01_Introducing_Python

September 26, 2014

1 Introducing Python

2 Python in HPC Tutorial

2.1 Supercomputing 2014

Presenter: Andy R. Terrel
Contributors: - Andy R. Terrel - Aron Ahmadia - Matthew G. Knepley

2.2 About This Tutorial

2.2.1 PyHPC Tutorial on GitHub

These presentation materials are part of a continuously updated tutorial on Python for High Performance Computing. Future versions of this presentation will be found at:
https://github.com/pyhpc/pyhpc-tutorial
Please set all permanent bookmarks to this URL.

2.2.2

Checkout from git

```
git clone https://github.com/pyhpc/pyhpc-tutorial.git
git checkout sc2014
```

2.2.3 Viewing the read-only version of this presentation on nbviewer:

- http://nbviewer.ipython.org/urls/raw.github.com/pyhpc/pyhpc-tutorial/master/notebooks/01_Introducing_Python.ipynb

---

2.3 Interacting with the Tutorial Slides

This tutorial is an interactive worksheet designed to encourage you to try out the lessons during the demonstration. If you are looking at the PDF version of these slides, we encourage you to download the updated version (see previous slide) and try the interactive version.

To run the interactive version of this notebook, you will need a Python 2.7 environment including:

- IPython version >= 13.0

- numpy version >= 1.6
- scipy >= 0.10
- matplotlib >= 1.0.0

Move to the directory containing the tarball and execute:

```
$ ipython notebook --pylab=inline
```

If you are installing the packages yourself, Continuum Analytics provides both free community as well as professional versions of the Anaconda installer, which provides all the packages you will need for this portion of the tutorial. The installer is available from the Anaconda download page at Continuum Analytics.

You are also welcome to use Enthought's Python Distribution (free for Academic users), available from the EPD download page at Enthought.

Unfortunately, you will need to upgrade your EPD's IPython to 0.13 if you go this route, and this may be non-trivial, please see the Enthought discussion thread here.

2.4 Introduction to Python (This Notebook)

2.4.1 Objectives

1. You will understand how scripting languages fit into the toolbox of a computational scientist.
2. You will see why Python is a powerful choice
3. You will get a taste of Python for actual scientific computing

The first part of this introduction is adapted from *Python Scripting for Computational Science* by Hans Petter Langtangen, Simula and also includes motivating material from Nathan Collier, KAUST.

2.5 Scripting vs Traditional programming

In traditional programming, large applications were typically written at a low level. Scripting, by contrast, is programming at a very high level with expressive, dynamic languages.

> Traditional programming: Fortran, C, C++, C#, Java
>
> Scripting: Python, Perl, Ruby, MATLAB

In the past, a major gain for scripting languages was that they could automate many tasks that otherwise would be performed by hand.

As domain-specific scripting languages such as MATLAB and Python evolved, scientists realized that there were also great benefits in programming at a high level, allowing them to naturally express their computations in fewer lines of code, resulting in greater productivity and less defects.

Computational scientists are now demonstrating that high level languages can also be more performant than their low-level counterparts, and Python is at the forefront of this.

2.6 Python Scripting - Has this ever happened to you?

2.6.1 Scenario 1

You are working on data for a presentation your advisor is giving at a conference. At the last minute, you realize that there is a major bug in your code and you need to regenerate all the images and graphs that you have given him. You spend 30 minutes regenerating the data and 6 hours regenerating the graphs because you had them in Excel and had done them by hand.

2.6.2 Scenario 2

You are working on your thesis and as you near the finish you review some graphs you generated months earlier and you aren't sure if they are now completely up to date. You spend half a day locating the old code that you wrote on your second laptop and hours more again creating and polishing the charts.

2.7 Buyer Beware

Learning to automate many of these common tasks can greatly increase your productivity (and make your research reproducible). However, beware that there is no end to the number of different ways to do essentially the same thing.

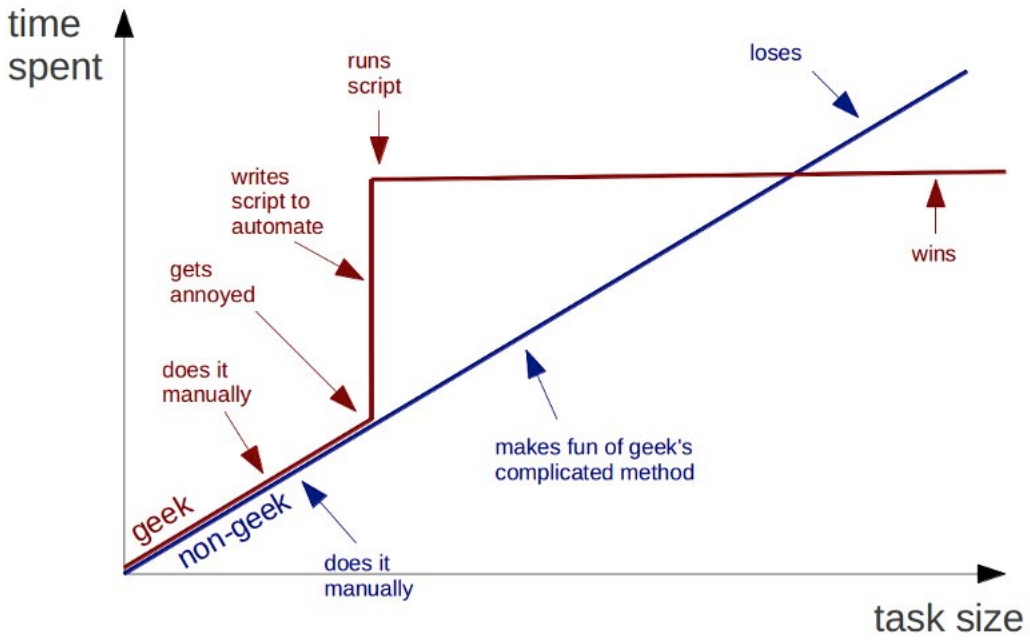


Figure 1: Recall we want to script to SAVE time

Image credit Bruno Oliveira

2.8 Why Python is useful

There are perhaps many reasons, here we list a few:

- like other scripting languages, has an easier interface to learn
- allow you to build your own work environment
- scientific computing is more than crunching numbers
- easier creation of GUIs, demos, and fancy HTML notebooks
- create modern interfaces to old codes
- allow you to explore, develop and test interactively
- cleaner, shorter, easy to read, write, and maintain code

2.9 Is Python > MATLAB?

2.9.1 Python == MATLAB

- Simple and easy to use syntax (dynamic typing, array and linear algebra language)
- Easy creation of interactive interfaces and GUIs
- Merge simulation and visualization
- Commercially supported (Continuum Analytics, Enthought)
- Vibrant, enthusiastic academic communities for getting support and learning more
- JIT support
- Amazon EC2 support

2.9.2 MATLAB > Python

- Several MATLAB toolboxes offer capabilities unavailable to the Python community
- Superior documentation
- Commercially robust and reliable scientific computing tools

2.9.3 Python > MATLAB

- Python is free
- Object-oriented programming is more convenient in Python
- Extensive tools within and beyond scientific computing
  - IPython notebook relies on a Python webserver as well as extensive asynchronous messaging support through PyZMQ
- Python runs on all modern Cray and IBM supercomputers (and anywhere C does)

---

3 Some Sample Applications

3.1 Teaching the finite element method

3.1.1 Stiffness matrix computation

3.2 Monitor program progress

3.2.1 Python parses a datafile and makes plots of current progress

---

3.3 Problem prototypes

3.3.1 Nonlinear, time dependent problem

3.4 Problem prototypes

3.4.1 Method for fitting surface to data

```
# compute K and F
K = np.zeros((n,n))
F = np.zeros(n)

for a in range(n): # loop over basis a
    F[a] += N(a)*dx*force(x)
    for b in range(n): # loop over basis b
        for e in range(n):
            K[a,b] += dN(a)*dN(b)*dx

# Neumann condition
F[0]=h

# Dirichlet condition
b=n
for a in range(n):
    for e in range(n):
        F[a] -= dN(a)*dN(b)*dx * g

# Solve system
d = np.linalg.solve(K,F)
```

Figure 2: Stiffness matrix

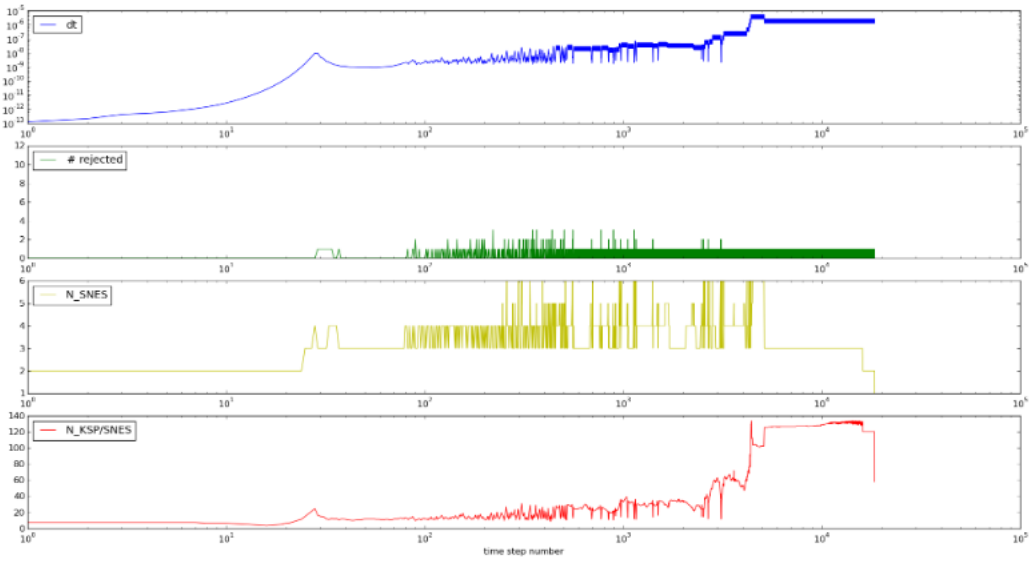


Figure 3: Log

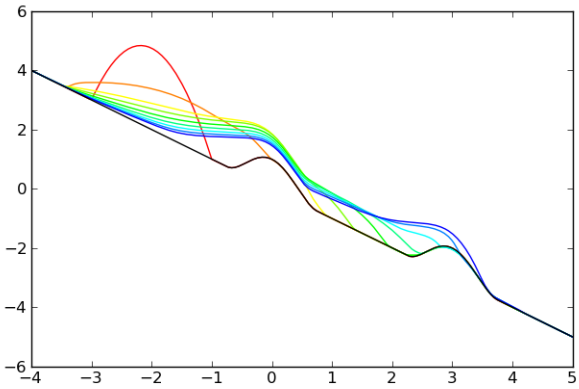


Figure 4: NonLinear problem

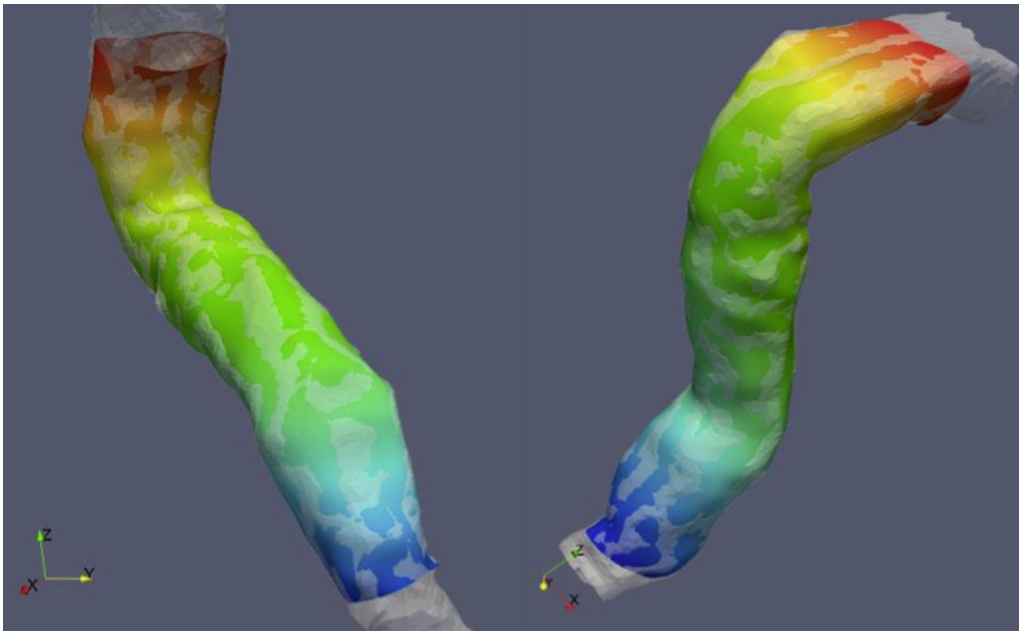


Figure 5: Prototypes

3.5 Structural program

3.5.1 Python manages floors and columns

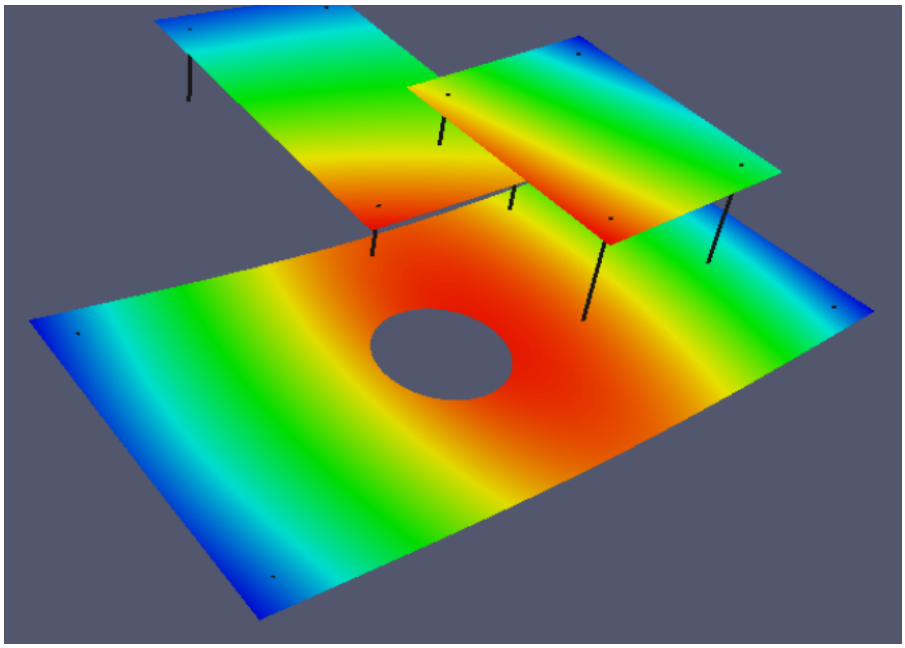


Figure 6: Structural program

3.6 Auto-generation of results

3.6.1 Python runs C code, post-processes the results, and generates a LaTeX table

---

4 A Tour of Python for Simple Scripting

4.1 Script 1

```
In [2]: #!/usr/bin/env python
        import math
        r = math.pi / 2.0
        s = math.sin(r)
        print "Hello world, sin(%f)=%f" % (r,s)

Hello world, sin(1.570796)=1.000000
```

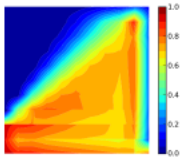
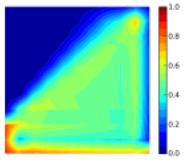
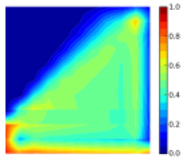
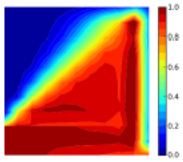
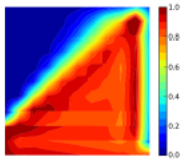
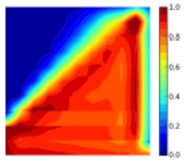
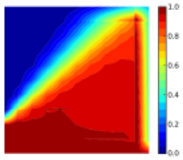
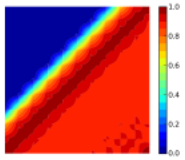
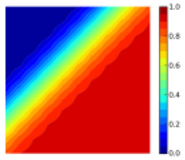
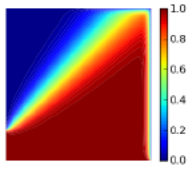
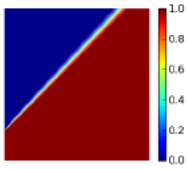
	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
SN			
	min/max = -0.13/0.96	min/max = -0.14/0.91	min/max = -0.14/0.91
WN			
	min/max = -0.09/1.14	min/max = -0.10/1.26	min/max = -0.10/1.25
QON			
	min/max = -0.12/1.15	min/max = -0.05/1.07	min/max = -0.04/0.96
Reference			N/A

Figure 7: Result Generation

4.2 Script 2

```
In [5]: import math
        infile = "files/examples/data/numbers"
        outfile = "files/examples/data/f_numbers"

        f = open(infile,'r')
        g = open(outfile,'w')

        def func(y):
            if y >= 0.0:
                return y**5.0*math.exp(-y)
            else:
                return 0.0

        for line in f:
            line = line.split()
            x = float(line[0])
            y = float(line[1])
            fy = func(y)
            g.write("%g %12.5e\n" % (x,fy))

        f.close()
        g.close()
```

4.3 Script 3

```
In [9]: import sys,os
        cmd = 'date'
        output = os.popen(cmd)
        lines = output.readlines()
        fail = output.close()
        if fail: print 'You do not have the date command'; sys.exit()
        for line in lines:
            line = line.split()
            print "The current time is %s on %s %s, %s" % (line[3],line[2],line[1],line[-1])
```

```
The current time is 10:14:20 on 4 Sep, 2014
```

4.4 Script 4

4.5 A bib-file (see examples/data/python.bib)

```
@Book{Langtangen2011,
  author =    {Hans Petter Langtangen},
  title =     {A Primer on Scientific Programming with Python},
  publisher = {Springer},
  year =      {2011}
}
@Book{Langtangen2010,
```

```
  author =    {Hans Petter Langtangen},
  title =     {Python Scripting for Computational Science},
  publisher = {Springer},
  year =      {2010}
}
```

```
In [8]: import re
        pattern1 = "@Book{(.*),"
        pattern2 = "\s+title\s+=\s+{(.*)},"
        for line in file('files/examples/data/python.bib'):
            match = re.search(pattern1,line)
            if match:
                print "Found a book with the tag '%s'" % match.group(1)
            match = re.search(pattern2,line)
            if match:
                print "The title is '%s'" % match.group(1)
```

```
Found a book with the tag 'Langtangen2011'
The title is 'A Primer on Scientific Programming with Python'
Found a book with the tag 'Langtangen2010'
The title is 'Python Scripting for Computational Science'
```

---

5 A Tour of Python for Scientific Computing

5.1 Arrays

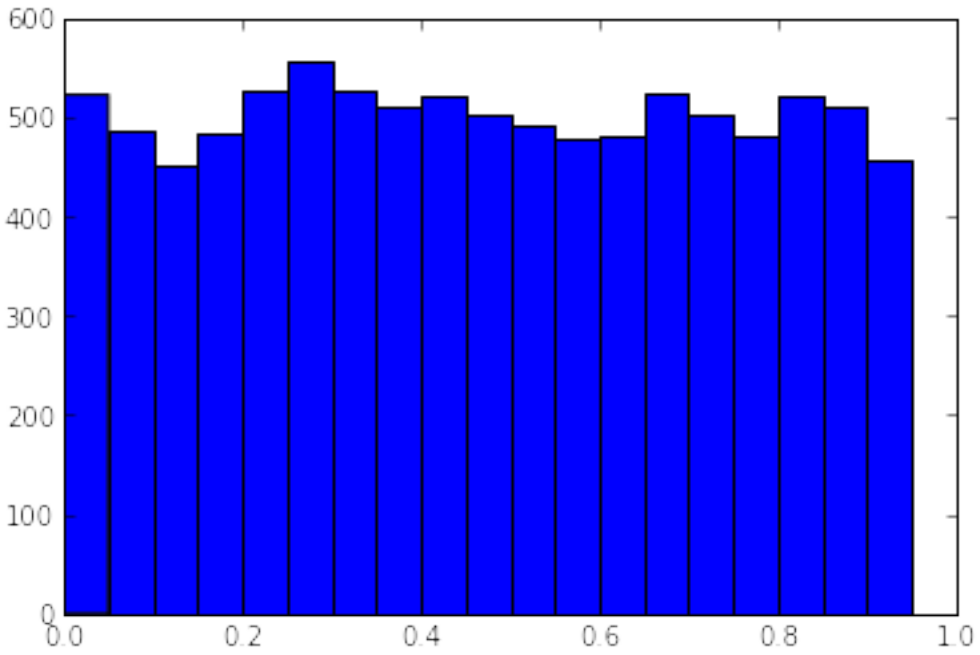
Python has built-in:
containers: lists (costless insertion and append), dictionaries (fast lookup)
high-level number objects: integers, floating point
Numpy is:
extension package to Python for multi-dimensional arrays
closer to hardware (efficiency)
designed for scientific computation (convenience)

```
In [11]: a = np.array([0,1,2,3,4])
         a

Out[11]: array([0, 1, 2, 3, 4])

In [12]: print a.ndim # Dimensionality
         print a.shape # Shape

1
(5,)

In [13]: b = np.array([[0, 1, 2], [3, 4, 5]])
         print b
         print b.ndim
         print b.shape
```

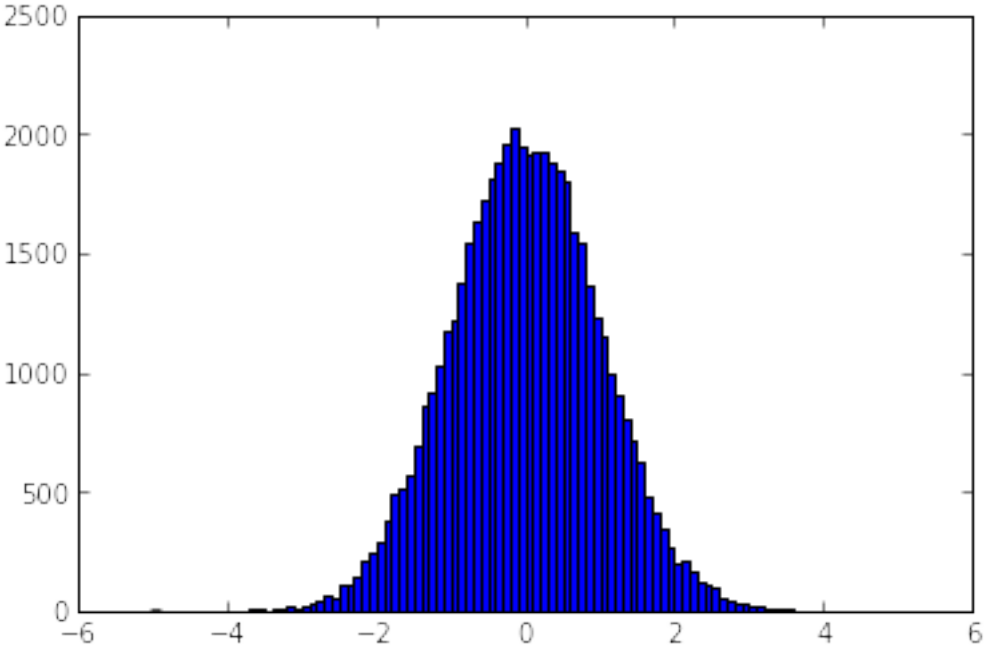
```
[[0 1 2]
 [3 4 5]]
2
(2, 3)
```

5.2 Common Arrays

```
In [14]: print np.arange(10) # Like range [0, 1, ..., 9]
         print np.arange(1,9, 2) # [1, 3, 5, 7]

[0 1 2 3 4 5 6 7 8 9]
[1 3 5 7]

In [15]: print np.linspace(0, 1, 6) # A linear space of [0,1] with 6 pts
         print np.linspace(0, 1, 6, endpoint=False) # [0,1[

[ 0.   0.2  0.4  0.6  0.8  1. ]
[ 0.          0.16666667  0.33333333  0.5         0.66666667  0.83333333]

In [16]: print np.ones((3,3)) # 3 X 3 2D array of 1's

[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]

In [17]: print np.eye(3)

[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]

In [18]: print np.diag(arange(4))

[[0 0 0 0]
 [0 1 0 0]
 [0 0 2 0]
 [0 0 0 3]]

In [21]: print np.random.rand(4) # Uniform distribution
         print np.random.randn(4) # Gaussian distribution

[ 0.9134486   0.7943014   0.40743464  0.8400922 ]
[-0.05662066 -0.99363921  0.35042966 -0.29096857]

In [27]: hist(np.random.rand(10000), bins=np.linspace(0, 1, 20, endpoint=False));
```




```
In [35]: hist(np.random.randn(50000), bins=np.linspace(-5, 5, 100, endpoint=False));
```



5.3 Other Numpy Features to be aware of

- Reshaping

```
In [36]: arr = np.arange(1000000)
         arr2 = arr.reshape((10,100000))
         print(arr2)
```

```
[[     0      1      2 ...,  99997  99998  99999]
 [100000 100001 100002 ..., 199997 199998 199999]
 [200000 200001 200002 ..., 299997 299998 299999]
 ...,
 [700000 700001 700002 ..., 799997 799998 799999]
 [800000 800001 800002 ..., 899997 899998 899999]
 [900000 900001 900002 ..., 999997 999998 999999]]
```

- Memory Views

```
In [37]: arr2.view?
```

- Index Slicing

```
In [38]: x = np.arange(0, 20, 2); y = x**2
         ((y[1:] - y[:-1]) / (x[1:] - x[:-1])) # dy/dx
```

```
Out[38]: array([ 2,  6, 10, 14, 18, 22, 26, 30, 34])
```

- Fancy Indexing

```
In [39]: evens = arr[arr%2 == 0]
         print(evens)
```

```
[     0      2      4 ..., 999994 999996 999998]
```

5.4 LinAlg, FFTs, and Random numbers

```
In [40]: np.dot(arange(10), arange(10))
```

```
Out[40]: 285
```

```
In [41]: np.dot(arange(9).reshape(3,3),
             arange(9).reshape(3,3))
```

```
Out[41]: array([[ 15,  18,  21],
                [ 42,  54,  66],
                [ 69,  90, 111]])
```

```
In [42]: np.fft?
```

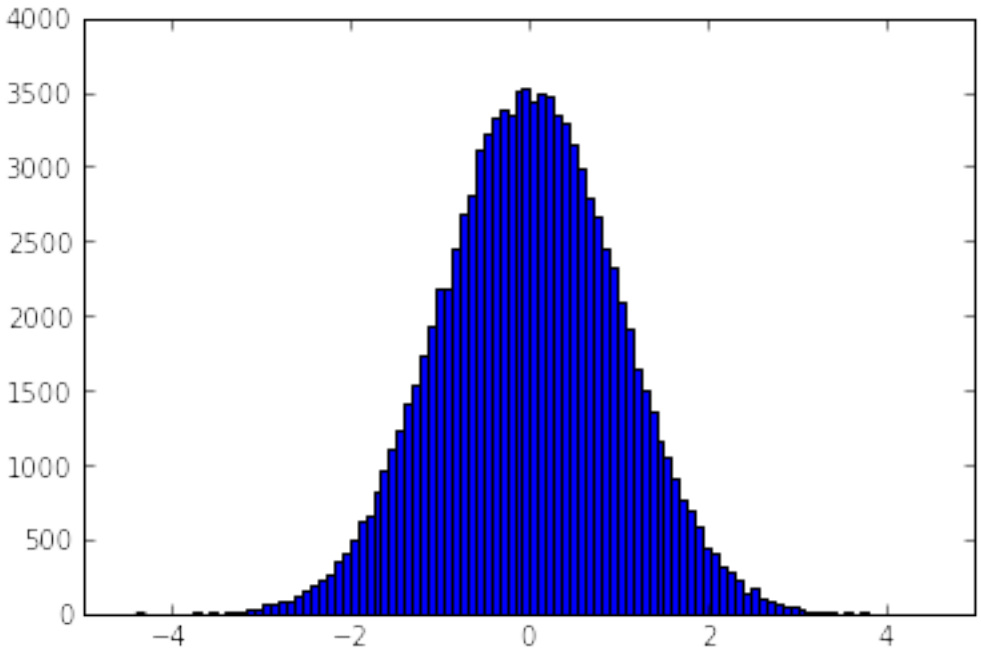
```
In [43]: np.linalg?
```

```
In [23]: np.random?
```

5.5 Quick Visualization / MatPlotLib

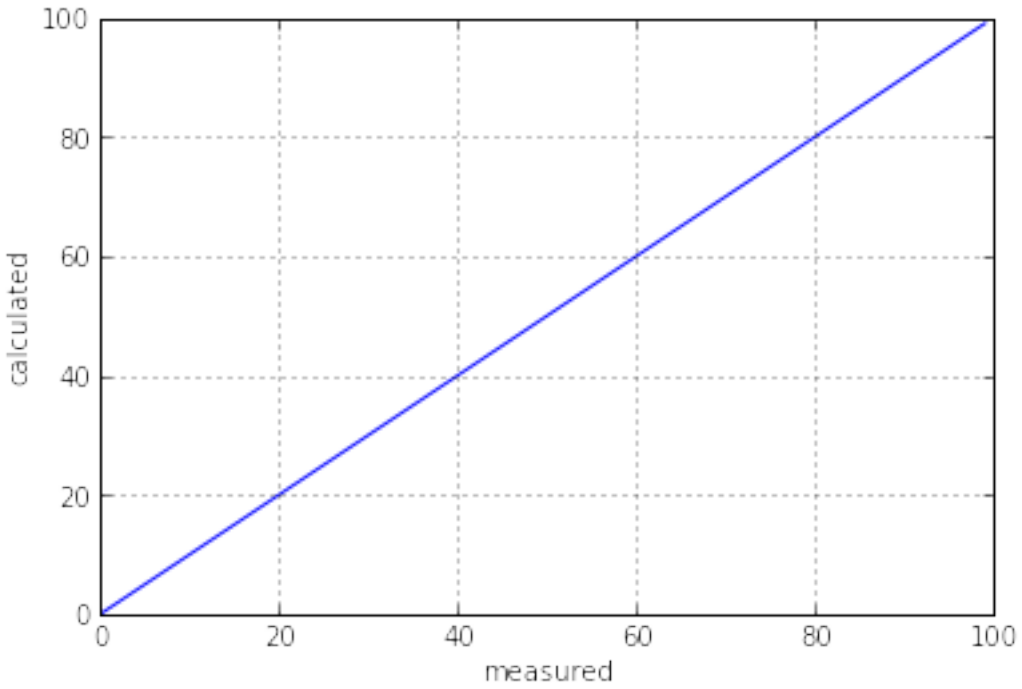
How quickly can we plot 10K numbers?

```
In [44]: _ = hist(random.randn(100000), 100)
```

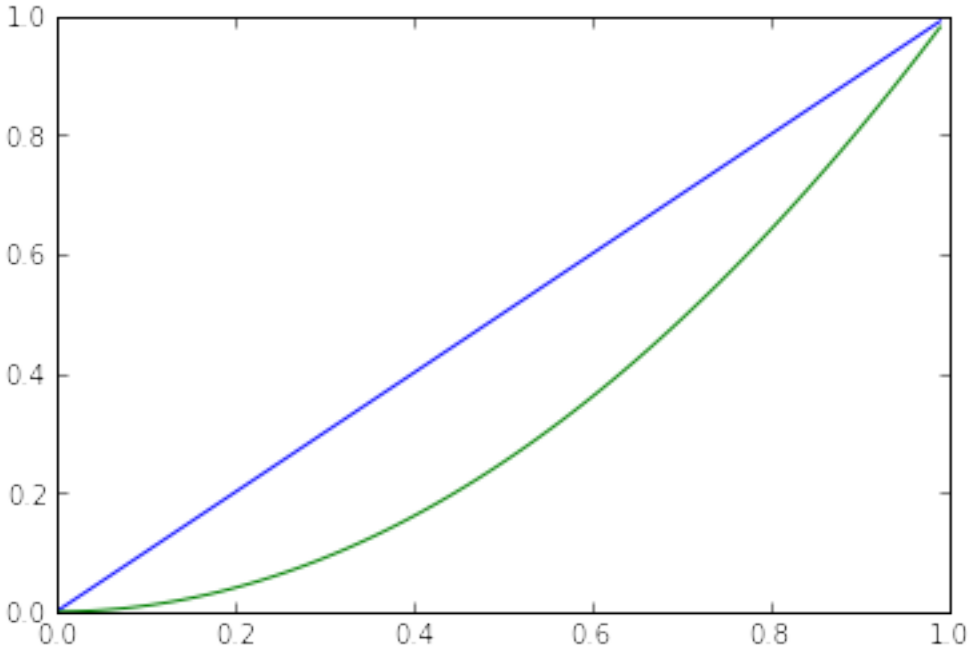


5.6 Basic Plotting

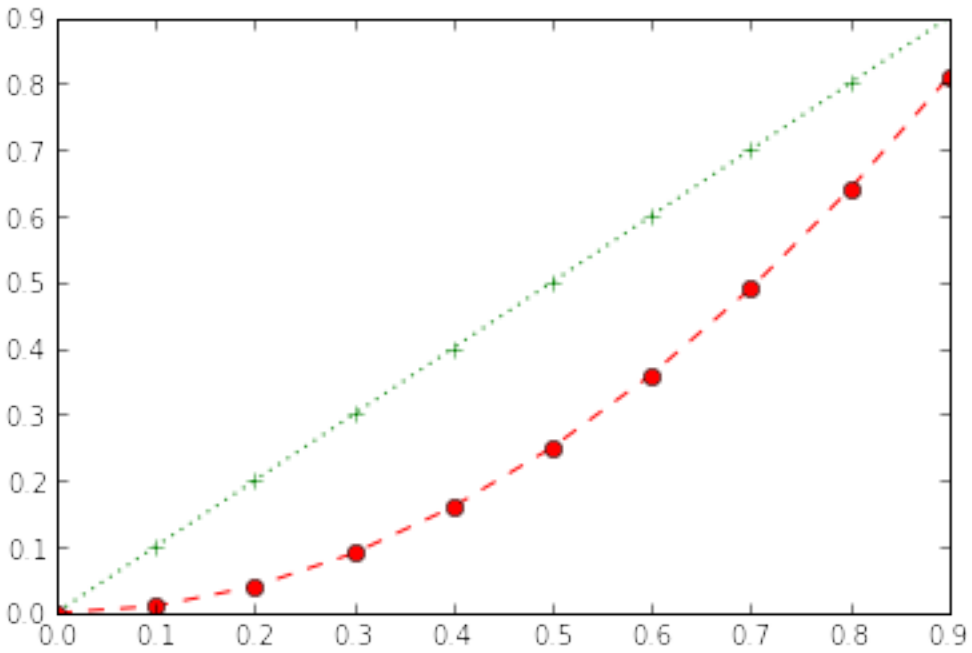
```
In [45]: _ = plot(range(100))
         _ = xlabel("measured")
         _ = ylabel("calculated")
         grid(True)
```



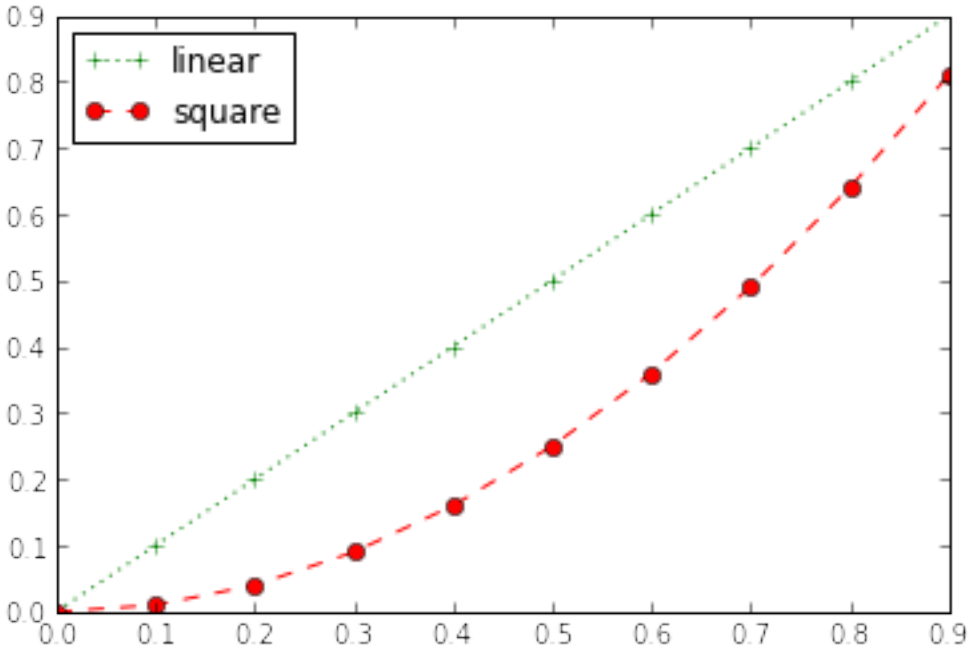
In [46]:
```
x = [val*.01 for val in range(100)]  # [0, .01, .02, ..., .98, .99]
linear = [val for val in x]
square = [val**2 for val in x]
_ = plot(x, linear, x, square)
```



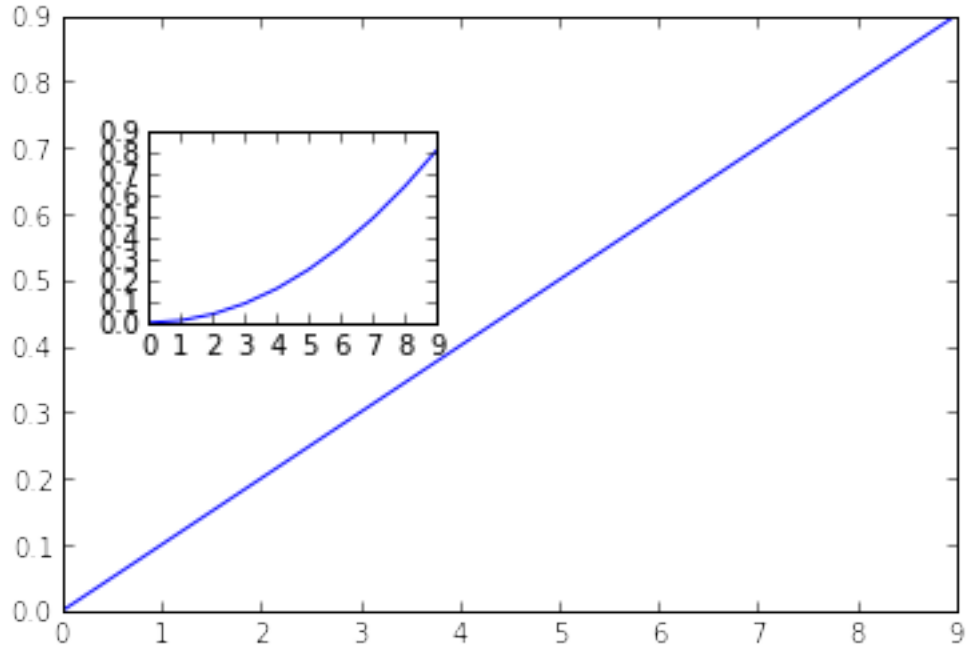
In [47]:
```
num_vals = 10
x = [float(val)/num_vals for val in range(num_vals)]  # [0, .01, .02, ..., .98, .99]
linear = [val for val in x]
square = [val**2 for val in x]
_= plot(x, linear, 'g:+', x, square, 'r--o')
```



In [53]:
```
num_vals = 10
x = [float(val)/num_vals for val in range(num_vals)]  # [0, .01, .02, ..., .98, .99]
linear = [val for val in x]
square = [val**2 for val in x]
_ = plot(x, linear, 'g:+', x, square, 'r--o')
_ = legend(('linear', 'square'), loc='upper left')
```




```
In [54]: _ = plot(linear)
         _ = axes([0.2, 0.5, 0.25, 0.25])
         _ = plot(square)
```



- mathematical algorithms and convenience functions built on the Numpy,
- organized into subpackages covering different scientific computing domains,
- a data-processing and system-prototyping environment rivaling sytems such as MATLAB, IDL, Octave, R-Lab, and SciLab

SciPy: Collection of High-Level Tools

- File IO (scipy.io)
- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems with ARPACK
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- Weave (scipy.weave)

scipy.io

```
In [55]: import scipy
         from scipy import io as spio
         a = np.ones((3, 3))
         spio.savemat('file.mat', {'a': a}) # savemat expects a dictionary
         data = spio.loadmat('file.mat', struct_as_record=True)
         data['a']
```

```
/Users/knepley/MacSoftware/lib/python2.7/site-packages/scipy/io/matlab/mio.py:267: FutureWarning: Using
  oned_as=oned_as)
```

```
Out[55]: array([[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]])
```

Also see:

- Images
- IDL Files
- Matrix Market Files
- Wav files

scipy.special

- Bessel function, such as scipy.special.jn() (nth integer order Bessel function)
- Elliptic function (scipy.special.ellipj() for the Jacobian elliptic function, ...)
- Gamma function: scipy.special.gamma(), also note scipy.special.gammaln() which will give the log of

Gamma to a higher numerical precision.

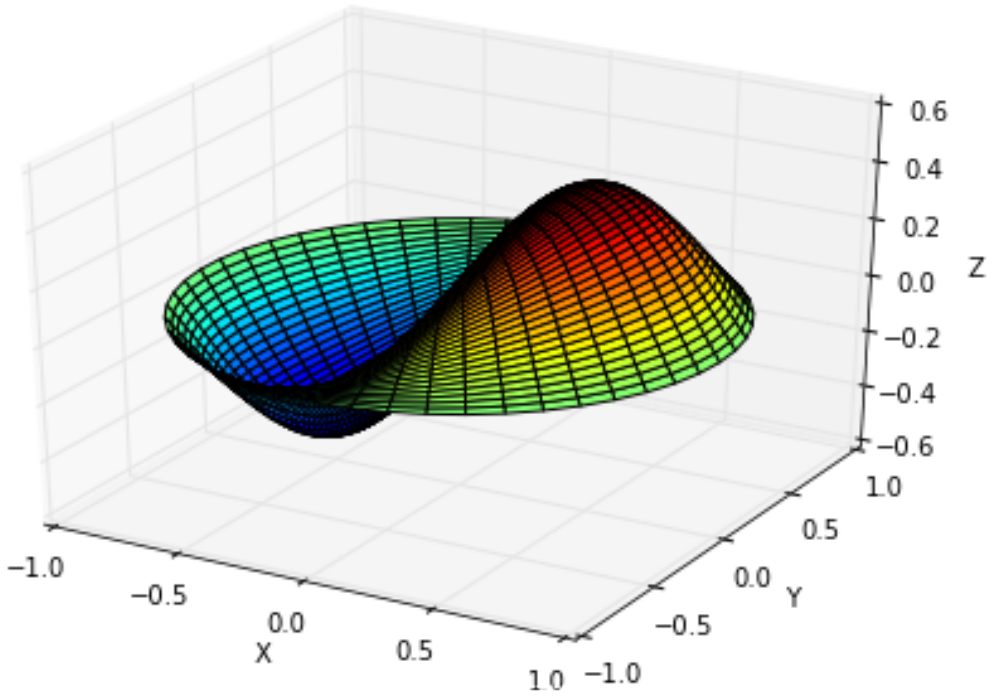
- Erf, the area under a Gaussian curve: scipy.special.erf()

```
In [56]: from scipy.special import jn
         x = np.linspace(0, 1, 10)
         print [jn(val, 1.0) for val in x]
```

```
[0.76519768655796649, 0.77042135015704238, 0.75775886140714455, 0.73087640216944849,
 0.69324418101264496, 0.64801456180958794, 0.59794997367362868, 0.54539073261127002,
 0.49225331977915776, 0.44005058574493355]
```

```
In [57]: from scipy import *
         from scipy.special import jn, jn_zeros
         def drumhead_height(n, k, distance, angle, t):
             nth_zero = jn_zeros(n, k)
             return cos(t)*cos(n*angle)*jn(n, distance*nth_zero)
         theta = r_[0:2*pi:50j]
         radius = r_[0:1:50j]
         x = array([r*cos(theta) for r in radius])
         y = array([r*sin(theta) for r in radius])
         z = array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])
```

```
In [58]: from mpl_toolkits.mplot3d import Axes3D
         from matplotlib import cm
         fig = pylab.figure()
         ax = Axes3D(fig)
         ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)
         _ = ax.set_xlabel('X')
         _ = ax.set_ylabel('Y')
         _ = ax.set_zlabel('Z')
```



scipy.linalg

- Matrix class (all routines still work with 2D NumPy arrays)
- Basic Linear Algebra
- Decompositions
- Matrix Functions
- Special matrices

```
In [59]: A = sp.mat('[1 3 5; 2 5 1; 2 3 8]')
         A

Out[59]: matrix([[1, 3, 5],
                 [2, 5, 1],
                 [2, 3, 8]])

In [60]: A.I

Out[60]: matrix([[-1.48,  0.36,  0.88],
                 [ 0.56,  0.08, -0.36],
                 [ 0.16, -0.12,  0.04]])

In [61]: from scipy import linalg
         linalg.inv(A)

Out[61]: array([[-1.48,  0.36,  0.88],
                [ 0.56,  0.08, -0.36],
                [ 0.16, -0.12,  0.04]])

In [62]: linalg.svd(A)

Out[62]: (array([[-0.52956045,  0.04298197, -0.84718255],
                 [-0.34647857, -0.92256717,  0.16977167],
                 [-0.77428569,  0.38343496,  0.50344742]]),
          array([ 11.13385134,   4.2134737 ,   0.53291053]),
          array([[-0.24888863, -0.50691635, -0.82528193],
                 [-0.24570758, -0.79117262,  0.56006577],
                 [ 0.93684696, -0.34217203, -0.07236068]]))

In [63]: linalg.toeplitz([1,2,3],[1,2,4,5,6])

Out[63]: array([[1, 2, 4, 5, 6],
                [2, 1, 2, 4, 5],
                [3, 2, 1, 2, 4]])
```

5.7 Pattern Formation Exercise

This exercise has been adapted from an example by David Ketcheson, KAUST

5.7.1 5 Point Laplace Operator

We start by constructing a dense matrix representing our stencil operator applied to a vector

```
In [64]: def five_pt_laplacian(m, a, b):
             """Construct a matrix that applies the 5-point Laplacian discretization"""
             e=np.ones(m**2)
             e2=([0]+[1]*(m-1))*m
             h=(b-a)/(m+1)
             A=np.diag(-4*e,0)+np.diag(e2[1:],-1)+np.diag(e2[1:],1)+np.diag(e[m:],m)+np.diag(e[m:],-m)
             A/=h**2
             return A

         five_pt_laplacian(8, 0.0, 1.0)
```

```
Out[64]: array([[-324.,   81.,    0., ...,    0.,    0.,    0.],
                [  81., -324.,   81., ...,    0.,    0.,    0.],
                [   0.,   81., -324., ...,    0.,    0.,    0.],
                ...,
                [   0.,    0.,    0., ..., -324.,   81.,    0.],
                [   0.,    0.,    0., ...,   81., -324.,   81.],
                [   0.,    0.,    0., ...,    0.,   81., -324.]])
```

Alternatively we can constuct a sparse representation.

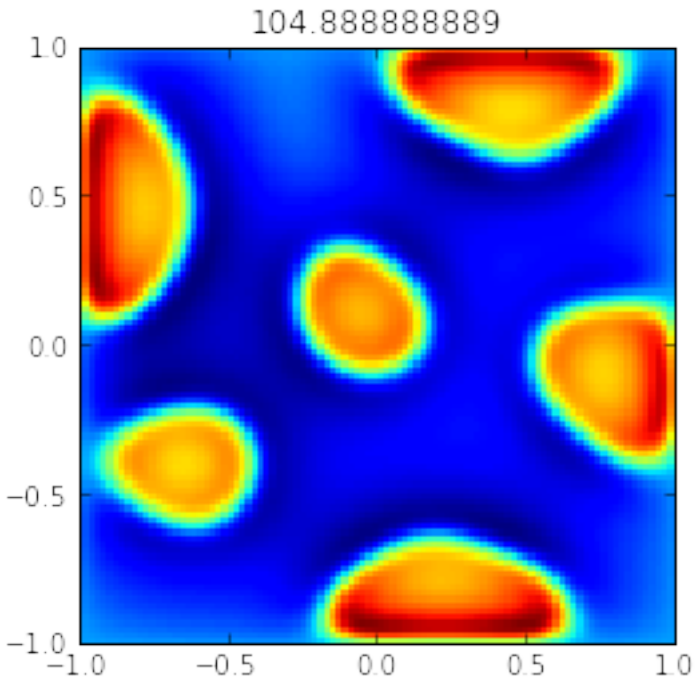
```
In [65]: from scipy.sparse import spdiags

         def five_pt_laplacian_sparse(m, a, b):
             """Construct a sparse matrix that applies the 5-point laplacian discretization"""
             e=np.ones(m**2)
             e2=([1]*(m-1)+[0])*m
             e3=([0]+[1]*(m-1))*m
             h=(b-a)/(m+1)
             A=spdiags([-4*e,e2,e3,e,e],[0,-1,1,-m,m],m**2,m**2)
             A/=h**2
             return A

         A = five_pt_laplacian_sparse(8, 0.0, 1.0)
         A?
```

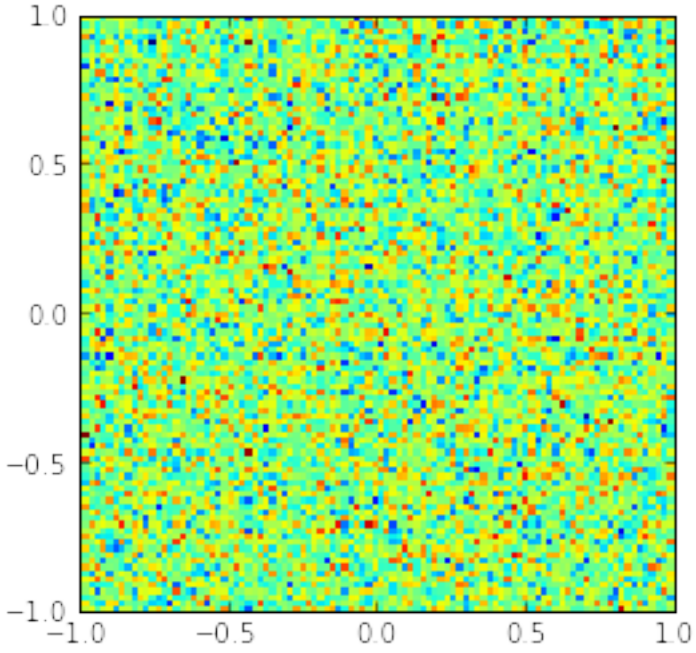
5.8 Pattern Formation Exercise

```
In [66]: from IPython.display import clear_output

         """Pattern formation code

             Solves the pair of PDEs:
                u_t = D_1 \nabla^2 u + f(u,v)
                v_t = D_2 \nabla^2 v + g(u,v)
         """

         #import matplotlib
         #matplotlib.use('TkAgg')
         import numpy as np
```

```
#import matplotlib.pyplot as plt
from scipy.sparse import spdiags,linalg,eye
from time import sleep

#Parameter values
Du=0.500; Dv=1;
delta=0.0045; tau1=0.02; tau2=0.2; alpha=0.899; beta=-0.91; gamma=-alpha;
#delta=0.0045; tau1=0.02; tau2=0.2; alpha=1.9; beta=-0.91; gamma=-alpha;
#delta=0.0045; tau1=2.02; tau2=0.; alpha=2.0; beta=-0.91; gamma=-alpha;
#delta=0.0021; tau1=3.5; tau2=0; alpha=0.899; beta=-0.91; gamma=-alpha;
#delta=0.0045; tau1=0.02; tau2=0.2; alpha=1.9; beta=-0.85; gamma=-alpha;
#delta=0.0001; tau1=0.02; tau2=0.2; alpha=0.899; beta=-0.91; gamma=-alpha;
#delta=0.0005; tau1=2.02; tau2=0.; alpha=2.0; beta=-0.91; gamma=-alpha; nx=150;

#Define the reaction functions
def f(u,v):
    return alpha*u*(1-tau1*v**2) + v*(1-tau2*u);

def g(u,v):
    return beta*v*(1+alpha*tau1/beta*u*v) + u*(gamma+tau2*v);


def five_pt_laplacian(m,a,b):
    """Construct a matrix that applies the 5-point laplacian discretization"""
    e=np.ones(m**2)
    e2=([0]+[1]*(m-1))*m
    h=(b-a)/(m+1)
    A=np.diag(-4*e,0)+np.diag(e2[1:],-1)+np.diag(e2[1:],1)+np.diag(e[m:],m)+np.diag(e[m:],-m)
    A/=h**2
    return A

def five_pt_laplacian_sparse(m,a,b):
    """Construct a sparse matrix that applies the 5-point laplacian discretization"""
    e=np.ones(m**2)
    e2=([1]*(m-1)+[0])*m
    e3=([0]+[1]*(m-1))*m
    h=(b-a)/(m+1)
    A=spdiags([-4*e,e2,e3,e,e],[0,-1,1,-m,m],m**2,m**2)
    A/=h**2
    return A

# Set up the grid
a=-1.; b=1.
m=100; h=(b-a)/m;
x = np.linspace(-1,1,m)
y = np.linspace(-1,1,m)
Y,X = np.meshgrid(y,x)

# Initial data
u=np.random.randn(m,m)/2.;
v=np.random.randn(m,m)/2.;
hold(False)
plt.pcolormesh(x,y,u)
plt.colorbar; plt.axis('image');
```

```
plt.draw()
u=u.reshape(-1)
v=v.reshape(-1)

A=five_pt_laplacian_sparse(m,-1.,1.);
II=eye(m*m,m*m)

t=0.
dt=h/delta/5.;
fig, ax = plt.subplots()
plt.colorbar

#Now step forward in time
for k in range(120):
    #Simple (1st-order) operator splitting:
    u = linalg.spsolve(II-dt*delta*Du*A,u)
    v = linalg.spsolve(II-dt*delta*Dv*A,v)

    unew=u+dt*f(u,v);
    v   =v+dt*g(u,v);
    u=unew;
    t=t+dt;

    #Plot every 3rd frame
    if k/3==float(k)/3:
        U=u.reshape((m,m))
        ax.pcolormesh(x,y,U)
        #ax.colorbar
        ax.axis('image')
        ax.set_title(str(t))
        #ax.draw()
        clear_output()
        display(fig)

plt.close()
```


104.888888889
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0