Analytic Modeling in Python

Why Choose Python for Analytic Modeling
Analytic Modeling in Python

Why Choose Python for Analytic Modeling

by Visual Numerics, a Rogue Wave Software Company

© 2009 by Visual Numerics, Inc. All Rights Reserved
Printed in the United States of America

Publishing History:

February 2009 – Initial publication
August 2009 – Update

Trademark Information

Visual Numerics, IMSL and PV-WAVE are registered trademarks. JMSL TS-WAVE, JWAVE, and PyIMSL are trademarks of Visual Numerics, Inc., in the U.S. and other countries. All other product and company names are trademarks or registered trademarks of their respective owners.

The information contained in this document is subject to change without notice. Visual Numerics, Inc. makes no warranty of any kind with regard to this material, included, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Visual Numerics, Inc, shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.
TABLE OF CONTENTS

Abstract ......................................................................................................................... 4
Prototyping versus Production .................................................................................. 4
Analytic Model Prototyping ..................................................................................... 4
Production Deployment of Analytics ........................................................................ 4
Prototype Modelers and Their Needs ....................................................................... 5
    Why Python? ........................................................................................................... 6
Production Deployment Challenges .......................................................................... 7
The Solution: PylIMSL Studio .................................................................................. 7
Summary .................................................................................................................... 9
Abstract
This paper explores in detail analytical modeling and production deployment of analytical applications and explains how they are fundamentally different in their requirements, goals and tools. The requirements of prototype modeling are mapped to the features of the Python dynamic language.

Bridging the gap between prototype modeling and production development is also discussed and it is shown how PyIMSL™ Studio addresses some of the key issues. PyIMSL Studio is the first and only commercially-available numerical analysis application development environment designed for deploying mathematical and statistical prototype models into production applications.

This paper is intended for those considering Python for prototype modeling who are unfamiliar with Python and how it fits with prototype/production requirements.

Prototyping versus Production
Many organizations developing analytic applications follow a process that includes creating a prototype model before developing the production application. To understand prototype modeling and production deployment better, it helps to understand the goals of each of these activities.

Analytic Model Prototyping
Modeling is an activity in which data and algorithms are explored to achieve some desired numerical results. This can take many forms in different industries but some common goals of this work are:

1. Identify the requirements for production analytics. The investigation often starts with some basic ideas of the available techniques, but the actual requirements for a production deployment are usually not clear until actual data is collected and examined and exploration of analytic techniques completed.
2. Prove that a given analytic approach addresses the identified goals for the project. For example, forecasting sales is not a goal, nor is supply chain optimization. Instead, the goal is to demonstrate forecast results and optimization efficiency that has real measured business value. Similar goals exist in other industries and problem sets.
3. Identify any performance and scalability issues that are important to consider in production deployment.
4. Get management approval to proceed with production deployment – usually through documentation and demonstration of the prototype models.

Production Deployment of Analytics
The goal of production deployment is to move the analytical model to an environment where the analytics are leveraged to achieve usable results. This also takes different forms in different settings, including:
1. Integrate algorithms, statistics, or business logic into an existing application used within a group, a department or company-wide. The integrated application might involve a user interface, possibly web based, and often is designed for use by non-analytic experts to perform repeated tasks or make actionable decisions on new data.

2. Use the code to operate on compute intensive problems which often involve large datasets, or run compute intensive simulations using developed models, possibly in high performance computing environments.

3. Batch processing of data to perform frequent analysis or analysis on many data sets, for example forecasting sales of many products based on a common forecast model or categorization of new data as it becomes available.

4. Integrate analytics into commercial products for resale.

Note that there are important challenges in putting analytic code into production that distinguish production deployment from the activities and concerns in the prototype stage. It is generally risky to simply deploy prototype code directly into a production environment. Some concerns for deployment include:

1. Improving application performance, often by writing compute intensive analytics in lower level languages like C.

2. On demand or scheduled data collection, cleansing and filtering of data which is usually done by hand in prototype. This data collection and processing may be spread out across the different activities in a production application: in a database, during ETL (Extraction Translation and Loading), and in the analytic code itself.

3. Robust error handling and reporting. It is especially important to trap and report analytic anomalies or errors rather than return possibly corrupt results.

4. Testing and quality control of analytic accuracy to insure that the quality of results in production are identical to those seen in prototyping.

Having the right numerical tools to achieve these production goals is important, and parity in the analytic tools used in prototyping and production is an important consideration.

**Prototype Modelers and Their Needs**

*Who does the prototyping?*

Prototype modelers are analysts who explore data and algorithms to achieve desired numerical results. They are often not trained as software developers; instead, they are usually domain experts: statisticians, mathematicians, business intelligence experts, financial quantitative analysts, scientists, or engineers.

*How do they typically go about such investigation?*

They often use off-the-shelf tools designed for flexibility and rapid development. Production deployment concerns are often less important than flexibility to easily manipulate, filter and transform data, apply and customize numerical analysis and create intermediate charts and tables of results. When prototype results are satisfactory the code is often turned over to a
programming staff who must find a way to replicate the numerical methods in a production environment using different tools, because the prototype tools are often ill-suited for the production environment.

**What tools are needed for prototyping?**
- The most important tool is a simple to use language that produces code where algorithm details are easy to read and understand. Lower level languages like C/C++, Java and C# can bury algorithm details in the syntactic baggage, and they lack rich array based processing. Ideally the prototype code closely resembles the algebraic formulas which it implements.
- Analytic tools or libraries needed for the problems under investigation are required. While some components are developed by the modeler, these components are often based on an underlying core analytic or statistical library of functionality.
- Charts and graphs are often used to visually display intermediate and final results as part of algorithm development, even in cases where it is not needed in production deployment.
- Many modelers prefer an interactive command line environment for prototyping. Others prefer a more formal integrated development environment with powerful tools for composing code using syntax highlighting, command completion, refactoring and formatting. Debugging tools to interrogate variables within code is a valuable part of these environments.

**What platforms/environments are used for prototyping?**
- Prototype platforms are usually desktop systems running Microsoft Windows, Linux or MacOS. These are usually different than the production deployment platforms which typically demand a wider range of hardware/OS environments, often on servers rather than desktop systems.

**What format does data typically take in prototyping?**
- Data access for prototyping may involve formal database access but in many cases it involves reduced-sized sample datasets in the form of simple ASCII data (e.g. comma separated data files) or Excel spreadsheets. This often is very different from data access needs in a production application.
- Data filtering and preprocessing in a production application might be performed within database access or by other parts of the production architecture, but for the prototype modeler, there needs to be a wide range of easy-to-use tools to perform data processing in an ad-hoc manner for rapid exploration.

**Why Python?**
Python is a leading open source dynamic language well suited for analytic prototype modeling for a number of reasons:
• It is a well rounded language, which can be used for either procedural or object-oriented development. Other dynamic languages are often more special purpose, with features that address certain kinds of problems but are not balanced for general programming.
• It is an open language and not a proprietary language, allowing for greater sharing of tools and analytic code across a wider audience of users. There are a large number of open-source toolkits for analytical modeling with Python. This is the result of more than a decade of strong adoption and contributions by the scientific community.
• It is a loosely typed language with simple syntax that makes it easy to read and understand.
• The industry-standard NumPy package for Python transforms it into a language for array based operations suitable for efficient storage and manipulation or large multi-dimensional arrays. NumPy includes a simple syntax to index, subset and perform operations on arrays, and is efficient in memory use and performance.
• While there are a number of open source analytic libraries and tools available for Python, the PyIMSL wrappers (part of the PyIMSL™ Studio environment) offer the most comprehensive collection of rich analytic and statistical techniques for both prototype modeling and production deployment, regardless of the final deployed hardware or operating system.

Production Deployment Challenges
Some of the benefits of prototyping in dynamic languages become obstacles for production deployment. Performance can sometimes, but not always, be a barrier for production deployment. For batch processing of large volumes or numbers of datasets, performance can be critical.

The deployment architecture can make integration of a dynamic language engine difficult. Existing applications where analytics are being added may define the language to be used. Web deployment and scalability may require different solutions than a dynamic language can provide.

Python can be used for the production deployment of applications because there are Python components available for creating high quality user interfaces for standalone applications and for web-based application development. However, for many production applications, the best performance and flexibility is achieved by deploying in code written in the C language. Code written in C can be optimized for specific deployment hardware/OS/compiler combinations.

The Solution: PyIMSL Studio
PyIMSL Studio\(^1\) combines the Python language and a selection of robust Python tools with the advanced analytics from the IMSL C Numerical Library, all in an easy to install package and with world-class support. Gaps in data I/O and cleansing are filled with additional functionality from

Visual Numerics. The components in PyIMSL™ Studio provide the functionality needed for prototype modelers as well as analytic functionality in C libraries needed to deploy into production environments.

At the heart of PyIMSL Studio is the IMSL C Library, a comprehensive set of mathematical and statistical algorithms that programmers can embed into their software applications. Within the IMSL Libraries, the actual count of available mathematics and statistical algorithms runs into the thousands, giving developers many options to mix and match algorithms as needed to create unique and competitively engineered analytical applications.

Within PyIMSL Studio, these mathematics and statistical functions are available to Python programmers for quick prototyping and to C developers for production application development. Most important is that it is same underlying algorithms available in each language. Current Visual Numerics customers estimate that by using the same algorithms in prototyping as in production, development time can be reduced significantly through the elimination of extra research, re-writing and testing.

For PyIMSL Studio, Visual Numerics has integrated and packaged a practical and robust set of Python tools to use for analytic modeling. These tools are tested, documented, and supported in a single installable package by Visual Numerics. This toolset includes:

- **Python**
- **NumPy**—A set of modules for powerful and efficient data array manipulation. The de-facto standard for array and matrix algebra in Python.
- **Data I/O and transformation components**—utilities for data filtering and transformation, including:
  - An ASCII data file reading utility available in Python and in production C code.
  - A missing value identification and substitution utility available in Python and in production C code.
  - PyODBC—A Python module for database access on Windows and Linux.
  - xlrd—A Python module for reading data from Microsoft Excel files.
- **matplotlib/pylab**—Python analytical charting components.
- **IPython**—A command line interface for interactive development and exploration in Python.
- **Eclipse/Pydev**—A full featured Integrated Development Environment (IDE) for Python.

**PyIMSL Wrappers**

When doing analytic modeling, the most important component for modelers is to have the necessary mathematical and statistical functionality required for analysis. Within PyIMSL Studio, this functionality is provided by the PyIMSL wrappers, a collection of Python wrappers to the IMSL C Library algorithms.
The PyIMSL wrappers expose all of the functionality of the IMSL C Libraries in a way that is true to the Python language philosophy. Functions requiring arrays can be called with anything that behaves like an array in Python. Error handling uses standard Python exception handling. Using PyIMSL delivers minimalist and readable code. “Pythonic” is the term used to describe this in the Python community. For the modeler, the PyIMSL wrappers provide a way to access the comprehensive, reliable and highly effective IMSL C Library functions without having to do any programming in C.

The PyIMSL wrappers documentation\(^2\) describes how to use all of the mathematics and statistics function in the library and is available for download from the Visual Numerics website.

Because the PyIMSL wrappers deliver a direct interface to each IMSL C Library function, those who will eventually translate the Python prototype code to C for deployment will have no trouble matching the algorithm routines and parameters in C for the production application.

**Summary**

This paper has shown that there are very significant differences in the goals, tools and concerns in prototype modeling and in the deployment of the resulting models into production. Python was shown to be an excellent tool for prototype work. In many cases however the production deployment calls for reimplementaiton of the analytics in another language. PyIMSL™ Studio provides a complete Python prototype environment and the analytics for both the prototype modeling and production deployment.