University of Zurich UZH

Department of Physics **Scientific Programming with Python**

Analytics tools Exercises

June 26, 2019

Before you start:

- create a suitable directory for this exercise
- download the zipped material from `http://www.physik.uzh.ch/~python/python/lecture_analysis/`

In case you are not familiar with the visualisation you can also postpone these parts to Friday after the lecture on it.

Exercise 1: Warm-up with Minimisation

Target libraries: `scipy.optimize`
The goal of the exercise is to use the Rosenbrock function $f(x, y) = (1-x)^2+100(y-x^2)^2$ to play around and get familiar with the SciPy's minimisation functionality.
The Rosenbroock function and its Jacobian (first derivative) and Hessian (second derivative) are available in `scipy.optimize` as `rosen`, `rosen_der` and `rosen_hess` and it is a common function design to benchmark minimisation algorithms.
You can investigate different questions:

- How do different algorithms perform in terms of run time?
- How does this depend on the initial guess?
- How do algorithms – where the first and second derivative are optional – perform with and without this information?

You can also define another function and run some tests.

Exercise 2: Numerical Integration

Target libraries: `scipy.integrate`
This exercise should familiarise you with the integration functionality in Scipy.

Take the function $f(x) = x/((x-1)^2+a)$ and perform the integration on $[0, \infty]$ with different approaches (Trapezoidal, Simpson's, Brent). How does the precision and performance vary for different values of a (go also to very small ones *i.e.* $a = 10^{-6}$ or $a = 10^{-12}$). How can the performance of the integration be improved for these cases?

Take the function $f(x) = x^2 \cos(2\pi x)$ and integrate on $[0, 1]$ using the `quad` method. The QUAD family offers also particular methods for calculating integrands that are weighted by trigonometric functions (see documentation of `quad`). So run the integration with this particular implementation and compare the performance in terms of speed and precision. What is the outcome if the integration range is $[0, 1000]$?

Exercise 3: Matrix Calculation

Targeted libraries: `scipy.linalg`
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 4: How Cold is the Universe?

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 column gives the frequency (actually the inverse of the wavelength in 1/cm), the second column the spectrum in MJy/sr (MJy: Mega-Jansky, 1 Jy= 10^{-26} W/Hz·m^2 ; sr: Steradian), so it is a measure of the spectral flux per solid angle. The third column shows the uncertainty on the spectrum in kJy/sr.

Use the `scipy.optimize.leastsq` or the fundamental minimisation method 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 $h \cdot c/k_B$ (h: Planck's constant, c: speed of light, k_B: Boltzmann's constant) in the chosen unit frame. So you can determine from the fit how cold the universe is.

Exercise 5: Maximum-Likelihood Fitting

Targeted libraries: `scipy` and `matplotlib`
The goal of this exercise is to develop an algorithm to perform a maximum-likelihood estimation of data with Python. The file `DJI_daily.csv` contains the daily returns of the Dow Jones Index since the start of 2000.

Perform a maximum-likelihood fit to this data. It's up to you to perform an normal or a binned maximum-likelihood fit. You can start by using a Gaussian distribution (`scipy.stats.gauss`) and later use other suitable distributions like the Student-t distribution (`scipy.stats.t`).

Plot the distribution of data using histograms or errobars and the estimated distribution.

What happens when you restrict the data to a shorter period of time *e.g.* 200 days? The data is chronologically ordered.

Tip: For numerical reasons using the negative log-likelihood function

$$_{log}\mathcal{L} = -\sum_i \log(f(x_i|p))$$

and perform a minimisation with respect to the parameters p is a way to go. f is the distribution and x_i are the observations.
The ordering of parameters in the distributions from the `scipy.stats` package are not necessarily intuitive. So please consult the documentation.

Exercise 6: Sun Spots

Targeted libraries: `scipy.fttpack`, `scipy.integrate` and `scipy.optimize`
The file `SN_m_tot.txt` contains monthly averages of the number of sun spots. (The data goes back to 1796, but let's only consider data after 1900.)

Load the file and analyse the correlation of the absolute and relative change in the number of sun spots at month t to month $t - k$, *i.e.* the autocorrelation for lag k. Assess how the values are changing as a function of k.

Use Fast-Fourier-Transforms to understand the frequency spectrum of the observations. What frequencies can you spot in the data? To what period are they corresponding?
Mask the frequencies which have an amplitude below 5% of the maximum of the spectrum and use the inverse FFT to create a model that describes the number of sun spots .

Exercise 7: Multidimensional Interpolation & Terrain Models

Targeted libraries: `scipy` and `matplotlib`
In this exercise we want to apply an easy way offered by `scipy` to interpolate multidimensional data.
The file `zrh_terrain.txt` contain about 8'000 rows of sampled terrain data in the area of Zurich. The rows are of the form latitude,longitude, and elevation (in metres).

Use the `griddata` function in the `scipy.interpolate` library to create a highly granular terrain model.
Tip: Study the doc string of the function in particular what the interpolation domain can be.

Visualise the resulting model. Try different plot types (heatmap, different 3D plots) offered by `matplotlib` or other visualisation packages.

Test the different interpolation methods in `griddata` and see how the models differ due to the different methods.

Optional: Create a very simple waterfall-type hydrology model to leverage the terrain model to simulate flooding.

Exercise 8: Airline Connections

Targeted libraries: `scipy.sparse` and `scipy.linalg`
The file `routes.txt` contains a list of almost all known commercial airline connections. It includes information about the carrier, the departure location and the destination.
Use matrix calculus to find out the answers to the following questions:

- How many different routes exist from Zurich (IATA: ZRH) to Ushuaia (IATA: USH), not counting code sharing, with less than four legs?
- Closing which airport will lead to the highest drop in number of possible connections?
- Closing which airport will lead to the highest increase in the average number of legs in connecting two airports?
- What is the answer for the above two questions in terms of the grounding of an airline?

Details on the airports, airlines and planes can be found in `airports.txt`, `airlines.txt` and `planes.txt`.