Introducing Python
Python in HPC
TACC Training, Oct. 15, 2012

Presenters:
Andy R. Terrel, PhD
Texas Advanced Computing Center
University of Texas at Austin

Yaakoub El Khamra
Texas Advanced Computing Center
University of Texas at Austin

0.1 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, we encourage you to download the updated version (see previous slide) and try the interactive version.

To run the interactive version, you need a good Python environment including:

- IPython version >= 13.0
- Numpy version >= 1.5
- Scipy
- Matplotlib

Move to the directory containing the tarball and execute:

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

We heartily endorse the Anaconda distribution and the Free Enthought Python Distribution.

0.2 Presentation mode

The slide show mode is only supported by an IPython development branch version. To get it I recommend cloning from the official branch, adding Matthias Carreau's remote, fetching and using his branch slideshow_extension2. Here are the commands:

```
git clone git://github.com/ipython/ipython.git # Official clone
cd ipython
git remote add carreau git://github.com/Carreau/ipython.git # Matthias' branch
git fetch carreau # Fetch the branches
git checkout carreau/slideshow_extension2 # Checkout the slideshow extension
python setup.py develop # Install the development version
ipython notebook # Check out the slideshows.
```

1 Introduction to Python (This Notebook)

1.0.1 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, material from Nathan Collier, and the SciPy Lecture Notes.

Texts in Computational Science and Engineering
3
Hans Petter Langtangen
Python Scripting for Computational Science
Third Edition
Springer
Texts in Computational Science and Engineering
6
Hans Petter Langtangen
A Primer on Scientific Programming with Python
Second Edition
Springer

Figure 1: Books

1.1 Scripting vs Traditional programming

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

Traditional programming: fortran, c, c++, c#, java

Scripting: python, perl, ruby, (matlab)

A major thrust of scripting is that you can automate many tasks that otherwise you would do by hand.

1.2 Has this ever happened to you?

1.2.1 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.

1.2.2 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.

1.3 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.

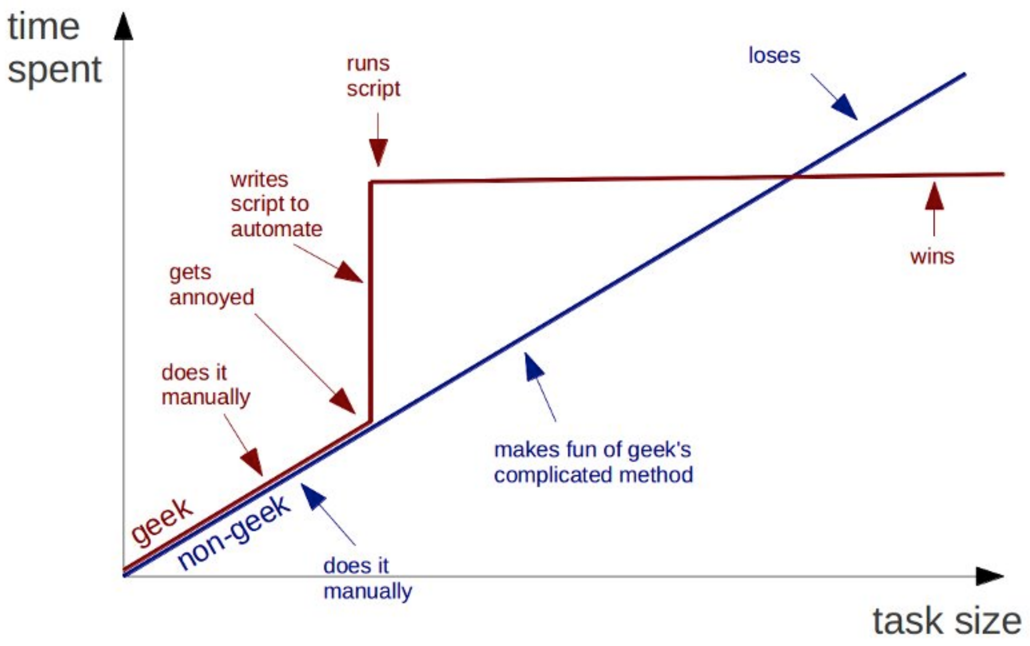


Figure 2: Recall we want to script to SAVE time

1.4 Why scripting is useful

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

- scripting languages have nicer interfaces
- allow you to build your own work environment
- scientific computing is more than crunching numbers
- easier creation of GUIs and demos
- create modern interfaces to old codes
- allow you test interactively
- cleaner, shorter, easy to read code

1.5 Is Python > MATLAB?

1.5.1 Similarities

- no mandatory variable declaration (dynamic typing)
- simple and easy to use syntax
- easy creation of GUIs
- merges simulation and visualization

1.5.2 Differences

- Python was designed to be completely open and to be integrated with external tools
- A Python module may contain a lot of functions and classes (compared to many m-files)
- Object-oriented programming is more convenient
- Interfacing C,C++,Fortran is better supported, this is important for running fast, parallel code
- scalar functions typically work with array arguments without changes to the arithmetic operators
- Python is FREE and runs on any platform C does, including supercomputers!

1.6 The right tool for the job

Many times people are looking for an easy way out. Scripting is easier to use, but not well suited for every situation. How do you know which tool is right for the job?

1.6.1 Traditional programming

- Does the application implement complex algorithms and/or data structures where low level control of implementation details is critical?
- Does the application manipulate large datasets and thus the memory has to be carefully controlled?
- Are you not likely to be changing the code once it is programmed?

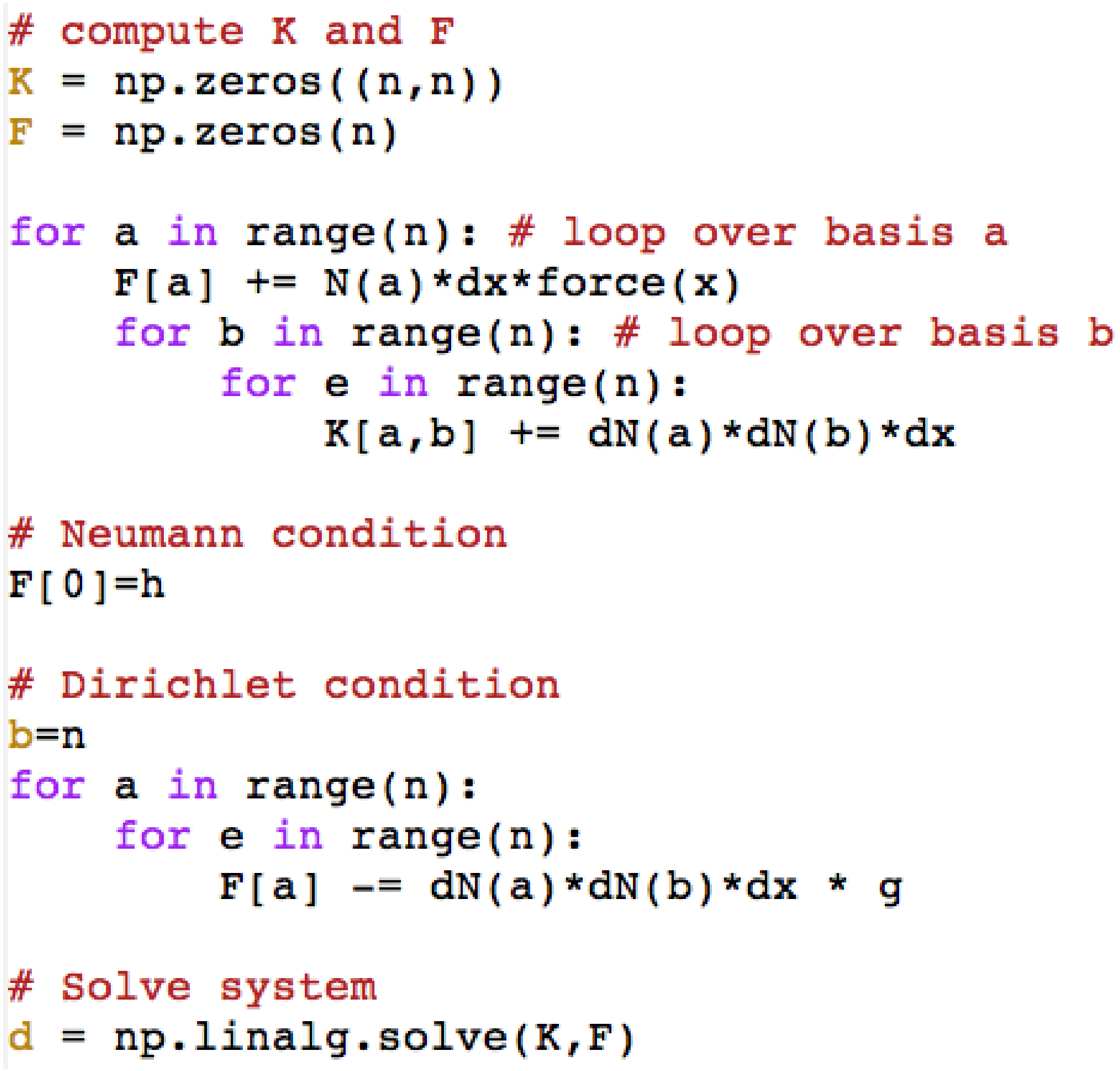
1.6.2 Scripting

- Your application's main task is to connected existing components
- The application depends on manipulating text
- The design of the application is expected to change over its life
- The CPU-intensive parts of your application may be migrated to C or Fortran
- Your application is largely based on common objects found in computer science

2 Some Sample Applications

2.1 Teaching the finite element method

2.1.1 Stiffness matrix computation



```
# 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 3: Stiffness matrix

2.2 Monitor program progress

2.2.1 Python parses a datafile and makes plots of current progress

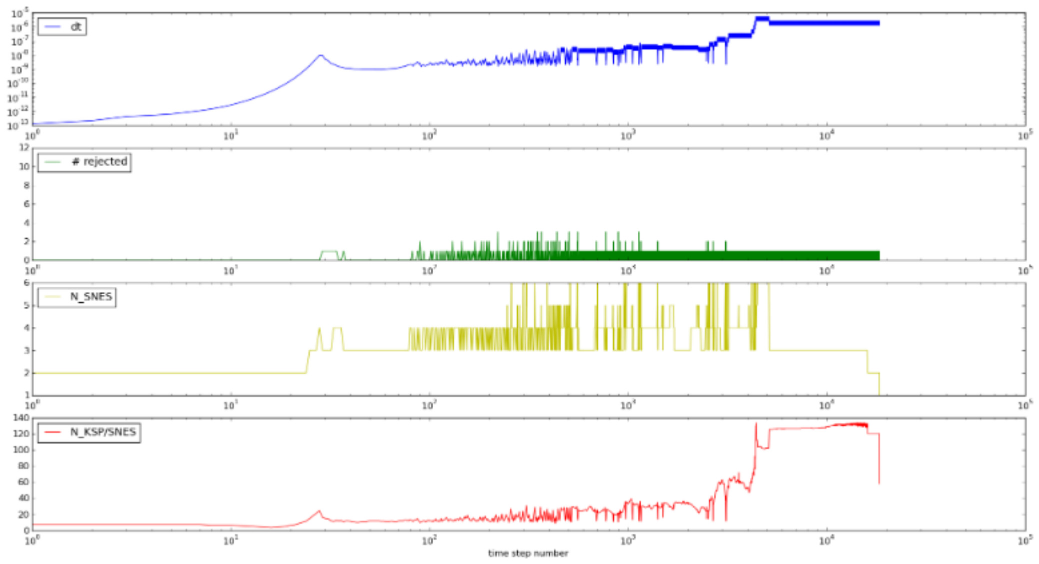


Figure 4: Log

2.3 Problem prototypes

2.3.1 Nonlinear, time dependent problem

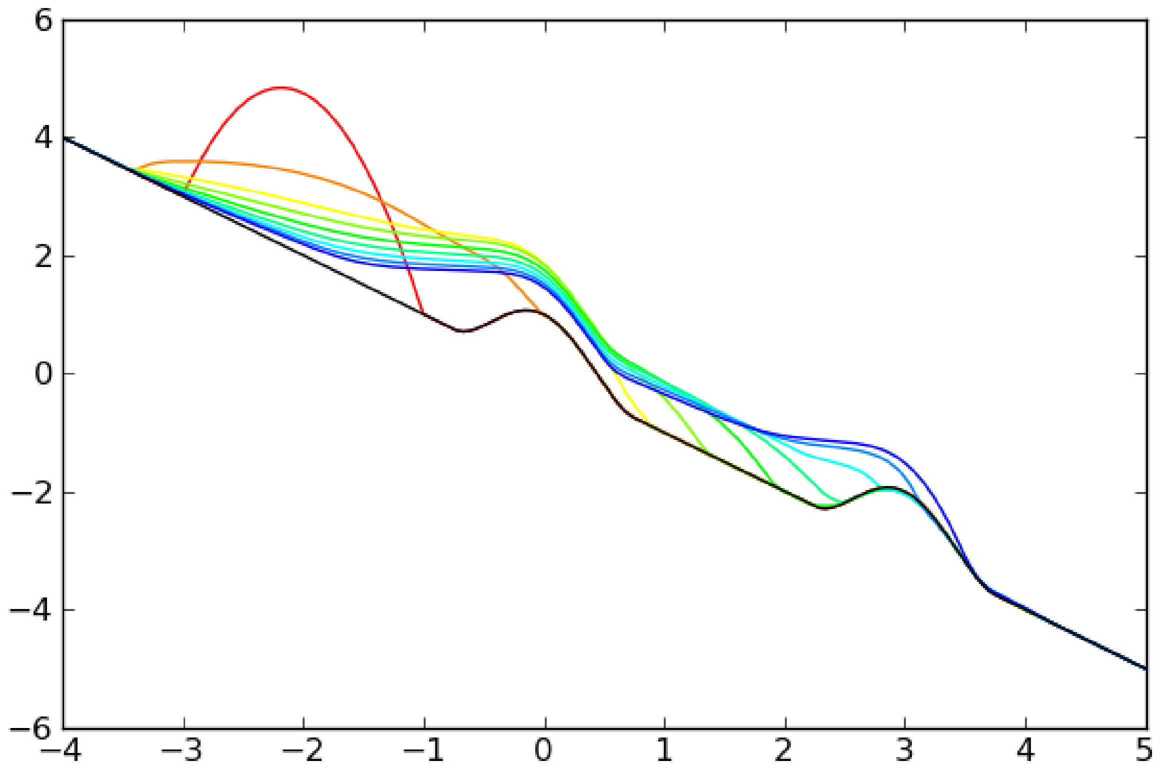


Figure 5: NonLinear problem

2.4 Problem prototypes

2.4.1 Method for fitting surface to data

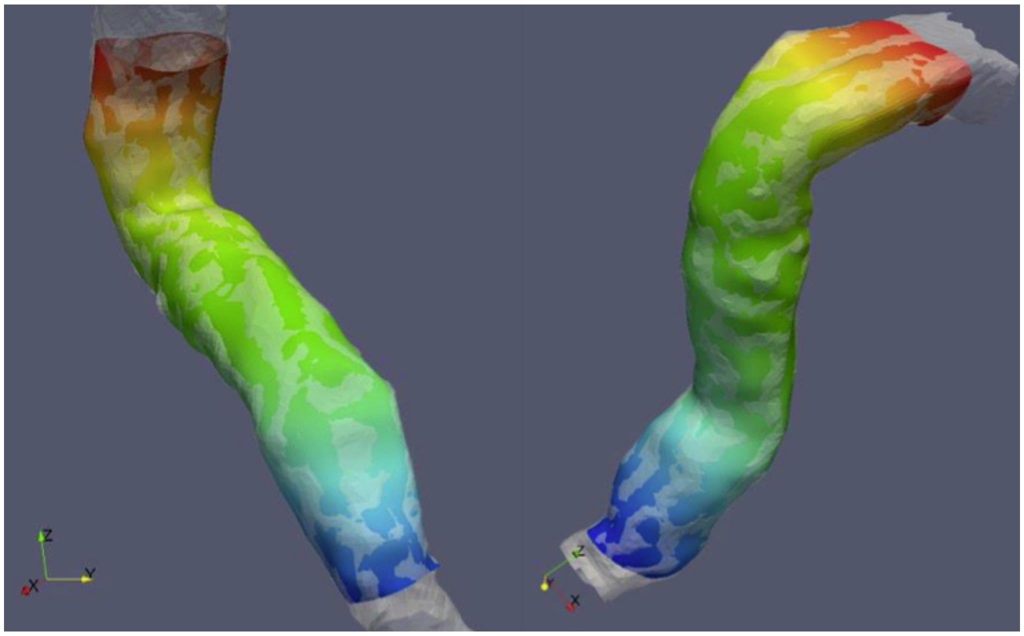


Figure 6: Prototypes

2.5 Structural program

2.5.1 Python manages floors and columns

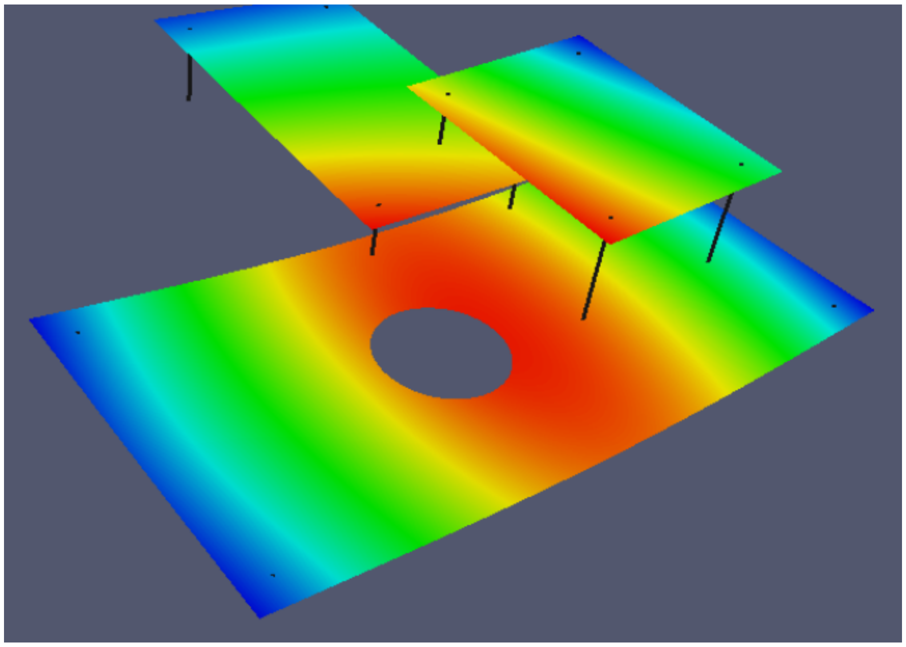


Figure 7: Structural program

2.6 Auto-generation of results

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

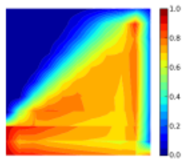
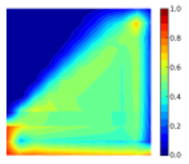
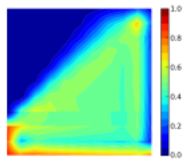
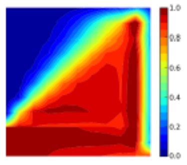
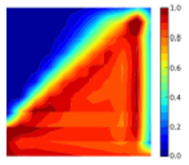
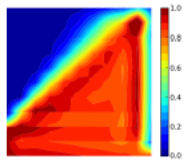
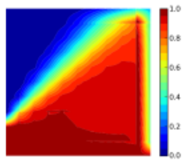
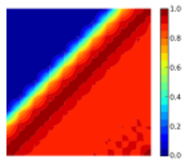
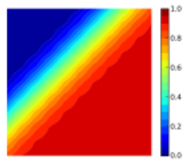
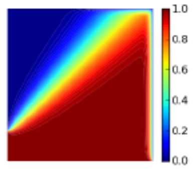
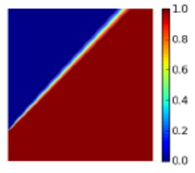
	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
SN			
	min/max = -0.13/0.96	min/max = -0.14/0.91	min/max = -0.14/0.91
WN			
	min/max = -0.09/1.14	min/max = -0.10/1.26	min/max = -0.10/1.25
QON			
	min/max = -0.12/1.15	min/max = -0.05/1.07	min/max = -0.04/0.96
Reference			N/A

Figure 8: Result Generation

3 Beginner's Guide

3.1 Scientific Hello World

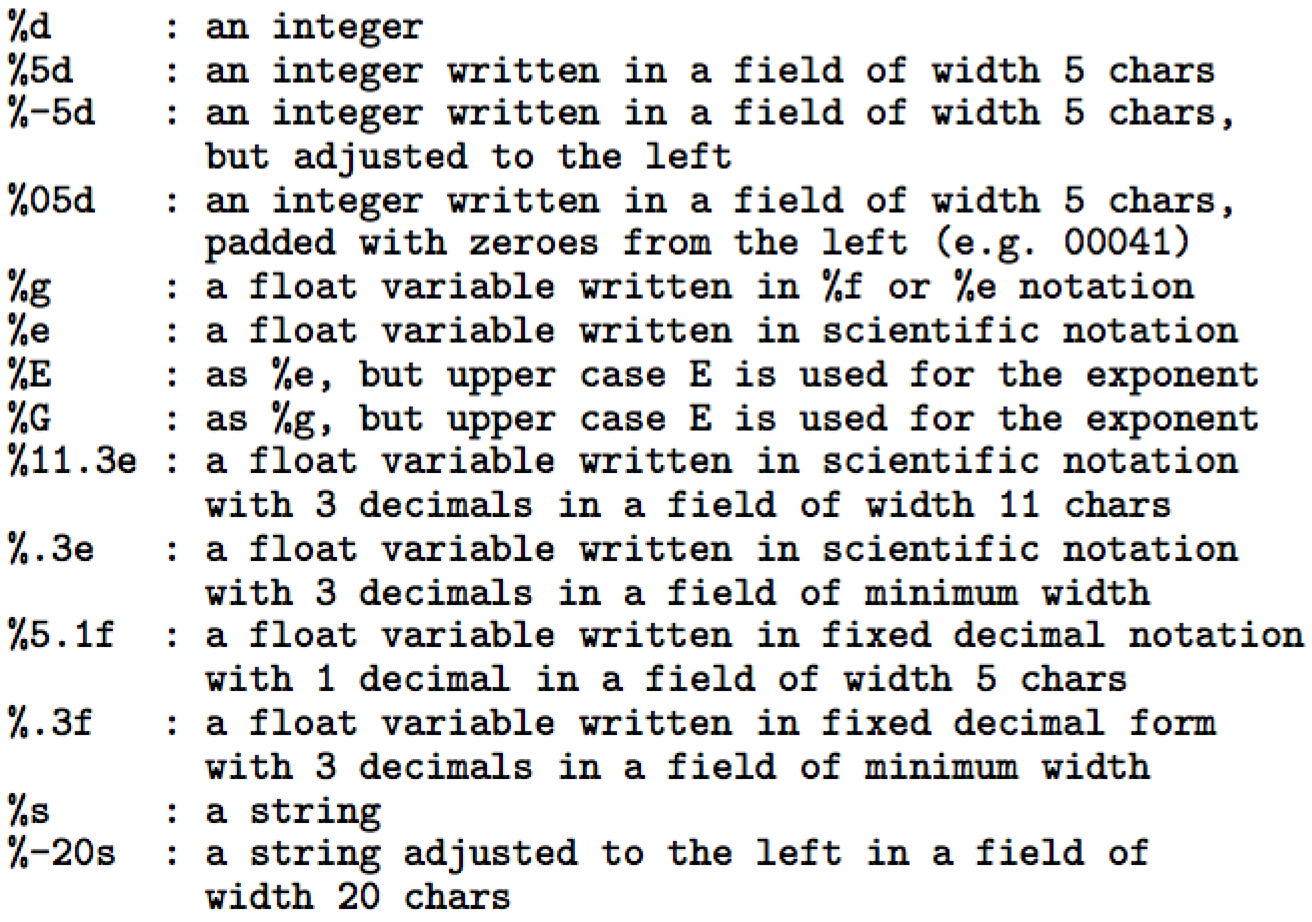
```python
#!/usr/bin/env python
import math
r = math.pi / 2.0
s = math.sin(r)
print "Hello world, sin(%f)=%f" % (r,s)
```

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

3.2 Simple I/O

```python
import math
infile = "data/numbers"
outfile = "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()
```

3.3 How to format the print statement, just like C!



%d	: an integer
%5d	: an integer written in a field of width 5 chars
%-5d	: an integer written in a field of width 5 chars, but adjusted to the left
%05d	: an integer written in a field of width 5 chars, padded with zeroes from the left (e.g. 00041)
%g	: a float variable written in %f or %e notation
%e	: a float variable written in scientific notation
%E	: as %e, but upper case E is used for the exponent
%G	: as %g, but upper case E is used for the exponent
%11.3e	: a float variable written in scientific notation with 3 decimals in a field of width 11 chars
%.3e	: a float variable written in scientific notation with 3 decimals in a field of minimum width
%5.1f	: a float variable written in fixed decimal notation with 1 decimal in a field of width 5 chars
%.3f	: a float variable written in fixed decimal form with 3 decimals in a field of minimum width
%s	: a string
%-20s	: a string adjusted to the left in a field of width 20 chars

Figure 9: format

3.4 Interacting with the system

```python
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 21:14:25 on 13 Oct, 2012
```

3.5 Regular expressions

3.5.1 A bib-file (see data/test.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}
}
```

```python
import re
pattern1 = "@Book{(.*),"
pattern2 = "\s+title\s+=\s+{(.*)},"
for line in file('data/test.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'
```

4 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)

```
a = array([0,1,2,3,4])
a
```

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

```
print a.ndim # Dimensionality
print a.shape # Shape
```

```
1
(5,)
```

```
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)
```

4.1 Common Arrays

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

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

```
print linspace(0, 1, 6) # A linear space of [0,1] with 6 pts
print 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]
```

```
print np.ones((3,3)) # 3 X 3 2D array of 1's
```

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

```
print np.eye(3)
```

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

```
print np.diag(arange(4))
```

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

```
print random.rand(4) # Uniform distribution
print random.randn(4) # Gaussian distribution
```

```
[ 0.72129428  0.34175057  0.7831168   0.25157102]
[-0.604022    1.41449551 -0.94909735  2.1459319 ]
```

4.2 Other Numpy Features to be aware of

- Reshaping

```
arr = 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

```
arr2.view?
```

- Index Slicing

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

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

```
((y[2:] - y[:-2])/(x[2:] - x[:-2])) # d^2y/dx^2 via center differencing
```

```
array([ 4,  8, 12, 16, 20, 24, 28, 32])
```

- Fancy Indexing

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

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

4.3 LinAlg, FFTs, and Random numbers

```
dot(arange(10), arange(10))
```

```
285
```

```
dot(arange(9).reshape(3,3),
    arange(9).reshape(3,3))
```

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

```
np.fft?
```

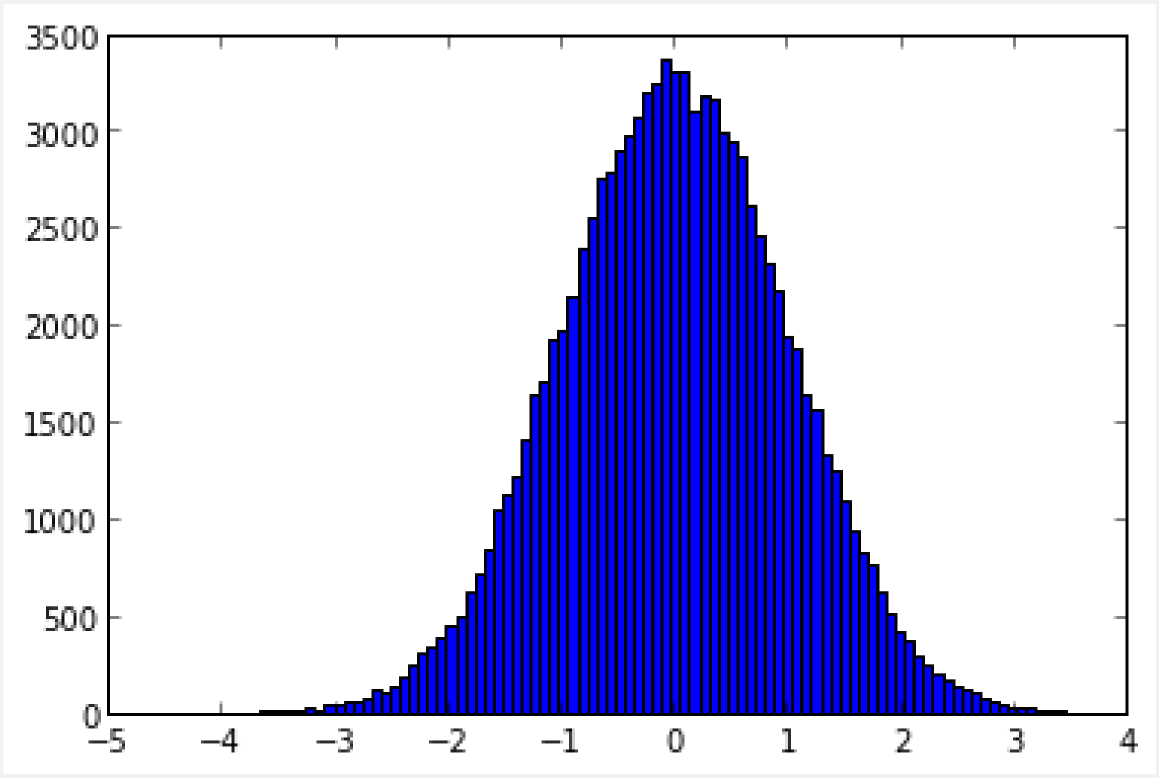
```
np.linalg?
```

```
np.random?
```

5 Quick Visualization / MatPlotLib

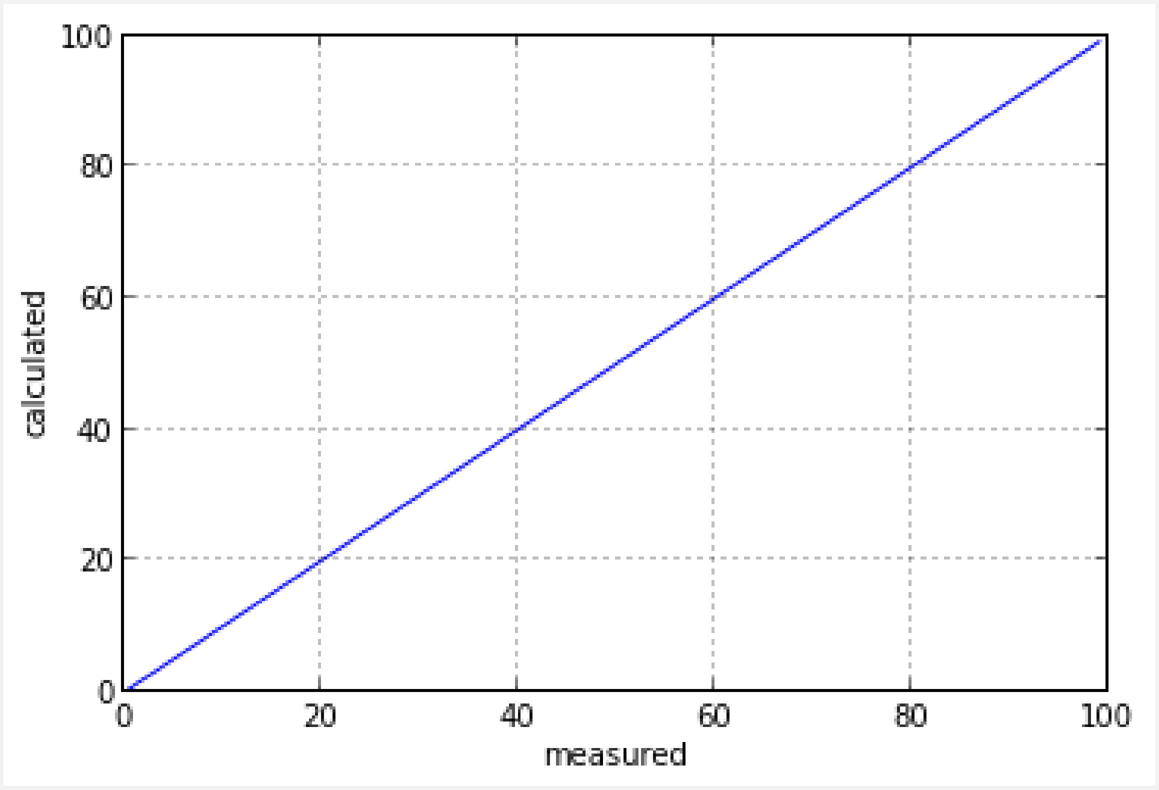
How quickly can we plot 10K numbers?

```
_ = hist(random.randn(100000), 100)
```

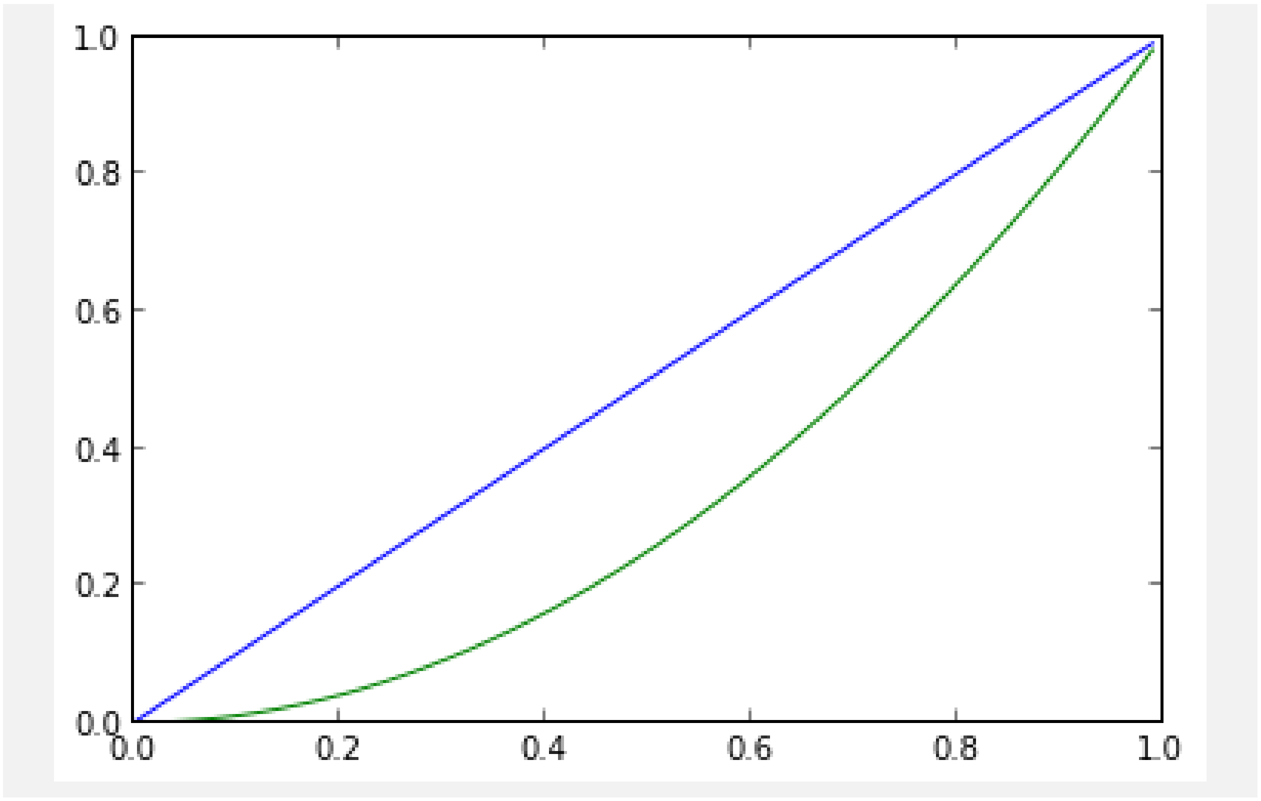


5.1 Basic Plotting

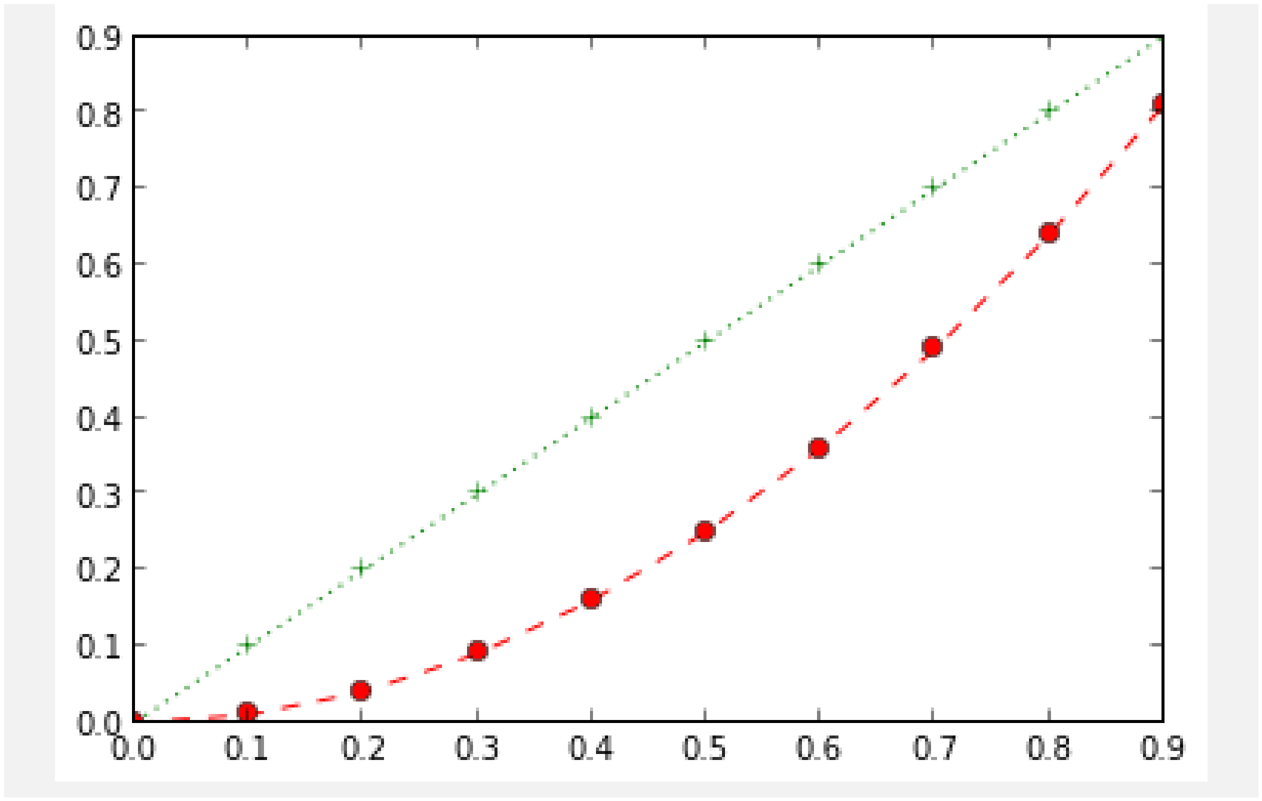
```
_ = plot(range(100))
_ = xlabel("measured")
_ = ylabel("calculated")
grid(True)
```



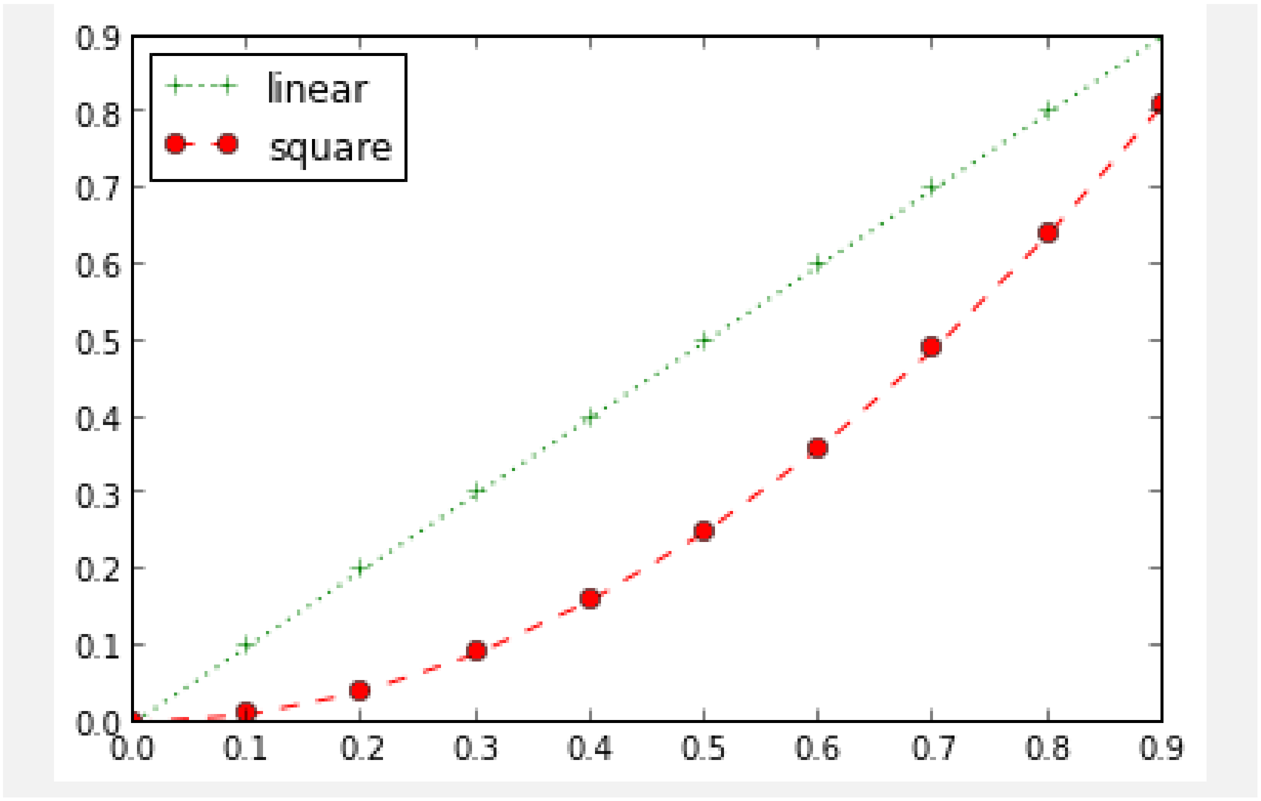
```
x = [val*.01 for val in range(100)] # [0, .01, .02, ..., .98, .99]
linear = [val for val in x]
square = [val**2 for val in x]
_ = plot(x, linear, x, square)
```



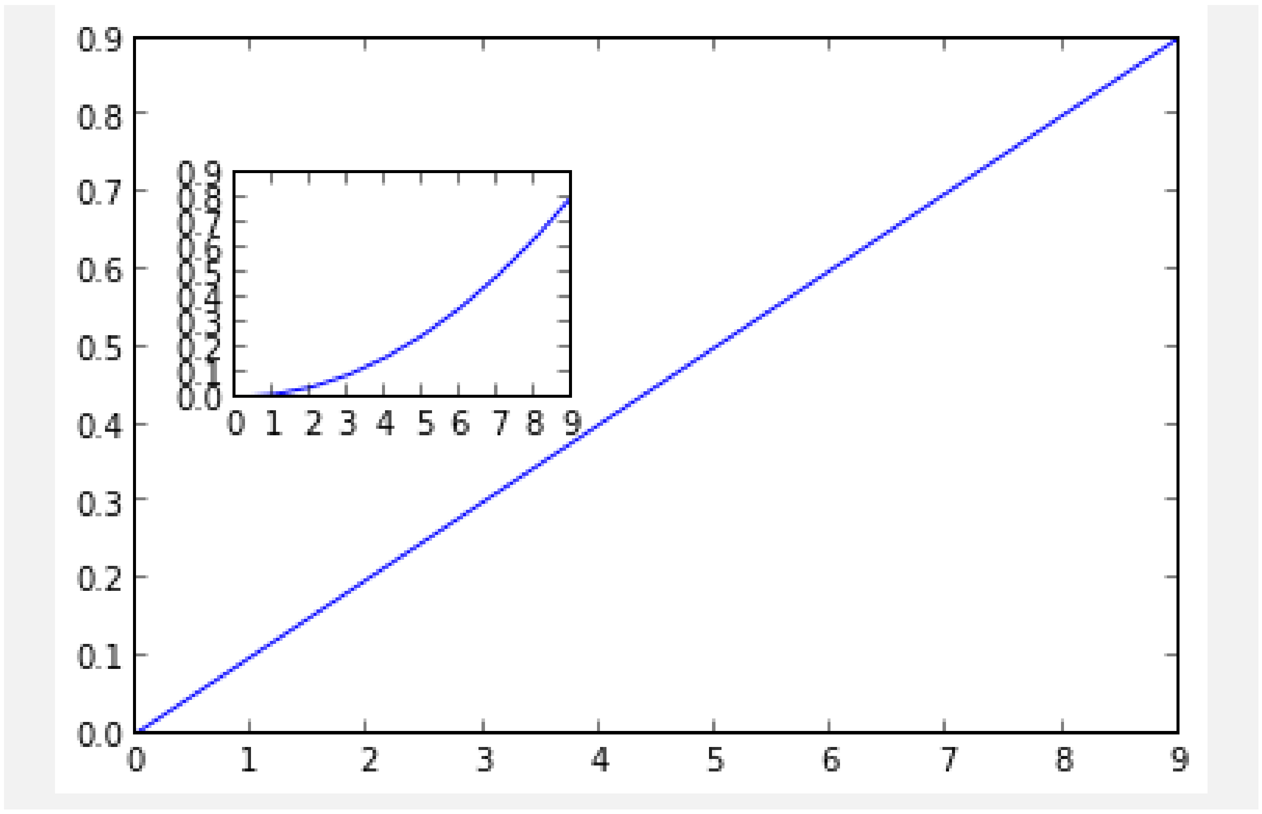
```
num_vals = 10
x = [float(val)/num_vals for val in range(num_vals)] # [0, .01, .02, ..., .98, .99]
linear = [val for val in x]
square = [val**2 for val in x]
_= plot(x, linear, 'g:+', x, square, 'r--o')
```



```
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')
```



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


0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
1
2
3
4
5
6
7
8
9

6 SciPy: Collection of High-Level Tools

- 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

6.1 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)

6.2 scipy.io

```python
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']
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/scipy/io/matlab/
  oned_as=oned_as)

array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Also see:

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

6.3 scipy.special

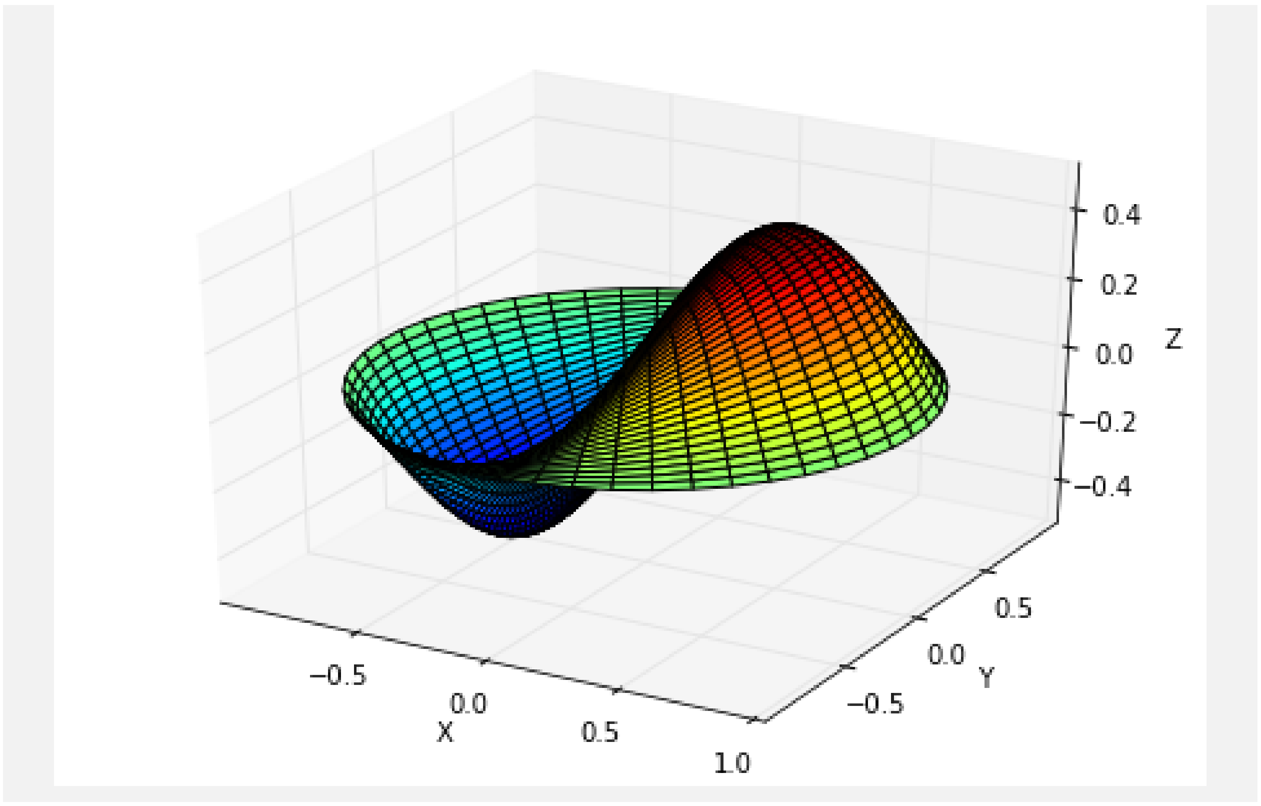
- Bessel function, such as scipy.special.jn() (nth integer order Bessel function)
- Elliptic function (scipy.special.ellipj() for the Jacobian elliptic function, . . .)
- Gamma function: scipy.special.gamma(), also note scipy.special.gammaln() which will give the log of Gamma to a higher numerical precision.
- Erf, the area under a Gaussian curve: scipy.special.erf()

```
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
```

```
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])
```

```
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')
```


0.4
0.2
0.0
−0.2
−0.4
Z
−0.5
0.0
0.5
1.0
X
−0.5
0.0
0.5
Y

6.4 scipy.linalg

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

```
A = mat('[1 3 5; 2 5 1; 2 3 8]')
A
```

```
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
```

```
A.I
```

```
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
```

```
from scipy import linalg
linalg.inv(A)
```

```
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
```

```
linalg.svd(A)
```

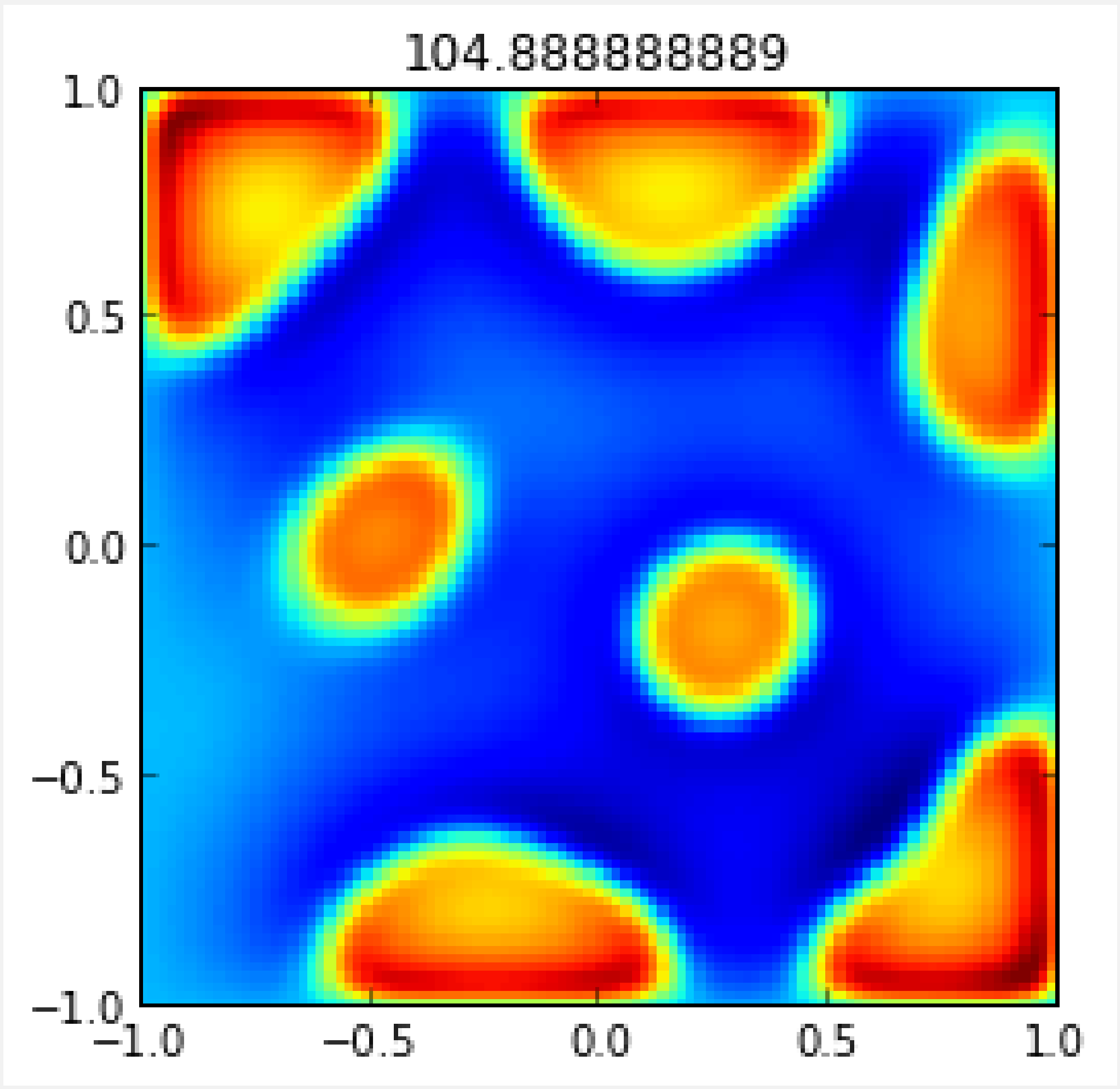
```
(array([[-0.52956045,  0.04298197, -0.84718255],
       [-0.34647857, -0.92256717,  0.16977167],
       [-0.77428569,  0.38343496,  0.50344742]]),
 array([ 11.13385134,   4.2134737 ,   0.53291053]),
 array([[-0.24888863, -0.50691635, -0.82528193],
       [-0.24570758, -0.79117262,  0.56006577],
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

```
linalg.toeplitz([1,2,3],[1,2,4,5,6])
```

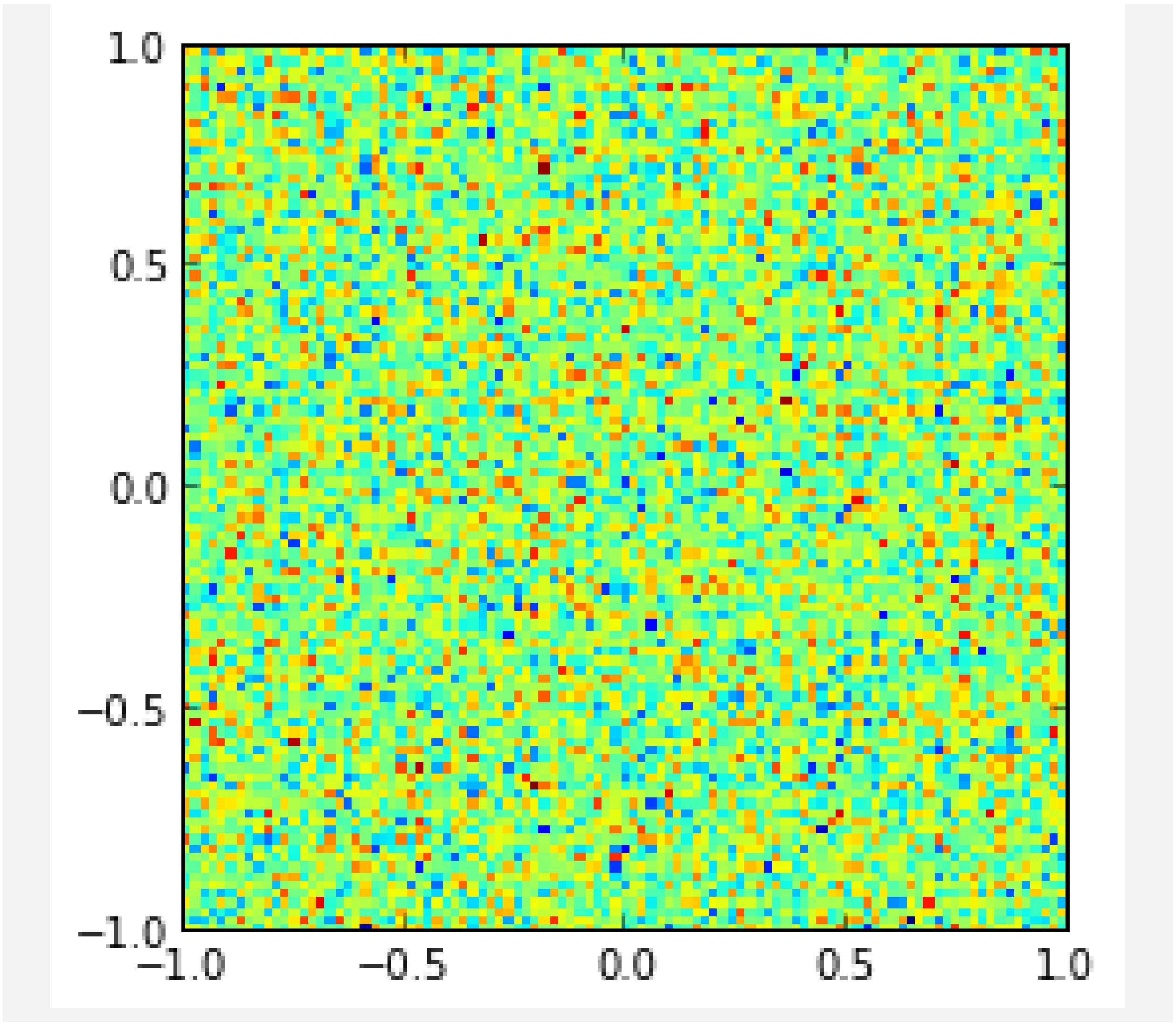
```
array([[1, 2, 4, 5, 6],
       [2, 1, 2, 4, 5],
       [3, 2, 1, 2, 4]])
```

7 Pattern Formation

```python
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()
```



```
<matplotlib.figure.Figure at 0x1077bb490>
```


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