Fortran in Python

based on "Python Scripting For Computational Science"
by Hans Petter Langtangen

Mehdi Rezaie
Ohio University
Fall 2015

Motivations

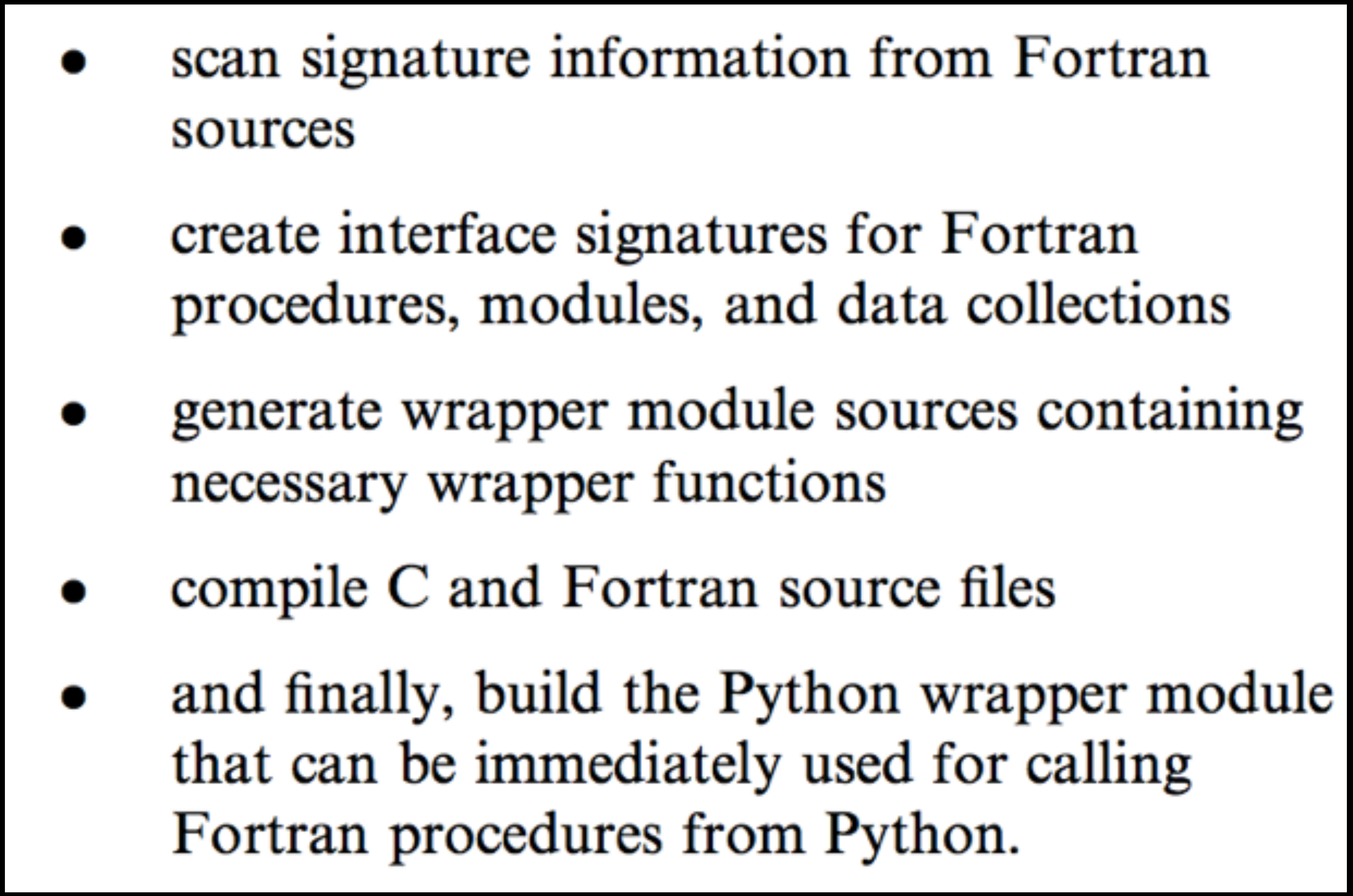
Python (dynamic)
- Pleasure
- Slow (eg. loops)
- More memory usage
- Great packages (eg. Scipy, Numpy, Matplotlib)

Static languages (e.g. Fortran, C/C++)
- Fast
- More control on memory usage
- Rich modules

Call out a fast machine-code routine (compiled C/C++/Fortran) from Python

How F2Py does that.

1. Converts Python objects to objects being used as arguments for Fortran procedures
2. Call the Fortran procedure
3. Re-converts the returned values and changed arguments



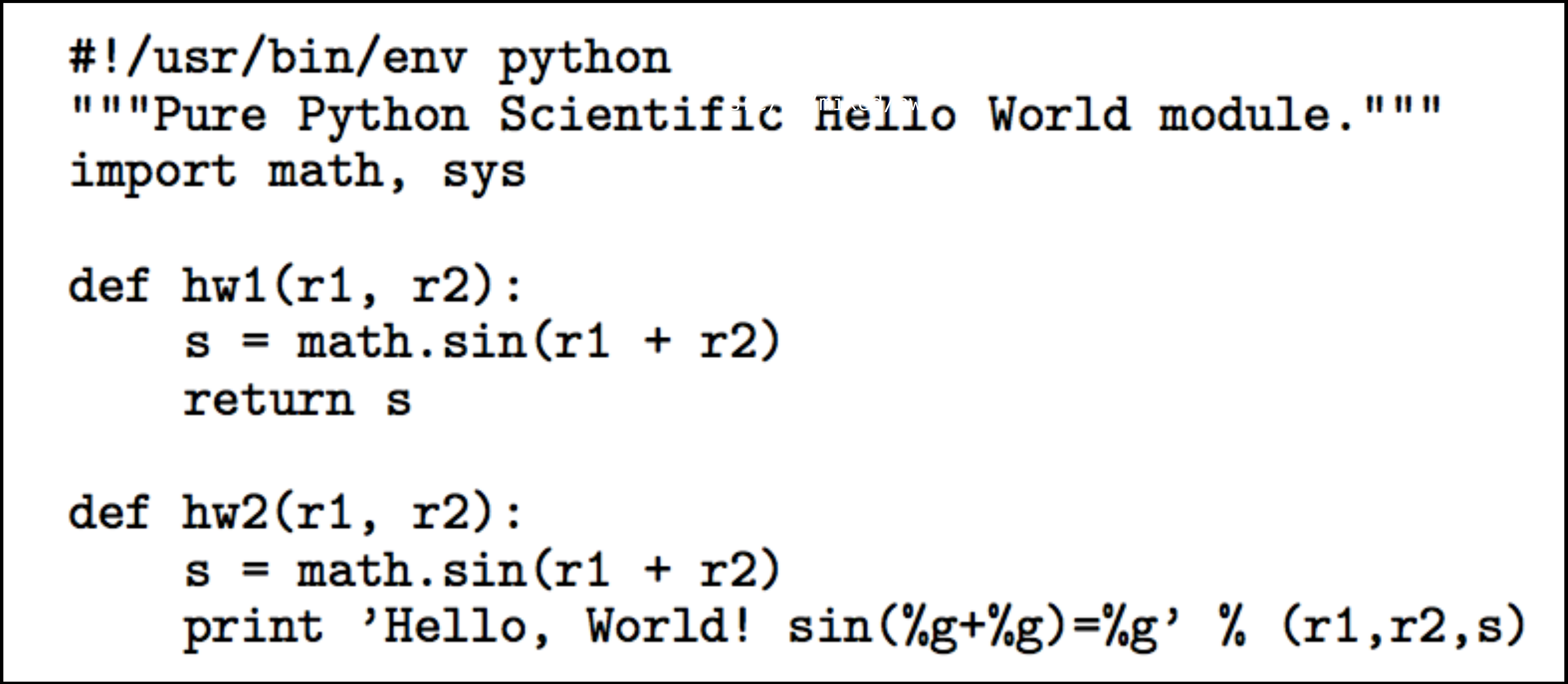
- scan signature information from Fortran sources
- create interface signatures for Fortran procedures, modules, and data collections
- generate wrapper module sources containing necessary wrapper functions
- compile C and Fortran source files
- and finally, build the Python wrapper module that can be immediately used for calling Fortran procedures from Python.

GOTO Ex. 2

(c) Peterson

Cut to the chase: Ex. 1

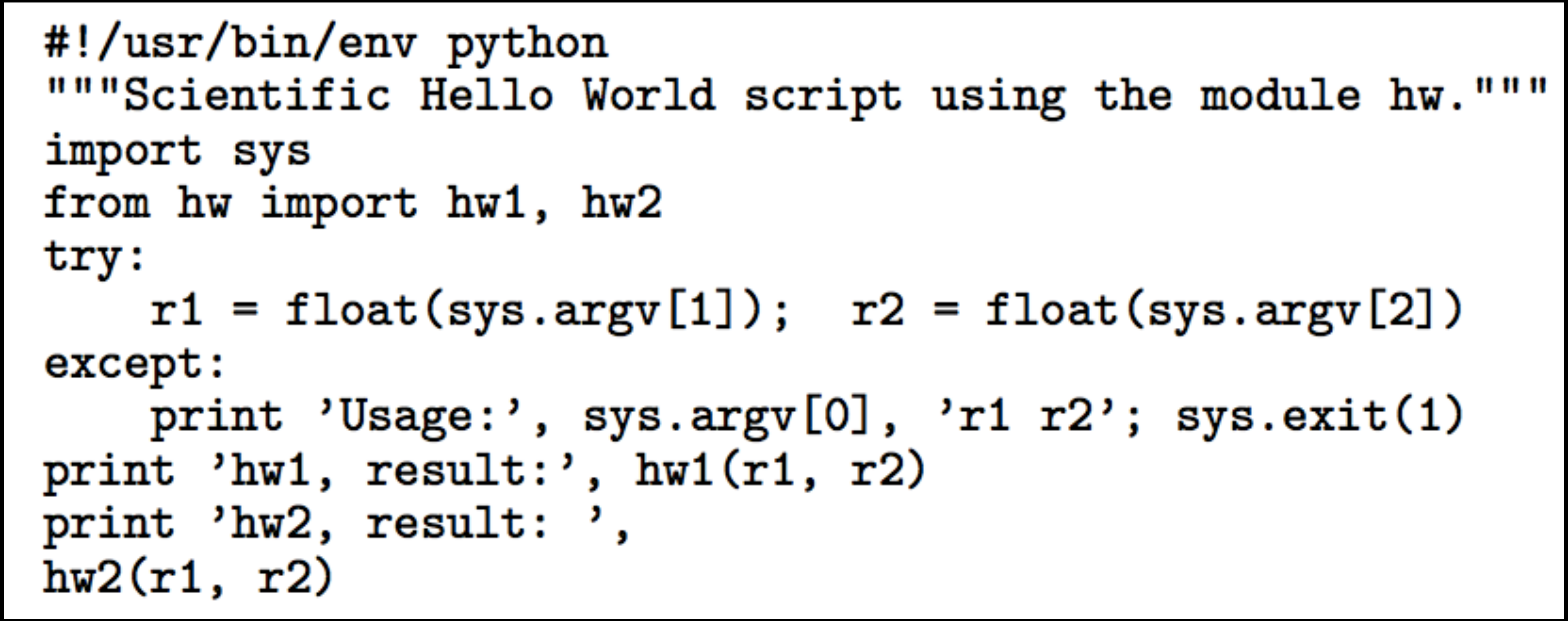
Extended “Hello World” (hw.py)
Find in: `src/py/mixed/hw`



```python
#!/usr/bin/env python
"""Pure Python Scientific Hello World module."""
import math, sys

def hw1(r1, r2):
    s = math.sin(r1 + r2)
    return s

def hw2(r1, r2):
    s = math.sin(r1 + r2)
    print 'Hello, World! sin(%g+%g)=%g' % (r1,r2,s)
```

Ex. 1

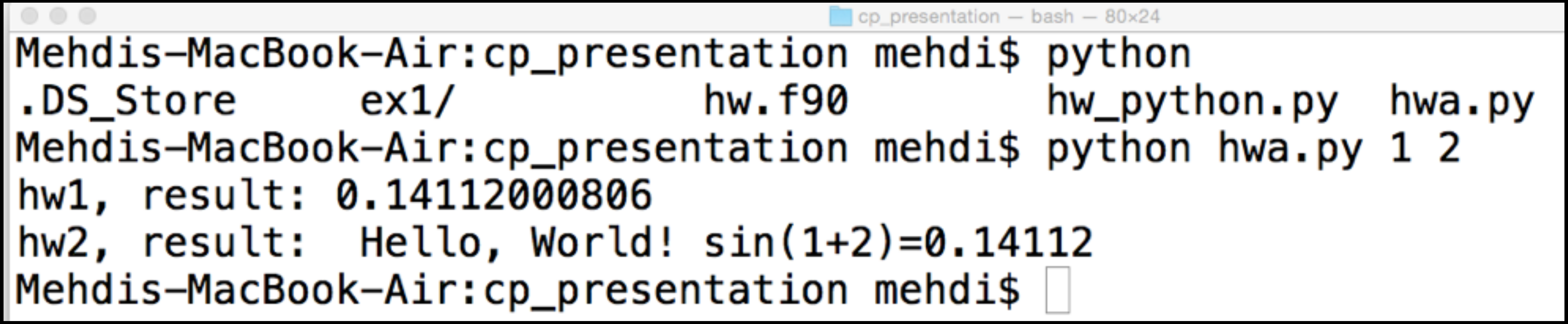
Script (hwa.py) to use hw.py



```
#!/usr/bin/env python
"""Scientific Hello World script using the module hw."""
import sys
from hw import hw1, hw2
try:
    r1 = float(sys.argv[1]);  r2 = float(sys.argv[2])
except:
    print ’Usage:’, sys.argv[0], ’r1 r2’; sys.exit(1)
print ’hw1, result:’, hw1(r1, r2)
print ’hw2, result: ’,
hw2(r1, r2)
```

Ex. 1

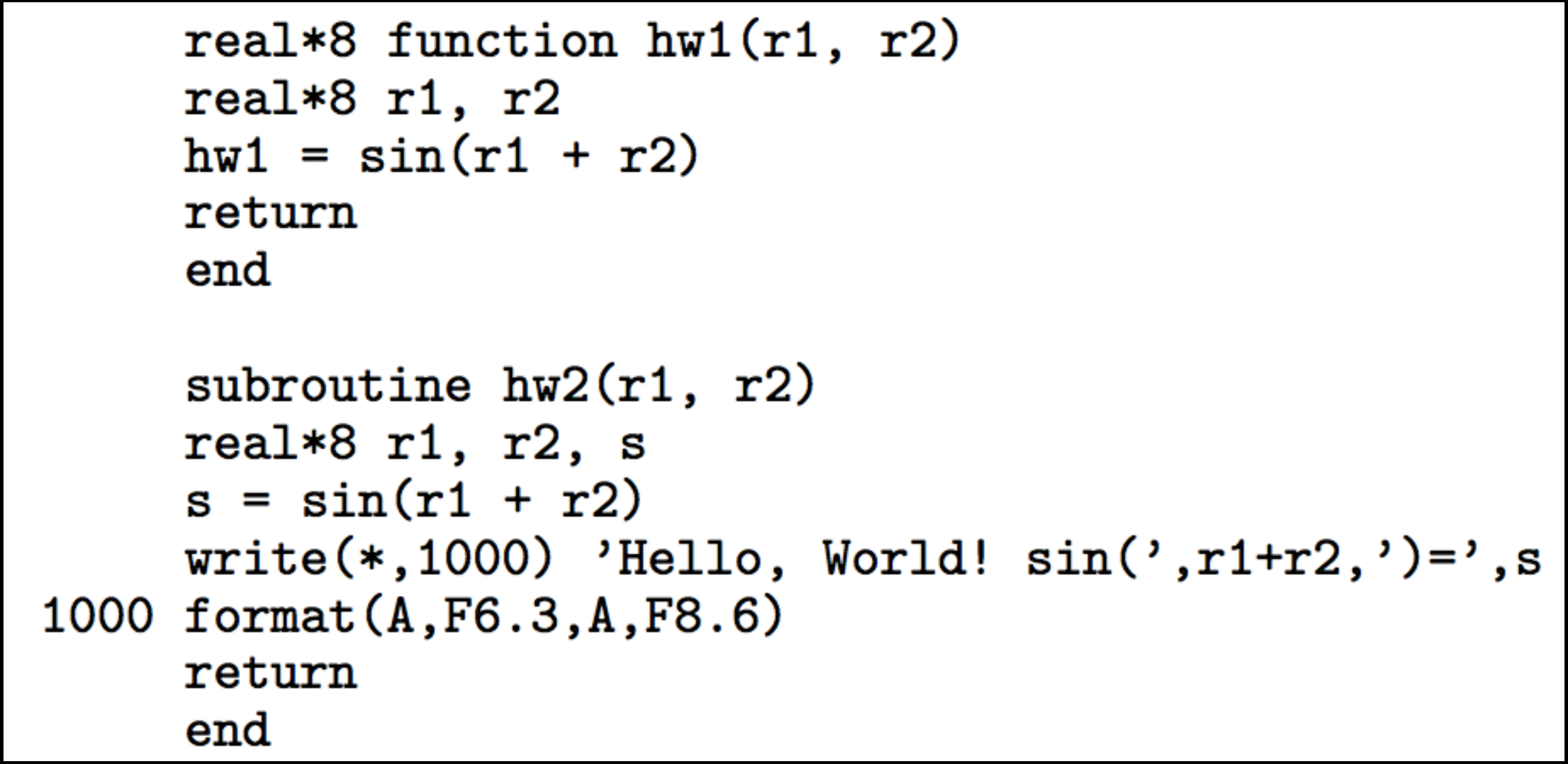
Output of Python



```
Mehdis-MacBook-Air:cp_presentation mehdi$ python
.DS_Store     ex1/           hw.f90         hw_python.py  hwa.py
Mehdis-MacBook-Air:cp_presentation mehdi$ python hwa.py 1 2
hw1, result: 0.14112000806
hw2, result:  Hello, World! sin(1+2)=0.14112
Mehdis-MacBook-Air:cp_presentation mehdi$ 
```

Ex. 1

Fortran version (hw.f90) of hw.py



```
      real*8 function hw1(r1, r2)
      real*8 r1, r2
      hw1 = sin(r1 + r2)
      return
      end

      subroutine hw2(r1, r2)
      real*8 r1, r2, s
      s = sin(r1 + r2)
      write(*,1000) 'Hello, World! sin(',r1+r2,')=',s
 1000 format(A,F6.3,A,F8.6)
      return
      end
```

Ex. 1

Preparing hw.f90 for our Python script by

```
F2py -m hw -c hw.f90
F2py -m <modulename> -c <fortranfile>
```

output: hw.so

Run:

```
python hwa.py <arg.1> <arg.2>
```

Some More

Specifying the compiler

```
-- fcompiler= ‘Gnu’
```

To see available F.Compiler on your system

```
f2py -c --help-compiler
```

Dealing with sophisticated libraries

```
only: <functions> :
```

```
f2py -m hw -c --fcompiler=’Gnu’ hw.f90 only: hw1 hw2 :
```

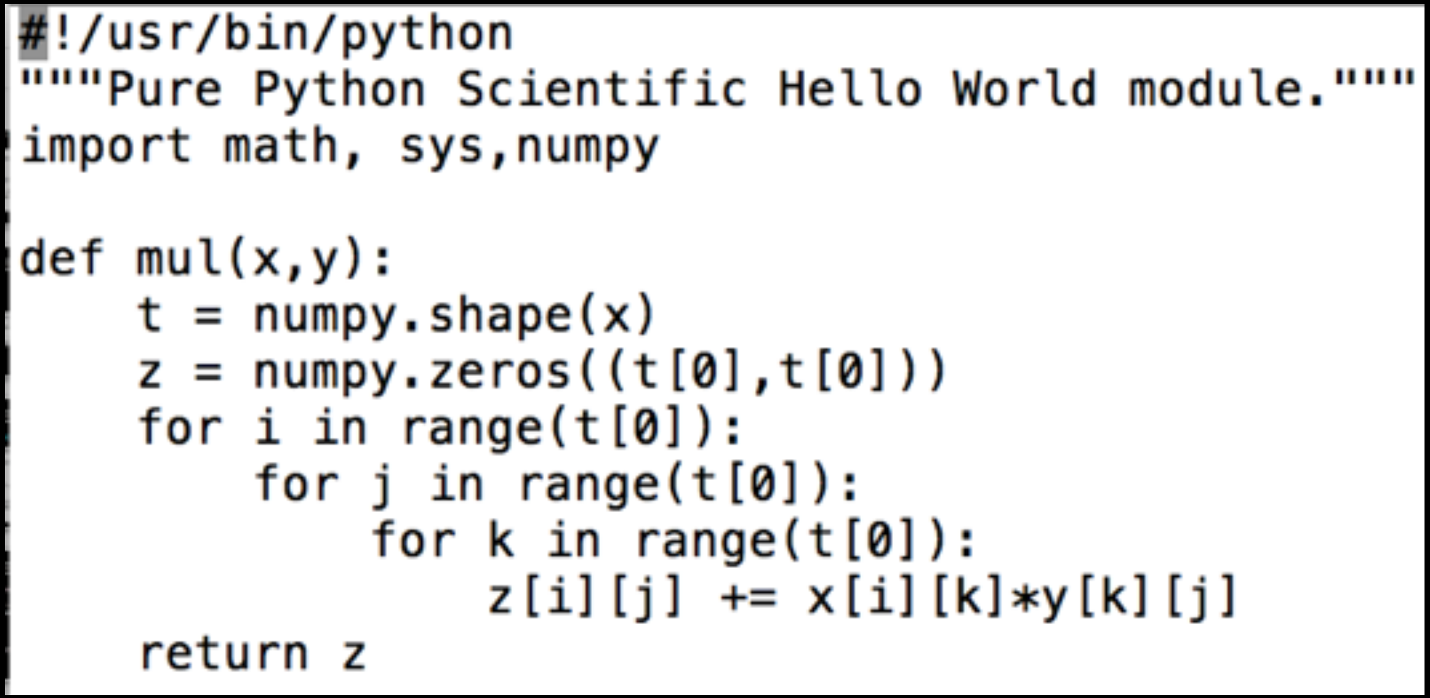
EX. 2

EX. 2: Matrix multiplication using:

1. Pure Python
2. Python using Fortran routine
3. Python using **`”Numpy.dot”`**

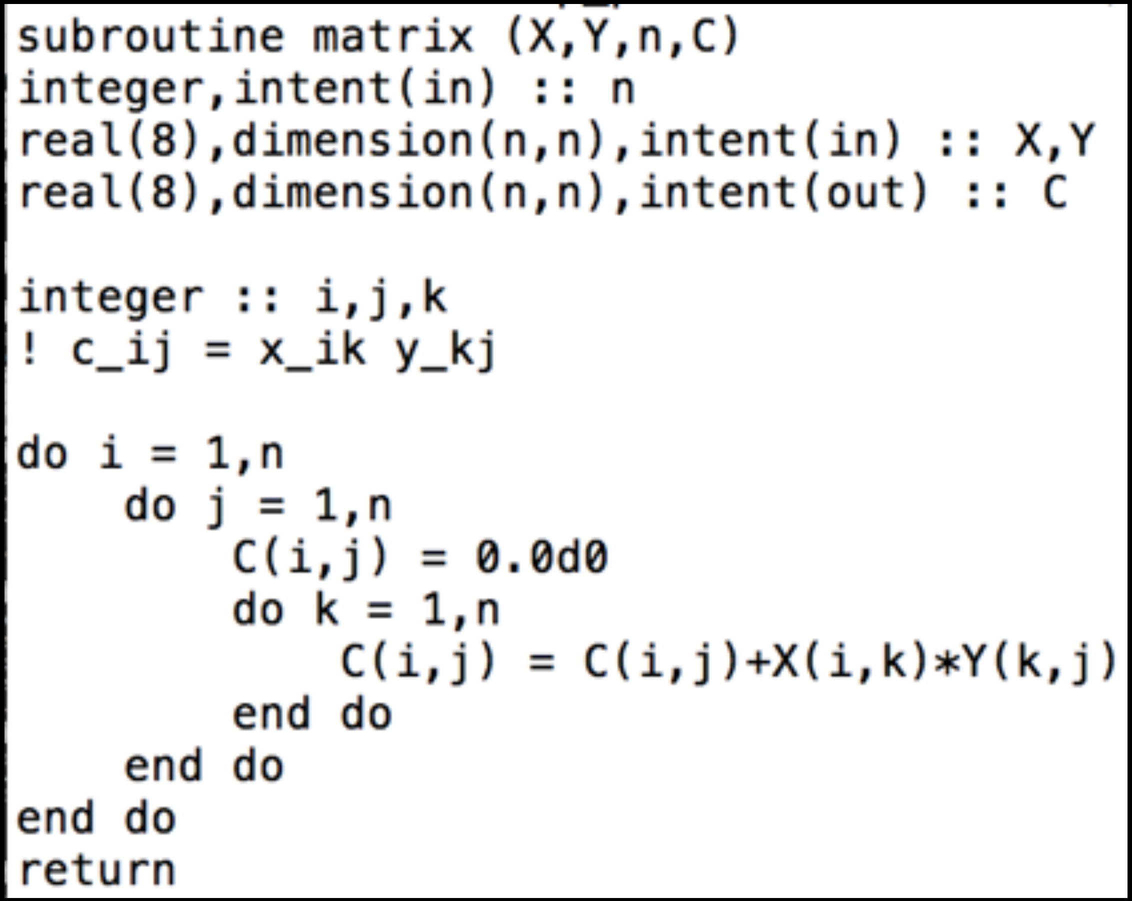
Routines

hw_python.py



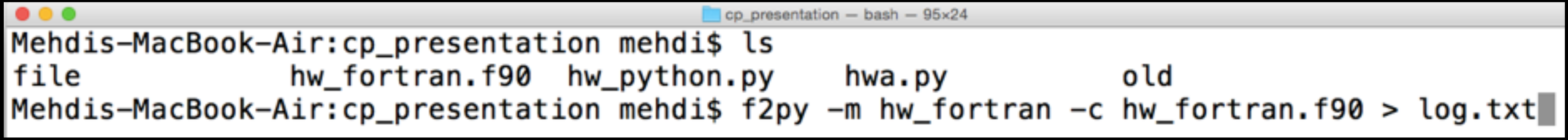
```
#!/usr/bin/python
"""Pure Python Scientific Hello World module."""
import math, sys,numpy

def mul(x,y):
    t = numpy.shape(x)
    z = numpy.zeros((t[0],t[0]))
    for i in range(t[0]):
        for j in range(t[0]):
            for k in range(t[0]):
                z[i][j] += x[i][k]*y[k][j]
    return z
```

hw_fortran.f90



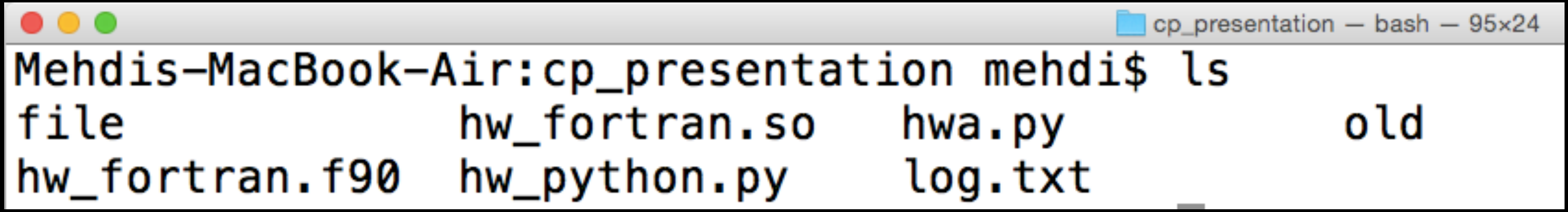
```
subroutine matrix (X,Y,n,C)
integer,intent(in) :: n
real(8),dimension(n,n),intent(in) :: X,Y
real(8),dimension(n,n),intent(out) :: C

integer :: i,j,k
! c_ij = x_ik y_kj

do i = 1,n
    do j = 1,n
        C(i,j) = 0.0d0
        do k = 1,n
            C(i,j) = C(i,j)+X(i,k)*Y(k,j)
        end do
    end do
end do
return
```

```
f2py -m hw_fortran -c hw_fortran.f90 > log.txt
```



```
Mehdis-MacBook-Air:cp_presentation mehdi$ ls
file              hw_fortran.f90  hw_python.py    hwa.py          old
Mehdis-MacBook-Air:cp_presentation mehdi$ f2py -m hw_fortran -c hw_fortran.f90 > log.txt
```

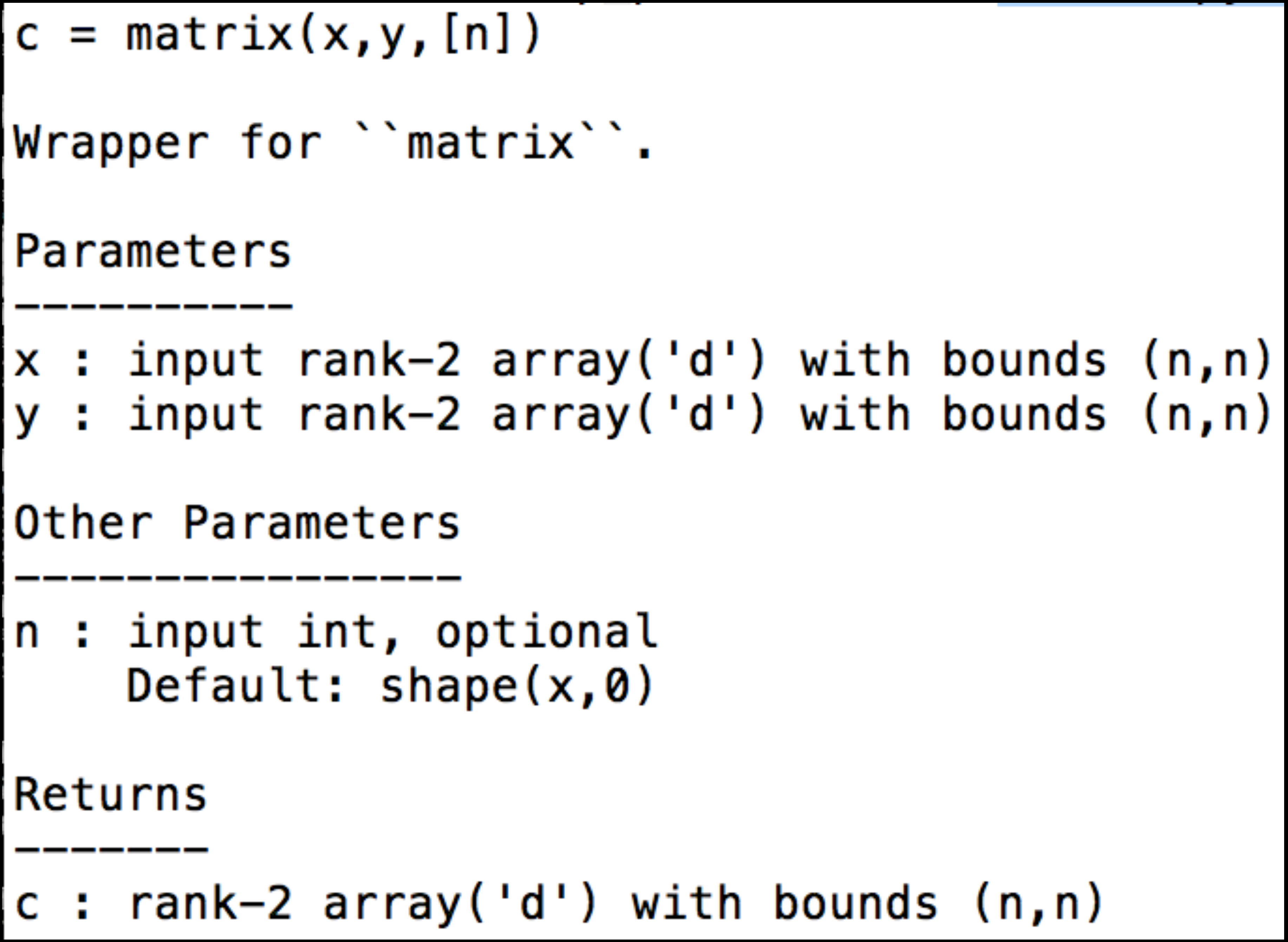
```
ls
```



```
Mehdis-MacBook-Air:cp_presentation mehdi$ ls
file              hw_fortran.so   hwa.py          old
hw_fortran.f90    hw_python.py    log.txt
```

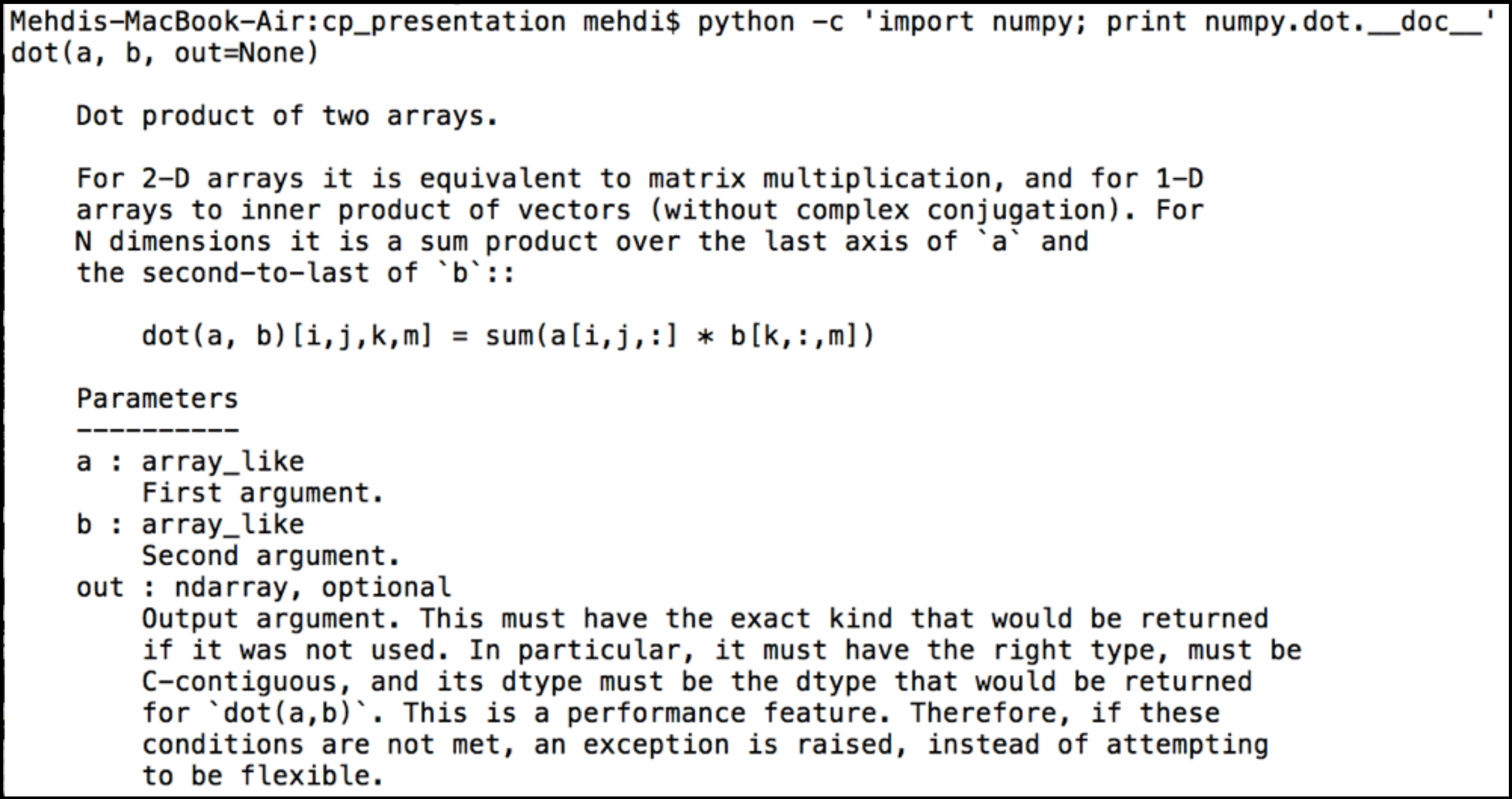
`hw_fortran.so` is the compiled extension module that is going to call the fortran routine.

```
python -c 'import hw_fortran; print hw_fortran.matrix.__doc__'
```



```
c = matrix(x,y,[n])

Wrapper for ``matrix``.

Parameters
----------
x : input rank-2 array('d') with bounds (n,n)
y : input rank-2 array('d') with bounds (n,n)

Other Parameters
----------------
n : input int, optional
    Default: shape(x,0)

Returns
-------
c : rank-2 array('d') with bounds (n,n)
```

Numpy.dot

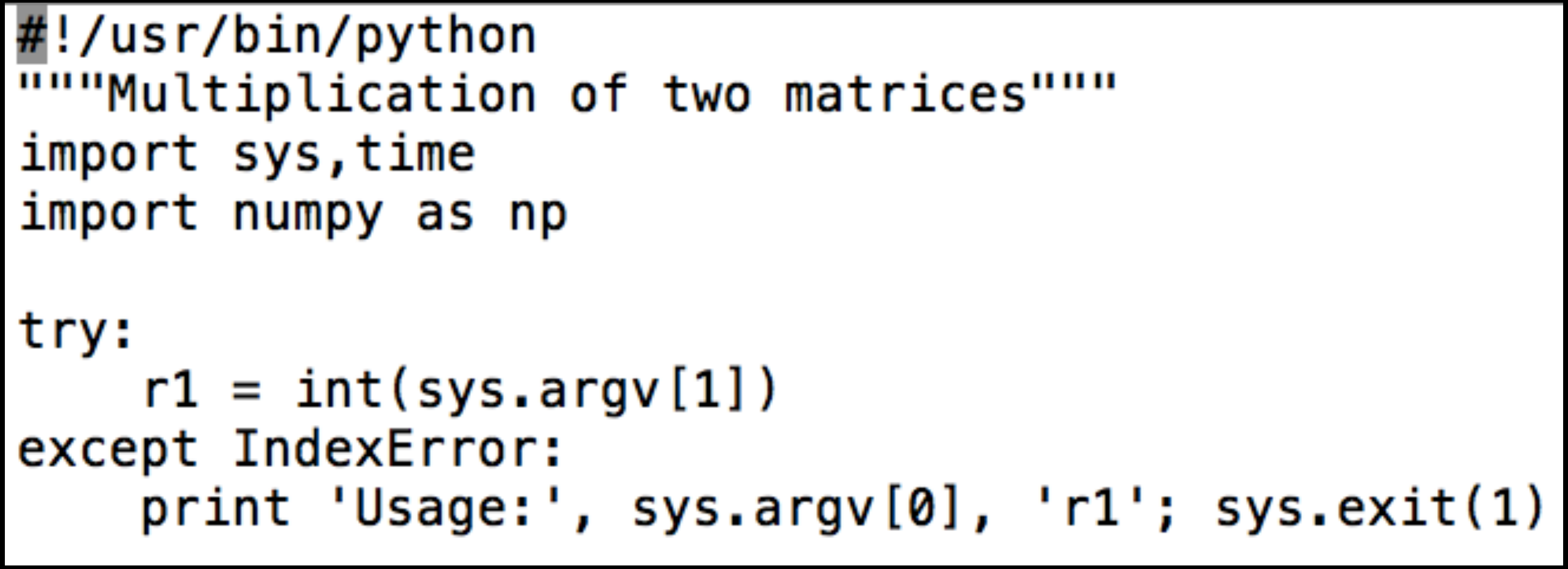


```
Mehdis-MacBook-Air:cp_presentation mehdi$ python -c 'import numpy; print numpy.dot.__doc__'
dot(a, b, out=None)

    Dot product of two arrays.

    For 2-D arrays it is equivalent to matrix multiplication, and for 1-D
    arrays to inner product of vectors (without complex conjugation). For
    N dimensions it is a sum product over the last axis of `a` and
    the second-to-last of `b`::

        dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])

    Parameters
    ----------
    a : array_like
        First argument.
    b : array_like
        Second argument.
    out : ndarray, optional
        Output argument. This must have the exact kind that would be returned
        if it was not used. In particular, it must have the right type, must be
        C-contiguous, and its dtype must be the dtype that would be returned
        for `dot(a,b)`. This is a performance feature. Therefore, if these
        conditions are not met, an exception is raised, instead of attempting
        to be flexible.
```

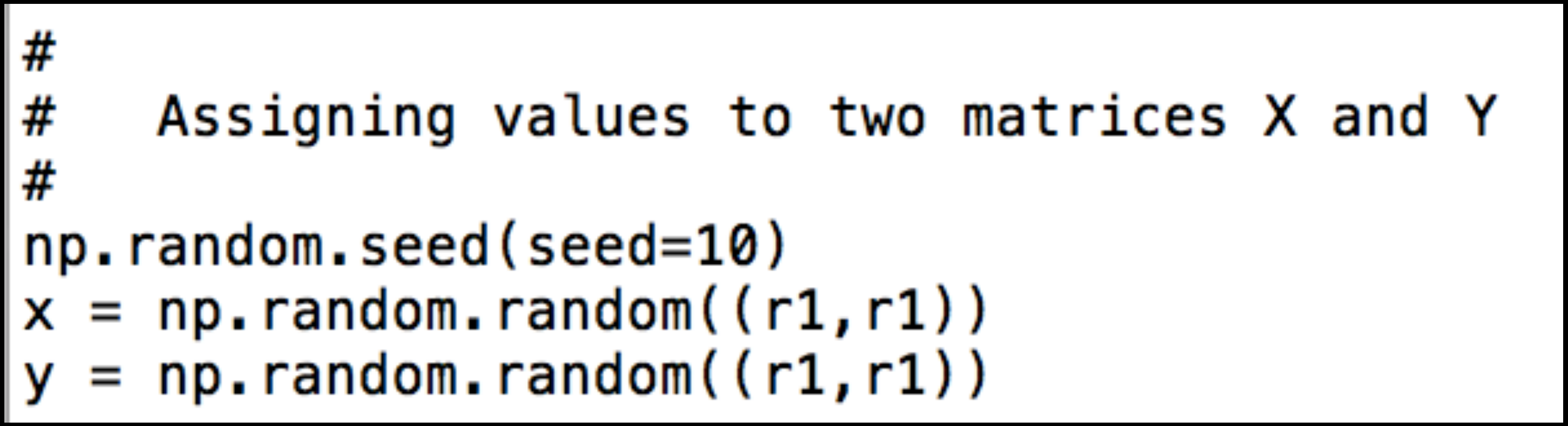
Let’s compare!

`hwa.py` is the driver script which

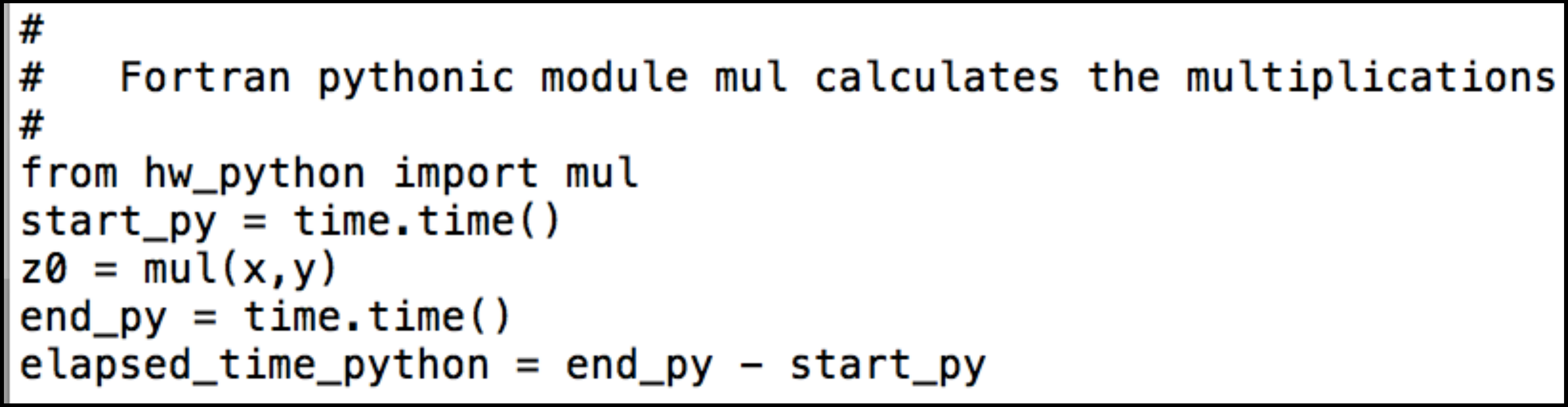
1. Gets one argument as the dimension (r1 = D)
2. Generates two random matrices (D*D)
3. Multiplies two matrices using three aforementioned approaches



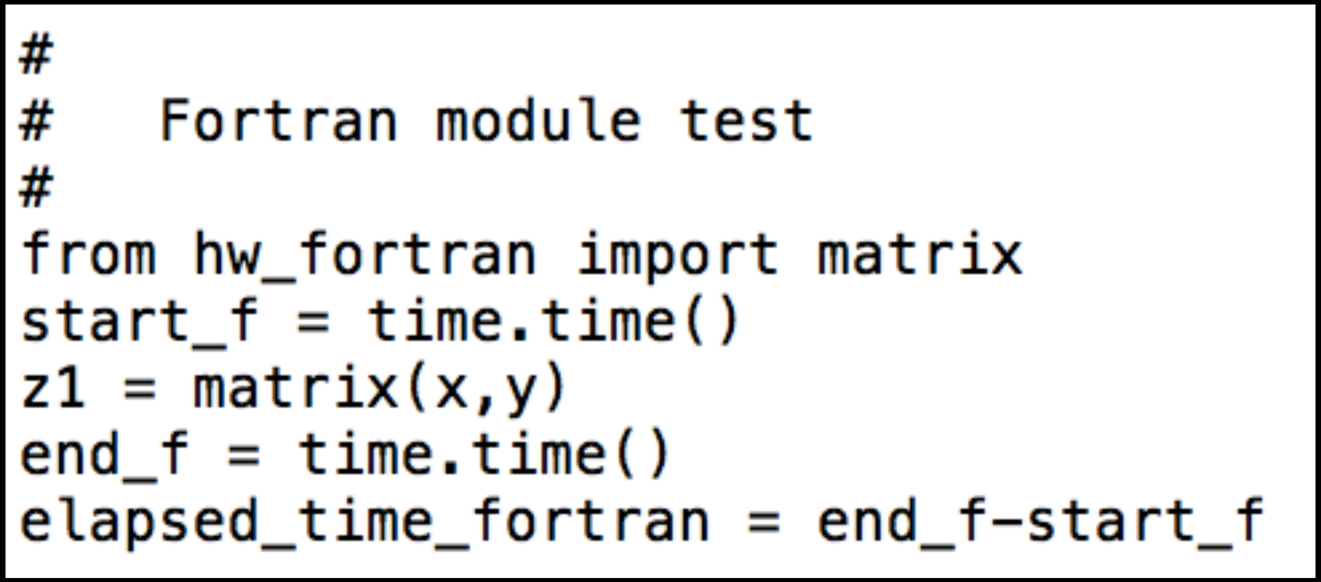
```python
#!/usr/bin/python
"""Multiplication of two matrices"""
import sys,time
import numpy as np

try:
    r1 = int(sys.argv[1])
except IndexError:
    print 'Usage:', sys.argv[0], 'r1'; sys.exit(1)
```



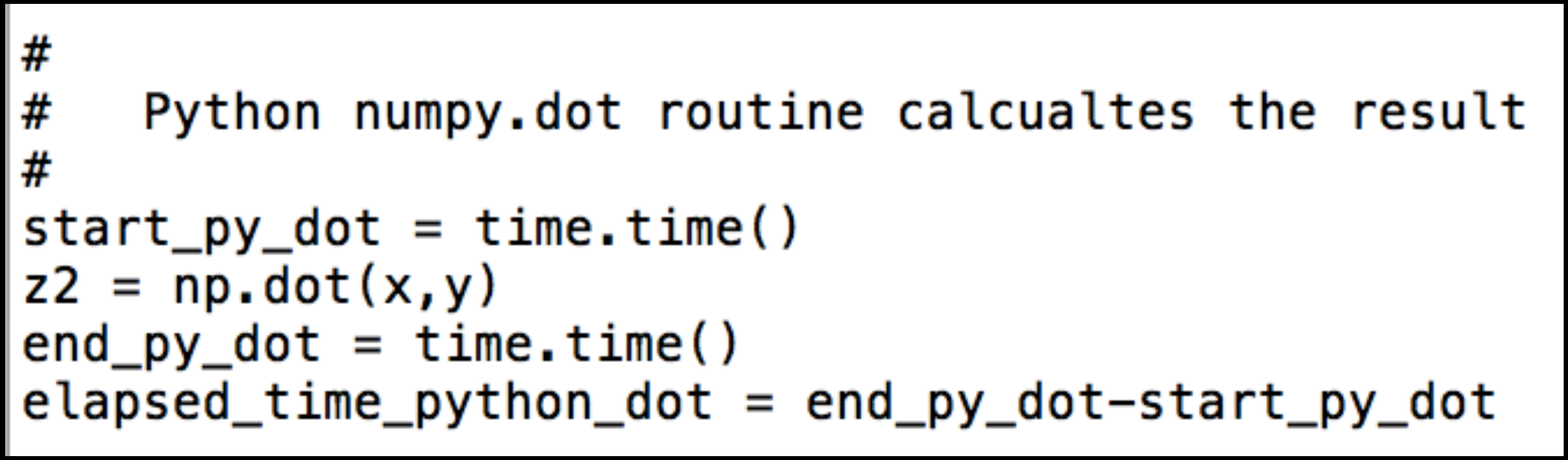
```python
#
#   Assigning values to two matrices X and Y
#
np.random.seed(seed=10)
x = np.random.random((r1,r1))
y = np.random.random((r1,r1))
```



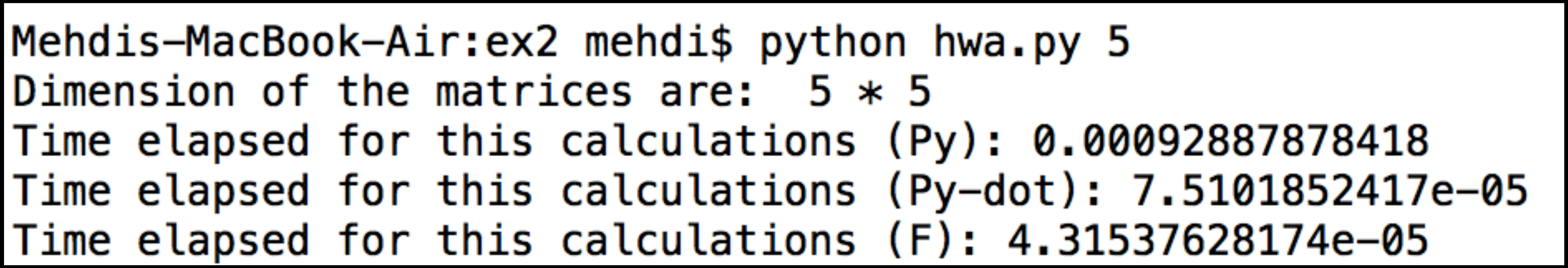
```
#
#   Fortran pythonic module mul calculates the multiplications
#
from hw_python import mul
start_py = time.time()
z0 = mul(x,y)
end_py = time.time()
elapsed_time_python = end_py - start_py
```



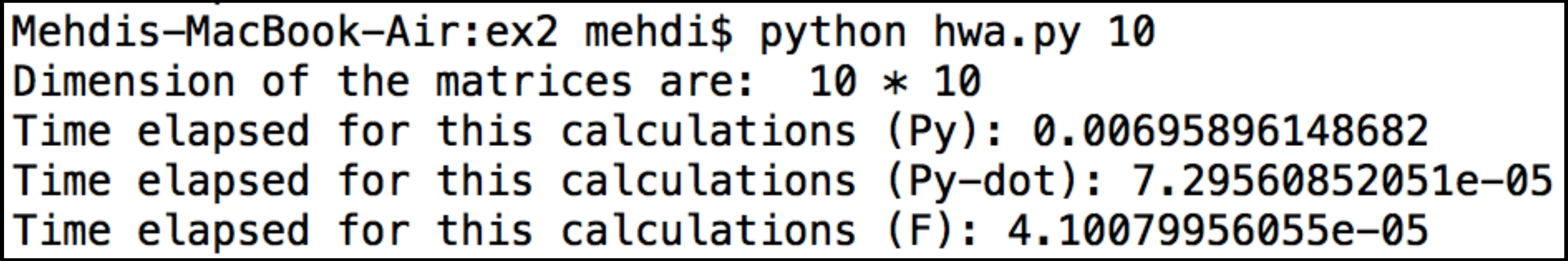
```
#
#   Fortran module test
#
from hw_fortran import matrix
start_f = time.time()
z1 = matrix(x,y)
end_f = time.time()
elapsed_time_fortran = end_f-start_f
```



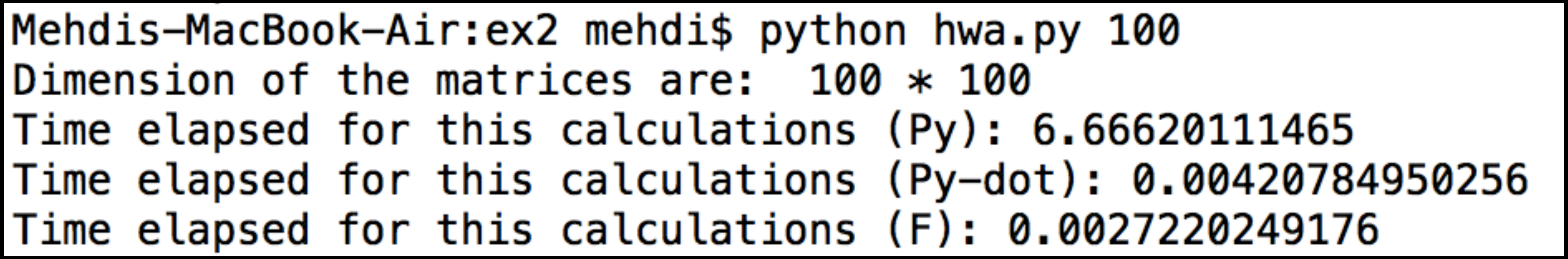
```
#
#   Python numpy.dot routine calcualtes the result
#
start_py_dot = time.time()
z2 = np.dot(x,y)
end_py_dot = time.time()
elapsed_time_python_dot = end_py_dot-start_py_dot
```



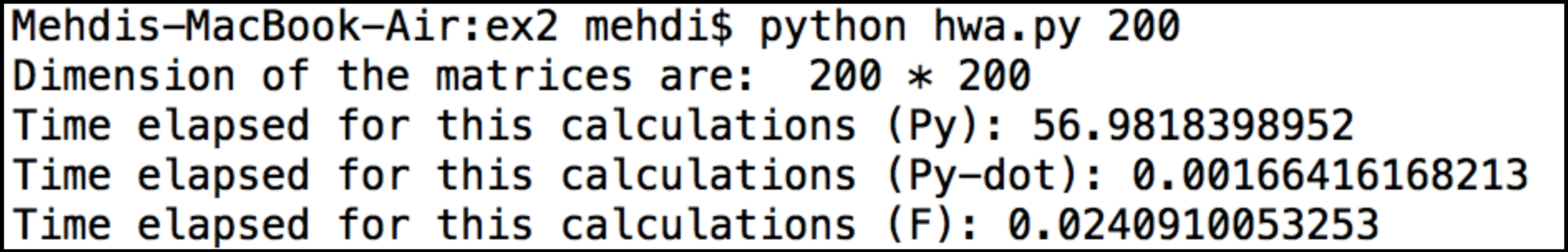
```
Mehdis-MacBook-Air:ex2 mehdi$ python hwa.py 5
Dimension of the matrices are:  5 * 5
Time elapsed for this calculations (Py): 0.00092887878418
Time elapsed for this calculations (Py-dot): 7.5101852417e-05
Time elapsed for this calculations (F): 4.31537628174e-05
```



```
Mehdis-MacBook-Air:ex2 mehdi$ python hwa.py 10
Dimension of the matrices are:  10 * 10
Time elapsed for this calculations (Py): 0.00695896148682
Time elapsed for this calculations (Py-dot): 7.29560852051e-05
Time elapsed for this calculations (F): 4.10079956055e-05
```



```
Mehdis-MacBook-Air:ex2 mehdi$ python hwa.py 100
Dimension of the matrices are:  100 * 100
Time elapsed for this calculations (Py): 6.66620111465
Time elapsed for this calculations (Py-dot): 0.00420784950256
Time elapsed for this calculations (F): 0.0027220249176
```



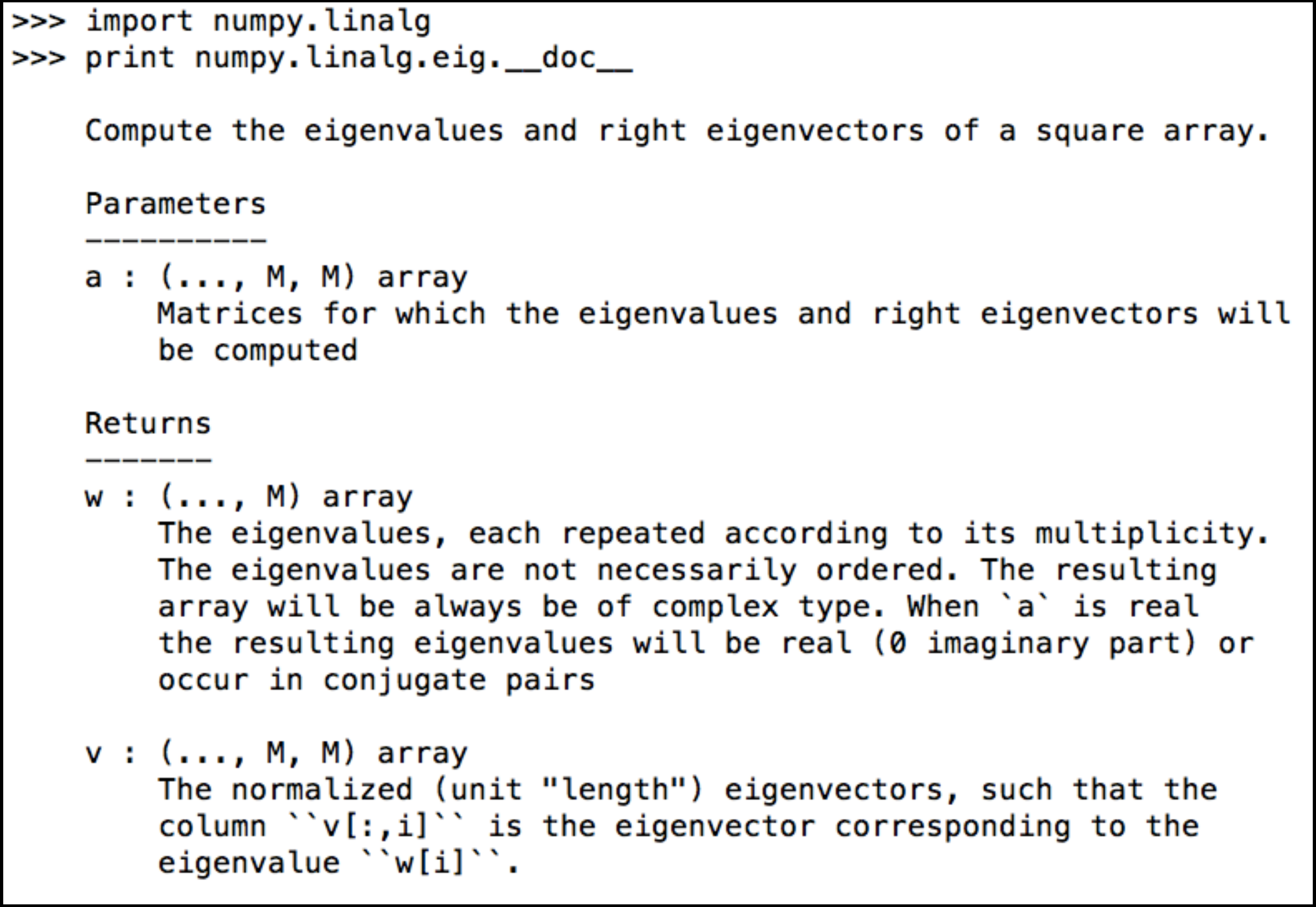
```
Mehdis-MacBook-Air:ex2 mehdi$ python hwa.py 200
Dimension of the matrices are:  200 * 200
Time elapsed for this calculations (Py): 56.9818398952
Time elapsed for this calculations (Py-dot): 0.0016641616821З
Time elapsed for this calculations (F): 0.024091005325З
```

EX. 3

EX. 3: Matrix diagonalization using:

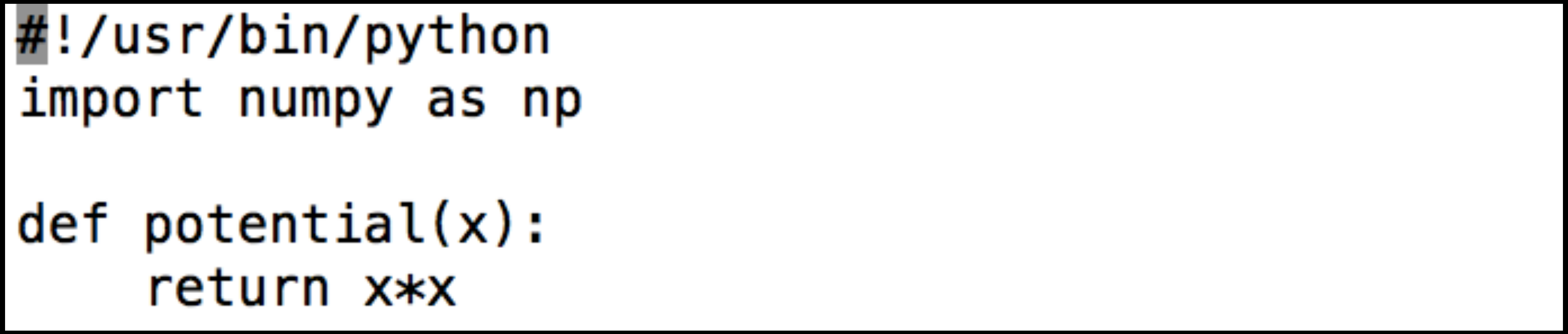
1. Python using **`Numpy.linalg.eig`**
2. Python using Fortran routine **`dsyev.f`**

Numpy.linalg.eig

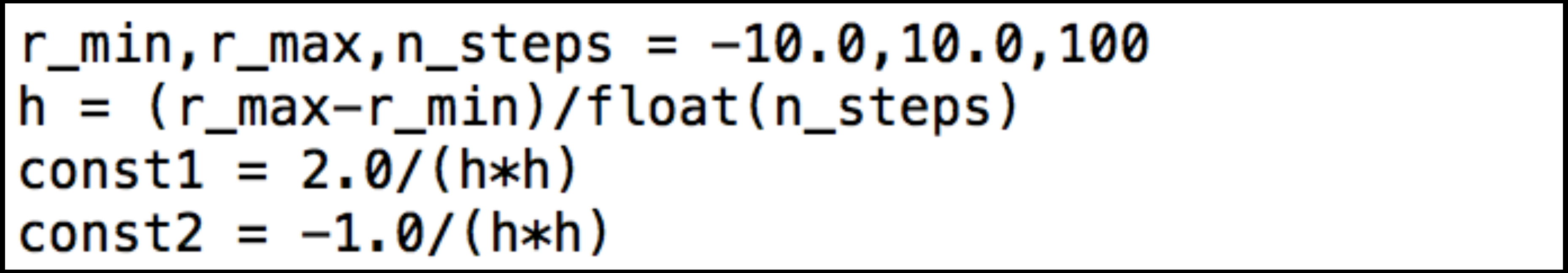


```
>>> import numpy.linalg
>>> print numpy.linalg.eig.__doc__

    Compute the eigenvalues and right eigenvectors of a square array.

    Parameters
    ----------
    a : (..., M, M) array
        Matrices for which the eigenvalues and right eigenvectors will
        be computed

    Returns
    -------
    w : (..., M) array
        The eigenvalues, each repeated according to its multiplicity.
        The eigenvalues are not necessarily ordered. The resulting
        array will be always be of complex type. When `a` is real
        the resulting eigenvalues will be real (0 imaginary part) or
        occur in conjugate pairs

    v : (..., M, M) array
        The normalized (unit "length") eigenvectors, such that the
        column ``v[:,i]`` is the eigenvector corresponding to the
        eigenvalue ``w[i]``.
```

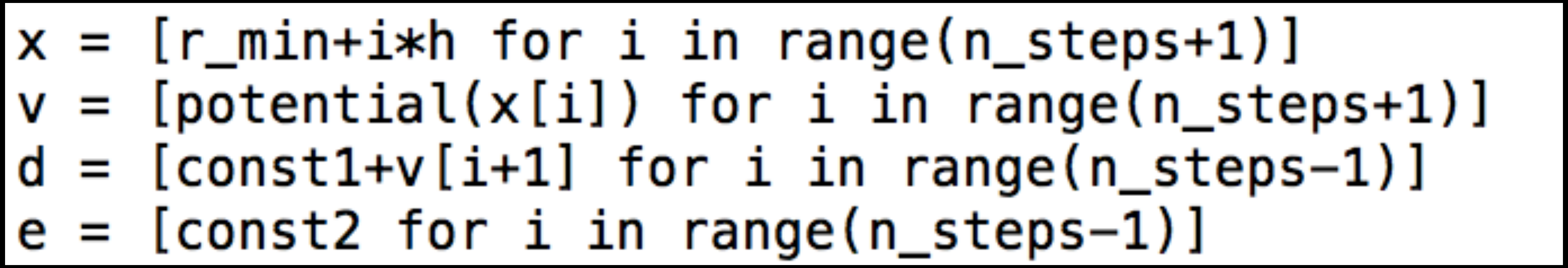
test.py



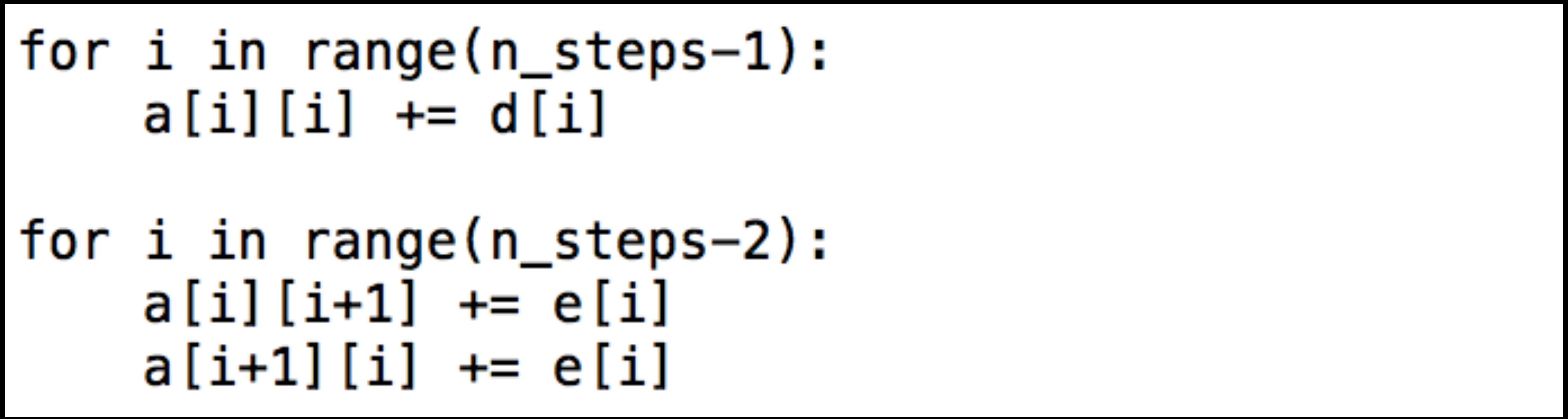
```python
#!/usr/bin/python
import numpy as np

def potential(x):
    return x*x
```



```python
r_min,r_max,n_steps = -10.0,10.0,100
h = (r_max-r_min)/float(n_steps)
const1 = 2.0/(h*h)
const2 = -1.0/(h*h)
```

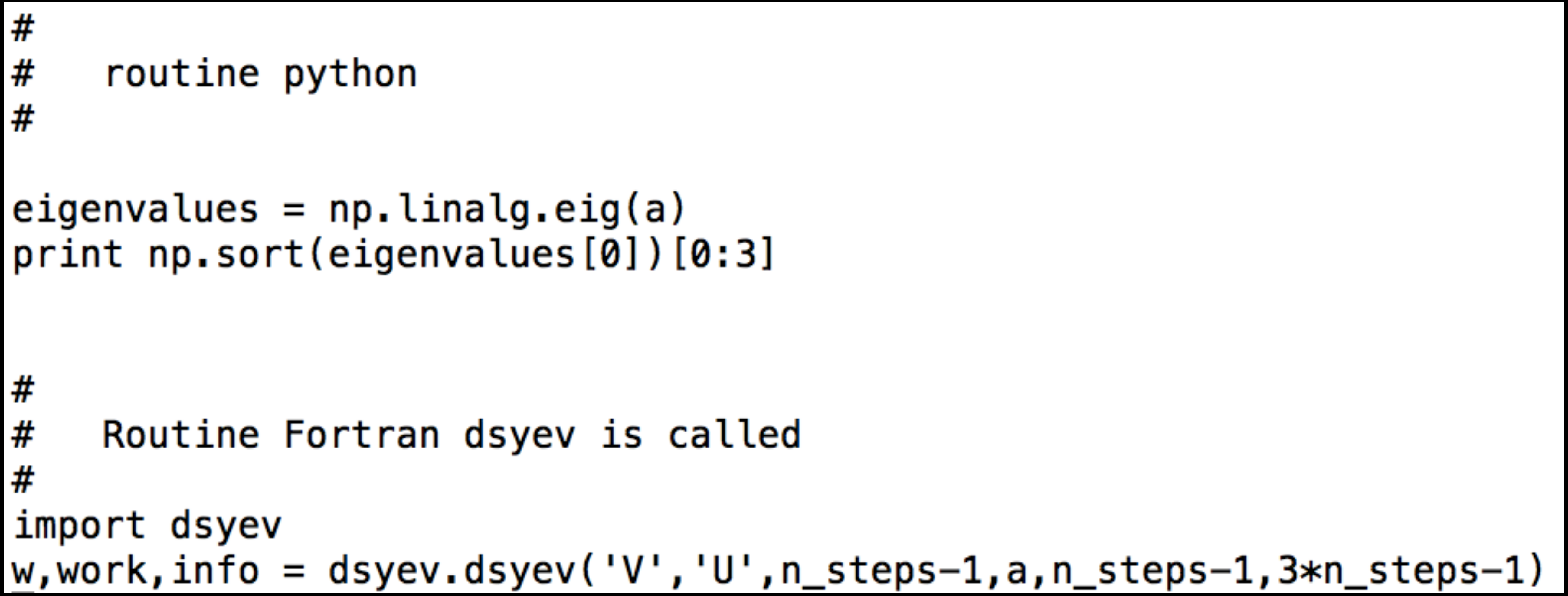


```python
x = [r_min+i*h for i in range(n_steps+1)]
v = [potential(x[i]) for i in range(n_steps+1)]
d = [const1+v[i+1] for i in range(n_steps-1)]
e = [const2 for i in range(n_steps-1)]
```



```python
for i in range(n_steps-1):
    a[i][i] += d[i]

for i in range(n_steps-2):
    a[i][i+1] += e[i]
    a[i+1][i] += e[i]
```

test.py



```
#
#   routine python
#

eigenvalues = np.linalg.eig(a)
print np.sort(eigenvalues[0])[0:3]



#
#   Routine Fortran dsyev is called
#
import dsyev
w,work,info = dsyev.dsyev('V','U',n_steps-1,a,n_steps-1,3*n_steps-1)
```

Building signature file:

```
f2py dsyev.f -m dsyev -h dsyef.pyf -llapack
```

Revising signature file, by adding:

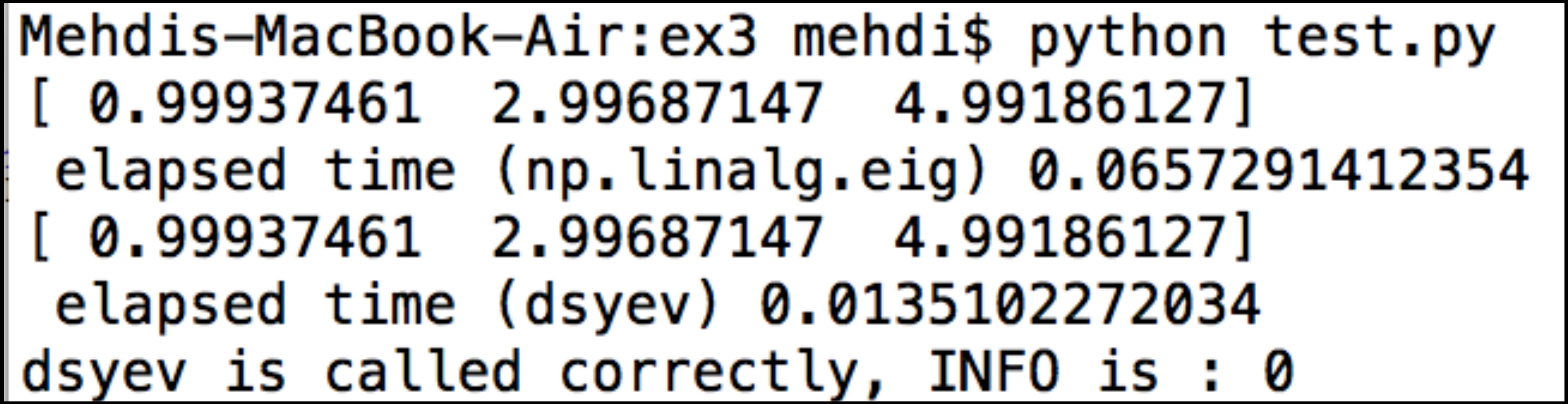
```
intent(in), intent(out), depend(n,n), …
```

Building the extension module:

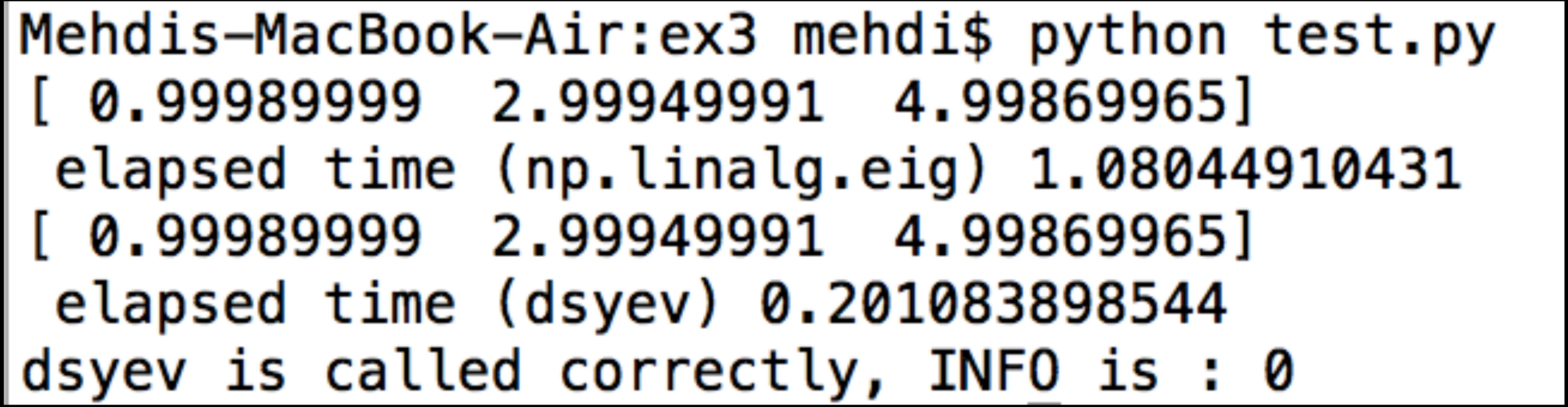
```
f2py -c dsyev.pyf dsyev.f -llapack
```

Bazinga! dsyev.so is ready to be imported in
Python!

200 vs. 500 steps



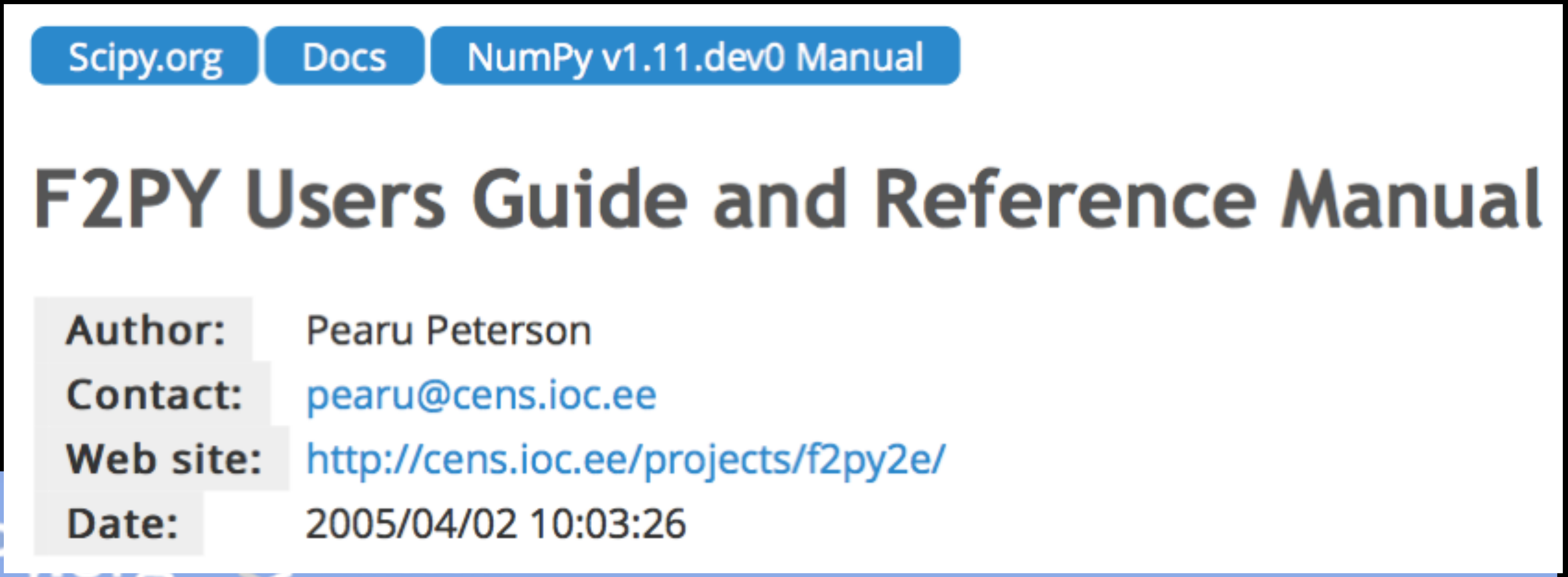
```
Mehdis-MacBook-Air:ex3 mehdi$ python test.py
[ 0.99937461  2.99687147  4.99186127]
 elapsed time (np.linalg.eig) 0.0657291412354
[ 0.99937461  2.99687147  4.99186127]
 elapsed time (dsyev) 0.0135102272034
dsyev is called correctly, INFO is : 0
```



```
Mehdis-MacBook-Air:ex3 mehdi$ python test.py
[ 0.99989999  2.99949991  4.99869965]
 elapsed time (np.linalg.eig) 1.08044910431
[ 0.99989999  2.99949991  4.99869965]
 elapsed time (dsyev) 0.201083898544
dsyev is called correctly, INFO is : 0
```

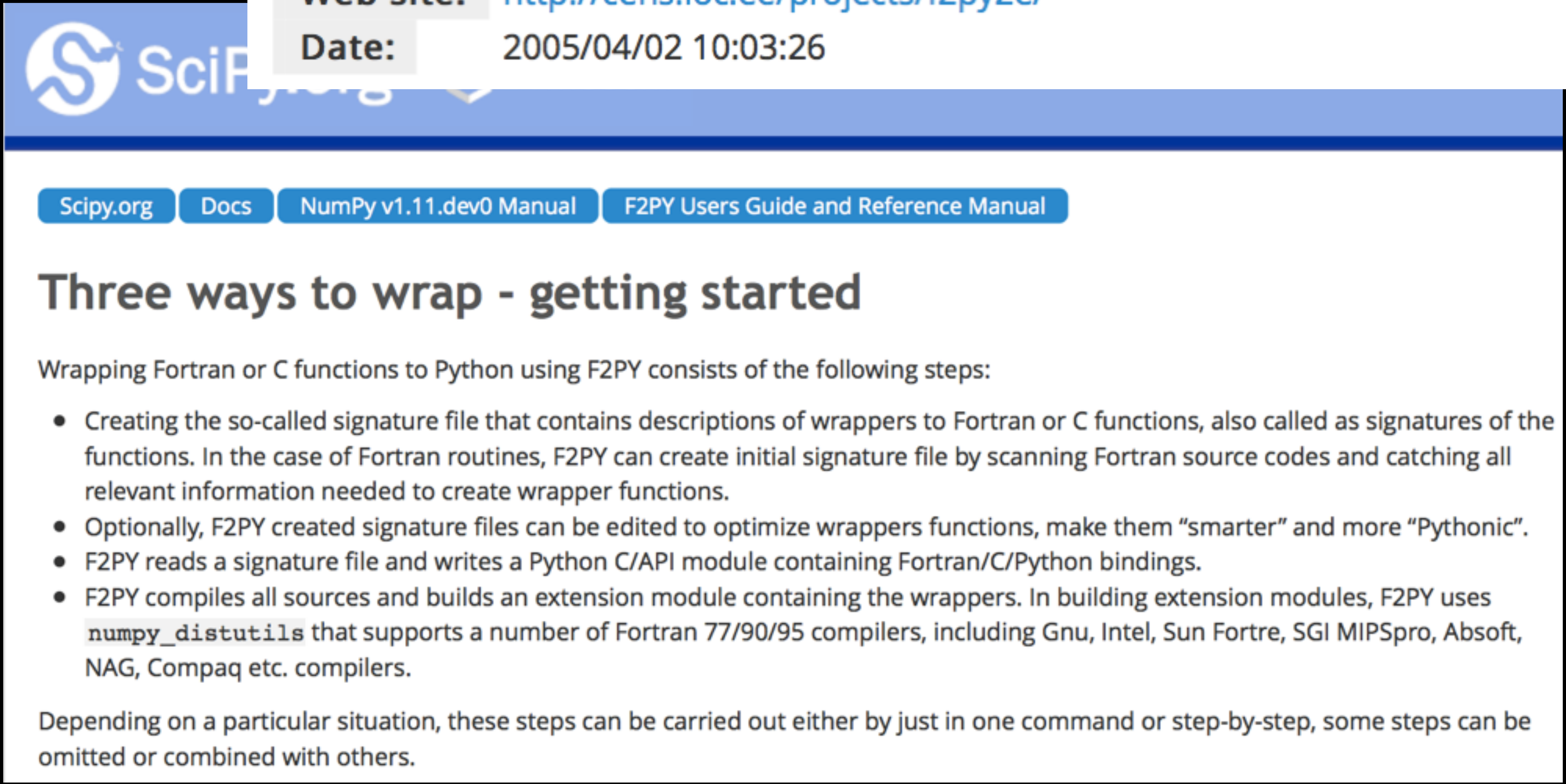
Thank you!

Special Thanks to **Prof. Pearu Peterson**!


Scipy.org | Docs | NumPy v1.11.dev0 Manual

F2PY Users Guide and Reference Manual

Author:	Pearu Peterson
Contact:	pearu@cens.ioc.ee
Web site:	http://cens.ioc.ee/projects/f2py2e/
Date:	2005/04/02 10:03:26


Scipy.org | Docs | NumPy v1.11.dev0 Manual | F2PY Users Guide and Reference Manual

Three ways to wrap - getting started

Wrapping Fortran or C functions to Python using F2PY consists of the following steps:

- Creating the so-called signature file that contains descriptions of wrappers to Fortran or C functions, also called as signatures of the functions. In the case of Fortran routines, F2PY can create initial signature file by scanning Fortran source codes and catching all relevant information needed to create wrapper functions.
- Optionally, F2PY created signature files can be edited to optimize wrappers functions, make them “smarter” and more “Pythonic”.
- F2PY reads a signature file and writes a Python C/API module containing Fortran/C/Python bindings.
- F2PY compiles all sources and builds an extension module containing the wrappers. In building extension modules, F2PY uses `numpy_distutils` that supports a number of Fortran 77/90/95 compilers, including Gnu, Intel, Sun Fortre, SGI MIPSpro, Absoft, NAG, Compaq etc. compilers.

Depending on a particular situation, these steps can be carried out either by just in one command or step-by-step, some steps can be omitted or combined with others.