poliastro: An Astrodynamics library written in Python with Fortran performance

ICATT, March 16th 2016

Juan Luis Cano Rodríguez

Outline for Section 1

1. Introduction

2. Python as a core computational language
    - 2.1 The Python programming language
    - 2.2 Just-in-time compilation using numba
    - 2.3 Benchmarks against Fortran
    - 2.4 Gradual typing

3. Interfacing with other languages

4. Software reusability and Open Development

The presenter

Introducing myself

- Programmer at the Commercial Flight Dynamics & Operations (CFDO) Department in **GMV**
- Studying **MSc Aeronautical Engineering** at Universidad Politécnica de Madrid
- Erasmus at Politecnico di Milano (**MSc Aerospace Engineering**)
- Free & Open Source Software (**FOSS**) advocate, **Python** enthusiast and practicioner

Outline for Section 2

1. Introduction

2. Python as a core computational language
    - 2.1 The Python programming language
    - 2.2 Just-in-time compilation using numba
    - 2.3 Benchmarks against Fortran
    - 2.4 Gradual typing

3. Interfacing with other languages

4. Software reusability and Open Development

The Python programming language

Its presence in the scientific and academic community

- Python was started in 1989 and v1.0 released in 1994
- One of the most used languages in fields like Astronomy[2] and small-to-medium Data Science
- “The Most Popular Introductory Teaching Language at Top U.S. Universities”[3]

The Python programming language

Simple, readable and easy to learn

```python
while count < numiter:
    y = norm_r0 + norm_r + A * (psi * c3(psi) - 1) / c2(psi)**.5
    # ...
    xi = np.sqrt(y / c2(psi))
    tof_new = (xi**3 * c3(psi) + A * np.sqrt(y)) / np.sqrt(k)

    if np.abs((tof_new - tof) / tof) < rtol:  # Convergence check
        break
    else:
        count += 1
        if tof_new <= tof:  # Bisection check
            psi_low = psi
        else:
            psi_up = psi
        psi = (psi_up + psi_low) / 2
```

The Python programming language

…however, dynamic and slow

```
In [2]: list = list(range(0,100000))
In [3]: %%timeit
   ...: sum(list)
   ...:
1000 loops, best of 3: 1.32 ms per loop

In [4]: array = np.arange(0, 100000)
In [5]: %%timeit
   ...: np.sum(array)
   ...:
[...]
10000 loops, best of 3: 38.9 us per loop
```

Just-in-time compilation using numba

Ahead-of-time (AOT) compilation

Code is compiled *before* it is executed.

Just-in-time (JIT) compilation

Code is compiled *during execution*.

With numba, the code goes through several stages of optimization using the LLVM compiler infrastructure until machine instructions for the desired platform are generated

Just-in-time compilation using numba

An example

```
# --- LINE 29 ---
#   $103.3 = unary(fn=-, value=psi)  :: float64
#   $103.4 = global(gamma: <built-in function gamma>)  :: [...]
#   $const103.5 = const(int, 5)  :: int64
#   $103.6 = call $103.4($const103.5)  :: (int64,) -> float64
#   $103.7 = $103.3 / $103.6  :: float64
#   delta = $103.7  :: float64

delta = (-psi) / gamma(2 + 2 + 1)
```

Just-in-time compilation using numba

An example

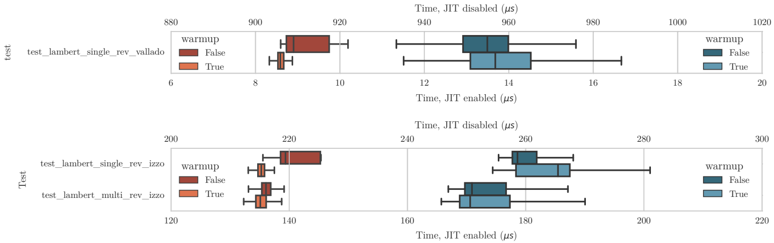
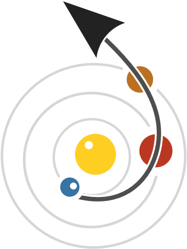


Figure: Comparison of running times of the BMW-Vallado and Izzo algorithms, with and without JIT compiling.

poliastro, a Python Astrodynamics library

- Pure Python, accelerated with numba
- Physical units (thanks to astropy[4])
- Analytical and numerical orbit propagation
- Conversion between position/velocity, classical and equinoctial orbital elements
- Simple 2D trajectory plotting
- Hohmann and bielliptic maneuvers computation
- Initial orbit determination (Lambert problem)
- Planetary ephemerides through SPK SPICE kernels (thanks to jplephem)



Benchmarks against Fortran

Fortran is fastest, but Python + numba is within the same order of magnitude

Version	Min	Max	Median	Relative
Intel ifort, -O2	594620.8	654121.4	623536.2	**1.0**
GNU gfortran, -O2	358478.2	505127.0	454613.6	**0.729**
Python + numba	197610.9	206153.2	203615.8	**0.327**
pure Python	3502.7	3703.0	3639.6	**0.006**

Table: Benchmarking results

Gradual typing

New in Python 3.5 (2015)

```
def greeting(name: str) -> str:
    return ’Hello ’ + name
```

Python 3.5 introduced a new provisional module adding *type hints* focusing on providing indirect help to be used by Integrated Development Environments (IDEs) and other tools to supply more useful information to the developer[1].

[1] https://www.python.org/dev/peps/pep-0484/

Outline for Section 3

1. Introduction

2. Python as a core computational language
    - 2.1 The Python programming language
    - 2.2 Just-in-time compilation using numba
    - 2.3 Benchmarks against Fortran
    - 2.4 Gradual typing

3. Interfacing with other languages

4. Software reusability and Open Development

C and C++: CFFI

```
from cffi import FFI

ffi = FFI()
ffi.cdef("""
double hyp2f1 ( double a, double b, double c, double x );
""")
ffi.set_source("_hyper", """
double hyp2f1 ( double a, double b, double c, double x );
""",
    libraries=["md"],
)
if __name__ == ’__main__’:
    ffi.compile()
```

C and C++: ctypes, Cython, SWIG

- ctypes[2] is similar to CFFI, it's available in the standard library but less powerful
- SWIG[3] generates bindings for parts written in C or C++ to several languages, including Python
- Cython[4] allows starting with pure Python code that gets compiled into C for an immediate improvement in performance
  - type declarations can be added to certain variables and functions to allow more code to run natively

[2] `https://docs.python.org/3/library/ctypes.html`

[3] `http://www.swig.org/`

[4] `http://cython.org/`

Fortran: f2py

Excellent for legacy FORTRAN 77 code

f2py[5] wraps FORTRAN 77 and a subset of Fortran 95[6] directly in Python by generating intermediate C wrappers.

```
$ f2py [-h module.pyf] -m module module.f90
```

[5] http://www.f2py.com/

[6] Notably, it does not support derived types

Java and MATLAB

- JCC[7] is "a C++ code generator that produces a C++ object interface wrapping a Java library via Java's Native Interface (JNI)". It is successfully used by the Orekit Python wrapper, which allows using the Orekit Java library from a Python program.
- pymatbridge[8] is a communication layer between MATLAB and Python based on the ZeroMQ socket library. oct2py[9] is an equivalent tool for the GNU Octave project. The latter had been successfully tested in poliastro v0.1.

[7] http://lucene.apache.org/pylucene/jcc/

[8] https://arokem.github.io/python-matlab-bridge/

[9] https://github.com/blink1073/oct2py

Outline for Section 4

1. Introduction

2. Python as a core computational language
   - 2.1 The Python programming language
   - 2.2 Just-in-time compilation using numba
   - 2.3 Benchmarks against Fortran
   - 2.4 Gradual typing

3. Interfacing with other languages

4. Software reusability and Open Development

Free/Open Source software

A complicated but important topic

- Examples of successful libraries for Astrodynamics: astropy[4] (Python), Orekit (Java)
- However a high percentage of the software available on the Internet has no license whatsoever (SOFA until 2009)
- Public Domain is not a sensible choice, since copyright law is different from country to country
- Viral licenses (GPL family) pose concerns for companies and cannot be combined with closed source products
- **Fortunately, most scientific Python libraries are released under permissive licenses (MIT and BSD)**[10]

[10] http://nipy.sourceforge.net/software/license/johns_bsd_pitch.html

Open development

Some features

- Carrying development discussions on public mailing lists
- Displaying a public list of issues and known defects
- Publishing the complete history of the project using SCM[11] tools
- Performing public code reviews
- Using Continuous Integration environments and striving for a high statement or branch coverage
- *Embracing democratic and transparent decision making processes, with a focus on diversity and safety*[12]

[11] Source Control Management

[12] Not specific to Open Development, but worth considering

Conclusions and future work

- **Python** is well considered in the scientific and technical community
- Using the right tools and under certains circumstances it can attain decent performance
- numba still misses support for high level functions and closures
- IDEs and libraries provide helpers for **type checking**
- Possibility to create powerful static analyzers
- There are several ways to *communicate Python with lower-level languages*
- Code reuse and **open development** approaches lead to high quality software

Bibliography

TEX, LATEX, and Beamer

[1] Dario Izzo. *Revisiting Lambert's problem*. Springer Science + Business Media, 2014.

[2] I. Momcheva and E. Tollerud. *Software Use in Astronomy: an Informal Survey*. Available at `http://adsabs.harvard.edu/abs/2015arXiv150703989M`.

[3] Philip Guo. *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*

[4] Thomas P. Robitaille and others. *Astropy: A community Python package for astronomy*. EDP Sciences, 2013.

Thanks!

- The software: `https://poliastro.github.io`
- Examples: `http://nbviewer.jupyter.org/github/poliastro/poliastro/blob/master/index.ipynb`
- The paper: `https://www.overleaf.com/read/kjjbwvfkgrxs`
- The mailing list: `https://groups.io/g/poliastro-dev`
- My email: 🖂 `hello@juanlu.space`

Per Python ad Astra!