Python in Neurocomputation - A Brief Introduction

Mike Hull

December 8, 2010

Motivation

- What?
- Why
- How?

A Brief Tour

- Basic Data Types
- Basic Data Types
- Lists & Dictionaries

Example - Manipulating an SWC File

- Background
- Algorithms
- Implementation

Example Code

- Libraries
- Examples

Conclusion

What is Python?

- High-level, general purpose programming language

What is Python?

- High-level, general purpose programming language
- In development since 1989. Currently Version 2.6

What is Python?

- High-level, general purpose programming language
- In development since 1989. Currently Version 2.6
- Opensource - Strong User Community

What is Python?

- High-level, general purpose programming language
- In development since 1989. Currently Version 2.6
- Opensource - Strong User Community
- Used by ...
  - Google - google-mail, google-groups & google-maps
  - Yahoo! - discussion boards

What is Python?

- High-level, general purpose programming language
- In development since 1989. Currently Version 2.6
- Opensource - Strong User Community
- Used by ...
  - Google - google-mail, google-groups & google-maps
  - Yahoo! - discussion boards
- Popular as an interface to software: GIMP, Blender, Maya, SPSS, neuron

Why is Python Popular?

- Design Philosophy

Why is Python Popular?

- Design Philosophy
  - 'remarkable power with very clear syntax'

Why is Python Popular?

- Design Philosophy
  - ‘remarkable power with very clear syntax’
  - ‘to describe something as clever is *NOT* considered a compliment in the Python culture’

Why is Python Popular?

- Design Philosophy
  - ‘remarkable power with very clear syntax’
  - ‘to describe something as clever is *NOT* considered a compliment in the Python culture’
- Learning Curve

Why is Python Popular?

- Design Philosophy
  - 'remarkable power with very clear syntax'
  - 'to describe something as clever is *NOT* considered a compliment in the Python culture'
- Learning Curve
- Scalability

Why is Python Popular?

- Design Philosophy
  - 'remarkable power with very clear syntax'
  - 'to describe something as clever is *NOT* considered a compliment in the Python culture'
- Learning Curve
- Scalability
- Cross Platform

Why is Python Popular?

- Design Philosophy
  - ‘remarkable power with very clear syntax’
  - ‘to describe something as clever is *NOT* considered a compliment in the Python culture’
- Learning Curve
- Scalability
- Cross Platform
- Standard Library - databasing, cryptography, networking, graphics, . . .

Why is Python Popular?

- Design Philosophy
  - ‘remarkable power with very clear syntax’
  - ‘to describe something as clever is *NOT* considered a compliment in the Python culture’
- Learning Curve
- Scalability
- Cross Platform
- Standard Library - databasing, cryptography, networking, graphics, . . .
- Extensibility

Why is Python Popular?

- Design Philosophy
  - 'remarkable power with very clear syntax'
  - 'to describe something as clever is *NOT* considered a compliment in the Python culture'
- Learning Curve
- Scalability
- Cross Platform
- Standard Library - databasing, cryptography, networking, graphics, ...
- Extensibility
- Rapid Development

Why is Python Popular?

- Design Philosophy
  - 'remarkable power with very clear syntax'
  - 'to describe something as clever is *NOT* considered a compliment in the Python culture'
- Learning Curve
- Scalability
- Cross Platform
- Standard Library - databasing, cryptography, networking, graphics, ...
- Extensibility
- Rapid Development
- ...Programming Python is Fun!!

How does Python look?

```
#Example 1
height = 12.5
weight = 23.8
area = height * weight
print area
>> 297.5

#Example 2
a = 'Xenopus'
b = 'laevis'
print 'Lets study ' + a + ' ' + b + '.'
>> Lets study Xenopus laevis.
```

Basic Built-in Data Types

- Integer

Basic Built-in Data Types

- Integer
- Float

Basic Built-in Data Types

- Integer
- Float
- String

Basic Built-in Data Types

- Integer
- Float
- String

- Lists

Basic Built-in Data Types

- Integer
- Float
- String

- Lists
- Dictionary

Basic Built-in Data Types

- Integer
- Float
- String

- Lists
- Dictionary

- Tuples

Basic Built-in Data Types

- Integer
- Float
- String

- Lists
- Dictionary

- Tuples
- `None True False`

Basic Data Types

- Integer, Float

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
'B'
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
'B'
>>> a[1]
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
'B'
>>> a[1]
'r'
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
'B'
>>> a[1]
'r'
>>> len(a)
```

Basic Data Types

- Integer, Float

```
>>> (50-5*6)/4
5
>>> 3 * 3.75 / 1.5
7.5
```

- String

```
>>> 'Hello, ' + 'plymouth'
'Hello, plymouth'
>>> a = 'Bristol has hills'
>>> a[0]
'B'
>>> a[1]
'r'
>>> len(a)
18
```

Lists

```
>>> l = [2,3,5,7,11,13]
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
>>> len(l)
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
>>> len(l)
6
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
>>> len(l)
6
>>> for x in l:
>>>     print 'Item: ', x, ', Doubled: ', x*2,
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
>>> len(l)
6
>>> for x in l:
>>>     print 'Item: ', x, ', Doubled: ', x*2,
Item: 2, Doubled: 4
Item: 3, Doubled: 6
Item: 5, Doubled: 10
Item: 7, Doubled: 14
Item: 11, Doubled: 22
Item: 13, Doubled: 26
```

Lists

```
>>> l = [2,3,5,7,11,13]
>>> l[0]
2
>>> len(l)
6
>>> for x in l:
>>>     print 'Item: ', x, ', Doubled: ', x*2,
Item: 2, Doubled: 4
Item: 3, Doubled: 6
Item: 5, Doubled: 10
Item: 7, Doubled: 14
Item: 11, Doubled: 22
Item: 13, Doubled: 26
>>> l = ['neuron', 'axon', 'dendrite', 4.6, 16]
```

Dictionaries

- Dictionaries map a *key* to a *value*

Dictionaries

- Dictionaries map a *key* to a *value*

```
>>> channel_density = {
                   'soma':  1.0,
                   'dendrite': 0.8,
                   'axon':0.7,
                   'axonhillock':  1.7,
                    }
```

Dictionaries

- Dictionaries map a *key* to a *value*

```
>>> channel_density = {
                  'soma':  1.0,
                  'dendrite': 0.8,
                  'axon':0.7,
                  'axonhillock':  1.7,
                   }


>>> channel_density['axon']
```

Dictionaries

- Dictionaries map a *key* to a *value*

```
>>> channel_density = {
                  'soma':  1.0,
                  'dendrite': 0.8,
                  'axon':0.7,
                  'axonhillock':  1.7,
                   }


>>> channel_density['axon']
0.7
```

Dictionaries

- Dictionaries map a *key* to a *value*

```
>>> channel_density = {
                  'soma':  1.0,
                  'dendrite': 0.8,
                  'axon':0.7,
                  'axonhillock':  1.7,
                   }


>>> channel_density['axon']
0.7

>>> channel_density['axonhillock']
```

Dictionaries

- Dictionaries map a *key* to a *value*

```
>>> channel_density = {
                    'soma':  1.0,
                    'dendrite': 0.8,
                    'axon':0.7,
                    'axonhillock':  1.7,
                     }


>>> channel_density['axon']
0.7

>>> channel_density['axonhillock']
1.7
```

Dictionary Iteration

Dictionary Iteration

```
>>> regions = channel_density.keys()
```

Dictionary Iteration

```
>>> regions = channel_density.keys()
>>> regions
```

Dictionary Iteration

```
>>> regions = channel_density.keys()
>>> regions
>>> ['axon','dendrite','soma','axonhillock']
```

Dictionary Iteration

```
>>> regions = channel_density.keys()
>>> regions
>>> ['axon','dendrite','soma','axonhillock']
>>> for location in residence.keys():
        print location + ' has channel density ',  channel_density[location]
```

Dictionary Iteration

```
>>> regions = channel_density.keys()
>>> regions
>>> ['axon','dendrite','soma','axonhillock']
>>> for location in residence.keys():
        print location + ' has channel density ',  channel_density[location]

axon has channel density 1.0
dendrite has channel density 0.8
soma has channel density 1.0
axonhillock has a channel density 1.7
```

Example 1

- Load SWC File and calculate the surface area of the neuron

Example 1

- Load SWC File and calculate the surface area of the neuron
- SWC File Format

  ID TYPE XPOS, YPOS, ZPOS, RAD, PARENT_ID

Example 1

- Load SWC File and calculate the surface area of the neuron
- SWC File Format

  ID TYPE XPOS, YPOS, ZPOS, RAD, PARENT_ID

  ```
  54 2 0.00 -3.00 -3.00 0.15 -1
  55 51 0.50 -5.00 -4.00 0.50 54
  56 51 0.00 -7.00 -7.00 0.15 55
  ```

Example 1

- Load SWC File and calculate the surface area of the neuron
- SWC File Format

  ID TYPE XPOS, YPOS, ZPOS, RAD, PARENT_ID

  ```
  54 2 0.00 -3.00 -3.00 0.15 -1
  55 51 0.50 -5.00 -4.00 0.50 54
  56 51 0.00 -7.00 -7.00 0.15 55
  ```

- Surface Area of cylinder: $A = (R + r) \times \pi \times length$

Human Algorithm

- For each line in the file (without a parent_id -1):

Human Algorithm

- For each line in the file (without a parent_id -1):
  - Find the position and radius of the parent.

Human Algorithm

- For each line in the file (without a parent_id -1):
  - Find the position and radius of the parent.
  - Find the length of the segment by subtracting this lines position for the parents (via Pythagorus)

Human Algorithm

- For each line in the file (without a parent_id -1):
  - Find the position and radius of the parent.
  - Find the length of the segment by subtracting this lines position for the parents (via Pythagorus)
  - Calculate the surface area of this segment using length and radii

Human Algorithm

- For each line in the file (without a parent_id -1):
  - Find the position and radius of the parent.
  - Find the length of the segment by subtracting this lines position for the parents (via Pythagorus)
  - Calculate the surface area of this segment using length and radii
- Add up the surface area of all lines.

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:
  $id \mapsto [x, y, z, r, parent\_id]$

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$

  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$

  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$

- Iterate over every id in the dictionary (without a parent_id of -1):

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:
  $id \mapsto [x, y, z, r, parent\_id]$
  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$
  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$
  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$
- Iterate over every id in the dictionary (without a parent_id of -1):

  Lookup the data($[x, y, z, r, parent\_id]$) for that id in the dictionary

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:
  $id \mapsto [x, y, z, r, parent\_id]$
  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$
  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$
  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$
- Iterate over every id in the dictionary (without a parent_id of -1):

  Lookup the data($[x, y, z, r, parent\_id]$) for that id in the dictionary
  Lookup the parent_data using parent_id (again using the dictionary)

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$

  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$

- Iterate over every id in the dictionary (without a parent_id of -1):

  Lookup the data($[x, y, z, r, parent\_id]$) for that id in the dictionary

  Lookup the parent_data using parent_id (again using the dictionary)

  Calculate the length of the segment

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:
  $id \mapsto [x, y, z, r, parent\_id]$
  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$
  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$
  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$
- Iterate over every id in the dictionary (without a parent_id of -1):

  Lookup the data($[x, y, z, r, parent\_id]$) for that id in the dictionary
  Lookup the parent_data using parent_id (again using the dictionary)
  Calculate the length of the segment
  Calculate the surface of the segment

Python Algorithm

- Read all the lines in the file & create a dictionary with the data,

  Map id's to a list containing the data on the line:

  $id \mapsto [x, y, z, r, parent\_id]$

  $54 \mapsto [0.00, -3.00, -3.00, 0.15, -1]$

  $55 \mapsto [0.00, -5.00, -4.00, 0.50, 54]$

  $56 \mapsto [0.00, -7.00, -7.00, 0.15, 55]$

- Iterate over every id in the dictionary (without a parent_id of -1):

  Lookup the data($[x, y, z, r, parent\_id]$) for that id in the dictionary

  Lookup the parent_data using parent_id (again using the dictionary)

  Calculate the length of the segment

  Calculate the surface of the segment

- Add up the surface area of all id's.

Reading the File I

Program

```
swcFile = open( "myNeuron.swc")
for line in swcFile.readlines():
    print line,
```

Output

```
'54 2 0.00 -3.00 -3.00 0.15 -1'
'55 51 0.50 -5.00 -4.00 0.50 54'
'56 51 0.00 -7.00 -7.00 0.15 55'
...
'77 2 8.00 -8.00 -27.00 0.15 76'
```

Reading the File II

Program

```
swcFile = open( "myNeuron.swc")
for line in swcFile.readlines():
    lineSplit = line.split()
    print lineSplit
```

Output

```
['54', '2', '0.00', '-3.00', '-3.00', '0.15', '-1']
['55', '51', '0.50', '-5.00', '-4.00', '0.50', '54']
['56', '51', '0.00', '-7.00', '-7.00', '0.15', '55']
...
['77', '2', '8.00', '-8.00', '-27.00', '0.15', '76']
```

Extracting the data

Program

```
swcFile = open( "myNeuron.swc")
for line in swcFile.readlines():
    lineSplit = line.split()

    id = int( lineSplit[0] )
    x = float( lineSplit[2] )
    y = float( lineSplit[3] )
    z = float( lineSplit[4] )
    r = float( lineSplit[5] )
    parent_id = int( lineSplit[6] )

    print 'ID: %d (%f,%f,%f) R:%f Parent:%d' % (id,x,y,z,r,parent_id)
```

Output

```
ID: 54 (0.00,-3.00,-3.00) R:0.15 Parent:-1]
ID: 55,(0.50,-5.00,-4.00) R:0.50 Parent:54]
ID: 56,(0.00,-7.00,-7.00) R:0.15 Parent:55]
...
ID: 77,(8.00,-8.00,-27.00) R:0.15 Parent:76]
```

Building the Dictionary

Program

```
idDict = {}
swcFile = open( "myNeuron.swc")
for line in swcFile.readlines():
    lineSplit = line.split()

    id = int( lineSplit[0] )
    x = float( lineSplit[2] )
    y = float( lineSplit[3] )
    z = float( lineSplit[4] )
    r = float( lineSplit[5] )
    parent_id = int( lineSplit[6] )

    idDict[id] = [x,y,z,r,parent_id]

print idDict
```

Output

```
{ 54 :[0.00,-3.00,-3.00,0.15,-1], 55:[0.50,-5.00,-4.00,0.50,54],
56:[0.00,-7.00,-7.00,0.15,55], ... 77:[8.00,-8.00,-27.00,0.15,76] }
```

Calculating the Surface Areas

```
from math import pi,sqrt
def calcLength(x1,y1,z1, x2,y2,z2):
    x = x1-x2
    y = y1-y2
    z = z1-z2
    return sqrt( x*x + y*y + z*z )

total_area = 0
for id, swcData in idDict.iteritems():
    #Shorthand for:x = swcData[0], y = swcData[1] ...
    x,y,z,rad,parent_id = swcData

    if parent_id == -1:
        print "ID: %d - Root node - nothing to do" %id
    else:
        parent_swc_data = idDictionary[ parent_id ]
        p_x,p_y,p_z,p_rad,p_parent_id = parent_swc_data

        l = calcLength(x,y,z, p_x,p_y,p_z)
        sa = (rad + p_rad) * pi * l
        print "ID: %d - Length %f, SA: %f" % (id,l,sa)
        total_area = total_area + sa
print 'Total Area:', total_area
```

```
ID: 55 - Root node - nothing to do"
ID: 56 - Length 3.64, SA: 7.43
ID: 57 - Length 1.50, SA: 3.06
...
ID: 77 - Length 2.24, SA: 2.11
Total Area: 98.78
```

Condensing

Simplfied Program

```
from math import pi,sqrt
from numpy import array
from scipy.linalg import norm

swcFile = open("myNeuron.swc")
for line in swcFile.readlines():
    lineSplit = line.split()

    id = int( lineSplit[0] )
    xyz = array( [float(lineSplit[2]), float(lineSplit[3]), float(lineSplit[4]) ] )
    r = float( lineSplit[5] )
    parent_id = int( lineSplit[6] )
    idDict[id] = [xyz,r,parent_id]

segmentAreas = {}
for id, (xyz,rad,parent_id) in idDict.iteritems():
    if parent_id == -1: continue

    p_xyz,p_rad,p_parent_id = idDictionary[ parent_id ]
    segmentAreas[id] = (rad + p_rad) * pi * norm(p_xyz-xyz)

print 'Total Area:', sum( segmentAreas.values() )
```

Notes

- Total: 17 Lines of code!

numpy, scipy & matplotlib

- numpy contains basic array and matrix manipulation functions

numpy, scipy & matplotlib

- numpy contains basic array and matrix manipulation functions
- scipy contains higher-level toolboxes
  - signal-processing, statistics,
  - optimisation, fft, image-processing

numpy, scipy & matplotlib

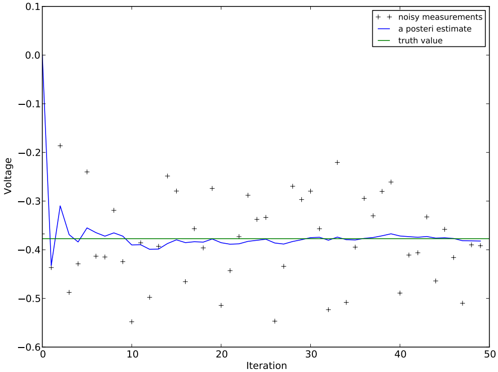
- numpy contains basic array and matrix manipulation functions
- scipy contains higher-level toolboxes
  - signal-processing, statistics,
  - optimisation, fft, image-processing
- matplotlib/pylab provide plotting functionality
  - Emulated matlab interfaces
  - High-quality output
  - Most imaginable plots - Polar, 3D, Map-Overlays, Shapes, Densities, [see gallery]
  - Flexible, Extensible architecture

Kalman Filter Example - Calculation

```
import numpy, pylab

# intial parameters
n_iter = 50
sz = (n_iter,) # size of array
x = -0.37727 # truth value (typo in example at top of p. 13 calls this z)
z = numpy.random.normal(x,0.1,size=sz) # observations (normal about x, s=0.1)
Q = 1e-5 # process variance

# allocate space for arrays
xhat=numpy.zeros(sz)      # a posteri estimate of x
P=numpy.zeros(sz)         # a posteri error estimate
xhatminus=numpy.zeros(sz) # a priori estimate of x
Pminus=numpy.zeros(sz)    # a priori error estimate
K=numpy.zeros(sz)         # gain or blending factor
R = 0.1**2 # estimate of measurement variance, change to see effect

# intial guesses
xhat[0] = 0.0
P[0] = 1.0

for k in range(1,n_iter):
    # time update
    xhatminus[k] = xhat[k-1]
    Pminus[k] = P[k-1]+Q

    # measurement update
    K[k] = Pminus[k]/( Pminus[k]+R )
    xhat[k] = xhatminus[k]+K[k]*(z[k]-xhatminus[k])
    P[k] = (1-K[k])*Pminus[k]
```

Kalman Filter Example - Plotting

```
from pylab import *
figure()
plot(z,'k+',label='noisy measurements')
plot(xhat,'b-',label='a posteri estimate')
axhline(x,color='g',label='truth value')
legend()
xlabel('Iteration')
ylabel('Voltage')
```



Conclusions

Any Questions. . .