Python in HPC

Supercomputing 2012

Presenters:

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

Travis Oliphant, PhD
Continuum Analytics

Aron Ahmadia, PhD
Supercomputing Laboratory
King Abdullah University of Science and Technoglogy

Updated Tutorial

These presentation materials are being continuously updated as we refine and improve our demonstrats. To get the latest version of this tutorial you can:

1) Download a zip or tar ball from the github SC2012 tag:

```
wget --no-check-certificate https://github.com/aterrel/HPCPythonSC2012/zipball/SC2012
wget --no-check-certificate https://github.com/aterrel/HPCPythonSC2012/tarball/SC2012
```

2) Checkout from git

```
git clone https://github.com/aterrel/HPCPythonSC2012.git
```

3) View the html version on nbviewer:
http://nbviewer.ipython.org/urls/raw.github.com/aterrel/HPCPythonSC2012/master/01_Introducing_Python.ipynb

4) As a last resort, head to https://github.com/aterrel/HPCPythonSC2012 for updated instructions (see the README at the bottom of the page).

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
```

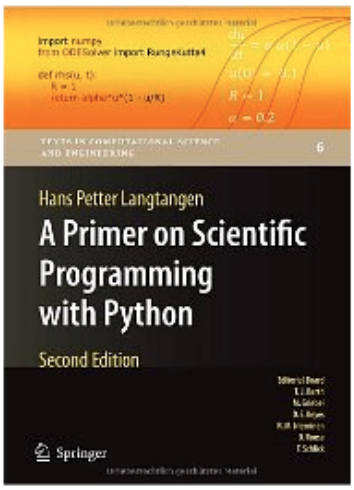
We heartily endorse the Free Enthought Python Distribution.

Introduction to Python (This Notebook)

Objectives

I. You will understand how scripting languages fit into the toolbox of a computational scientist.
II. You will see why Python is a powerful choice
III. 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 and is based on material from Nathan Collier.



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.

Has this ever happened to you?

Scenario 1

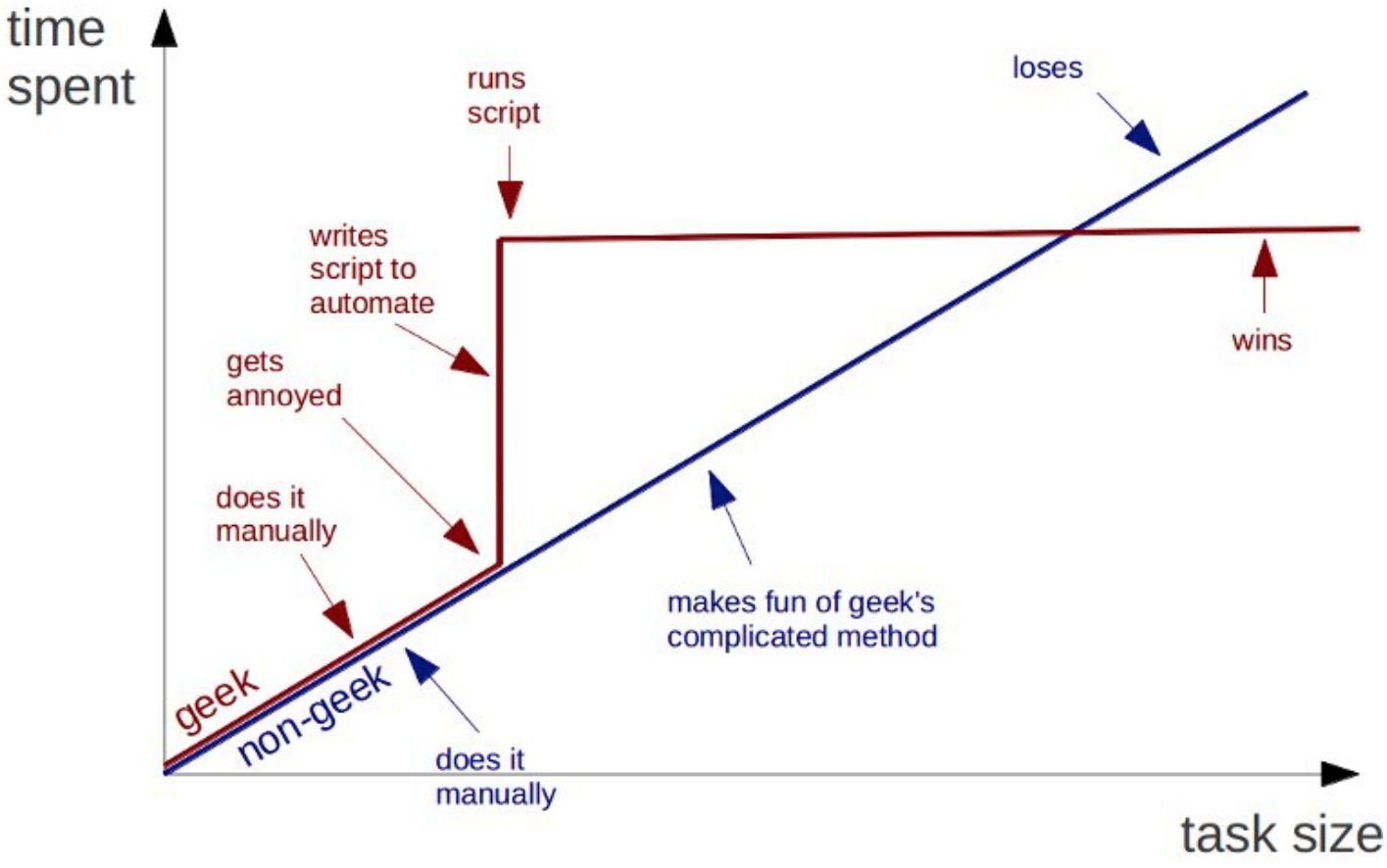
You are working on data for a presentation your advisor is giving at a conference. At the last minute, you realize that there is a major bug in your code and you need to regenerate all the images and graphs that you have given him. You spend 30 minutes regenerating the data and 6 hours regenerating the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

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.

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.



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

Is Python > MATLAB?

Similarities

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

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!

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?

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?

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

---

Some Sample Applications

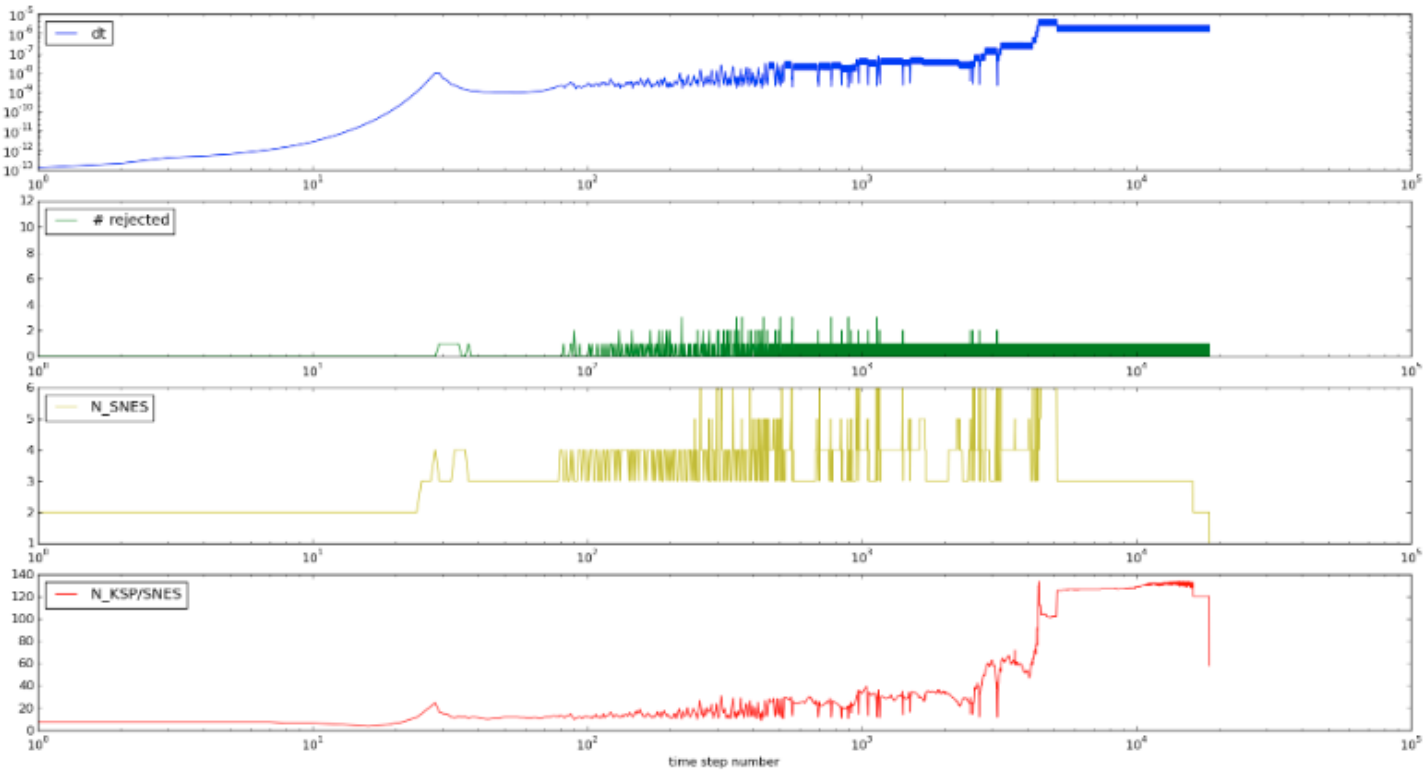
Teaching the finite element method

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)
```

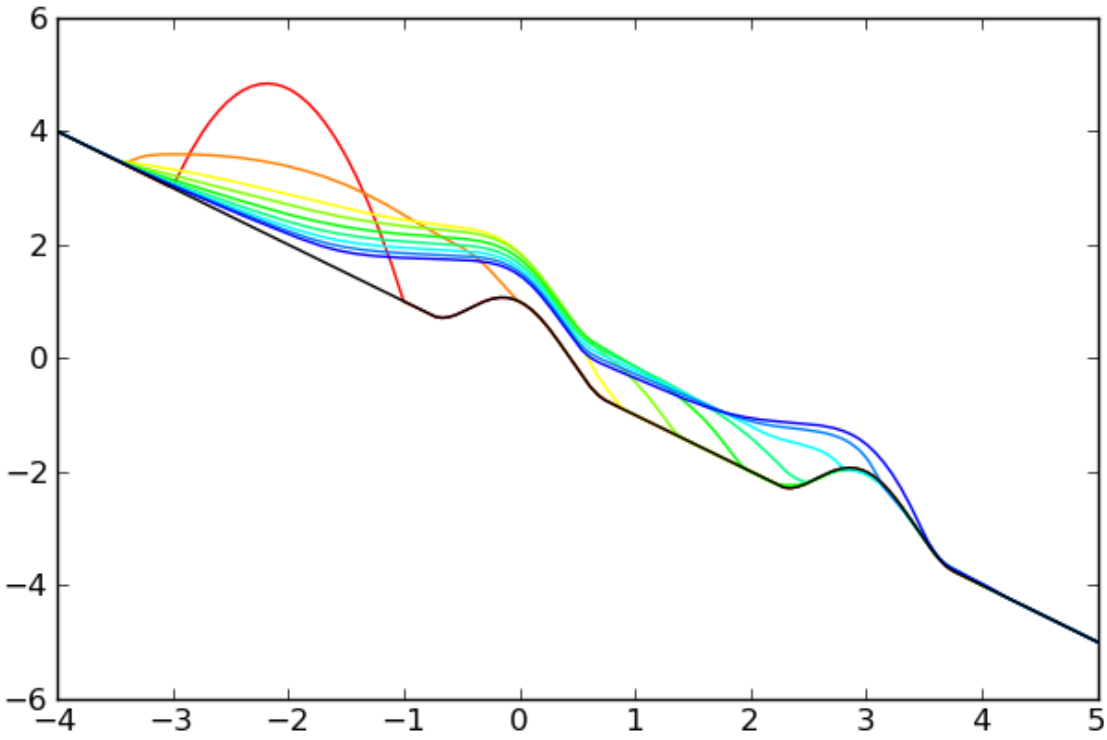
Monitor program progress

Python parses a datafile and makes plots of current progress



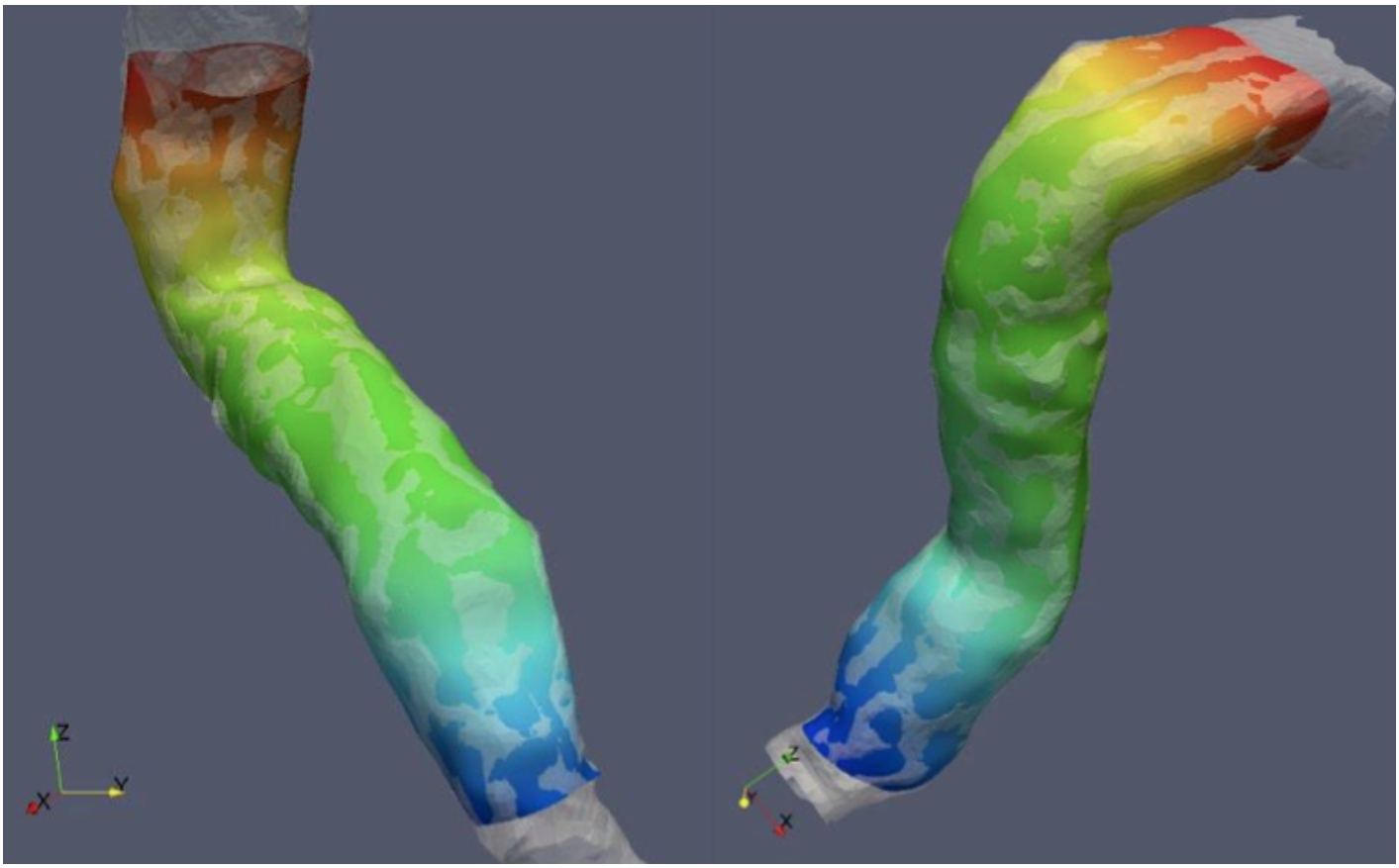
Problem prototypes

Nonlinear, time dependent problem



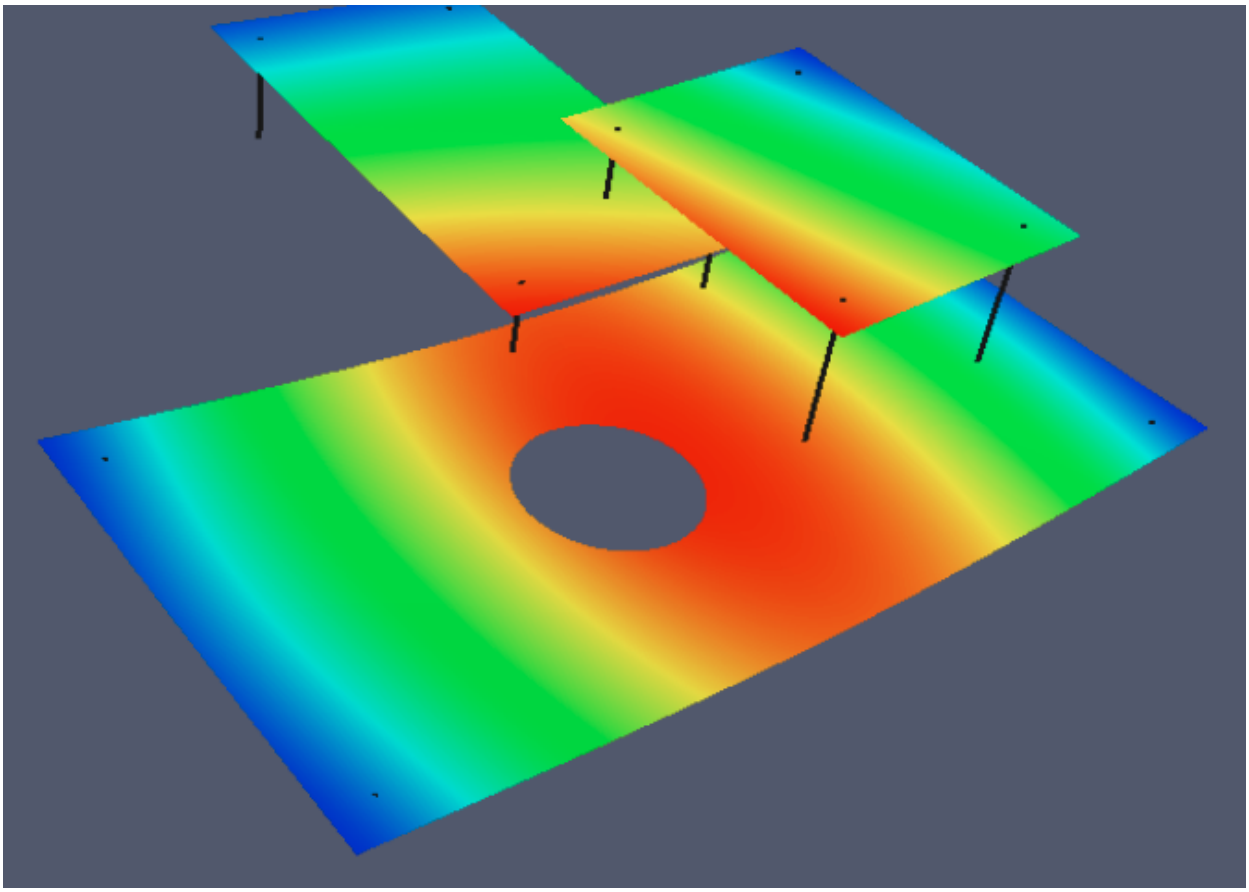
Problem prototypes

Method for fitting surface to data



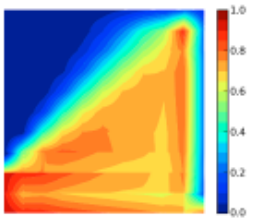
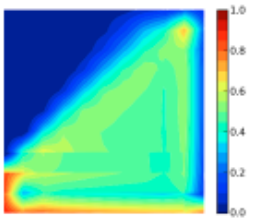
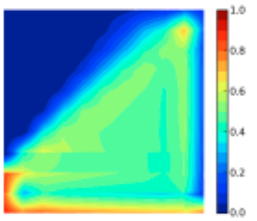
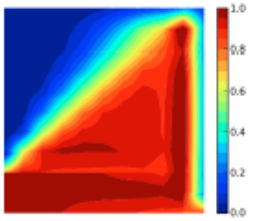
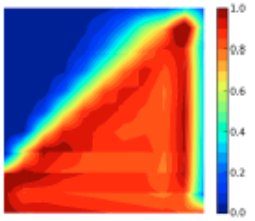
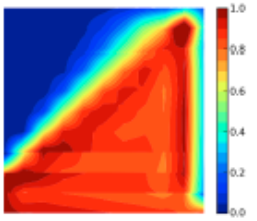
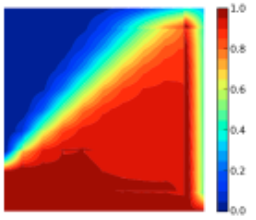
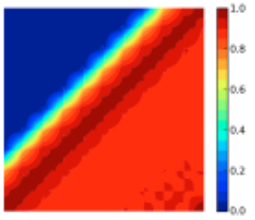
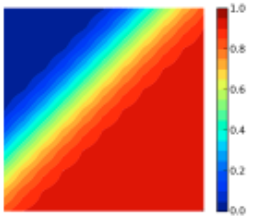
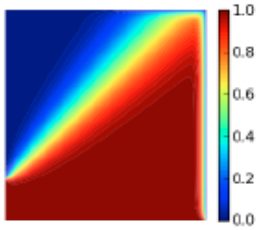
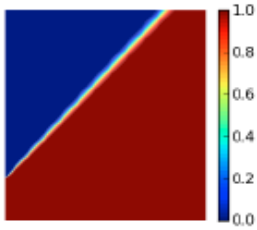
Structural program

Python manages floors and columns



Auto-generation of results

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

Beginner's Guide

Script 1

In [3]:
```
#!/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
```

Script 2

In [5]:
```
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()
```

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
```

Script 3

In [6]:
```
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 16:32:23 on 9 Sep, 2012
```

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}
}
```

Script 4

In [8]:

```
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'
```

Pattern Formation

In [8]:

```
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:
    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()
```

. . .