with the closest points having larger impact. We can consider as a predictor the following linear combination (weighted average):

$$\hat{z}(s_0) = w_1 z_1 + w_2 z_2 + \cdots + w_n z_n = \sum_{i=1}^{n} w_i z_i$$

The weights $w_i, i = 1, \cdots, n$ are inversely related to the distance of point i to the point being predicted. For example, $w_i = \frac{1}{d_i}$. This makes sense since we give more weight to the closest data points. Of course other functions can be used, such as, $\frac{1}{d_i^2}$, $\frac{1}{d_i^3}$, $\frac{1}{\sqrt{d_i}}$, e^{-d_i}, etc.

Example:
Suppose we want to predict point z_o from 12 surrounding points with distances and values as follows:

d_i	150	145	153	100	120	125	130	135	136	110	115	103
z_i	40	37	44	42	50	55	38	41	39	47	51	58

The distances are in feet and the values of z are in %.
What is the problem here?

We need to generate weights that are unit-free numbers. One way to do this is to divide each weight with the sum of the weights. The inverse distance weighted predictor is equal to:

$$\hat{z} = \frac{\sum_{i=1}^{n} w_i z_i}{\sum_{i=1}^{n} w_i} = z_1 \frac{w_1}{\sum_{i=1}^{n} w_i} + z_2 \frac{w_2}{\sum_{i=1}^{n} w_i} + \cdots + z_n \frac{w_n}{\sum_{i=1}^{n} w_i}.$$

where, $w_i = \frac{1}{d_i}$. The sum of the new weights is equal to 1, and this will make the predictor unbiased.

A more general expression is the following based on some power we choose for the inverse distance:

$$\hat{z}(s_0) = \frac{\sum_{i=1}^{n} w_i^p z(s_i)}{\sum_{i=1}^{n} w_i^p} = z_1 \frac{w_1^p}{\sum_{i=1}^{n} w_i^p} + z_2 \frac{w_2^p}{\sum_{i=1}^{n} w_i^p} + \cdots + z_n \frac{w_n^p}{\sum_{i=1}^{n} w_i^p}.$$

where, $w_i^p = \frac{1}{d_i^p}$.

If p is large then only the points closer to the point being estimated are important. If $p = 0$ then $\hat{z} = \bar{z}$. The default value for p in `gstat` is $p = 2$. With the method of inverse distance the weights are guaranteed to be between 0 and 1.

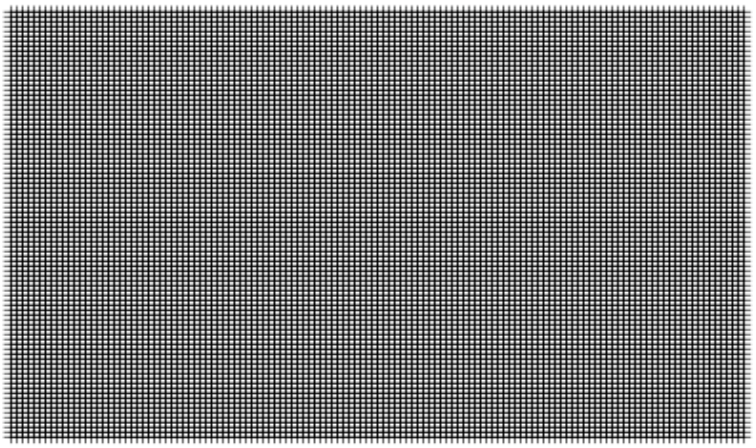
Example using R:

```
# Read the data:
a <- read.table("http://www.stat.ucla.edu/~nchristo/statistics_c173_c273/
     wolfcamp.txt", header=TRUE)

# Load the gstat library:
library(gstat)

# Create the grid:
x.range <- as.integer(range(a[,1]))
x.range
y.range <- as.integer(range(a[,2]))
y.range
grd <- expand.grid(x=seq(from=x.range[1], to=x.range[2], by=2),
y=seq(from=y.range[1], to=y.range[2], by=2))
coordinates(grd) <- ~ x+y

# Plot the grid (see figure below):
plot(grd)

# Plot the observed data on the grid (see figure below):
points(a)
```

```
#Predict data points on the grid:
idw.out <- idw(level~1, locations=~x+y, a, grd)
```

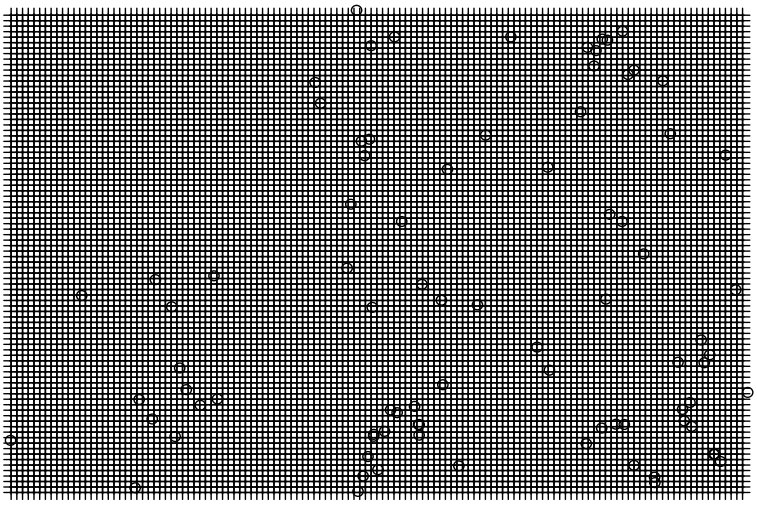
Here is the grid:

```
plot(grd)
```



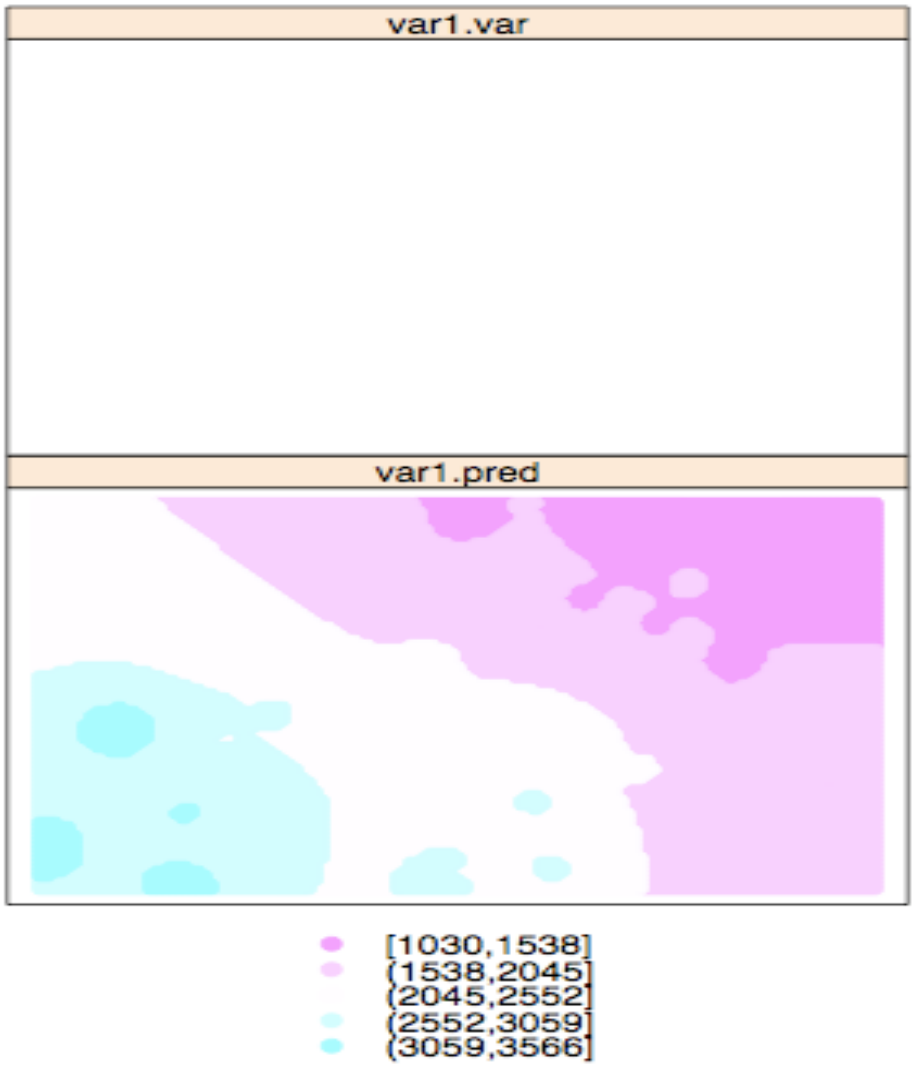
Here is the grid with the data points:

```
points(a)
```



Here is the plot of the predicted values:

```
# Plot the raster map (see figure below):
spplot(idw.out)
```




```
#Use different color patterns:
spplot(idw.out, col.regions=heat.colors(4))
```

The `var1.var` window above is empty because the inverse distance weighted method does not provide the variance of the predicted values.

Plot of the predicted values using the `image` function:
Another way to plot the predicted values is by using the `image` function.
Important note: Before you use the `gstat` library you must save the original `image` function. You can do for example the following:

```
image.orig <- image
```

This is necessary because `gstat` has its own `image` function which does not work the same way as the `graphics image` function.

First, we need to collapse the vector of the predicted values (`idw.out`) and form a matrix. This is needed because the `image` function requires a matrix input. The following function will do it for us.

```
vec.2.Mx <- function( xvec, nrow, ncol ) {
Mx.out <- matrix(0, nrow, ncol )

for(j in 1:ncol) {
for(i in 1:nrow) {
Mx.out[i, j] <- xvec[ (j-1)*nrow + i ]
}
}

return( Mx.out )
}
```

Now you can use the function to create a matrix (*qqq* below):

```
> qqq <- vec.2.Mx( idw.out$var1.pred,
  length(seq(from=x.range[1], to=x.range[2], by=2)),
  length(seq(from=y.range[1], to=y.range[2], by=2)) )
```

And you can use the original image function to create the raster map (see figure on page 10):

```
> image.orig(seq(from=x.range[1], to=x.range[2], by=2),
  seq(from=y.range[1], to=y.range[2], by=2), qqq,
  xlab="West to East", ylab="South to North")
```

The color of the resulting image can be changed using the argument `col` with options:
- `rainbow(n)`
- `heat.colors(n)`
- `topo.colors(n)`
- `terrain.colors(n)`

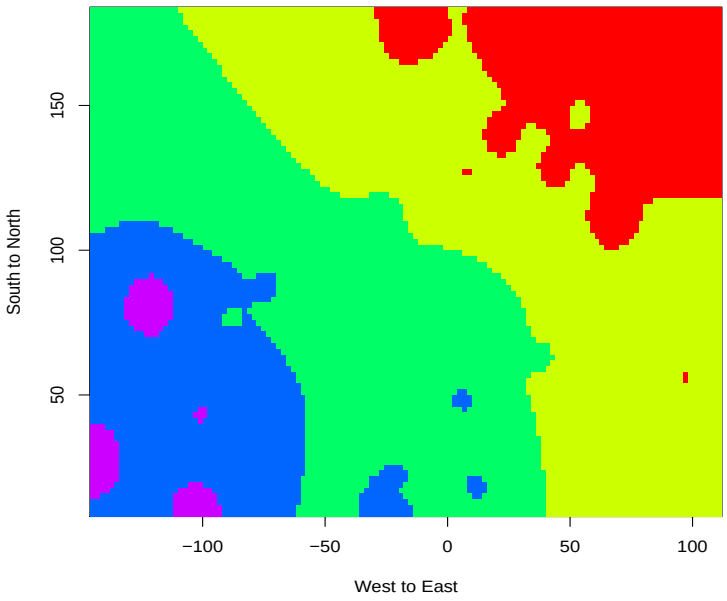
where, `n` is the number of colors.

Much easier the vector of the predicted values can be collapsed into a matrix with the `matrix` function:

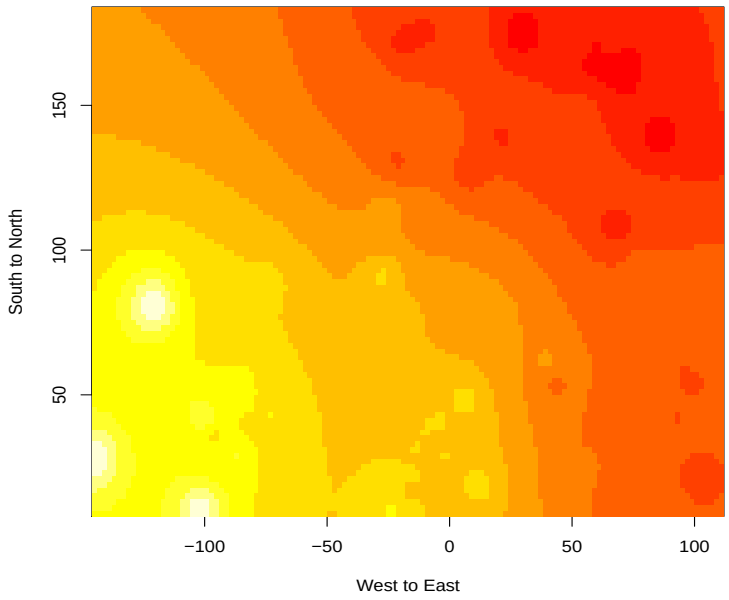
```
#Collapse the predicted values into a matrix:
qqq <- matrix(idw.out$var1.pred,
length(seq(from=x.range[1], to=x.range[2], by=2)),
length(seq(from=y.range[1], to=y.range[2], by=2)))
```

Using `col=rainbow(5)`:

```
image.orig(seq(from=x.range[1], to=x.range[2], by=2),
seq(from=y.range[1], to=y.range[2], by=2), qqq,
xlab="West to East", ylab="South to North",
col=rainbow(5))
```

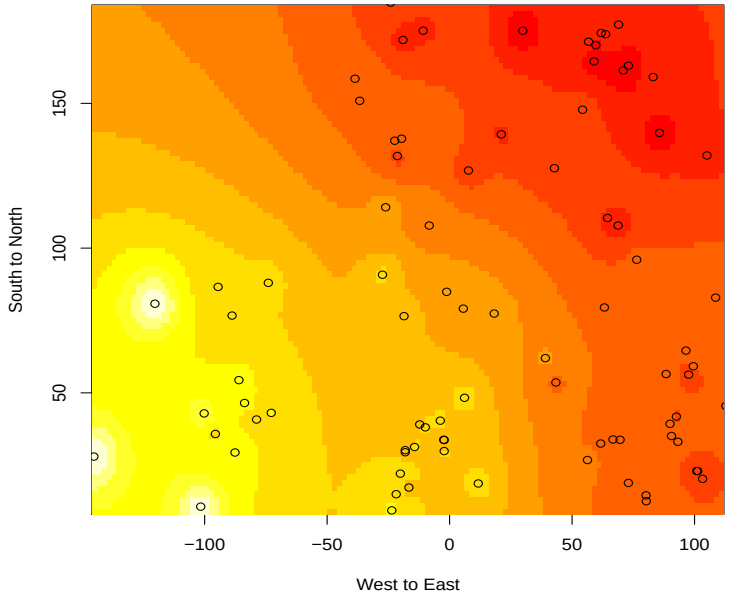


This is what we get in our example (see commands on page 8):



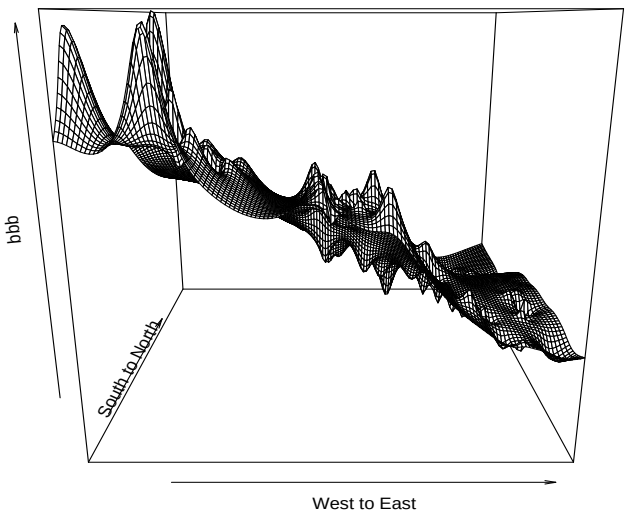
In case we want the observed points to be plotted on the previous raster map we can simply use the `points` function as follows:

```
points(a)
```



Another type of graph:

```
persp(seq(from=x.range[1], to=x.range[2], by=2),
seq(from=y.range[1], to=y.range[2], by=2), qqq,
xlab="West to East", ylab="South to North")
```



Exercise:
Access the data from the *Maas* river at:

```
a <- read.table("http://www.stat.ucla.edu/~nchristo/statistics_c173_c273/
     soil.txt", header=T)
```

a. Create a grid (use `by=50`).

b. Use the `idw` function of `gstat` to predict the value of the logarithm of `lead` at every point of the grid.

c. Plot the predicted values and create a raster map using the `spplot` function.

d. Collapse the vector of the predicted values into a matrix and then use the `image` function to repeat question (c).

Linear Regression:
For spatial prediction we can use the function `krige` of the `gstat` library as follows:

```
> a <- read.table("http://www.stat.ucla.edu/~nchristo/statistics_c173_c273/
  wolfcamp.txt", header=T)
# Load the gstat library:
> library(gstat)
# Create the grid:
> x.range <- as.integer(range(a[,1]))
> x.range
> y.range <- as.integer(range(a[,2]))
> y.range
> grd <- expand.grid(x=seq(from=x.range[1], to=x.range[2], by=2),
  y=seq(from=y.range[1], to=y.range[2], by=2))
> coordinates(grd) <- ~ x+y

#Predict the value at the grid points:
> q1 <- krige(formula=level~x+y, locations=~x+y, a, grd)
> names(q1)
[1] "var1.pred" "var1.var"
```

Note: The `id="level"` is useful. Here is what you get:

```
> q1 <- krige(id="level", formula=level~x+y, locations=~x+y, a, grd)
> names(q1)
[1] "level.pred" "level.var"
```

To display the predicted values you can type `q1$level.pred`.

Alternatively, the `lm` function can be used:

```
> q1.r <-  lm(level ~ x+y, data=a)
> q1.r_pred <- predict(q1.r, se.fit=TRUE, grd)
> names(q1.r_pred)
[1] "fit"            "se.fit"         "df"             "residual.scale"
```

To display the fitted values you can type `q1.r_pred$fit`.

Fitting a polynomial of spatial coordinates:
We can also use a second order polynomial of the coordinates as follows:

```
> q2 <- krige(id="level", formula=level~1, locations=~x+y, a, newdata=grd,
  degree=2)
```

With the argument `degree=2` we run a regression on $x + y + x^2 + y^2 + xy$.

The same result can be obtained using the `lm` function:

```
> q2.r <- lm(level ~ x+y+I(x^2)+I(y^2)+I(x*y), a)
> q2.r_pred <- predict(q2.r, se.fit=TRUE, grd)
```

Questions that need to be addressed:

1. What function of distance should we use? We can use $\frac{1}{d_i}$, $\frac{1}{d_i^2}$, etc.
2. How do we take into account different continuity in different directions? This is called *anisotropy*.
3. How many of the observed data points should we use in predicting a new point? Is it, "the more points we use the better estimate we get"?
4. How do we handle clusters of data points or highly irregular spaced data points?
5. How far should we go to include data points? Note: This is not the same as 3 above.
6. Can we produce confidence intervals for the predictor assuming normality?
7. What happens when we have non-normal data?
8. Why are predictions smooth?
9. What happens if there is a strong trend in the data?
10. How do we estimate areas and volumes? For example, in mining we want to estimate the ore deposit.