Multivariate Statistics

In this third lecture, we will shift from univariate statistics to multivariate statistics. Here we will have a look into the special case of dependence on the distance between observations. This is the most fundamental assumption of **geostatistics**. We assume that proximity in space leads to proximity in value, or in other words: Two observations that are close should observe something more similar than two distant observations.

Task 1

Geostatistics is all about distances. Instead of calculating the mean or the standard deviation of the whole sample, we will calculate them only for points at a specific distance. Therefore the first thing we need is a neat and clever way to store all distances that can occur in the sample.

Consider the points:

location	coordinates
s1	(1, 1)
s2	(1, 0)
s3	(0, 2)

We can store all combinations into a NxN matrix, where N is the sample size like:

| | s1 | s2 | s3 | | s1 | 0 | 1 | 1.41 | | s2 | 1 | 0 | 2.24 | | s3 | 1.41 | 2.24 | 0 |

Implement the method `distance_matrix.m`, that takes a matrix `X` of shape `(N, 2)` where N is the sample size and returns a distance matrix `M` as explained above.

If you want to test your function against working code use this snippet, which yields exactly the same result:

```
% pkg load statistics  % octave only
squareform(pdist(X))
```

Task 2

The distance matrix `M` from task 1 has some redundant information. First, the diagonal of `M` contains only zeros by definition. Second, it is a diagonal matrix where the upper and lower triangle are mirrored, as long as the distance measure is not direction depended. This is always the case for us, as we use the Euclidean distance.

To decrease the workload, we could only use the upper triangle of `M`. Secondly, we store all elements of the upper triangle into a one dimensional array, as long as define how the elements are ordered. For example row-wise. Then the resulting array `d` would be composed like:

- the first `N - 1` elements give the distance of s1 to s2 ... sN.
- the following `N - 2` elements give the distance of s2 to s3 ... sN.
- the next `N - 3` elements give the distance of of s3 to s4 ... sN.
- and the last element gives the distance of s(N - 1) to sN.

One of the main advantages, beside avoiding repetetive calculations, is that all other functions and calculations we use in geostatistics can easily be mapped onto this array. This simplifies the algorithms and is much faster.

Implement the algorithm given above as `distances.m`. You can test it against the `pdist` function of Matlab. For Octave you have to `pkg load statistics` first, to make this function available.

Task 3

Now calculate the distances of the given sample file `PREC_GAUGE_COOR.csv`.

a) How do they look like?

b) How are they distributed?

c) Utilize a histogram to decide on a suitable binning for the distances? How many classes do you define?

The file can be opened and read similar to the example given below:

```
data = dlmread('PREC_GAUGE_COOR.csv', ' ');
coords = data(:,2:3);
```

Task 4

In Task 3 we calculated the distance in space between all observation points. The same can be done with all observation values. For a first shot we will use the absoulte difference `abs(v1 - v2)` as the *distance* metric for observations.

Create a scatter plot that relates each separating distance between any of the point pairs to the corresponding distance in value.

Task 5

Last, but not least, we will formulate a more functional relationship between the two distance measures formulated in the previous tasks. Calculate the arithmetic mean and variance (first and second univariate statistical moment) for each of the distance lag classes. Plot these relationships.

Now consider the results from Task 4. Are these relationships sufficiently describing the interconnection of proximity in distance and proximity in value?