KIT-HYD / **geostatistics_lecture**

Code | Issues 1 | Pull requests | Actions | Projects

master | **geostatistics_lecture** / exercises / **univariate** /

Jan Wienhoefer Ex 1- 3 Jan W ... | 7 days ago | **History**

..	
README.md	7 days ago
average.m	12 months ago
histogram_counts.m	12 months ago
sampling.m	7 days ago
standard_deviation.m	12 months ago
synthetic_signal.m	7 days ago
temperature_study.m	7 days ago
tests.m	12 months ago
variance.m	12 months ago

README.md

Univariate statistics

This exercise is a collection of important metrics, concepts and tools that are helpful to dive into geostatistics.

Note: These tasks are even more fun when you split up into two groups and run task 2 against the code of the other group.

Task 1

Matlab/Octave are great as they offer a lot of pre-built functions. Especially for statistics. Calculating a arithmetic mean, standard deviation, variance and a histogram of the sample stored in an array `x` is as easy as:

```
mean(x)
std(x)
var(x)
hist(x, 25)
```

But what have you actually done? Do you know that? What is the variance or mean? Easy? Then reprogramm it. There are 4 files with a prepared function. Remove the call to the generic Matlab/Octave function with your own code.

Task 2

Now it's time to write your own unit test. You produced 4 functions in task 1. Now fill the `test.m` file. After running this file one has to be sure that your code is running properly and there is no way to break it by misusage. Look into the idw folder `test_file.m` if you need some inspiration (But note, that this file does not test everything).

How should a unit test be designed? Well, that's a long story and you can buy books about it. We'll keep it simple. Answer yourself these questions and program the `test.m` accordingly:

- Create defined input.
- What is the expected outcome of a function using this input?
- Test it.
- Is there more than one option, how the code can be run? Well, then you need more than one test...
- Now, think of a stupid user. What could go wrong? Test against it.

Task 3

With real data we usually never observe exhaustively. We use a **sample** of the reality, calculate some statistical properties of that sample and assume that these properties also apply to reality. Aim of this task is to get familiar with some of the fundamental statistical measures and their dependence of sample size.

The script `sampling.m` creates an artificial soil moisture dataset using the lines:

```
soil_moisture = randn(500000,1) .* 5 + 25
```

Note that this is an artificial **univariate** dataset. Unlike real soil moisture data, the (temporal) ordering of the data is uncorrelated and does not have any meaning. The `sampling.m` prints the statistical moments created with your code from the prveious tasks to the screen and plots an histogram function.

You can use the following code to draw samples from `soil_moisture`：

```
datasample(soil_moisture, 50, 'Replace', false)
```

which will produce 50 *observations*. How well are the statistical properties of soil moisture estimated, when using a sample size of:

- 15
- 25
- 50
- 100
- 1000
- 5000
- 50000

Task 4

This task will use more complex input data. There is a function in the folder called `synthetic_signal.m` which can be used to create sine-based signals, like air temperature. Here, the ordering of the values has a meaning. Run the `temperature_study.m` file, which will create two timeseries and compare them value by value. The two small plots are the histograms of each signal individually.

- What are the differences between the signals?
- Can you describe the scatter plot? What does it show?
- Change mean values and amplitude of one and both signals. What is the influence of each one on the correlation between both?
- How *different* can you make the the parameters to still judge both series as *similar*?

Task 5

In task 4, the correlation coefficient between the two time series is calculated. Now, write a new script or function that will take only one of the series. Duplicate the series and *shift* it by one element. That means, take the first element of the array and push it to the end. Then the formerly second element should be the first now. You call this a **lag** of 1.

a)

What is the correlation coefficient now? Did it change significantly?

b)

Repeat the procedure for a lag of 10, 100, and 500.

c)

Repeat the procedure for all lags: 1, 2, 3, ... N-1 and plot the resulting correlation coefficient a against the lag. Describe this result.