Data Analysis with Python

Craig Finch

Barcamp Orlando 2008
About me

• An engineer by training—currently a PhD candidate in Modeling and Simulation at UCF
• Started coding in Apple Basic around 1987
• Moved on to Basic and Pascal on Macs
• Then to C on DOS/Windows
• Started using Unix around 1993
• Linux since 1998
Part I: Quick Overview
Programming languages: a family tree

FORTRAN
Basic
Pascal
C

C++

smalltalk
eiffel

shell scripts
Tcl
early perl
early PHP
Matlab™

ruby
Python
perl 5
PHP 5

Java
Programming languages: a family tree

- FORTRAN
- Basic
- Pascal
- C

- C++

- smalltalk
- eiffel

- shell scripts
  - Tcl
  - early perl
  - early PHP
  - Matlab™

- ruby
- Python
- perl 5
- PHP 5

- Java
Quick description

• Object oriented (since Day One)
• Automatic memory management
• Minimal language, large standard library
• Can be used on the command line
• Implementations:
  – CPython (most common interpreter)
  – Stackless Python
  – Jython (Python implementation on JVM)
  – IronPython (Python implementation on .NET)
Types

- strong
- safe
- dynamic

For example:

```python
In [1]: x = 5
In [2]: y = "37"
In [3]: x + y
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```
What it looks like

```python
# Function to extract sample times

def getTime(table):
    time = [x['time'] for x in table.iterrows()
        if x['label'][0] == 's' and x['cumEnzymeLost'] < 50.0]

    # convert to hours for plotting
    for i in range(len(time)):
        time[i] = time[i]/60.0/60.

    # print time
    return time
```

- Minimal amount of symbols
- Indent levels are used to denote code blocks, in place of `{ }` or BEGIN END
Part II: Handling Ridiculous Amounts of Data

I CAN HAS DATA PLZ???
HDF and the PyTables package

- Hierarchical Data Format
- HDF is comprised of
  - A data model
  - A portable file format
  - A software library
- PyTables is a Pythonic interface to HDF
- HDF/PyTables deal with arbitrarily large datasets (ie larger than memory)
NumPy package

• A powerful, n-dimensional array class
• Highly optimized for numerical computing
• Also useful as a multi-dimensional container for generic data
• Includes basic linear algebra functions

• Gives Python capabilities similar to Matlab™

http://numpy.scipy.org/
# Define HDF5 file structure

```python
# root node
file = openFile("AP_PEG.h5", mode = "w", title = "AP on PEG Capillary")

# concentrations: three groups branching from root node
conc90 = file.createGroup("/", 'conc90', '90 ng/ml')
conc300 = file.createGroup("/", 'conc180', '180 ng/ml')
conc300 = file.createGroup("/", 'conc300', '300 ng/ml')

# flow rates: two groups branching from each concentration node
file.createGroup("/conc90", 'flow60', 'Flow Rate 60 microliters/hour')
file.createGroup("/conc90", 'flow100', 'Flow Rate 100 microliters/hour')
file.createGroup("/conc180", 'flow60', 'Flow Rate 60 microliters/hour')
file.createGroup("/conc180", 'flow100', 'Flow Rate 100 microliters/hour')
file.createGroup("/conc300", 'flow60', 'Flow Rate 60 microliters/hour')
file.createGroup("/conc300", 'flow100', 'Flow Rate 100 microliters/hour')
```
HDF data structure

- Root node
  - Concentration 1
  - Concentration 2
  - Concentration 3
  - Flow rate 1
  - Flow rate 2
  - Trial 1
    - Row 1
    - Row 2
    - Row 3
    - ...
    - Row n
  - Trial 2
    - Row 1
    - Row 2
    - Row 3
    - ...
    - Row n
From flatfiles to HDF

```python
## Opening the file
textFile = open( textFileName, 'r' )

# First line--the important data is concentration
words = textFile.readline().split(\'\t\'
concentration = float(words[0])
print concentration

# Next line--headers
words = textFile.readline().split(\'\t\'
print words

# Data lines
isData = True
while isData:
    words = textFile.readline().split(\'\t\'
    if words[0] == '':
        isData = False
    else:
        print words
data = table.row
```
Part III: Data Processing
Accessing data with PyTables

• **List comprehensions**
  – A nice shorthand for creating lists

```python
newList = [ expression for-statement if-statement ]
```

• **Dictionaries**
  – Associative arrays that are indexed by keys rather than integers

```python
delegates = {'clinton':1486 , 'obama':1629}
delegates['obama'] = 1486
```
Accessing data with PyTables

• The code:

```python
# ------------------- Function to extract sample times-----------------
def getTime(table):
    ## Access data in table ##
    time = [x['time'] for x in table.iterrows()]
    if x['label'][0] == 's' and x['cumEnzymeLost'] < 50.0:

    # convert to hours for plotting
    for i in range(len(time)):
        time[i] = time[i]/60./60.

    return time
```
Plotting with matplotlib

• Matplotlib: a package with plotting functions that are highly compatible with Matlab™
• Publication-quality 2D graphics

http://matplotlib.sourceforge.net
Plotting all the data

AP on PEG, 100μl/hr, Syringe Pump

Surface Concentration (ng/ml) vs. Time (hours)
Data reduction

• Combine multiple runs into a single data set: mean and standard deviation

```python
if len(values) > 1:
    stdevSurfConc.append(pylab.std(values))
else:
    stdevSurfConc.append(0.0)
```

• Could have also used exceptions

```python
try:
    stdevSurfConc.append(pylab.std(values))
except DivideByZeroError:
    stdevSurfConc.append(0.0)
```
Plotting the final data

![Graph showing surface concentration over time with experimental data and model prediction.](image-url)
# ----- Overall Mean Plot -----
pylab.figure()
pylab.errorbar(times_all, average_all, fmt='ko', xerr=None, yerr=stdev_all, label='Averaged Experimental Data')
pylab.hold(True)

pylab.plot(modelTimes, optimalPrediction[0], 'k-', label='Model Prediction')

# Scale and decorate
#pylab.title('AP on PEG, 90, 180, and 300ng/ml, 100ul/hr')
pylab.xlabel(r'Time (hours)', size='medium')
pylab.ylabel(r'Surface Concentration $(nmol/m^2)$', size='medium')
pylab.legend(loc="lower right", numpoints=2, pad=0.2, handletextsep = 0.025, prop=fontprop)
pylab.axis([0., 25., 0., 7.])
pylab.savefig('SyringePump_AP_PEG_fitted_100ulhr.eps')

# Show plots and close files
pylab.show()
Conclusion

• What I need in a language:
  – Rapid development
  – Highest possible level of abstraction
  – Clean, readable code for easy maintenance
  – Extensive open-source libraries

• What I don't need:
  – Blinding fast execution
  – Licensing fees and proprietary source

• Python meets my needs—consider whether it will meet yours!
Further Reading

• Why Eric S. Raymond likes Python (over Perl)
  http://www.linuxjournal.com/article/3882

• http://www.prescod.net/python/why.html