Numerical Programming in Python

Part III: Using Python for Numerics

Nick Maclaren

Computing Service

nmm1@cam.ac.uk, ext. 34761

February 2006

Post-Processing (1)

Often need to munge output to stay sane
May need arithmetic on values in it
Python is ideal tool for this purpose

Problem occurs when output may be corrupted
Common if running multiple processes
Schedulers, NFS etc. can interleave weirdly
Or overflow causing Fortran's '********'

Post-Processing (2)

Real code from HPCF benchmarking
Replaced twice the length of awk script
And became a lot more robust, too!

(cd Examples; python streams.py)
(cd Examples; python linpack.py)
(cd Examples; cat linpack.py)

Post-Processing (3)

```
from re import compile
fpnum = r' \d+ \. \d+E[+–] \d \d'
fpnum_1 = fpnum + r' +'
str = r' ^   *' + fpnum_1 + fpnum_1 +  \
      fpnum_1 + r'(' + fpnum + r') +' +  \
      fpnum_1 + fpnum + r' * \n$'
print str
pattern = compile(str)
```

Matches just the lines we want to use
Then use Python's numerics to summarise

Python Output

```
^ *\d+\.\d+E[+–]\d\d +
    \d+\.\d+E[+–]\d\d +
    \d+\.\d+E[+–]\d\d +
    (\d+\.\d+E[+–]\d\d) +
    \d+\.\d+E[+–]\d\d +
    \d+\.\d+E[+–]\d\d *\n$
```

Pre-Processing (1)

Data in one format, needed in another
Can also use for data input utility
Easy to do fairly thorough checking
Could prompt for corrections, etc.

Example is using Maple output in Matlab
(cd Examples; python maple.py)
(cd Examples; cat maple.py)
(cd Examples; cat maple.out)
(cd Examples; more maple_1.py)

Pre-Processing (2)

Can both munge data and run program
(cd Examples; python matlab.py)
(cd Examples; cat matlab.py)

Or both programs (Maple is not on PWF)
(cd Examples; cat matlab_1.py)

Becomes a complete application harness

Pre-Processing / Harness

```
from os import popen
from sys import stdout
from maple_1 import grind
strm_1 = popen("ssh nmm1@cus.cam.ac.uk  \
       maple < maple.in", "r")
strm_2 = popen("matlab –nodisplay –nojvm", "w")
grind(strm_1, strm_2)
x = strm_1.close()
if x != None : print x
x = strm_2.close()
if x != None : print x
```

Application Harnesses

Used on HPCF for submission script etc.
Can write high–quality commands easily
Start/stop/control subprocess commands

Including on different systems (see above!)
If not PWF, could use SSH keys
Then don't need to provide password

Won't go into any more detail here
But strongly recommend Python for harnesses

Numpy/Scipy

Intended for efficient work in Python
Generally does what it intends
Consider as alternative to Matlab

Very limited experience here with it
So check that it does what you need

Will give VERY simple examples
And then will describe how to use it

Starting Off

Examples assume the following:

```
python
from numpy import *
```

You can also use one or more of:

```
from numpy.linalg import *
from numpy.fft import *
from numpy.random import *
```

And others . . .

Arrays

Arrays are the basic numpy data type
Matrices are specialised 2–D arrays

Come in int, float, complex forms
New arrays usually get correct type
Updating element does NOT change type

Simple use is just as you would expect
If shapes match or one argument a scalar

Simple Arrays (1)

```
a = array([[1,2],[3,4]])
b = array([5,6])
print a
print b

print dot(a,b)
print a[0,1]*100
a[0,1] = 5
print a
```

Python Output

```
[[1 2]
 [3 4]]
[5 6]
[17 39]
200
[[1 5]
 [3 4]]
```

Simple Arrays (2)

```
a = array([[1,2],[3,4]])
print a*a
print dot(a,a)
print a/a

b = matrix([[1,2],[3,4]])
print b*b

c = array([[[1,2],[3,4]],[[5,6],[7,8]]])
print c
```

Python Output

```
[[ 1  4]
 [ 9 16]]
[[ 7 10]
 [15 22]]
[[1 1]
 [1 1]]
[[ 7 10]
 [15 22]]
[[[1 2]
  [3 4]]
 [[5 6]
  [7 8]]]
```

Array Types

```
a = array([[1,2],[3.0,4]])
b = array([[5,7],[9,1]])
print a.dtype
print b.dtype

print a+b
c = array([[5,7],[9,1]],dtype=float)
print c.dtype

b[0,1] = 6.78
print b
```

Python Output

```
float64
int64
[[  6.   9.]
 [ 12.   5.]]
float64
[[5 6]
 [9 1]]
```

Manipulating Arrays

Various construction methods (see doc.)
Creating a zero–filled matrix is simplest

Can reshape arrays, much as in Fortran 90
Can slice arrays, just as in rest of Python
Can use stride (step), as in Fortran 90

- Both ALIAS the (sub)array, not copy it

Creation and Reshaping

```
a = zeros((3,2,4),dtype=complex)
print a
b = identity(5,float)
print b

c = array([[[1,2],[3,4]],[[5,6],[7,8]]],float)
print c
d = reshape(c,(4,2))
print d
```

Python Output

```
[[[ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]
  [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]]

 [[ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]
  [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]]

 [[ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]
  [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j]]]
```

Python Output

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

Python Output

```
[[[ 1.  2.]
  [ 3.  4.]]

 [[ 5.  6.]
  [ 7.  8.]]]
[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]
 [ 7.  8.]]
```

Slicing

```
c = array([[1,2,3,4],[5,6,7,8],[9,0,1,2]])
print c
d = c[1:3,1:4]
print d

e = c[:,1:4:2]
print e

d[1,1] = 123
print c
```

Python Output

```
[[1 2 3 4]
 [5 6 7 8]
 [9 0 1 2]]
[[6 7 8]
 [0 1 2]]
[[2 4]
 [6 8]
 [0 2]]
[[  1   2   3   4]
 [  5   6   7   8]
 [  9   0 123   2]]
```

Universal Functions

Ufuncs apply to each element separately
Equivalent to Fortran 90 elemental functions
Include most of the basic numeric operators
Plus the basic functions from cmath
All can also operate on mixed scalars and arrays

```
a = array([[[1,2],[3,4]],[[5,6],[7,8]]],float)
print power(1.23,a)
print power(a,a)
print a > 5.23
```

Python Output

```
[[[ 1.23       1.5129    ]
  [ 1.860867   2.28886641]]

 [[ 2.81530568  3.46282599]
  [ 4.25927597  5.23890944]]]
[[[  1.00000000e+00   4.00000000e+00]
  [  2.70000000e+01   2.56000000e+02]]

 [[  3.12500000e+03   4.66560000e+04]
  [  8.23543000e+05   1.67772160e+07]]]
```

Python Output

```
[[[False False]
  [False False]]

 [[False True]
  [True True]]]
```

Array Functions Proper

Large number, mainly for manipulation
Array shapes must match, as appropriate
Already mentioned reshape and dot

```
a = array([[1,2],[3,4]])
print transpose(a)
print sort(a)
print trace(a)
```

- Some alias and some copy – watch out!

Python Output

```
[[1 3]
 [2 4]]
[[1 2]
 [3 4]]
5
```

Linear Algebra

Bare minimum features from LAPACK

```
from numpy.linalg import *
a = array([[1,2,3],[4,6,8],[9,6,1]])
b = array([10,11,12])
print det(a)
print inv(a)
print solve(a,b)
print eig(a)
```

Python Output

```
4.0
[[-10.5   4.   -0.5]
 [ 17.   -6.5   1. ]
 [ -7.5   3.   -0.5]]
[ -67.   110.5  -48. ]
(array([ 13.28983218,  -0.05752375,  -5.23230844]),
 array([[-0.26604504, -0.48835717, -0.2366787 ],
        [-0.77502303,  0.79568926, -0.50631919],
        [-0.57320096, -0.35830974,  0.82923101]]))
```

Fast Fourier Transforms

Bare minimum from FFTPACK

```
from numpy.fft import *
a = array([1,2,4,7,0,3,5,8,6,9])
b = fft(a)
print b
print ifft(b)
```

Python Output

```
[ 45.       +0.00000000e+00j  1.30901699 +9.90659258e+00j
 -11.28115295 +2.48989828e+00j  0.19098301 +9.64932244e+0
  -1.21884705 +2.24513988e-01j -13.       +1.33226763e-
-14j
     -1.21884705  -2.24513988e-01j        0.19098301  -
-9.64932244e+00j
   -11.28115295  -2.48989828e+00j        1.30901699  -
-9.90659258e+00j]
[  1.00000000e-00 -8.88178420e-17j   2.00000000e+00 -
-3.64153152e-15j
                   4.00000000e+00        +1.36026581e-
-15j   7.00000000e+00 +1.44435090e-15j
   1.06581410e-15 +5.13552392e-15j   3.00000000e+00 -
-1.89109605e-15j
                   5.00000000e+00        +1.01999232e-
-15j   8.00000000e+00 +1.82590004e-16j
```

Random Numbers

Some facilities from ranlib

```
from numpy.random import *
c = array([[1,2,3],[4,5,6],[7,8,9]])
d = 2*c+1
print uniform(c,d)
print normal(d,c)
```

Python Output

```
[[  2.88170183   3.690092     6.39022939]
 [  4.40842594   8.04713813  11.67611954]
 [  9.67607436  12.55491236  11.83973885]]
[[  4.28898176   5.57995933   6.64778541]
 [  9.01649799  15.84265958  14.7575158 ]
 [ 17.85013703  29.66551119   3.99543002]]
```

f2py

f2py is a numpy command to call Fortran
Looks useful for pure Fortran90 interfaces
Unreliable and tricky for Fortran90 ones

I haven't been able to make it work
It claims to support NAG f95, but doesn't
Even more environment–dependent than numpy

It may be worth trying with different compilers
Please tell me if you have any success

Installation

Packages available for some systems
Installed on PWF, as you found
Also available for Microsoft Windows

Seems to compile fairly easily under Linux

- Version 0.9.6 for Python 2.4 and 1.0.1 for 2.5

Library details built into setup mechanism

- Supports only specific libraries, not ACML

Porting the build mechanism could be hard

Documentation (1)

Code is open source, documentation isn't
$40 (electronic) from http://www.tramy.us/

Numpy is close to Numeric, but with changes
http://numpy.scipy.org//numpydoc/numdoc.htm

There is an online course for NumPy/SciPy:
http://www.rexx.com/~dkuhlman/...
.../scipy_course_01.html

Documentation (2)

Can use dir in Python to see what's there

```
import numpy
import numpy.linalg
print dir(numpy.linalg)
```

Not always easy to work out specification

```
x = numpy.matrix([[2,0],[0,–3]])
numpy.invert(x)
numpy.linalg.inv(x)
```

Extending numpy is for experts only

Python Output

```
['LinAlgError', ..., 'det', ..., 'test']
[[-3 -1]
 [-1  2]]
[[ 0.5         0.        ]
 [ 0.         -0.33333333]]
```

Exception Handling

Simple tests indicate that it is fairly robust
Against crashing – NOT other errors
Not even Python's standard numeric handling

```
b = array([123456789])
print b*b*b
print b/0
c = array([1.0e200])
print c*c
```

Python Output

```
Warning: divide by zero encountered in
    divide
Warning: overflow encountered in multiply
[–22041936616612446 27]
[0]
[             inf]
```

Usage Errors

Many errors are diagnosed:

```
a = array([[1,2,3],[4,5,6]])
print dot(a,a)
```

Many more just do weird things:

```
print trace(a)
b = array([[[1,2],[3,4]],[[5,6],[7,8]]])
print transpose(b)
```

Python Output

```
Traceback (most recent call last):
  File "Demos/demo_44a.py", line 3, in
                    <module>
    print dot(a,a)
ValueError: objects are not aligned

6
[[[1 5]
  [3 7]]

 [[2 6]
  [4 8]]]
```

What Can We Do?

Remember about complex exceptions?
Exactly the same applies here!

- Put in plenty of sanity checks

NEVER rely on automatic diagnostics

This applies WHATEVER language you use!
Similar problems apply to all of them
Some libraries (e.g. NAG) better than others