Artificial Neural Networks

Prof. Dr. Sen Cheng

Oct 7, 2019

Introduction to Scientific Computing

Tutors: Richard Görler (Richard.Goerler@rub.de)

Further Reading: pyhon, numpy and matplotlib/pyplot documentation online

These exercises are meant to give you some first practice in scientific programming with python. The problems that are to be solved computationally here themselves are not relevant for the exam.

Problem Set

1. **Make a Q-Q-plot to check visually whether a sample distribution is normal.**

   A Q-Q-plot can be applied to visually judge whether a given sample was drawn from a certain theoretical distribution. This can be useful for instance before choosing a statistical test that requires normally distributed samples. Q-Q stands for quantile-quantile because each point in the scatter plot is a quantile of one sample distribution plotted against the same quantile of another sample distribution. If the points deviate systematically from a straight line, then the sample was probably not drawn from the theoretical distribution. In our case, we want to compare a sample to a normal distribution.

   The task is to first generate a sample as well as normal distributed data, then plot each distribution separately, and, finally, the Q-Q-plot for the two distributions (Fig. 1). Below you find step-by-step instructions to help you with this task.

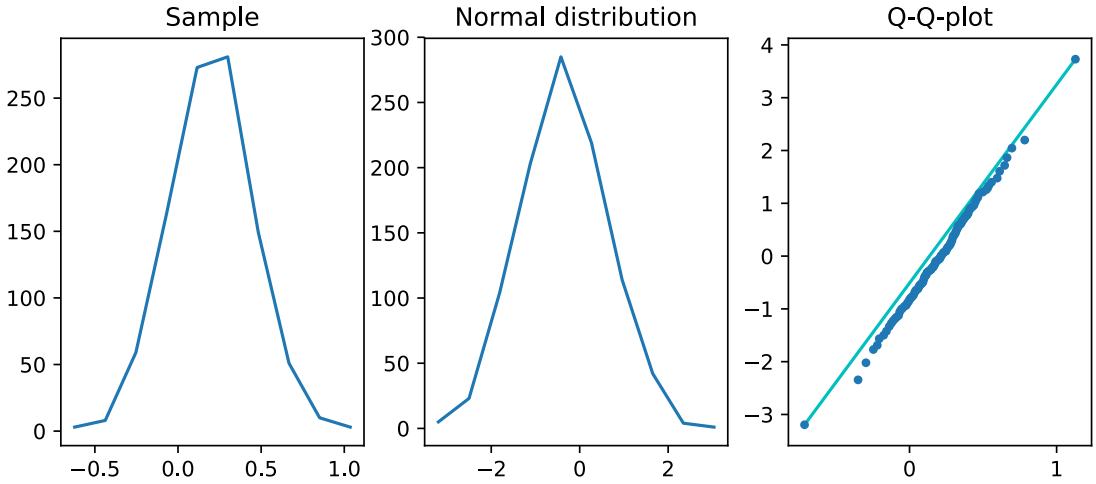

Figure 1: Example figure output

   (a) Obtain sample data. This can be any series of single measurements. You can be creative here, or use the following suggestion.

   Generate a set of points uniformly distributed within a square. Compute your data as the Euclidean distance of each point to the center of the square.

- Generating $N = 1000$ data points should work well.
- With numpy you can generate arrays with random values.
  The function `numpy.random.rand()` produces uniformly distributed values between 0 and 1 in the given shape. For instance, the call `numpy.random.rand(10)` generates a 10-element vector and the call `numpy.random.rand(5,4)` generates a 5-by-4 matrix containing random values.
  By multiplying the result with a constant you can scale the number range. Also, you can add a constant to shift.
- What is the Euclidean distance?
  The Euclidean distance d from q to p is the "length" of the vector pointing from q to p.

$$d = \sqrt{(q-p)\cdot(q-p)} = ||q-p|| \tag{1}$$

  Calculating this requires several steps, which can at the end be integrated in a single line.

  (i) *Difference.* With numpy you can use operators between arrays of identical shape to perform element-wise operations. However, you can also use operators between arrays with a different number of dimensions when the shared dimensions are of the same size. For instance, you can add a 3-element vector to a 5-by-3 matrix. This is called "broadcasting", try it out!
  With that in mind you can calculate the difference between each each point and the center of the square using only one operator.

  (ii) *Scalar product with itself.* You can do this in two steps: First, square each element. Second, sum the entries of each vector up. You can do this for all vectors at the same time by using the `numpy.sum()` function with the `axis` argument (see the slides or the numpy docs).

  (iii) *Square root.* You can use the `numpy.sqrt()` function.

(b) Plot the sample distribution. You do not need to normalize the distribution. See Fig. 1 (left) for an example. Your distribution will look different.

- The non-normalized distribution is a histogram. You can have a quick look at it by using the function `pyplot.hist()`. This function generates a bar plot. However, what we want is a line plot to make it look like a distribution function. The function `numpy.histogram()` returns the histogram data (values and bin edges) without generating a plot. You can look it up in the numpy docs.
- The function `pyplot.plot()` needs an array for each the x- and y-coordinate of the data points. Conveniently, the y's are the histogram values. For x, though, you cannot use the bin edges because they have one too many data points. If you want to be precise, calculate the histogram bin centers from the edges and use those as x.
- You have to use subplots to generate several plots next to each other in one figure. You can use `pyplot.subplot()`. Check the slides or the pyplot docs.

(c) Generate normal distributed data for comparison and plot it next to the sample distribution.

- The function `numpy.random.randn()` generates a sample of normal distributed values in a given shape, similar to `numpy.random.rand()` that was introduced in (a).

(d) Make the Q-Q-plot. Generate an ascending range of numbers q_i between 0 and 1 and calculate the q_i-quantiles. Make a scatter plot. You can add the diagonal line from the 0- to the 1-quantile. Note that this is not necessarily the best line to reference the data to.

- What is a quantile?
  The 0.25-quantile, for instance, is the x-value of the distribution such that 25% of the observations have a lower value (= are to the left of the quantile). Similarly, the 0-quantile is always the lowest value of the distribution and the 1-quantile the highest.
- You can use the function `numpy.quantile()`. You can look it up in the numpy docs.

2. **Bonus: Test whether a number is prime**

Write a function that tests whether a given integer $x >= 2$ is a prime number and returns `True` or `False` accordingly. In that function test whether any whole number in an appropriate range divides x.

- How to test whether a number divides another number?
  `9//4` yields `2`, but `9/4` yields `2.25`.
  `9//3` yields `3`, and `9/3` also yields `3`.
- What is an appropriate range?
  Every number (even a prime number) is divisible by itself and by 1.
  What is the smallest, and what is the largest number (potential divisor) we need to test?

3. **Bonus: Calculate π by random sampling**

   "Construct" a square that contains a circle. The radius of the circle is equal to the side length of the square. Randomly sample uniformly distributed points within the square. Observe the number of points C that appear within the circle. See Fig. 2 for a visualization of that sampling process.

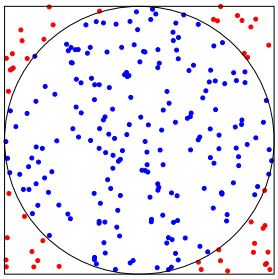

Figure 2: Sampling points within the square

   (a) Do not use loops, but make use of the functionality of numpy.
   - How to calculate π from C and N, the total number of points?
     Start with the ansatz that the ratio $\frac{C}{N}$ equals the ratio of the surface areas of the circle and the square.
     Solve for π.
   - $N = 10^6$ should work well.

   (b) To improve the approximation, put your sampling into a function that returns its approximation for π. Average over multiple (e.g. 1000) calls of the function. Here you can use a loop.