Lecture Notes in Earth System Sciences

Gabriele Morra

Pythonic Geodynamics

Implementations for Fast Computing

Springer

Lecture Notes in Earth System Sciences

More information about this series at http://www.springer.com/series/10529

Gabriele Morra

Pythonic Geodynamics

Implementations for Fast Computing

Springer

Gabriele Morra
Department of Physics and School
of Geoscience
University of Louisiana at Lafayette
Lafayette, LA
USA

ISSN 2193-8571 ISSN 2193-858X (electronic)
Lecture Notes in Earth System Sciences
ISBN 978-3-319-55680-2 ISBN 978-3-319-55682-6 (eBook)
DOI 10.1007/978-3-319-55682-6

Library of Congress Control Number: 2017937696

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

In recent years, there arises a great need for a freshly crafted beginners text book in computational geosciences, because of the rising levels of capable university students with improved background in mathematics and physical sciences. Today in the USA and other countries, many freshmen and high school seniors have sufficient background and sophistication from advanced placement courses to appreciate a firm knowledge of computational knowledge is needed to solve real problems in geosciences without going through the drudgery of analytical solutions and analysis typically found in traditional textbooks, written in the twentieth century. So, they are more than ready to try some new pedagogical undertaking, as present in this volume prepared by Gabriele Morra.

The most direct link between simple physical laws and the real world is the numerical solutions of time-dependent partial differential equations from real systems in terms of realistic examples with physical meaning. On the one hand, experimental technique and data collection have seen such dramatic progress that by now many fundamental properties of geophysical fluid dynamics can be demonstrated in laboratory experiments, such as thermal convection and mixing. On the other hand, great efforts are being made to exploit ideas from nonlinear dynamics in cases where the system is not necessarily deterministic but the data displays more structure than can be captured by traditional analytical methods. Problems of this kind are typical in many areas of science where multi-scale phenomena prevail as in geosciences but in many other sciences, like in biology and physiology.

This book by Morra provides the young students in geosciences with a sound knowledge of tools such as Python and modern numerical techniques for more

incursions into other burgeoning areas such as Big Data and High-Performance Computing. I hope that this book will inspire others to write similar books for future generation of students. This book will indeed bring a fresh approach to computational geosciences.

February 2016

David A. Yuen
University of Minnesota
Minneapolis, MN
USA

Preface

Adults, like children, learn by playing. With this insight in mind, I wrote this book as a non-exhaustive and non-overambitius text that aims at introducing computational geodynamics to young students, either undergraduates or beginning graduates. The goal, more than trying to cover a topic that is way too vast for a book to contain it, is to show some of the fundamental strategies available with a strong focus on practical, playful, easy-to-use techniques.

Python is now a standard tool for programming in science and even more in the industry, where the explosion of the Big Data, mining, and artificial intelligence research fields has convinced many professionals to learn this simple and powerful language. While I write, Python, combined with its visualization and numerical libraries, is replacing MATLAB in many scientific and technological contexts, because it offers the same capabilities but with the great advantage of being open source. Open projects attract more developers which, in the long term, always overcomes closed/commercial tools if the user base is sufficiently large. Many developers are in this moment turning existing C and Fortran libraries into Python, among them the ones that allow parallel computing, programming NVIDIA GPUs, running machine learning algorithms, and many more. Although Numerical Python is not the only fast growing scientific/technological language, as others such as Ruby and Scala are emerging as well, Python is the present state of the art in scientific computing.

Geodynamics, the study of the evolution of our planet and similarly of the other solid planets, is a fast evolving field. It constantly englobes new research topics ranging from the dynamics of the Earth's core, to the convection of its slowly deforming solid mantle, from the study of the dynamics of the Geoid (the *gravitational* shape of the Earth) to the global resonance of the entire planet to the stimulus of a large Earthquake on its surface, from the dynamics of the two-phase gas-magma system in a volcano conduit to the estimation of the risks related to small and giant eruptions, from the dramatic formation of our planet to the increase of its mineralogical variety during its evolution, from the detection of the chemical spectrum of the atmosphere of some exoplanets modelling their interior dynamics.

Given that most of these systems are at the same time complex and inaccessible to direct measurements, numerical modeling plays a key a role to their understanding. I have been teaching numerical modeling applied to geosciences, and geodynamics in particular, in my past academic positions at the University of Sydney, Seoul National University, and presently at the University of Louisiana at Lafayette. In my courses I have used numerous books that covered several aspects of geodynamics, however many of the key progresses in the past 30 years have been driven or confirmed by well crafted numerical experiments. I have, therefore, decided to write a textbook that focuses on the practice of designing these numerical experiments, organized as a hands-on manual, almost like one of these online tutorial that programmers use every day, but with a greater structure, designed to teach undergraduate and graduate students to create their own numerical models.

In my courses, I verified that the insight that students gain by designing and writing a code is often greater than the understanding gained by the solution itself. To create a software that simulates the Earth's interior behavior is like to teach geodynamics to a machine, and requires a very clear understanding of the underlying physics behind geological phenomena. This book combines the material taught in three different occasions. One is the introduction to computation to honor sophomore students of general physics at the University of Louisiana at Lafayette. The second is a course of computational geophysics taught at the same university to graduate students during the academic years 2013/2014 and 2014/2015. Finally, I added advanced topics such as the one taught at a summer school on large-scale Boundary Element Method (BEM) at the University of Brescia, Italy, in 2011.

The book is structured in four parts. In the first one, I introduce in three chapters a (i) bird's eye view of the computational capabilities of Python, (ii) the visualization tools available in Python, and (iii) how to use the powerful Numerical Python libraries and other numerical tools to embed C, OpenMP, MPI capabilities to our Python programs.

In the second part, I illustrate few examples of how to use Numerical Python to solve typical problems of a standard general physics course. These problems are normally solved only using analytical tools, while here I illustrate how to approach them numerically, and at the same time familiarize with momentum and energy equations, the main ones that are solved in geodynamic modeling. The goal of this part is to introduce the less expert reader to using Python, solve numerically some simple problems, and teach at the same time how to calculate standard physics quantities such as momentum, work, power, dissipative energy, and the action. In this part, we will for the first time learn how to numerically calculate derivatives and integrals, how to monitor the accuracy of a numerical solution, and how to visualize the results of a computational model.

In the third part of the book, I introduce the main laws of continuum mechanics that every geodynamicist needs to know, diffusion and momentum equations in a continuum context, and how to use Finite Differences, and the advanced *Particle in Cell (PIC)* technique to solve them. The goal of this part is to describe how to write compact and elegant but still very fast routines that allow implementing a

sophisticate and advanced code such as *PIC* in simple and straightforward manner. Applications to solving the equations that describe strongly viscous flow, porous media flow, elasticity, both in a linear and nonlinear setup are introduced in a simplified and introductory manner. Also some important issues such as numerical stability/instability are shown using examples.

The fourth and last part of the book covers a small set of more advanced topics such as numerical approaches that do not require the discretization of the space, but are built from the summation of fundamental solutions. This will give the reader the opportunity to familiarize with tree-based codes, that are at the core of many modern numerical techniques and can efficiently solve the many body interaction problem that applies to many aspects of geosciences and beyond, such as planetary science. A specific application to the calculation of the dynamics of a multiphase fluid is finally introduced and applied to the motion of gas bubbles in a magmatic system.

I believe that this book still lacks much material that I would like to add. In particular more details on the solution techniques for the non-linear Navier-Stokes equation, lagrangian multiplier method, detailed implementations of the tree fast multipole method, and many more examples of how to parallelize and benchmark these codes. I have however to stop here and plan these additions for a second edition of this book. I hope that the very incomplete material of this first edition will anyway result useful to some young students who approach the fascinating field of computational geodynamics for the first time.

Who Should Read this Book

This book has been written for ambitious undergraduate college students and for graduate students in geophysics/geodynamics. I eliminated most of the calculus based derivations, while I focus on in implementations and examples, with the idea that it is the 'practice' that makes the 'scientist', and that 'creativity' is the product of 'perseverance'. This book is, therefore, designed for readers of every background who desire to learn about the behavior of our planet by explicitly modeling it.

This book is also addressed to more advanced practitioners who have been modelling geodynamics using different programming languages than Python and aim at trying new numerical techniques. In many ways Python is the best language for prototyping scientific implementations, and also for running some high performance simulations. I remember that when I begun my Ph.D. in Geodynamics, I dreamed about learning everything, mantle convection, earthquakes, seismic wave propagation, oceanography, glacial rebound, exoplanets, surface processes, and so on, but quickly I realized that to write an efficient, robust, and reliable software for modeling is a huge work. I certainly do not affirm that now that with the scientific environment of Python, and the emerging new tools that come from Machine Learning, this is today possible, but certainly presently a numerical modeler can aim at a much broader and multidisciplinary project than it was possible 15 years ago.

How Should this Book be Read

This volume is structured with increasing difficulty, starting from a level accessible to freshman college student. Through the book the level rises until reaching some topics chosen from the present-day geodynamics research. The goal of this book is to offer to the reader the key instruments not only to create general and powerful computational tools, but also a clear understanding of the difficulty of implementing them.

The reader is expected to test all the examples proposed and try to do as many exercises as possible. Real learning is achieved only by writing a software by our own, and this is much easier to achieve in Python compared to standard scientific languages such as C and Fortran.

The reader with little background in computational sciences will find easier to study the book in the same order as the material is presented, but the expert programmer can safely skip the first chapters where the main tools for achieving high performance with Python are introduced. Only from Chap. 7 the book starts to build on past shown examples, therefore sequential reading is recommended.

Great software for Geodynamics already exist today, such as *Access*, *Underworld*, and Terra. Still it is possible today, by using smart programming to quickly but gradually guide a student to building a geodynamic modeling software. The goal of this book is not to push the student toward competing against large projects, but to prepare them to understand how they work in order to work within these projects, or to develop new modules for running techniques never implemented elsewhere. In other word the idea of this book is to invite students to learn by experimenting in freedom.

Lafayette, Louisiana, USA Gabriele Morra
December 2016

Acknowledgements

This book was written for one year and a half from summer 2015 until the end of 2016. During this time I took a year leave from my university position to stay close to my family, who lived in another continent. Far from the university environment, from teaching and from the possibility to go to meetings or anyway interact with other scientists, I decided to write this book. I have, therefore, to thank the many people who supported and stimulated me, both emotionally and physically during this time.

Certainly in this time my family made the greatest sacrifice. Much of time that I planned to spend, during weekends and holidays, with my wife Dorothea and my daughters Marlen and Renee, was instead invested into writing many of the pages of this book. Many nights were spent into rewriting entire chapters of this book, with the result that I was not fully there the day after for them. I thank them immensely for their support and patience.

I want to thank my graduate student Prasanna Mahesh Gunawardana. Much of the material in this book has been shown to him first and he used to develop a software. Based on his feedback, and his questions on what to better explain and how, I added many of the routines that I selected for this book. I also want to thank you my entire class of the computational geological modeling at UL at Lafayette of 2014: with them I tested first how to teach Numerical Python to young graduate geology students.

I would like to thank those who provided me with useful feedback on the entire manuscript or on part of it. Among them, Sang-Mook Lee, Erik Sevre, Fabio Capitanio, Manuele Faccenda. I would like to thank Paul Russell Woodward for suggesting to use the word *Pythonic* in the title. A special thank to Peter Mora who carefully read the entire book and made many suggestions.

Last but not least, I have to thank David A. Yuen. His continuous support and patience has been incredible. Every time I started to become distracted with some new interesting topic he immediately called me back and forced me to focus into finishing this book. Without his tireless stimulus, this book would have probably never seen the light.

Contents

Part I Introduction to Scientific Python

1 Bird's Eye View 3
- 1.1 Bird's Eye View 3
- 1.2 History 4
- 1.3 Programming or Scripting 5
- 1.4 Python Interfaces 5
  - 1.4.1 IPython: Interactive Python 6
- 1.5 Few Words on Syntax 8
- 1.6 Extending Python 10
  - 1.6.1 Importing Libraries 11
- 1.7 NumPy: Numerical Python 11
- 1.8 Visualization 12

Summary 13
Problems 13

2 Visualization 15
- 2.1 The MatPlotLib Visualization Library 16
  - 2.1.1 Plotting a 2D Field 17
  - 2.1.2 Plotting a Map 18
  - 2.1.3 NetCDF and ETOPO 20
  - 2.1.4 Plotting a Seismic Waveform 22
- 2.2 Plotting in 3D with MatPlotLib 24
  - 2.2.1 VTK File Format 26
- 2.3 Example: Length of the Day 27
- 2.4 IPython and Jupyter Notebooks 29
- 2.5 Paraview and VisIt 30
- 2.6 Python as a wrapper: SEATREE and Underworld 31

Summary 32
Problems 32

3 Fast Python: NumPy and Cython 35
3.1 How Fast is Your Computing Machine?. 36
3.2 Numerical Python 37
3.2.1 NumPy Types 38
3.2.2 ndarrays. 39
3.3 Indexing and Slicing. 42
3.3.1 N-Dimensional Indexing 44
3.3.2 Boolean Indexing 44
3.3.3 Transposing and Axis Rotation 46
3.4 Strides. 47
3.5 Vector Products 48
3.6 Linear Algebra 50
3.7 Cython 52
3.7.1 Cython in iPython 53
3.8 Going Parallel: mpi4py and PETSc4py. 54
3.9 Other Computational Modules 58
Summary. 59
Problems 60

Part II Second Part: Mechanics

4 Mechanics I: Kinematics. 63
4.1 Computation of Velocity and Acceleration 64
4.2 Integrate Acceleration 68
4.3 Projectile Trajectory 71
4.4 Circular Motion 73
Summary. 74
Problems 74

5 Mechanics II: Newtonian Dynamics. 77
5.1 Analytical Solutions for 1D Dynamics 77
5.1.1 1-D Dynamics 79
5.1.2 2D Dynamics 80
5.1.3 Potential, Dissipated, Kinetic, Mechanical Energies for the Droplet 82
5.2 Monte Carlo Simulation of the Pyroclastic flow During the 1944 Mt Vesuvio Volcanic Eruption 84
5.3 Precession of a Gyroscope 87
Summary. 90
Problems 90

6 Physics of Stokes Flow 93
6.1 Momentum and Continuity Equations. 93
6.1.1 Navier Stokes Equation 96

6.2 Stokes Flow: Simple but Not Obvious 98
6.2.1 Stokes' Paradox 98
6.2.2 Flow Reversibility 100
6.2.3 Origin of the Paradoxes 101
6.3 Fundamental Solutions of Stokes Flow 102
6.3.1 Rotlet 103
6.3.2 Stokeslet 103
Summary 104

Part III Lattice Methods

7 Lagrangian Transport 107
7.1 Strain and Strain Rate 107
7.2 Rigid Rotation 109
7.2.1 Cell-Particles Projections 112
7.2.2 Motion of the Particles 114
7.3 Thinning Flow 116
7.4 Lagrangian Advection of a Continuous Field 119
7.5 Upwind Scheme Versus Lagrangian Transport 125
Summary 127
Problems 127

8 Operator Formulation 129
8.1 Strain Rates 131
8.2 Cell-Centered Strain Rates from Linear Operators 134
8.2.1 Sparse Derivative Operator 138
8.3 Reversible and Irreversible 139
Summary 140
Problems 141

9 Laplacian Operator and Diffusion 143
9.1 Diffusion Processes in Geodynamics 144
9.2 Explicit Diffusion Implementation 146
9.3 Explicit Formulation Using Operators 148
9.4 Implicit Formulation 150
9.5 Two-Dimensional Diffusion Equation 153
9.6 Biharmonic Equation 157
Summary 160
Problems 160

10 Beyond Linearity 161
10.1 Operator Form of the Stokes Equation 161
10.2 Implementation of the Homogeneous Stokes Equation 163
10.3 The Finite Volume Method 165

10.4 Implementation of the Nonhomogenous Stokes Equation 167
10.5 Long-Range Interaction 170
10.6 Advection–Diffusion Equation 175
Summary 176
Problems 176

Part IV Advanced Techniques

11 Trees, Particles, and Boundaries 181
11.1 Tree Building 182
11.1.1 The Barnes and Hut Tree 182
11.1.2 The Warren and Salmon Solution 183
11.2 SciPy k-d Tree 186
11.3 Boundary-Based Simulations 186
11.3.1 Drag over a Rigid Particle 189
11.4 Quadratic Triangular Elements Mesh 190
11.4.1 Calculation of the influence matrix 193
11.4.2 Calculation of the Resistance Matrix 196
Summary 199
Problems 199

12 Applications to Geodynamics 201
12.1 Plate Tectonics 201
12.2 Raise of Gas in a Volcanic Conduit 203
12.3 Interaction Between Faults 205
12.4 Convection in 2D 205

13 The Future 209
13.1 Jupyter 209
13.2 Machine Learning 210
13.2.1 Theano and Tensor Flow 212
13.3 Big Data 215
13.4 Final Outlook 216

References 217

Index 225

Part I
Introduction to Scientific Python

In the first chapter a birds's eye view on how to use Python for scientific purposes is proposed. Then a chapter is devoted to visualization of small and large scientific data and modeling results. Examples range from retrieving and plotting time series of Geodetic data obtaining and visualizing large,open geophysical datasets.

In the third chapter, Numerical Python (Numpy) functions are introduced focusing on fast manipulation of *ndarrays*. It is illustrated in detail how to operate with indexing, slicing, and Boolean operators on arrays in order to achieve the maximum performance, with specific applications to linear algebra. Finally, it is shown how nonexisting functions can be added using *Cython*, *Numba* and others. Parallel programming tools for Python are quickly introduced.

Chapter 1
Bird's Eye View

Readability counts.
Simple is better than complex.
Complex is better than complicated.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
— from The Zen of Python of T. Peters

Abstract In this chapter it is reviewed the history of Python, the main differences between the 2.x and the 3.x versions, and how to choose the appropriate version to start with. It is then shown how to work either with Python in an interactive mode, using iPython (now Jupyter), or in the standard productive mode for large projects, with the main two implementations at present: Anaconda Spyder and Enthought Canopy. The main libraries used through the book are quickly introduced, among them MatPlotLib, NumPy, and Cython.

1.1 Bird's Eye View

The purpose of this book is to introduce the reader to some fundamental techniques used to model the evolution of solid planets, such as Earth, using computers of small as well as great power. I will treat these numerical methods from a very general standpoint, using examples of how to solve fundamental equations in physics and fluid dynamics, and only finally showing the application to geodynamics. The intention is to leave to the reader the freedom to develop the geodynamic model that they desire, without sticking to a specific numerical approach or predetermined assumptions for geodynamics.

To teach these techniques I will employ *Python*, a free programming language that uniquely shares the property of being simple, powerful, and widely used in science and engineering. During the development of Python many projects have grown together with it. For example, *NumPy* has been developed to increase performance for all vectorial operations, the main tool necessary to solve numerical problems. *MatPlotLib* has been introduced to offer a visualization experience similar to the

G. Morra, *Pythonic Geodynamics*, Lecture Notes in Earth System Sciences,
DOI 10.1007/978-3-319-55682-6_1

commercial package MATLAB, but has now grown to being the standard for Python users.

In the first part of the book I will teach to use *iPython* (interactive Python), a powerful front end that allows to easily familiarizing with the language and testing it. Later in the book, I will present more advanced and sophisticated implementations that can be run from a bash shell or from a comprehensive environment. While I write, the two largest projects aimed at developing a universal environment for Python are *Anaconda Spyder* (`https://www.continuum.io/`) and *Enthought Canopy* (`https://www.enthought.com/products/canopy/`). Both are available and come with the numerical and visualization libraries on Linux, OS X, and Windows platforms, allowing every user to skip the tedious procedure of installing a large number of library set.

1.2 History

The genesis of Python goes back to the work of the Dutch programmer Guido van Rossum, who created the first implementation in December 1989, and released it for the first time in 1991. For this reason, he has been informally nominated *Benevolent Dictator for Life* by the Python developers community. This simply means that in case of an unsolvable controversy he will take the definitive decision.

Since then, Python together with other popular dynamic programming languages such as Perl and Ruby, has taken off becoming the standard in many sectors of software development. Definitive versions have been released on January 1994 (1.0) and on October 2000 (2.0).

The development of Python faced a twist on December 2008 when a backwards-incompatible 3.0 version was released. Suddenly some Python software could run only on Python 2.x while others only on Python 3.x, which irritated many users. After 8 years from the first release of Python 3.x, however, the new software is almost exclusively developed for the new version of Python. In fact, Python 2.x will continue to be supported only until 2020. After that, Python 3.x will become the only supported version, therefore it is wise to write your software on Python 3.x only. It is important to emphasize that most incompatibilities between Python 2.x and 3.x are not related to the content of this book, but concern mostly I/O (input/output). Still some new features are essential in allowing Python programmers to use *threads* that are essential in the connected present world, therefore Python 3.x has to be the choice for the new developer. Most examples of this book run equally well on both platforms, with some possible conflicts arising when exporting the results and data.

In many ways computer science is an art. When two different implementations exist for solving the same problem, computer scientists debate about which one is the 'best' one, or the 'most elegant', often related to subjective opinions. Languages are, therefore, preferred by computer scientists when for each problem there is clearly one and only one 'best' way to implement its solution. Python often displays this property, however as T. Peters in *The Zen of Python* observed: "There should be one —and preferably only one —obvious way to do it. Although that way may not be obvious at first unless you're Dutch."

Many books can introduce you to the Python Language. For the ones who did not have any experience at all with programming, ***Think Python*** from Allen B. Downey, O'Reilly, is a clear and simple guide. The book is presently freely available from the website http://www.greenteapress.com/thinkpython/, where a PDF copy of the book can be downloaded, both for the 2.x and 3.x versions. More details, the complete manuals, and tutorials are freely available on the Python Website (http://python.org/), as well as many on other sources such as Code Academy (https://www.codecademy.com/learn/python) and Python Course (http://www.python-course.eu/).

The rise of Python has triggered the development of a huge number of tools to expand its functionality. Its ability to integrate existing compiled software, C and C++ above all, but also Fortran, has allowed building hybrid projects, where Python plays the role of glue between them. In the past years, in particular, Cython (http://cython.org/) has become the principle way for creating compiled extension for Numerical Python. It is as well possible to integrate C and C++ code inside the Python programs, for example, by using CPython.

1.3 Programming or Scripting

Often in literature one reads about *Python scripts*. We will, however, work with programs and not with scripts. What is the difference?

Generally scripts are not structured, but simply a list of instructions that operate on a dataset organized ad hoc for the script. Programs are more autonomous, create own data structures, allocate memory, and in general take more control of the machine. As we will see more in detail in the next chapter using NumPy, Cython, and analogue tools we will take full control of the machine as with a standard programming language, literally *programming* in Python.

The file containing a Python program ends always with *.py* . Python programs are not compiled, but execute instruction by instruction (which ends at the end of line or at a the next semicolon). Python programs, however, are also parsed before execution to check whether evident syntactic errors already exist, and will not run until they are fixed.

1.4 Python Interfaces

In this book I will either show examples from the *Enthought* edition of Python, *Canopy* (https://www.enthought.com/products/epd/) or from *Anaconda*, *Spyder* (https://pypi.python.org/pypi/spyder), both with a similar graphical style to MATLAB. Each of these interfaces are based on a 'three windows' system in which one has an (i) editor, an (ii) object/variable explorer, and a (iii) standard or iPython console. By using these three at the same time, on a large

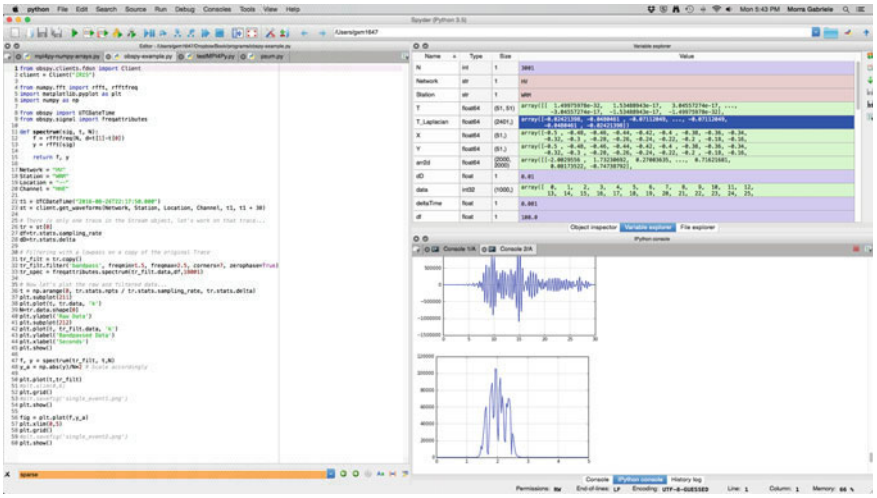

Fig. 1.1 Screenshot of Spyder, the development environment of Anaconda. On the *left* there is the editor. On the *right* on the *top* the variables created in this Python session and on the bottom this *IPython* session with the graphical output

screen, writing, debugging, and testing Python programs is very fast and practical. Fig. 1.1 illustrates a screenshot of Anaconda Spyder.

Installing the *Enthought* and *Anaconda* distributions on a Linux or OSX terminal can be easily done using *bash* or *pip*. Graphical installers exist for both Windows and OSX. Most of the examples shown in this book have been created by using the free version of *Anaconda*.

The easiest way to learn Python and also to begin a new project is to use *iPython*, a project created by the programmer Fernando Perez in 2001 while he was a graduate student in particle physics at the University of Colorado Boulder. During his research Perez realized that the existing tools for scientific computing were insufficient and that Python was the perfect base to start to build new ones. Since then he has been involved in the development of the interactive version of Python, that is today the most used approach to prototyping and experimenting new procedures.

1.4.1 IPython: Interactive Python

When you are ready to learn new computational techniques the question is always what is the fastest and less painful way to start. A straightforward approach is to use *iPython* from the *Anaconda* distribution. After installing the libraries from `https://www.continuum.io/` one can start the iPython with the command:

```
$ ipython
Python 3.5.2 |Anaconda 4.2.0 (x86_64)| (default, Jul  2 2016, 17:52:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
```

Since Python is a relatively new programming language, its tools have been often built in a way to resemble existing well-known software. Aesthetically the *iPython* prompt looks similar to Mathematica. *iPython* has also a number of properties that make it a very practical testing tool, together with its twin environment *Spyder*. People who used MATLAB will find *Spyder* and *Canopy* very familiar.

The online help of *iPython* is very clear and self-explanatory. If this is your first experience with Python I suggest you to play with it and test commands such as:

- Type *help()*, or `help('for')`, or look for other instructions
- `print('I like Geodynamics!')`
- Assign values to string, integer and float variable types.
  - `In[1]: a=1; print(a)`
  - `In[2]: a=1.0; print(a)`
  - `In[3]: a='a'; print(a)`
- Make some mathematical operations (*,/,+,-,**) using the interpreter as a calculator. Notice how python automatically decides which one of the three types of variable, integer, float and string must the answer be.
  - `In[4]: 2+2`
  - `In[5]: 2+2.0`
  - `In[6]: 3**3`
  - `In[7]: 3.0**3`
  - `In[8]: 6/2`
  - `In[9]: 2/6`
  - `In[10]: 'a'+'b'`
- You can also change the type of each variable:
  - `In[11]: int('32')`
  - `In[12]: str(3.2)`
  - `In[13]: int(5.99)`
  - `In[14]: int('5.99') \#this will give you error`
  - `In[15]: int(float('5.99')) \#this works`

If this is your first programming experience, reflect on the answer that *Python* is giving to you. Programming languages have, like in the algebra that we learn at school, the equivalent of *integer numbers*, of *real numbers*, and of *text strings*, therefore its answers reflect these categories, that in Python are called integer (`int()`),

float (`float()`), and string (`str()`). Python has always to choose one among them and the interpretation of your instruction will depend to this choice. This is different from languages such as C or Fortran, where you define to start with the type that you are using.

1.5 Few Words on Syntax

An original and practical characteristic of Python (compared to other programming languages) is that it uses the indentation to determine the body of a loop or a function. It is custom to define your functions at the beginning of the file, or in a separate file and import it at the beginning of the main body. You can start with the following example:

```
In[20]: def printSomething():
            print('I like Geodynamics!')
In[21]: printSomething()
```

Here the **`print()`** instruction is part of the function `printSomething()` while the call `printSomething()` is not indented, therefore it calls the above function.

Functions can be nested one inside another:

```
In[22]: def repeatPrint():
            printSomething()
            printSomething()
```

and inside themselves (recursive functions):

```
In[23]: def fact(n):
            if n==1:
                return 1
            else:
                return n*fact(n-1)

In[24]: print(fact(20))
2432902008176640000
```

Notice how the level of indentation indicates to which loop every instruction belongs. This choice, different from any other language, makes Python closer to human written language, without the large amount of opened and closed parentheses of the C language.

In standard Python one does not need to define the type of every variable. This means that a function will operate differently depending on the type used as input.

```
In[25] : def doublePrint(a):
             print(a+a)

In[26] : doublePrint(2)
Out[26]: 4
In[27]: doublePrint('ciao')
ciaociao
```

The `if` instruction can become a sequence of alternative when combined with `elif`. The standard `or` and `and` logical conjunctions apply:

```
In[28]: def congratulateGrade(grade):
          if grade=='F':
             print('Mmm, something went wrong')
          elif grade=='D':
               print('Not the best performance')
          elif grade=='C':
               print('It is ok!')
          elif grade=='B':
               print('Very good!')
          elif grade=='A':
               print('Fantastic!')
```

Several types of loops exist in Python, however we will exclusively use the `for` instruction. `for` in Python iterates through a list that can contain any kind of object, also belonging to different types. For example,

```
In [29]: for x in [6,'ciao',8*8,congratulateGrade]: print(x)
s6
ciao
64
<function congratulateGrade at 0x10df1d7b8>
```

The last print refers to the function defined above. More in general we will use the `for` loop combined with `range()` or `np.arange()` instructions. For example,

```
In [30]: sum=0
In [31]: for x in range(1000):
             sum+=x
In [32]: print(sum)
Out[33]: 499500
```

As in *C*, `Break` exits a loop and continue to the next iteration. `Pass` does nothing. Another loop instruction that is sometimes used is `while do`. A much more detailed tutorial through all the control flow instructions is available at http://docs.python.org/3.3/tutorial/controlflow.html.