University of Zurich[UZH]

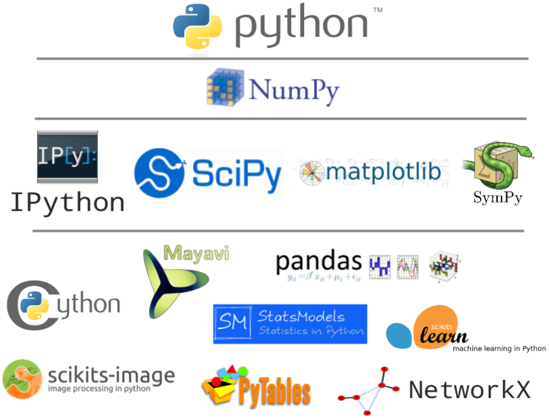
Department of Physics

# Scientific Programming: Analytics

Scientific Programming with Python

Christian Elsasser

# The Ecosystem of Homo Python Scientificus




[Ondřej Čertík/LANL]

# Table of Contents

**Three use cases**

- Financial engineering
- Graph analysis
- Signal and time series analysis

$\Rightarrow$ What methods we are going to look at

- Minimisation/Optimisation
- Numerical integration
- Fast-Fourier Transformation
- Matrix calculus/Sparse matrices
- Distributions

**We will not be able to go in the very details! But you find a lot of resources in the Scipy Lectures here!**

# Fundamental Tools – SciPy & NumPy

# SciPy – or Where the Fun Really Starts

- Offering a large number of functionality for numerical computation
  - `scipy.linalg` → Linear Algebra
  - `scipy.optimize` → Numerical optimisation (incl. least square)
  - `scipy.integrate` → Numerical integration
  - `scipy.stats` → Statistics including a large set of distributions
  - `scipy.spatial` → Spatial analysis like creation of Voroni sets, etc.
  - more at http://docs.scipy.org/doc/scipy/reference/
- Eco-system of more advanced packages for data analysis, *e.g.*
  - scikits.learn: Machine-learning algorithms
  - scikits.image: Image processing
  - pytables: data structure (based on HDF5)
  - ...

**Remark:** `import scipy as sp` only imports the most basic tools ⇒ `from scipy import stats`

# Use case 1 – Financial Engineering

**Situation:**

- Three different assets
  - Two stock indices Dow-Jones Industrial (DJI) and Swiss-Market Index (SMI) (performance yet unknown)
  - One risk-free investment (*e.g.* government bonds) at an annual return of 1%.

**Problem:**

- Evaluate the last year performance of the two stock indices ...
- ... and build a portfolio that minimises the risk (volatility) while having a minimum expected return of 14% p.a.

**Approach:**

1. Take the daily stock returns of two indices
2. Use Maximum-Likelihood Estimation to infer average return and volatility (standard deviation).
3. Use these parameters together with the correlation to build the optimum portfolio using optimisation under constraints.

**Libraries discussed:** Optimisation, Distributions

# Maximum-Likelihood Estimation

**Fundamentals:**

- For a given sample of (observed) values $x_i$ find the parameters $\theta_j$ that are maximising the likelihood of the observation based on the distribution $f(x|\theta)$
- $$\mathcal{L} = \prod_i f(x_i|\theta)$$
- Problem equivalent to minimise:

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

**Concrete case:**

- Estimation of the daily returns by using a Gaussian distribution

$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Single Gaussian case is trivial as the problem can be solved analytically with $\hat{\mu} = \overline{x}$ and $\hat{\sigma} = \sqrt{\overline{x^2} - \overline{x}^2}$

# Minimisation Algorithms

**Questions to ask:**

- Is the objective function smooth?
- Is the objective function convex?
- Can I help the algorithm by providing the exact Jacobian vector or Hessian matrix?
- Are the parameters bound?
- Are the constraints?

**Available algorithms:**

- Simplex (Nelder-Mead)
- Bi-directional (Powell)
- (Quasi-)Newton (BFGS)
- Trust-method (Dogleg,Newton)

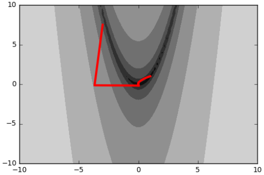
**Check documentation of**
`scipy.optimize.minimize`

- **Choose the algorithm carefully based on your problem!**
- **A good conditioning (*i.e.* comparable scaling) is always beneficial**

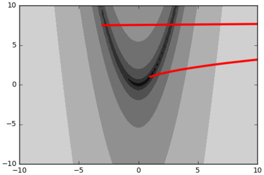
# Minimisation Algorithms – Differences

Comparison of different algorithms with the Rosenbrock function
$f(x, y) = (x - 1)^2 + 100(y - x^2)^2$ and starting point $(-3, 7.5)$

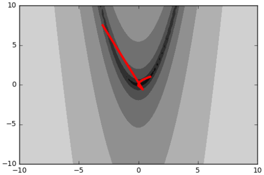
### Nelder-Mead




### BFGS




### Conjugate Gradient




Convergence heavily dependent on the choice of the algorithm and the initial starting point.

# Optimisation with Constraints

**Problem:**

- Find the fraction of investment in the two indices $p_{\text{DJI}}$ and $p_{\text{SMI}}$ such that the overall expected risk is minised ...
- ... with an expected return of at least 14%.

**Mathematical formulation:**

Total expected risk:

$$\sigma^2 = (p_{\text{DJI}}\sigma_{\text{DJI}})^2 + (p_{\text{SMI}}\sigma_{\text{SMI}})^2 + 2|p_{\text{DJI}}||p_{\text{SMI}}|\rho\sigma_{\text{DJI}}\sigma_{\text{SMI}}$$

Total expected return:

$$\mu = p_{\text{DJI}}\mu_{\text{DJI}} + p_{\text{SMI}}\mu_{\text{SMI}} + (1 - p_{\text{SMI}} - p_{\text{DJI}})\mu_{\text{rf}}$$

**Formulation in Python:**

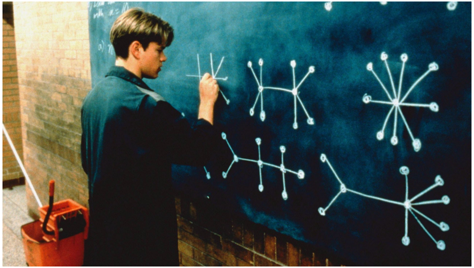
- Specialised minimisation algorithms for constraints: L-BFGS-B, SLSQP
- `scipy.optimize.minimize` understands bounds on parameters (*i.e.* trivial constraints) and constraints as equality or inequality
- Normal constraints have to be formulated as function that has to be equal/larger than zero.

# Use case 2 – Graph Theory

## Approach

- Graphs can be represented by matrices ($a_{ij}$ represents the connection from node $i$ to node $j$) called adjacency matrices.
- By exponentiating the matrix ($A^n$) we see which nodes are connected via $n$ sequential edges.
- The spectrum of $A$ reveals information about the structure of the graph.

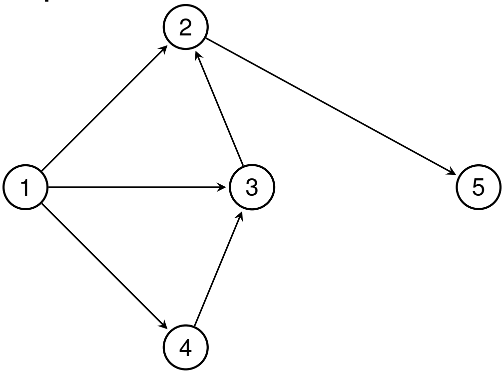
We are using the airline connections of the world as playground.



**Libraries discussed:** (Sparse) matrices

# One-page Introduction to Graph Theory

**Graph:**




**Adjacency matrix:**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 |

Row = From, Column = To

- If there is an edge to a node itself, entries on the diagonal
- Symmetric graph leads to symmetric adjacency matrix

# More than Arrays – NumPy and Matrices

NumPy offers a matrix framework for linear algebra calculations, allowing to defining one- and two-dimensional arrays as matrices

## Matrices

```
>>> a = np.matrix([[1,2],[3,4]])
>>> b = np.matrix(np.random.rand(4))
>>> c = np.matrix(np.random.rand(3,3))
```

One-dimensional arrays $\rightarrow 1 \times n$ matrices, *i.e.* row vectors

Matrices have some additional functionality (*e.g.* inverse: `a.I`, hermitian: `a.H`)

# Linear Algebra with SciPy – Bringing High-Performance Libraries to the Table

Light version of SciPy's linear algebra implementation at `np.linalg`

**Examples of available functionality:**

| | | |
|---|---|---|
| `np.linalg.cholesky` | `np.linalg.det` | `np.linalg.eig` |
| `np.linalg.eigh` | `np.linalg.qr` | `np.linalg.svd` |

The functions are wrappers of the LAPACK linear algebra package

More functionality is embedded in the full SciPy implementation `scipy.linalg`, *e.g.*

### Matrix Exponential

```
>>> a = np.matrix([[1,2],[3,4]])
>>> scipy.linalg.expm(a)
```

# Sparse Matrices

**Purpose:**

- Representation of graphs
- Representation of corpora

**Implementation in Python:**

- Different representations available in `scipy.sparse`
- `scipy.sparse.linalg` contains certain method to make calculations with sparse matrices

**Available types/flavours:**

| | | |
|---|---|---|
| Block Sparse Row | `bsr_matrix` | |
| COOrdinate format | `coo_matrix` | good for random access; tuple of indices and values |
| Compressed Sparse Column | `csc_matrix` | values, column/row indices |
| Compressed Sparse Row | `csr_matrix` | and non-zero entries up to row/column |
| DIAgonal storage | `dia_matrix` | |
| Dictionary of Keys | `dok_matrix` | good for construction |
| Row-based linked list | `lil_matrix` | |

# Use case 3 – Signal/Time Series Analysis

**Situation:**

- You have data in the form of signals (*e.g.* from a sensor) or time series.
- And you want to analyse them in terms of their frequency spectrum.

**Problem:**

- Typically a problem to be performed over and over again . . .
- . . . in certain applications is should go fairly fast.

**Approach:**

- Applying a Fast-Fourier-Transformation for a periodical function
- Calculating “by hand” the Fourier transformation for different functions

**Caution:** For certain functionalities in terms of signal analysis there is the library `scipy.signal`

**Libraries discussed:** Fast-Fourier-Transform, Integration

# Fast-Fourier-Transformations

**Problem to solve:**
Given a sample of (complex) numbers $x_n$ calculate

$$X_k = \sum_{n=0}^{N-1} x_n e^{2\pi kn/N}$$

- Like this algorithm of complexity $O(n^2)$
- FFT algorithm = way to bring complexity to $O(n \log n)$ or even below

**Implementation in Python:**

- Cooley-Tukey algorithm (breaking down of the problem recursively into smaller samples leading to the reusability of calculations)
- Dedicated algorithms for samples of real numbers (`rfft`)
- Or in case of cosine or sine series
  $X_k = \sum_{n=0}^{N-1} x_n \cos 2\pi kn/N$ (`dct`)
  $X_k = \sum_{n=0}^{N-1} x_n \sin 2\pi kn/N$ (`dst`)

# Fourier Transformation

**Problem to solve:**

- Calculate for a given function $f(t)$ and frequency $\omega$ the amplitude

$$A(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t e^{-i\omega t} f(t)$$

- Depending on the convention you might have an additional factor $(2\pi)^{-1/2}$.
- Idea: Evaluate the above integral numerically.

**Integration in Python:**

- `quad` as most generic integration algorithm based on QUADPACK (also available for multi-dimensional problems)
- It allows to indicated necessary precision.
- Options to indicate singularities
- Options to have a weight function $w$ *i.e.*

  $I = \int_a^b \mathrm{d}x f(x) w(x)$
- Also methods available to apply Trapezoidal and Simpsons rules as well as Romberg's method.

# Advanced Python Modules

We omitted any modules with a large and specific purpose $\rightarrow$ otherwise you would sit here tomorrow

Left to the interested audience to explore them further

- NLTK (www.nltk.org) $\rightarrow$ Natural language processing
- scikit-learn (scikit-learn.org) $\rightarrow$ Machine learning
- scikit-image (scikit-image.org) $\rightarrow$ Image processing and analysis
- ...

Rapidly growing and improving landscape of python modules, but with still some “whitish” spots (*e.g.* time series) $\Rightarrow$ Reflection of available alternatives?

# Conclusion

- Scipy together with Numpy offers a large number of fundamental tools for your everyday work in science and beyond
- Take the time to understand the content of the package ...
- ...to avoid a reinvention of the wheel

- Many specialised modules are based on the Scipy/Numpy foundation.
- We leave it to the interested audience to explore them further:
  - NLTK (www.nltk.org) $\rightarrow$ Natural language processing
  - scikit-learn (scikit-learn.org) $\rightarrow$ Machine learning
  - scikit-image (scikit-image.org) $\rightarrow$ Image processing and analysis
  - ...

Other relevant (fundamental) libraries will be discussed on Friday by Andreas together with the topic of visualisation.