Getting Started with Python

Kevin Sheppard
http://www.kevinsheppard.com

Department of Economics
University of Oxford

This version: February 10, 2014

February 10, 2014

UNIVERSITY OF
OXFORD

Today

- Array Slicing
- Slicing
- Array Mathematics and Broadcasting
- Basic use of Logical
- Loops
- Conditional Execution

Activating the Virtual Environment

- Activate
  - Windows: `c:\Anaconda\Scripts\activate.bat econ`
  - *nix: `source ~/anaconda/bin/activate econ`
- One final package to install
  - All: `pip install pylint`
  - pip is the (soon to be) default Python package installer
- Using conda
  - Updating Anaconda
    `conda update conda`
    `conda update anaconda`
  - Updating a virtual environment:
    `conda update -n` *environment packages*
  - Installing additional packages in a virtual environment:
    `conda install -n` *environment packages*

The Python Interpreter

UNIVERSITY OF OXFORD

- From inside the virtual environment, run `python`
- Success will show a banner and >>>
- The default Python interpreter is not suited to interactive work
- `exit()` to quit

IPython

- Interactive Python
  - Run `ipython` from an active environment
- Integrated
  - History
  - Tab completion
  - Help
  - Magic keywords
- Command line switches
  - `pylab`

IPython QtConsole

- A more sophisticated IPython terminal
  - On Windows, clearly better than IPython/cmd
  - On *nix, better but gains are smaller
- Run `ipython qtconsole` from an active environment
- Command line switches
  - `colors=linux`
  - `ConsoleWidget.font_size` (number)
  - `ConsoleWidget.font_family` (Installed font)
  - `pylab`
- Pop-up help

IPython Notebook

UNIVERSITY OF OXFORD

- IPython in a web browser
- Allows mixing code, formatted text and $\LaTeX$ math
- Install MathJax locally
  - Run `ipython qtconsole` and execute

    ```
    from IPython.external.mathjax import install_mathjax
    install_mathjax()
    ```

  - Need to be connected to the internet
- Should have Chrome or Firefox installed
  - Internet Explorer and Safari both have issues
- Run `ipython notebook` to start
- Main cell types
  - Code
  - Markdown
  - Heading

Spyder and PyCharm

- Spyder - Scientific PYthon Development EnviRonment
- More similar to RStudio or MATLAB than other Python IDEs
- Major features:
  - Code editor
  - Object inspector (help)
  - Variable and File explorers
  - Integrated console (Python or IPython)
  - Support for debugging
  - Cell mode
  - Reconfigurable
- PyCharm is a commerical IDE, but has free community edition
  - Steeper learning curve
  - More advanced features:
    Static analysis, Code completion, Code formatting, Block commenting, Git Integration, PEP integration, Spell checking, Quick help, Refactoring and Renaming, more…

Immutable Native Data Types

- Integers
- Floats
- Complex Data
- Long Integers
- Boolean
- Strings
- NoneType
- Tuples
- `xrange`

Mutable Data Types

Tasks 2.11–2.13, 2.18–2.19

- Lists
  - Revisiting strings
- Dictionaries

Note: more native data types available, but less useful to us

Slicing

Tasks 2.14–2.17

- Lists, tuples and strings can all be sliced
  - `[first:last:step]`
  - Lists also support assignment from same-sized variable
- $n-$element array elements are indexed $0, 1, \ldots, n-1$
- Shorthands
  - `[0:n:1]`
    - `[:]`,`[::]`,`[0:]`,`[:n]`,`[0:n]`,`[0::]`,`[:n]`,`[:n:]`
  - `[s:n:1]`, `[s:]`,`[s::]`, `[s:n]`,`[s:n]`
  - `[0:e:1]`, `[:e]`,`[:e:]`
  - `[0:n:st]`, `[::st]`
- Negative step counts down
  - `[::-1]` is reverse, and is similar to `[n-1::-1]` (but not `[n:0:-1]`)

NumPy Data Types

Tasks 3.1 – 3.6

- Array
- Matrix

UNIVERSITY OF OXFORD

- Similar to list and string slicing, except for nested lists
- Array slicing supports explicit multi-dimensional slices
  - 2-d Lists: `things[3][3]`
  - Arrays: `x[3,3]`, `x[:3,:3]`, `x[2::2,1::3]`
  - Higher-dimensional arrays use high-order slicing
- Arrays can be indexed using two other methods
  - Numeric indexing
  - Logical indexing

`flat` Slicing

- `flat` can be use to index the “flattened” version of an array
- `A=array([[0,1],[2,3]])`
- `A.flat[:]` is then `array([0, 1, 2, 3])`
- By default, NumPy arrays are stored in *row major* format
  - First across rows, then down columns
  - This is why 0 was next to 1 and not 2
  - Consider `A.T` and `A.T.flat[:]`

Array Mathematics

Tasks 4.3–4.5

- Array mathematics operates element by element
  - Addition: `+`
  - Subtraction: `-`
  - Multiplication: `*`
    - Linear algebra definition: `dot()`
  - Division: `/`
  - Exponentiation: `**`
    - Linear algebra definition: `matrix_power()`
    - Must be square, $X^2 = XX$

Matrix Mathematics Differences

- Matrices follow the rule of linear algebra
  - Matrix multiplication: `*`
    - Element-by-element: `mulitply()`
  - Matrix exponentiation (square): `**`
    - Element-by-element: `power()`

Broadcasting

Tasks 4.6–4.8

- NumPy does not respect the laws of matrix addition, subtraction and Hadamard multiplication and division
- Under some circumstance arrays can be *promoted* to be as-if bigger than they are
- Let $s_1 = (s_{11}, s_{12}, \ldots, s_{1k})$ be the shape of array 1, and $s_2 = (s_{21}, s_{22}, \ldots, s_{2m})$ be the shape of array 2
  - Assume WLOG $m \geq k$, let $n = m - k$
- Two arrays are broadcastable if:
  - Let $\tilde{s}_1 = (1, 1, \ldots, 1, s_{11}, s_{12}, \ldots, s_{1k})$, then for all j $\tilde{s}_{1j} = s_{2j} \cup s_{1j} = 1 \cup s_{2j} = 1$
    - Alternative $\max\left(\tilde{s}_{1j}, s_{2j}\right) / \min\left(\tilde{s}_{1j}, s_{2j}\right) \in \left\{1, \max\left(\tilde{s}_{1j}, s_{2j}\right)\right\}$
  - Either same size *or* at least one is unity
  - Common broadcast size is $b_j = \max\left(\tilde{s}_{1j}, s_{2j}\right)$

Broadcasting Examples

Tasks 4.6–4.8

- Note: You can always choose to avoid broadcasting using functions like `tile` to explicitly replicate arrays
  - Still exposed to accidental or unintended broadcasting
- Which are broadcastable?

x	y	$x+y$?
`(1,2)`	`()`	
`(5,5)`	`(5,)`	
`(5,5)`	`(1,5)`	
`(5,5)`	`(5,1)`	
`(1,2,3)`	`(4,1,3)`	

Array Assignment

Task 4.9

- Array slices can be used for assignment
  - Simple if dimension of target same as slice
  - Also can use broadcasting
    - Scalar: `A[:]=1`
    - Note that 1 is 0-dimensional, so always broadcastable

Array Manipulation

Tasks 4.11–4.12

- Basic Information
  - `shape`, `ndim`, `size`
- Reshaping an array
  - `shape`, `reshape`, or `[:,None]`(technically a slice)
  - `squeeze`
  - `transpose`, `.transpose()`, `.T`
- Copying an array
  - `copy`
  - `+0.0`
  - Call `array` or `matrix`
- Building
  - `tile`
  - `concatenate`
    - `hstack` or `vstack`

Array Memory Management

- Slices are views into arrays
- Contain same elements
- Good from performance point of view
  - Use `.flags` to determine if original
- Slices *do not copy*
- Use `copy()` to copy data
  - Or `+0.0`
    - Math ops produce copies
  - Or `array`/`matrix`

Operator Precedence

Tasks 4.12

Operator	Name	Rank
`( ), [ ] , ( ,)`	Parentheses, Lists, Tuples	1
`**`	Exponentiation	2
`+,-`	Unary Plus, Unary Minus	3
`*, /, //, %`	Multiply, Divide, Modulo	4
`+,-`	Addition and Subtraction	5
`<, <=, >, >=`	Comparison operators	9
`==, !=`	Equality operators	9
`=,+=,-=,/=,*=,**=`	Assignment Operators	13

Scalar Logical Operations

- NumPy arrays can be used in logical operations
  - Discussion later
- Scalar logical operations evaluate to `True` or `False`
  - ==, >, <, >=, <=, `!=`
    - == also works with strings or lists
  - Combine using `and`, `or`, `not`
  - If testing for None, use `is` as in `x is None`
- Empty things are generally `False`
  - None, `[]`, `(,)`, ”, “ ”
  - Test using `not`
- Care is needed when testing equality with floating point numbers
  - `allclose` from NumPy may be better

Loops

Task 5.1

- `for` and `while` loops
- Generic structure of for loop:

```
for i in iterator:
    # Do something with i
```

- Note: **whitespace** matters!
  - Tabs or spaces are whitespace
  - Use only spaces – 4 per level of indention
- iterator is anything that supports iteration:
  - `array` and `matrix`
  - `range`, `arange` and `xrange`
  - `list` and `tuple`
- `enumerate` can be useful when using arrays or lists

Nested For Loops

Task 5.2

- `for` loops can be nested

```
for i in iterator1:
    for j in iterator2:
    # Do something with i and j
```

- `while` loops are similar, but end when a condition is met
- Generically, they are given by

```
while some_condition:
    # Do something
    # Update condition
```

- It is important that the condition is updated inside the `while` loop

Conditional Flow Control

6.1 – 6.7

- if statements implement conditional flow control
- Always use *scalar* logical values
- Generic structure

```
if condition:
    # Code to run if condition true
elif other_cond:
    # Code to run if other_cond and not condition
else:
    # Code to run if not (condition and other_cond)
```

- elif and else are optional
- Whitespace delimited
- Python has a ternary operator

```
x = a if condition else b
```

List Comprehensions

- List comprehensions are syntactically dense method to build lists
- Basic

```
x = [item for item in iterable]
```

- Can be combined with logicals

```
x = [item for item in iterable if item>0]
```

  - Only items that satisfy the logical condition will be added
- Can use nested loops
- Mostly syntactic sugar, but also have additional optimizations over using `for` and `list.append`

Calling Functions

Tasks 5.1 – 5.2

- Function calls are simple: `function()`
- Functions can return multiple outputs
  - Take them as a tuple: `out = function()`
  - Unpack them `a,b,c = function()`
    - Tuple can be unpacked later: `a,b,c = out`
    - Also similar to multiple assignment `a,b,c = x,y,z`
- Two input methods
  - In order (positional): `function(x,y)`
  - Keyword: `function(file='input.csv', skiprows=10)`
- Two special constructs
  - `*args`: Additional positional arguments
  - `**kwargs`: Additional keyword arguments
- Mandatory vs. optional arguments

```
array(object, dtype=None, copy=True, order=None,
        subok=False, ndmin=0)
```

Array generating functions

Task 5.3

- `arange`
- `linspace`, `logspace`
- `zeros`, `ones`, `empty`
  - See also `zeros_like`, `ones_like`, `empty_like`
- `eye`
- `r_` and `c_`
  - These are special purpose, are not normal functions
  - Use slice-like inputs

Core Array Functions

Tasks 6.1 – 6.7

- `sum`, `prod`, `cumsum`, `cumprod`
  - Importance of `axis`
- `exp`, `log`, `log10`
- `sqrt`, `square`
- `abs`, `sign`
- `diff`
- `around`, `floor` and `ceil`
- Set functions: `unique`, `in1d`, `intersect1d`, `union1d` and `setdiff1d`

Sorting and Extremes

Task 6.8

- `sort` vs `.sort`
- `amax` and `amin` or `.max` and `.min`

Functions vs. Methods & Properties

- Many operations can be used as a function or method
  - `x.dot(x.T)`, `dot(x,x.T)`
  - `x.sum(axis = 0)`, `sum(x, axis=0)`
- Generally a personal choice, some uses may be easier to read than others

Linear Algebra Functions

- Large library of useful linear algebra routines
  - `inv`, inverse
  - `det` and `trace`, determinant and trace
  - `eig` and `svd`, eigenvalues/eigenvectors and singular values
  - `slogdet`, the signed log determinant
    - More accurate than `log(det(x))`
  - `pinv` (Moore-Penrose) Pseudo Inverse $\left(X'X\right)^{-1}X'$
    - Note that `dot(pinv(x),y)` is more accurate than `dot(inv(dot(x.T,x)),dot(x.T,y))`
  - `diag`, `tril`, `triu`, diagonal and triangular matrices

`inf` and `nan`

Tasks 7.1 and 7.4

- `inf` represents infinity
  - `exp(800)`
- `nan` is not a number
  - `exp(800)/exp(1000)`
  - `nan` corrupts other functions, such as `mean`
  - Can use nan-functions to compute values using only non-nan values
    - `nansum`
    - `nanmin` and `nanmax`
    - `isnan`, a logical function
    - NumPy 1.8.0 adds `nanmean`, `nanstd`, `nanvar`

Numeric Limits

Tasks 7.2–7.4

- `inf` represents infinity: `exp(800)`
- `nan` is not a number: `exp(800)/exp(1000)`
- Computer math has limits for floating point data
- Computer numbers are of the form $a \times 10^b$ where a has about 15 digits stored and b is within ± 308.
  - Absolute magnitudes
    - `finfo(np.float64).max`, `finfo(np.float64).tiny`
  - Relative Magnitudes
    - `finfo(np.float64).eps`
- Examples
  - `1.0 == 1.0 + (finfo(np.float64).eps)/2`
  - `1.0 == 1.0 + (finfo(np.float64).eps)`
  - `10.0**30 + 10.0**12 == 10.0**30`
- Best Practices: Scale large/small data

Importing Data using pandas

- pandas is the main data-management package in the Scientific Python stack
- pandas provides a `DataFrame` based on NumPy `arrays`, but with additional features
  - Meaningful row and column indices
  - Support for merge/join operations
  - Lots of other helpful features
- pandas includes a number of importers from common formats
  - csv, Excel, formatted text, STATA, tables on the web
- Also includes exporters to most common formats

Examples of Importing Data

- Federal Reserve Economic Database
  - Real GDP
  - `http://research.stlouisfed.org/fred2/data/GDPC1.csv`
- Ken French Data
  - `http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/F-F_Research_Data_Factors.zip`
- WorldBank Data: Central Government Debt, % of GDP
  - `http://api.worldbank.org/v2/en/indicator/gc.dod.totl.gd.zs?downloadformat=excel`
- Files at `http://kevinsheppard.com/wiki/Python_Course`

Examples of Saving Data

Tasks 8.2 – 8.4

- pandas DataFrames can be directly exported to a range of formats
- When working with data, better to use binary formats since there is never a loss of precision
- Preferred format is h5, which is provided by PyTables
- `rgdp.to_hdf(`*filename*`,`*key*`, complevel=6, complib='zlib')`
- Saving multiple: `HDFStore`
  - Works like a dictionary
  - `store = HDFStore(`*filename*`, `*mode*`, complevel=6,complib='zlib')`
  - `store['`*var_name*`']=`*variable*
  - `store.close()`
- More on pandas next term

Logical Operator

Tasks 10.1, 10.7

- Logical operators are similar to mathematical operations
  - x *L* y where L is one of >, >=, <, <=, ==, !=
  - Scalars or broadcastable arrays
- Logical operators can be joined using `logical_and`, `logical_or`, `logical_not`, `logical_xor`
- `all` and `any` can be used to test multiple operations
- Scalars support short-circuit operators: `and`, `or` and `not`
- Long list of logical information functions: `is*`
  - `isinf`, `isnan`, `isfinite`, `iscomplex`, `isreal`, `isposinf`, `isneginf`, `isscalar`

Numeric Indexing

Tasks 10.3–10.5

- Numeric indexing returns arrays with the same shape as the arrays used as indices
  - Arrays used as indexes ***must*** be broadcastable
- Simply solution is to use `ix_` if interested in selecting rows and columns
- Scalar selection is different than numeric indexing
  - `x[0]` vs `x[[0]]` or `x[array([0])]`
  - Also `x[0]` vs `x[[:1]]`

Logical Indexing

Task 10.2, 10.6

- Logical statements can be used to select elements of an array
- One dimensional selection is simple
  - `x = arange(5.0)`
  - `y = x<3`
  - `x[y]`
- Higher dimensional selection vectors ***must*** be broadcastable
  - Usually use `ix_`
  - See `nonzero` which is used to turn logical indexes into numeric indices

Writing functions

Tasks 11.1–11.2

- Functions start with `def`
- Return single value or tuple using `return`
- Whitespace used to denote function boundaries
  - Can use `pass` to explicitly end function
- Functions are available in the same file after they are declared
- Order of inputs determines positions
- Name determines keyword use
- Default values given using form `input = default`
  - Do not use mutable values as defaults, use None and test

UNIVERSITY OF OXFORD

- Docstrings are the embedded help in IPython
- Standard NumPy-encouraged format for numerical code

```
def l2_norm(x,y)
"""
Computes the L2-normed distance between two vectors

Parameters
----------
x : array-like, 1-dimensional
y : array-like, 1-dimensional

Returns
-------
L2 distance between functions

Notes
-----
The L2 distance is the squareroot of average the squared deviation
of the elements of the vectors
"""
return np.mean((x-y)**2.0)
```

Allowing arbitrary args and kwargs

Tasks 11.3–11.4

- Arbitrary arguments can be passed in using `*args`
  - Appear as tuple in function
- Arbitrary keyword arguments using `**kwargs`
  - Appear as dictionary in function
  - Only keywords not in function appear
  - See `.keys()`

Calling functions in other files

Task 11.6

- Use `from` *file* `import` *function*
  - Also `from` *file* `import *`
  - Also `import` *file* and then *file.function*

Variable scope

- Variables declared in a function are local to that function
  - These do not overwrite variables with the same name
- Variables declared before the function are available *read only*
  - Unless used with `global`
  - Rarely needed
- Mutable variables passed to functions can be changed, so care is needed
  - Copy if required
  - Important to not use inputs for temporary values
- Immutable cannot be changed in a function

Random Number Generation

- Two methods to generate random numbers
  - `numpy.random`
  - `scipy.stats`
    - `scipy.stats.DIST`
- Important to be able to re-produce random sequences
  - `numpy.random.RandomState()`
  - `get_state/set_state`

Working with Common Distributions

- SciPy contains a large statistics library
  - `scipy.stats`
- Faster to use frozen RV objects for repeated calls
  - Especially true if calls are simple

Common Statistics Functions

- Normal
  - norm
- χ^2
  - chi2
- f
  - f
- Γ
  - gamma
- Log-normal
  - lognormal

Non-linear Optimization

- Non-linear optimization is provided by `scipy.optimize`
- Usually imported using `import scipy.optimize as opt`
- Main routines
  - `fmin_bfgs` - Gradient descent (Broyden−Fletcher−Goldfarb−Shanno)
  - `fmin` - Derivative free (Simplex)
  - `fmin_slsqp` - Constrained (Sequential LS Quadratic Programming)
- All optimizers *minimize*
  - Not a problem since can multiply by -1

Nonlinear optimization

- Optimizers require an objective function
  - Function should take values as first input, usually a 1d numpy array
- Other values (e.g. data) can be passed as additional arguments
  - Usually passed through a tuple
  - Positional arguments, so order matters
- Some optimizers take additional arguments like gradient
  - Improves performance and precision, but not required

pandas

UNIVERSITY OF OXFORD

- Canonical import name: pd
- Provides three key structures that provide meaning to NumPy arrays
  - `Series` - a 1-dimensional array with a name and index
  - `DataFrame` - a collection of `Series`, 2-dimensional
  - `Panel` - a collection of `DataFrames`, 3-dimensional
- pandas data structures are both array-like and dict-like
  - `np.log(df)`: array-like
  - `df['series']`: dict-like
- Other features
  - Reading and writing data
  - Merging/joining multiple datasets
  - Quick access to common plots and statistics
  - Access to underlying NumPy arrays using `values`

pandas: Reading Data

- `read_csv`
  - Important keyword arguments: `skiprows`, `index_col`, `parse_dates`, `na_values`
- `read_excel`
  - Requires filename and sheet name, same keyword args
- `read_table`: Read text files, such as tab delimited
  - Important keyword arguments: `sep`
- `read_stata`: Read Stata .dta files
- `read_hdf`: Read data from HDF files (h5), which provide compression
  - Important keyword arguments: `complevel`, `complib`
- `read_pickle`: Read the native Python pickle format
- Other: json, sql, clipboard, html

pandas: Exporting Data

- `df.to_csv`: Export to csv
  - Keyword argument sep allows tab delimited
- `df.to_excel`: Export to excel (97 (xls) or 2007+ (xlsx))
  - Advanced usage allows multiple sheets in a single file
- `df.to_stata`: Write Stata .dta files
- `df.to_hdf`: Writes files to HDF files (h5)
  - Important keyword arguments: `complevel`, `complib`
- `df.to_pickle`: Writes the native Python pickle format
- `df.to_string`: Output tabular data
- `df.to_latex`: Writes to a latex table
- Others: json, sql, dict

Series

- Series is building block of DataFrame
- `.head()`, `.tail()` (or `.tail(n)`), `.info()`, `.dtype()`
- `.name` to assign a name, or create using the keyword argument `name`
- `.index` to assign an index, or create using the keyword argument `index`
  - `.index.name` to assign name to an index
- `Series` is array-like, and work with NumPy functions (e.g. `np.log`)
  - Math on multiple series works aligning indices!
- `dropna()` to remove NaNs.
- `Series` can be created from arrays (or other lists) or dictionaries.
- Slicing a `Series` can be done
  - Numerically, scalar or slice
  - Using index labels, scalar or slice of these as well

DataFrames

- Main data structure in pandas
- `.head()`, `.tail()` (or `.tail(n)`), `.info()`, `.dtypes()`
- `.describe()` to get a simple summary
- Columns have names, and can be changed or reordered
- Series can be accessed using
  - Dictionary style syntax – `df['series_name']`
  - Attribute syntax `df.series_name`
    - Only Dictionary if name has spaces (avoid this)
- Series can be added to DataFrames
  - Dictionary style adding a series - left join
  - `pd.concat` performs an outer join using tuple of series input
- Multiple series extracted with dictionary-syntax and list of column names
  - Also how columns are re-ordered

More DataFrame

- Indices can be subseted or supersetted using `reindex`
- Removing rows or columns with NaNs: `dropna`
- Rows extracted using
  - `.ix[slice]`
- Extracting rows and columns
  - `ix[slice,columns]`
  - Pure numeric `ix[:2,:2]`or mixed labels
- Selection comes with same caveats with scalar selection vs slice selection (or arrays)
- Extracting rows: `xs` which is a function (`xs()`)
- Deleting columns or rows
  - `del`
  - `pop`
  - `drop(rows, axis=1)`

More DataFrames

- Underlying NumPy with `.values`
- Get or set index using `.index`
- Construction from arrays or lists of lists
  - Keyword arguments: `index`, `columns`
  - Can also set later (correct number of elements)
- Also can construct from dict of series
- `.copy()` to copy
- Logical indexing works on rows
- `.drop` to remove rows
  - Can also slice rows to keep
- Sorting
  - `sort_index`
  - `sort()`
    - Keyword arguments: `inplace`
- `pivot` transforms flat data to be converted to more meaningful array

Multiple Indices

- Can construct indices composed of multiple items
  - Example country and year
- Use ix[outerIndex] or xs(outerIndex)
- xs(innerIndex, level=1) to access inner index
- Direct access to specific elements with ix[outer, inner]
- `swaplevel` to alter order for easier access
- `sortlevel` to sort a specific level
- Can fill or drop missing values
  - `.fillna`, `.ffill`, `.bfill`
  - `.interpolate`
  - `.dropna`

Newton-Cotes Quadrature

- See Judd 1998, Ch. 6–7
- Integrals are the limits of sums, so sums can be used to approximate integrals
- Newton-Cotes

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

- Midpoint Rule
  - Simple Version

$$\int_a^b f(x)\,dx \approx (b-a)\, f\left(\frac{b-a}{2}\right)$$

  - Composite Version

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{n} \sum_{j=1}^{n} f(x_j) \text{ where } x_j = a + (j - 1/2)\,\frac{(b-a)}{n}$$

Better Approximations

- Trapezoid Rule
  - Simple Version

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$$

  - Composite Version

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2n}\left(\sum_{j=1}^{n} f(x_j) + \sum_{k=2}^{n-1} f(x_k)\right) \text{ where } x_i = a + i\frac{(b-a)}{n}$$

- Simpson's Rule, based on piecewise quadratic
  - Simple Version

$$\int_a^b f(x)\,dx \approx \left(\frac{b-a}{6}\right)\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

  - Composite Version

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n-1}\left(\frac{x_{i+1}-x_i}{6}\right)\left[f(x_i)+4f\left(\frac{x_{i+1}+x_i}{2}\right)+f(x_{i+1})\right] \text{ where } x_i = a+i\frac{(b-a)}{n}$$

Problems

Compute values for

1. $f(x) = x^3$, where x is $[0, 1]$, with $n = 100$.
2. $f(x) = 1/x$, where x is $[1, 100]$ with $n = 1000$
3. $f(x) = x$ where x is $[0, 5000]$ with $n = 5,000,000$.

Change of Variables

- Integrals can be modified to change the range of integration

$$\int_a^b g(y)\,dy = \int_{\phi^{-1}(a)}^{\phi^{-1}(b)} g(\phi(x))\,\phi'(x)\,dx$$

- Where
    - $y = \phi(x)$

Orthogonal Polynomials

- Orthogonal polynomial are useful for approximating non-polynomial functions
- Polynomials are orthogonal if

$$\langle f, g\rangle = \int_a^b f(x) g(x) w(x)\, dx = 0$$

  with respect to a weighting function $w(x)$
- A family $\{\phi_n(x)\}$ is mutually orthogonal if

$$\left\langle \phi_i, \phi_j \right\rangle = 0 \ \forall i \neq j$$

- Can also be orthonormal if

$$\langle \phi_i, \phi_i\rangle = 1$$

Families

Family	$w(x)$	$[a,b]$	Definition
Legendre	1	$[-1,1]$	$P_n(x) = \frac{(-1)^n}{2^n n!}\frac{\partial^n}{\partial x^n}\left[\left(1-x^2\right)^n\right]$
Chebyshev	$(1-x)^{-1/2}$	$[-1,1]$	$T_n(x) = \cos\left(n\cos^{-1}x\right)$
Gen. Chebyshev	$\left(1-\left(\frac{2x-a-b}{b-a}\right)^2\right)^{-1/2}$	$[a,b]$	$T_n\left(\frac{2x-a-b}{b-a}\right)$
Laguerre	$exp\,(-x)$	$[0,\infty]$	$L_n(x) = \frac{e^x}{n!}\frac{\partial^n}{\partial x^n}\left(x^n\exp(-x)\right)$
Hermite	$exp\left(-x^2\right)$	$[-\infty,\infty]$	$H_n(x) = (-1)^n\exp\left(x^2\right)\frac{\partial^n}{\partial x^n}\left(\exp\left(-x^2\right)\right)$

Recusions

- Legendre, $P_0(x) = 1,\ P_1(x) = x$

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x)$$

- Chebyshev , $T_0(x) = 1,\ T_1(x) = x$

$$T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x)$$

- Laguerre, $L_0(x) = 1,\ L_1(x) = 1 - x$

$$L_{n+1}(x) = \frac{1}{n+1}(2n + 1 - x) L_n(x) - \frac{n}{n+1} L_{n-1}(x)$$

- Hermite, $H_0(x) = 1,\ H_1(x) = 2x$

$$H_{n+1}(x) = 2x H_n(x) - 2n H_{n-1}(x)$$

Gaussian Quadrature

UNIVERSITY OF OXFORD

- Gaussian quadrature is similar but solves a related problem

$$\int_a^b f(x)\, w(x)\, dx \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

- The weighting function defines different classes
- Have the property that if n nodes are used, and if f is a polynomial in x of degree $2n-1$, then the integral is exact
  - Newton-Cotes are generally not exact in most non-trivial cases

Guass-Chebyshev Quadrature

$$\int_{-1}^{1} f(x)\left(1-x^2\right)^{-1/2} dx \approx \frac{\pi}{n}\sum_{i=1}^{n} f(x_i)$$

where $x_i = \cos\left(\frac{2i-1}{2n}\pi\right), i = 1, \ldots, n$ are nodes

- Change of variables to get

$$\int_a^b f(y)\,dy = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(x+1)\left(b-a\right)}{2}+a\right)\frac{\left(1-x^2\right)^{1/2}}{\left(1-x^2\right)^{1/2}}dx$$

  - Linear: $x = -1 + 2\,(y-a)\,/\,\left(b-a\right)$

Guass-Chebyshev Quadrature

- So that

$$\int_a^b f(y)\,dy \approx \frac{\pi\left(b-a\right)}{2n}\sum_{i=1}^{n} f\left(\frac{(x_i+1)\left(b-a\right)}{2}+a\right)\left(1-x_i^2\right)^{1/2}$$

  where x_i are the Gauss-Chebyshev nodes nodes on $[-1, 1]$.

- Compute the integral of x^α for $\alpha \in \{0.5, 1\}$, $a \in \{0.2, 1\}$ and $b \in \{2, 5\}$ for $n \in \{3, 6, 15\}$

Guass-Legendre Quadrature

- Similar to Gauss-Chebyshev, only using Gauss-Legedre nodes and weights
  - Legendre orthogonal polynomials
- After change of variables

$$\int_a^b f(y)\,dy \approx \frac{b-a}{2}\sum_{i=1}^{n}\omega_i f\left(\frac{(x_i+1)\left(b-a\right)}{2}+a\right)$$

  - ω_i and x_i are the G-L weights and nodes on $[-1, 1]$.
  - Note: G-C uses $\omega_i = (1 - x_i)^{-1/2}$
- Guass-Legendge has a smaller error than Gauss-Chebyshev in most cases

Guass-Legendre Nodes and Weights

- Interested in n nodes, which are the roots of the nth order Legendre polynomial
- Polynomials can be constructed using $P_0(x) = 1,\ P_1(x) = x$

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x)$$

- Derivatives are also recursive

$$P'_{n+1}(x) = \frac{n+1}{x^2-1}\left(x P_n(x) - P_{n-1}(x)\right)$$

Guass-Legendre Nodes and Weights

- Roots are not analytical, but can use

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

to iterate starting from

$$x_{0,j} = \cos\left(\pi \frac{j - \frac{1}{4}}{n + \frac{1}{2}}\right)$$

for the jth root

- Weights can be computed from root using

$$w_i = \frac{2}{\left(1 - x_i^2\right)\left[P_n'(x_i)\right]^2}$$

Topics Next Term

- Random Number Generation
- Statistical Distributions and Related Quantities
- Plotting
- Non-linear Optimization
- Working with Dates and Times
- Using pandas to Manage Data
- Performance Considerations
- Using Classes to manage Complex Code