Scientific computing with Python

Robert Johansson
(robert@riken.jp)

RIKEN
Content

- Role of computing in Science
- Efficient workflow for scientific computing
- Why Python for scientific computing
- Python environments for interactive computing
- The Scientific Python modules
- Introduction to the
  - Python language
  - Scientific Python modules
The role of computing in Science

Computing is an increasingly important complement to the traditional disciplines of theoretical and experimental science, and can now be viewed as a third branch of science.
When is computing scientific?

• The scientific method:
  – A set of principles that describes how science should be conducted

• Important concepts are:
  – Theory → prediction → experiment → confirmation/rejection of the hypothesis
  – Each step should be published, analyzed by peers, replicable and reproducible

• How does this relate to computing? My view is that:
  – Simulations and calculations must be replicable and reproducible by the original authors as well as independent researchers.
  – We need to publish source codes and fully document simulations and calculations (parameters, conditions, etc.) used for scientific publications.
  – We cannot use black-box software.
  – We need to make sure that code and the environment in which it was used is stored and available for a long time (5, 10, 25 years? Very difficult problem...)

What we need for efficient scientific computing

Some important components in an efficient workflow for scientific computing:

- **Number crunching**
  - High-level computing environment for interactive computing and exploration
    - E.g. Python and IPython, or other scripting language
  - Low-level environment for efficient computing
    - Fortran, C, C++, or other compiled language

- **Symbolic mathematics (CAS – Computer Algebra System)**

- **Visualization and presentation**
  - For interactive exploration
  - Publication quality figures

- **Archiving, reproducibility**
  - Version control software: git, hg, ...
  - Public repositories: github, Figshare.com, datadryad.org, ...
High and low level programming languages

High-level languages:
   Optimizes programming time at the expense of CPU time

Low-level languages:
   Optimize CPU time at the expense of programming time and flexibility of the code

Best of both worlds?
   A good balance between low and high level programming languages
Why Python for scientific computing?

- Python is a general purpose, high-level, interpreted language
  - Simple, clean, efficient syntax
  - Readable and intuitive code
  - Maintainable, extensible, adaptable code
- Suitable for exploratory and interactive computing
- Useful as a glue language (ex. providing high-level interfaces to existing low-level codes)
- Fully open source
  - No license fee
- Vibrant scientific computing community
  - Large number of high-quality packages available
  - Commercial products/support also readily available
Python environments for interactive computing

IPython: Interpreter prompt

Spyder: traditional IDE

IPython notebook: web-based interactive interface
The scientific python modules

The core packages for scientific computing with Python:

- **Numpy**: matrices, vectors, data arrays, linear algebra, ...
- **Scipy**: calculus, statistics, higher level computational routines, ...
- **Matplotlib**: plotting and visualization
- **SymPy**: symbolic computing

Many hundreds other domain specific packages available
Introduction to the Python language

(IPython notebook)

- Modules
- Variables and types
- Operators
- Compound types: Lists, tuples, and dictionaries
- Control Flow
  - if/else, loops
- Functions
- Classes
- Exceptions
Numpy

- The numpy package provides high-performance vector, matrix and higher-dimensional data structures.
  - Statically typed and homogeneous.
- Implemented in C and Fortran so when calculations are vectorized (formulated with vectors and matrices), performance is very good.
  - Optionally linked to efficient BLAS implementations, such as ATLAS, OpenBLAS or Intel MKL.
- Efficient representation of large data (n-dimensional arrays, such as vectors, matrices etc.)
  - Slicing, fancy indexing, for extracting and assigning data
  - Arithmetic and mathematical operations on data
  - Statistics functions
The SciPy framework builds on top of NumPy and provides a large number of higher-level scientific algorithms. Some of the topics that SciPy covers are:

- Special functions [scipy.special]
- Integration [scipy.integrate]
- Optimization [scipy.optimize]
- Interpolation [scipy.interpolate]
- Fourier Transforms [scipy.fftpack]
- Signal Processing [scipy.signal]
- Linear Algebra [scipy.linalg]
- Sparse Eigenvalue Problems [scipy.sparse]
- Statistics [scipy.stats]
- Multi-dimensional image processing [scipy.ndimage]
- File IO [scipy.io]
Matplotlib

• Publication quality 2D and 3D figures and animations
• Object-oriented API for programatically construct figures
Sympy

SymPy is a Python library for symbolic mathematics, and a feature-rich computer algebra system (CAS). A simple example:

```
In [1]: from sympy import *

In [2]: x = Symbol("x")

In [3]: s1 = cos(x).series(x, 0, 5)
s1
Out[3]: 1 - \frac{x^2}{2} + \frac{x^4}{24} + \mathcal{O}(x^5)

In [4]: s2 = sin(x).series(x, 0, 4)
s2
Out[4]: x - \frac{x^3}{6} + \mathcal{O}(x^4)

In [5]: expand(s1 * s2)
Out[5]: x - \frac{2x^3}{3} + \mathcal{O}(x^4)
```
Conclusions

• Discussed general principles and requirements for scientific computing

• Introduced Python and its stack of scientific packages as a programming environment for scientific computing

• Introduced the Python programming language

• Introduced scientific packages
  – NumPy/SciPy/Matplotlib/Sympy

• More information:
  – Longer version of this lecture: http://jrjohansson.github.io