Advanced Information and Knowledge Processing

For other titles published in this series, go to
www.springer.com/series/4738

Sergei V. Chekanov

Scientific Data Analysis using Jython Scripting and Java

Springer

Dr. Sergei V. Chekanov
Argonne National Laboratory (ANL)
9700 S. Cass Ave
Argonne
60439 IL, USA
chakanau@hep.anl.gov

AI&KP ISSN 1610-3947
ISBN 978-1-84996-286-5 e-ISBN 978-1-84996-287-2
DOI 10.1007/978-1-84996-287-2
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2010930974

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

This book is dedicated to my family

Preface

Over the course of the past twenty years I have learned many things relevant to this book while working in high-energy physics. As everyone in this field in the yearly to mid-90s, I was analyzing experimental data collected by particle colliders using the FORTRAN programming language. Then, gradually, I moved to C++ coding following the general trend at that time. I was not too satisfied with this transition: C++ looked overly complicated and C++ source codes were difficult to understand. With C++, we were significantly constrained by particular aspects of computer hardware and operating system (Linux and Unix) on which the source codes were compiled and linked against existing libraries. Thus, to bring the analysis environment outside the high-energy community to the Windows platform, used by most people, was almost impossible.

I began serious development of ideas that eventually led to the jHepWork Java analysis environment in 2004, when I was struck by the simplicity and by the power of the Java Analysis Studio (JAS) program developed at the SLAC National Accelerator Laboratory (USA). One could run it even on the Windows platform, which was incredible for high-energy physics applications; We never really used Windows at that time, since high-energy physics community had wholly embraced Unix and Linux as the platform of choice, together with its build-in GNU C++ and FORTRAN compilers. More importantly, JAS running on Windows had exactly the same interface and functionality as for Unix and Linux! It was a few months after that I made the decision to focus on a simplified version of this Java framework which, I thought, should befit from Java scripting, will be simpler and more intuitive. Thus, it should be better suited for general public use. I have called it "jHepWork" ("j" means Java, "HEP" is the abbreviation for high-energy physics, and "work" means a sedentary lifestyle in front of a computer monitor).

Indeed, I was able to simplify the language and semantic of the JAS analysis environment by utilizing more appropriate short names for classes and methods, which are more suited for scripting languages. The entire project had grown tremendously after inclusion of many new GNU-licensed packages and extending the functionality of JAS in many areas, such as 3D graphics, serialized I/O and numerous numerical packages. At present, jHepWork covers an impressive list of Java-written packages

ranged from basic mathematical functions to neural networks and cellular automation. And, eventually, a little of JAS has left inside jHepWork! One important thing, however, has remained: As JAS, jHepWork was still an open-source software that can be downloaded freely from the Web.

For this project, Python was chosen as the main programming language because it is elegant and easy to learn. It is a great language for teaching scientific computation. For developers, this is an ideal language for fast prototyping and debugging. However, since the whole project was written in Java, it is Jython (Python implemented in Java) that was eventually chosen for the jHepWork project.

This book is intended for general audience, for those who use computing power to make sense of surrounding us data. This book is a good source of knowledge on data analysis for students and professionals of all disciplines. Especially, this book is for scientists and engineers, and everyone who devoted themselves to the quest of where we find ourselves in the Universe and what we find ourselves made of.

This book is also for those who study financial market; I hope it will be useful for them because the methods discussed in this book are undoubtedly common to any scientific research. However, I have to admit that this book may have little interest for a commercial use since financial-market analysts, unlike researches in basic scientific fields, could afford costly commercial products.

This book is about how to understand experimental data, how to reduce complexity of data, derive some meaningful conclusions and, finally, how to present results using Java graphical packages. It concentrates on computational aspects of these topics: as you will see, due to the simplicity of Python, one could catch ideas of many examples of this book just by looking at the code snippets without even explaining them in words. This book is also about how to simulate more or less realistic data samples which can mimic real situations. Such simulated data are used in this book in order to give simple and intuitive examples of data analysis techniques using Java scripting.

In this book I did not go deep inside of particular statistical or physics topics, since the aim was to give concrete numeral receipts and examples using Jython scripting language interfaced with Java numerical packages. My aim was also to give an introduction to many data-analysis subjects with sample code snippets based on Jython and jHepWork Java libraries. In cases when I could not cover the subject in detail, a sufficient number of relevant references was given, so the reader can easily find necessary information for each chapter using external sources.

Thus, this book presents practical approaches for data analysis, focusing on programming techniques. Each chapter describes the conceptual and methodological underpinning for statistical tools and their implementation in Java, covering essentially all aspects of data analysis, from simple multidimensional arrays and histograms to clustering analysis, curve fitting, metadata and neural networks. This book includes a comprehensive coverage of various numerical and graphical packages currently implemented in Java that are part of the jHepWork project.

The book was written by the primary developer of the software, and aimed to present a reliable and complete source of reference which lays the foundation for future data-analysis applications using Java scripting. The book includes more than

200 code snippets which are directly runnable and used to produce all graphical plots given in the text. A detailed description and several real-life data-analysis examples which develop a genuine feeling for data analysis techniques and their programming implementation are given in the last chapter of this book.

Finally, I am almost convinced myself that this book is self-contained and does not depend on knowledge of any computing package, Java, Python or Jython (although knowledge of Python and Java is desirable for professionals).

Chicago-Hamburg-Minsk Sergei V. Chekanov

Acknowledgements

Several acknowledgements are in order. Much of this project grew out of fruitful collaboration with many of my colleagues who devoted themselves to high energy physics.

My scientific career in experimental high-energy physics was most influenced by Prof. Dr. V.I. Kuvshinov and Prof. Dr. E.W. Kittel, who were my supervisors almost fifteen years ago. I've learned experimental computation in the yearly 90x from Dr. L.F. Babichev and Dr. W.J. Metzger. I've learned experimental physics and its computational aspects from Dr. M. Derrick, Prof. Dr. E. Lohrmann, Dr. J. Repond, Dr. R. Yoshida, Dr. S. Magill, Dr. C. Glasman, Prof. Dr. J. Terron, Dr. J. Proudfoot, Dr. A. Vanyashin and many others.

The author is grateful to many authors writing free scientific software for their dedication to science and open-source analysis tools. I would like to thank many of my collegues for checking and debugging the jHepWork package, especially J. Dale, E. May, L. Lee, T. Johnson, P. Di Stefano and many others.

I would like to thank my parents and sister for their support, guidance, and love. Not least, I would like to express my eternal gratitude for my dear wife and children for their love and patience to a husband and father who wrote this book at home and thus was only half (mentally) present after coming from his work. Without their patience and understanding, this book would not have been possible.

Contents

Conventions and Acronyms

In this book, we will use the following typographical convention: A box with a code inside usually means interactive Jython commands typed in the Jython shell. All such commands start with the symbol >>>, which is the usual invitation in Python to type a command. This is shown in the example below:

```
>>> print 'Hello, jHepWork'
```

Working interactively with the Jython prompt has the drawback that it is impossible to save typed commands. In most cases, the code snippets are not so short, although they are still much shorter than in any other programming language. Therefore, it is desirable to save the typed code in a file for further modification and execution. In this case, we use Jython macro files, i.e. we write a code using the jHepWork (or any other) editor, save it in a file with the extension ".py", and run it using the keyboard shortcut `[F8]` or the button "`run`" from the jHepWork tool bar menu. In the book, such code examples are also shown inside the box, but code lines do not start with the Jython invitation symbol >>>. In such situations, the example codes will be shown as:

```
print 'Hello, jHepWork'
```

If a code snippet should be used as a Jython module for inclusion into other programs, then we should write our program inside a file. A Python code always imports an external module using its file name. Since the file names are important, we always indicate which exactly file name should be used on the top of the box with a code. For example, if a program code is considered as a module to be imported by another code example, we will show it as:

Module 'hello.py'

```
print 'Hello, jHepWork'
```

with the box title indicationing the file name. For instance, we call the module above as:

```
>>> import hello
Hello, jHepWork
```

when using the Jython prompt (recall the `>>>` symbol!). The code imports the file `'hello.py'` and executes it, printing the string `'Hello, jHepWork'`. In other cases, one can use any file names for the code snippets.

We use `typewriter` font for Jython and Java classes and methods. For file names and directories, we also use the same font style after adding additional parentheses.

We remind also that the directory name separators are backward slashes for Windows, and slashes for Linux and Mac computers. In this book, we use the latter convention. For example, the directory with examples will be shown as:

```
... /macro/examples/
```

while for Windows computers, the same directory should be shown as:

```
C:...\macro\examples\
```

The dots in this example are used to indicate the jHepWork installation directory.

This is all. We will try to avoid using abbreviations. When we use abbreviations, we will explain their meaning directly in the text.