The Python Interpreter - Part II

Remi Lehe, Daniel Winklehner

US Particle Accelerator School (USPAS) - Winter Session

Simulation of Beam and Plasma Systems

D. Bruhwiler, R. Lehe, S. Lund, J.-L. Vay, & D. Winklehner

Old Dominion U., Hampton, VA, 15-26 January, 2018

Python interpreter: Outline

1. Reusing code: functions, classes, modules
2. Faster computation: Forthon
3. Faster computation: Parallel Python

Modules

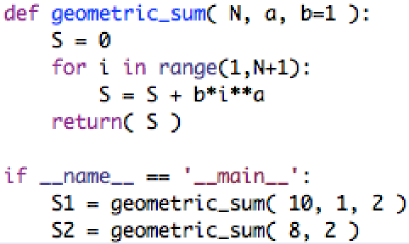
Module

Defines variables to be **imported** by **other Python sessions**.

- Any Python script can be treated as a module.
  `numpy` is a set of modules.
- The section
  `if __name__ == ’__main__’:`
  is executed if the script is **run** (e.g. `python geometric.py`) but not when it is **imported** (`import geometric as gm`)

Example module

In file `geometric.py`:



```
def geometric_sum( N, a, b=1 ):
    S = 0
    for i in range(1,N+1):
        S = S + b*i**a
    return( S )

if __name__ == '__main__':
    S1 = geometric_sum( 10, 1, 2 )
    S2 = geometric_sum( 8, 2 )
```

Example import and use

In e.g. `ipython`:

```
import geometric as gm
S = gm.geometric_sum(8, 2)
```

Importing modules

Different import styles:

- `import geometric`
  → `S = geometric.geometric_sum(8,2)`
- `import geometric as gm`
  → `S = gm.geometric_sum(8,2)`
- `from geometric import geometric_sum`
  or `from geometric import *` (imports all variables)
  → `S = geometric_sum(8,2)`

The source file of the module needs to be:

- in the same directory
- or in the default Python path
  (case of installed packages like `numpy`, `matplotlib` or even `warp`)

Functions and modules: task

Task 5

Download the file http://github.com/RemiLehe/uspas_exercise/raw/master/euler.py and put the last section (which creates an instance of EulerSolver) in a if __name__ == '__main__' clause.

Then use this file as a module, inside ipython

- In the shell, type ipython --matplotlib
- Then, inside ipython, type from euler import *
- Then create instances of EulerSolver for N1=100 and N2=100
- Then call the methods euler_integration and evaluate_result on each instance. Compare the results.

(NB: Do not hesitate to use tab completion in ipython)

How to install publicly-available modules/packages

Use a **package manager**!

- Automatically installs dependencies of requested packages
- Keeps track of the packages that you installed and their version

pip

- Example: `pip install Forthon`
- Can install any package that has been uploaded to pypi.python.org

conda

- Example: `conda install numpy`
- Only works for the **Anaconda distribution** of Python
- Automatically downloads binaries that are requested for certain Python packages (e.g. MPI for `mpi4py`, HDF5 for `h5py`)

How to write your own module/package

Structure (from http://docs.python-guide.org)

```
README.rst
LICENSE
setup.py
requirements.txt
sample/__init__.py
sample/core.py
sample/helpers.py
docs/conf.py
docs/index.rst
tests/test_basic.py
tests/test_advanced.py
```

Minimal Structure

```
setup.py
sample/__init__.py
sample/core.py
```

How to write your own module/package

setup.py

```
from setuptools import setup, find_packages
setup(
 name=’sample-package’,
 packages=find_packages(’./’)
)
```

sample/__init__.py

```
from .core import CoreClass
```

(Note: `sample-package`, `sample`, `core` and `CoreClass` are example names ; they depend on your code.)

Install the module using pip

From the directory that contains `setup.py`, type:

```
pip install .
```

Python interpreter: Outline

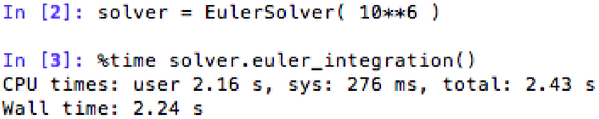
1. Reusing code: functions, classes, modules
2. Faster computation: Forthon
3. Faster computation: Parallel Python

Faster computation

Problem

Large `for` loops are slow in Python.

Example:



```
In [2]: solver = EulerSolver( 10**6 )

In [3]: %time solver.euler_integration()
CPU times: user 2.16 s, sys: 276 ms, total: 2.43 s
Wall time: 2.24 s
```

Solution

- If the operation is of type **element-wise** or **reduction**: Use `numpy` syntax
- Otherwise, rewrite the `for` loop in a **compiled** language (e.g. Fortran, C) and link it to the rest of the Python code

→ **High-level control** with Python (modularity, interactivity)
→ **Low-level number-crunching** with e.g. Fortran or C (efficiency)

Faster computation: Forthon

Forthon

- Generates links between Fortran and Python
- Open-source, created by D. P. Grote (LLNL) `https://github.com/dpgrote/Forthon`
- Heavily used in Warp for low-level number crunching

On the user side:

- Write Fortran **subroutines** and **modules** in a `.F` file
- Write a `.v` file to tell which variables to link to Python
- Compile with Forthon → produces a Python module
- Import the module in Python and use the linked variables

NB: Other similar solutions exist: `f2py` (links Fortran code), `Cython` (generates and links C code), `Numba` (compiles Python code), etc...

Faster computation: task

Task 6

Download and decompress the code from
http://github.com/RemiLehe/uspas_exercise/raw/master/Forthon_task.tgz
The files `acc_euler.F` and `acc_euler.v` are the files needed by Forthon, while `euler.py` is the code from task 5.

- The Fortran file `acc_euler.F` contains an error in the line that starts with `x(i) =` . Spot it and correct it.
- Compile the code with Forthon by typing `make` in the shell. A new file `acc_eulerpy.so` should be created.
- At the beginning of the file `euler.py`, add `from acc_eulerpy import forthon_integration` then create a new method `acc_euler_integration(self)`, which calls `forthon_integration` (see `acc_euler.F` for its signature).

In `ipython`, create an instance with `N=10**6`, and compare the runtime of `euler_integration` and `acc_euler_integration`

Python interpreter: Outline

1. Reusing code: functions, classes, modules
2. Faster computation: Forthon
3. Faster computation: Parallel Python

Faster Computation: Multiprocessing and MPI

- multiprocessing is a python module that introduces an API to access multiple processors on the same **node**.
- very useful for tasks that have many independent repetitive steps (e.g. particle tracing without space charge)

Typical (simple) usage with map():

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    p = Pool(5)
    print(p.map(f, [1, 2, 3]))
```

Message Passing Interface (MPI)

- python can also be used with MPI (e.g. on a big cluster)
- using mpi4py (but necessary to install underlying MPI binaries)
- Remi will talk about parallel computing on Friday, Jan 18th

References

Scipy lecture notes:
`http://www.scipy-lectures.org/` (G. Varoquaux et al., 2015)

Python tutorial:
`https://docs.python.org/3/tutorial/` (Python Software foundation, 2016)

Forthon:
`https://github.com/dpgrote/Forthon` (D. Grote et al., 2016)