



NumPy, SciPy & Matplotlib Exercises

January 20, 2015

Exercises 1 & 5 from
Stéfan van der Walt
Licence: CC-by-sa

Exercise 1: Exploring the NumPy Data Structure Further

This exercise has the goal to explore the NumPy data structure a bit further.
Generate two 2×2 arrays by

```
>>>>> x = np.array([[1, 2], [3, 4]], order='C', dtype=np.uint8)
>>>>> y = np.array([[1, 2], [3, 4]], order='F', dtype=np.uint8)
```

which gives you a C- and a F-contiguous array.

Check the order of the elements and bytes in the data by looking at the string representing the data behind the arrays.

Now define these arrays as of data type `uint32`. Consider again the ordering of the bytes in the data.

Is the order of type Little (least significant byte first) oder Big (most significant byte first) Endian?

What happens if you switch to signed integers (*i.e.* `int32`)?

What happens if you switch the sign of the values in the input array?

Exercise 2: Zebra Tracking

You find the position data of several zebras (Plain zebras, *Equus quagga burchelli*) in northern Botswana (Source: movebank.org) in the file `ZebraBotswana.txt`.

The data consists of the date and time of the measurement in unix format (*i.e.* seconds since 1970-1-1), the longitude and latitude of the measured position (in degrees) and the number of the corresponding Zebra.

Read in the data and plot for each zebra its path.

Calculate for each zebra the average movement per day as a function of the month and plot it. (The distance of one degree in latitude corresponds to about 111.3 km and in longitude at this latitude to about 104.6 km).

Calculate the average position per zebra in each month and calculate the correlation between the different zebras.

Tip: Make usage of the function `np.where`.

Exercise 3: Financial Analysis

The file contains as time series the adjusted closing prices of the six highest-weighted stocks (IBM, Goldman Sachs, 3M, Boeing, Chevron and United Technologies; VISA is not included due to its initial public offering only in 2008) in the Dow Jones Industrial (DJI) index plus the index itself. The first row is the date in unix format.

Read in the data and plot the different stocks and the DJI index.

Calculate the daily increase or decrease of their value in percent.

Calculate the mean and the standard deviation of these changes over the full time range.

Optional: Consider the distribution of the daily changes.

Tip: Use the `histogram` function from NumPy and – to plot – the `bar` function from Matplotlib.

Simulate for one time serie the change as a function of time – based on the assumption that the percent changes are Gaussian distributed – and compare it to the actual development of the corresponding stock.

Tip: Use the `np.multiply.accumulate` function.

Repeat the last step, but do not use the full time range, but rather for each date the last 30 days. Does it look more similar to the actual development.

Calculate also the correlations between the stocks and the DJI index and incorporate this into your simulation.

Tip: Use the NumPy's `cov` function and the Cholesky decomposition as well as the multivariate normal distribution implementation of SciPy `scipy.stats.multivariate.normal`.

Exercise 4: Matrix Calculation

You can find in the file `matrix.txt` a 100×100 matrix. Read it in and use SciPy to calculate the determinant, the eigenvalues and -vectors. Since it is a positive definite symmetric matrix you can also try the dedicated functions for such matrices, *i.e.* `eigh` and `eigvalsh`, in particular in terms of CPU timing. You can also test matrix addition and multiplication using the matrix and its inverse and some other matrix operations you are interested in as well as the Cholesky and QR and SVD (Singular value decomposition) decomposition. Use `%timeit` to monitor the performance.

Exercise 5: Fancy Indexing

Create an $n \times n$ array and try to construct a one-dimensional array containing all the diagonal elements with fancy indexing.

Create a 10×5 array with random numbers between 0 and 1. Construct the (one-dimensional) array returning the values of the array closest to 0.66 for each row using fancy indexing. Do the same for the columns.

Tip: Make use of `np.abs` and `np.argmax`, and check their documentation.

Predict and verify the shape of array resulting from the following slicing operation:

```
>>>>> x = np.empty((12, 7, 5))
>>>>> idx0 = np.zeros((2, 9)).astype(int)
>>>>> idx1 = np.zeros((2, 1)).astype(int)
>>>>> idx2 = np.zeros((1, 1)).astype(int)
>>>>> x[idx0, idx1, idx2]
```

Exercise 6: Some more Data Analysis

You can also try out some data analysis you have done with other tools (*e.g.* MATLAB, R) and try to figure out if it is possible in Python with the introduced libraries. If you have a lack of data, the file `Cobe.txt` contains data from the COBE satellite (more info about COBE at <http://lambda.gsfc.nasa.gov/product/cobe/>). It shows the spectrum of the cosmic microwave background. The first row gives the frequency (actually the inverse of the wavelength), the second row the spectrum in MJy/sr (MJy: Mega-Jansky, $1 \text{ Jy} = 10^{-26} \text{ W/Hz}\cdot\text{m}^2$; sr: Steradian), so it is a measure of the spectral flux per solid angle. The third row shows the uncertainty on the spectrum in kJy/sr.

Use the `scipy.optimize.leastsq` to perform a least-square fit of the data. The function that should describes the data is the Planck law $f(x) = A_0 \cdot x^3 / (\exp(1.439x/T) - 1)$ where x is the frequency in 1/cm. A_0 and T are the fit parameters, where A_0 is the amplitude and T the temperature of the universe. The factor 1.439 K·cm comes from $h \cdot c / k_B$ in the chosen unit frame. So you can determine from the fit how hot the universe is.