Experience with Using Python for Teaching Scientific Computing

Hans Petter Langtangen (hpl@simula.no)

Apr 29, 2014

Abstract

This essay explains why and how we have chosen Python as the language of choice for teaching scientific computing at the University of Oslo. By teaching students Python and numerics from day one, various bachelor programs have taken advantage of this knowledge and reformed classical science courses by using programming and numerical simulation to solve mathematical problems. We have learned several lessons:

- The choice of Python as a teaching language for scientific computing has been a great success and is highly recommended.
- Python provides a natural, continuous glue between MATLAB-style "flat programs", procedural programming, object-oriented programming, generative programming, and even functional programming.
- It is possible to treat quite advanced problems very early in the studies. For example, at the end of the first semester our students implement an object-oriented toolbox for solving a wide class of nonlinear vibration problems (!).
- Replacing classical mathematical solution techniques by programming and numerical simulation in science courses is indeed challenging, but possible and natural. The experience is very positive and such reforms are strongly encouraged.
- Students come with all sorts of laptops. Force everyone to use Ubuntu as this minimizes the technical hassle with installing complicated mathematical software.

We started with a course on Python scripting

In 1999, I started a course (IN228, renamed to INF3331) about scientific scripting at the University of Oslo. The purpose was to use a real scripting language, primarily Perl, to do all the administrative computer work associated with scientific investigations. During the 90s we had tried to throw the "Camel Book" for Perl4 and the "Unix Power Tools" book at people, but few (if any) understood why and how they should apply such tools.

Within a couple of years, we naturally moved from Perl to Python as the primary scripting language, because Python did the same as Perl, but with a more readable syntax and hence easier maintenance. Students picked up Python significantly faster than Perl, and we started realizing the pedagogical strengths of the Python language. I wrote a book[1] for the course,

[1] http://www.amazon.com/Python-Scripting-Computational-Science-Engineering/dp/3642093159/ref=sr_1_2?ie=UTF8&qid=1340530481

but the publisher was initially only modestly interested because the market was considered too small. Our gut feeling, however, told that scripting with Python with time would gain a significant position in the scientific computing community. At the time of this writing, this is a fact. Over 1500 students have completed the course, and the book has been very popular world wide.

The course addressed experienced Fortran, C, C++, or Java programmers and aimed at teaching them the power of a dynamically typed environment. Early in the 2000s, our research group started using Python for scientific computations, not only scripting. Our students used Python (with NumPy, SciPy, and friends) as a MATLAB replacement and migrated slow parts of the code to Fortran or C++ when needed. It stroke us that students should learn about this effective numerical computing environment as early as possible.

We chose Python for learning computer programming

Most universities buy an introductory programming course from their computer science department. Very often the offer is a standard Java course. Exceptionally the choice is C, C++, or perhaps something more exotic like Scheme. Beyond the very basics of programming, the content is usually typical computer science algorithms rather than numerical algorithms. The result is often that students for all practical purposes must relearn to program when they meet programming in science courses, and then the language is usually MATLAB, Fortran, C, or C++.

At the University of Oslo, the introductory Java course was free of any mathematics and not synchronized with the needs in later science courses. The mentioned relearning of programming consumed much time when they met a totally different language and totally different applications of programming at the end of the bachelor studies. We therefore decided around 2006 to develop an introductory programming course that could lay a firm foundation for continued programming in all later courses that could benefit from integrating numerical simulations. We would therefore teach programming in the context of relevant algorithms like numerical integration, differentiation, solution of differential equations, and Monte Carlo simulation.

The introductory programming course was scheduled for the very first semester, meaning that the students would learn to program and use the computer to solve mathematical problems from day one. But what should be the language of choice? MATLAB or C++ were obvious candidates because of their popularity. These two languages are, however, very different in nature, and it is not natural to integrate them in a course. Python, on the other hand, can be used for MATLAB-style programming and object-oriented or generative C++-style programming. In this way Python provides a natural glue between different programming styles. We also knew that Python was much easier to use and teach than C++. The most convincing argument was, nevertheless, that Python may act as a MATLAB replacement for students and professors, doing the same things as MATLAB, and for free, but with a much more powerful programming language at disposal.

Again I wrote a book[2] for the introductory programming course, this time addressing newcomers to programming and concentrating the examples on numerical computing. Needless to say, there were many skeptics: Python was of marginal interest to scientific computing, Python was a bad first language (because of its dynamic typing), mixing programming and mathematics is inferior to first mastering calculus and Java separately, etc. etc.

The course[3] is now a huge success, and the book is a best-seller in its category. Since 2007, more than 1500 students have passed the exam and used the programming technology to solve mathematical problems in about 20 other courses in the Bachelor programs at the

[2] http://www.amazon.com/Scientific-Programming-Computational-Science-Engineering/dp/3642183654/ref=sr_1_1?ie=UTF8&

[3] http://www.uio.no/studier/emner/matnat/ifi/INF1100/index-eng.xml

University of Oslo. The introduction of programming and numerical simulation in all these other courses was a huge undertaking known as the Computing in Science Education project. This project has attracted worldwide attention as it represents a long-awaited reform of the science education. The idea is trivial: let programming and computer simulations change the curriculum! However, the implementation among professors is extremely demanding most places. The fortunate situation in Oslo is that professors across disciplines have managed to collaborate on how programming and simulation can be integrated in classical courses and used to replace the classical strong attention on complicated and specialized mathematical techniques for very simplified physical problems. Recently, the Computing in Science Education project has been acknowledged by the University and the government in Norway through prestigious awards[4] and generous funding. The ideas are being implemented at other Norwegian colleges and universities as well as at several institutions abroad.

There is impact of teaching Python

The world is changed only when people do new things. When students learn Python, they bring this knowledge to later courses and then to industry. Their knowledge automatically creates a demand for using the tools they prefer. Many companies in the Oslo region that apply Python today discovered the language through new employees who had picked up Python in the scripting course. Similarly, students create Python examples and projects in other courses and thereby attract attention to the tool and its applications.

The science education at the University of Oslo puts much emphasis on learning other languages, in particular MATLAB, R, C, C++, and Fortran. We teach (a subset of) Python in a way that eases the transition to and from MATLAB. For high performance we emphasize reusing Fortran and C libraries from Python or migrating slow Python code to compiled languages. Our experience shows that users who gain a good knowledge of Python tend to prefer that language and do as much as possible in Python before the quest for high performance demands using compiled languages. It seems that many students and professors, when their knowledge of Python is at the level of their MATLAB skills, steadily drift to do most of their work in Python. The numerical functionality in Python is not superior to MATLAB, so this drift is more rooted in Python's clear syntax, powerful software engineering tools, easy integration with Fortran and C/C++, and rich set of libraries for non-numerical tasks.

The described trend among students and professors are also found in industry: Python steadily eats of the MATLAB market because people find the language stronger and more convenient. Our efforts in teaching and professional use of Python contribute to the observed exponential growth.

We let the students play with advanced problems

As mentioned, we start with numerics and programming when the 19-year old new students arrive at the University. About 300 students enter courses on classical calculus, numerical calculus, and programming of the latter. The programming course covers basic MATLAB-style procedural programming in the first half of the fall semester, while the second half is spent on statistical simulations, ordinary differential equations, and class programming (recall that classes and object-oriented programming were invented in Oslo so we have no other choice!). The final project is to create a flexible, object-oriented toolbox for studying nonlinear vibration problems described by the standard model

[4] `http://simula.no/news/computing-in-science-education-receives-learning-environment-award`

$$m\ddot{x} + f(\dot{x}) + s(x) = F(t)$$

The development environment for programming consists of the well-proven terminal window and a text editor (`gedit`, `emacs`, or `vi`).

Many reviews of the course book[5] claim that the book is quite advanced and may fit the graduate level. However, most of the book is in fact covered the very first semester in Oslo. This works well and the course has received excellent student critiques. Scientists usually think of Monte Carlo simulation and nonlinear differential equations as advanced topics, not belonging to first semester courses, but the methods and their applications can be exposed in very intuitive ways suitable for newbies. At least the students can train their programming skills on such problems and be able to produce solutions to what many traditionally think of as advanced problems. A deeper understanding of what they actually do needs to wait until they master a broader set of courses. My main message is that what is basic and what is advanced should be continuously up for discussion.

We use Ubuntu to minimize installation problems

Many fear to introduce programming and simulation in courses because there are so many technical details. For example, how shall students install Python and 30 software packages on their personal laptops? This question quickly scares teachers. We have found a solution that has proven to be successful over the last four years: *force all students to use Ubuntu*, and provide support for Ubuntu only!

Mac users typically run Ubuntu in a virtual machine (VirtualBox[6], or preferably, VMWare Player[7] or VMWare Fusion[8]), while Windows users can run a virtual machine or have a dual boot. The Wubi[9] software makes installing the dual boot solution very easy. On Ubuntu we have one package (`python-scitools`) that installs everything the students need in one command. Later courses can just provide a one-line command with their needs for additional packages.

There are several reasons why we dare to force the use of only one operating system:

1. Installation of programming tools and mathematical software on Windows and Mac OS X requires comprehensive competence that very few students have and want to gain.
2. On Debian Linux systems, including Ubuntu, installation of complex mathematical software is done by a one-line command.
3. Debian Linux has the largest collection of mathematical software today.
4. When the students need to compile and link their own software, this seems to be easier on Linux systems than on Mac OS X and Windows. Although Mac OS X is basically Unix, there are many technical peculiarities that quickly calls for competence beyond what students and the average teacher have.
5. Ubuntu's graphical interface is very similar to Windows or Mac OS X so students pick up Ubuntu without any noise.

[5] `http://www.amazon.com/Scientific-Programming-Computational-Science-Engineering/dp/3642183654/ref=sr_1_1?ie=UTF8&`
[6] `http://www.virtualbox.org/`
[7] `http://www.vmware.com/products/player`
[8] `http://www.vmware.com/products/fusion/overview.html/`
[9] `http://wubi-installer.org`

6. Students struggle much more with the logic of programming than with Emacs and Unix commands.

I should add that Fedora Scientific[10] may provide a viable alternative to Ubuntu, especially on sites where Fedora is already the supported Linux system. Fedora Scientific now offers a lot of very useful mathematics software[11]

In future cloud supercomputing I imagine one can just upload the Linux image (running in a virtual machine) to the cloud service and avoid tedious installation processes. By archiving the image along with scientific results, one can at any time rerun simulations - the complete operating system, all the needed software, and all data reside in the image. This is key to reproducible science.

So far, I have not heard one single negative comment that we support only one operating system. When we tried to support Unix, Linux, Windows, and Mac, there were a lot of complaints from students that they did not have access to the right software and that technical problems with computers stole too much attention. These complaints have simply disappeared with the standardization on Ubuntu. So we have learned that giving people more choices does not necessarily make them happier.

A final comment. The use of Ubuntu is mainly motivated by the total need of mathematical packages and programming tools during a student's entire stay at the university. If the need is basically an environment for doing Python programming, the Enthought Python Distribution[12] or the Python(x,y)[13] package provide user-friendly environments on any standard Windows computer.

Resources

- Published formats of this essay: PDF[14], HTML[15]
- The book Python Scripting for Computational Science[16]
- The book A Primer on Scientific Computing with Python[17]
- Student information[18] on our introductory programming course where Python is used
- The Computing in Science Education[19] reform of science courses at the University of Oslo
- How to install Ubuntu[20] on any computer using VMWare Fusion

[10] http://www.floss4science.com/fedora-scientific-amit-saha/
[11] http://spins.fedoraproject.org/scientific-kde/
[12] http://www.enthought.com/products/epd.php
[13] http://code.google.com/p/pythonxy/
[14] http://hplgit.github.com/teamods/edu/uiopy/uiopy.pdf
[15] http://hplgit.github.com/teamods/edu/uiopy/uiopy.html
[16] http://goo.gl/q8tM7D
[17] http://goo.gl/SWEQlz
[18] http://www.uio.no/studier/emner/matnat/ifi/INF1100/index-eng.xml
[19] http://www.mn.uio.no/english/about/collaboration/cse/
[20] http://hplgit.github.com/teamods/ubuntu/vmware/index.html