UNIVERSITY OF DUNDEE

Introduction to python

Why Python?

Highly expressive:

```
friends = ['john', 'pat', 'gary', 'michael']
for i, name in enumerate(friends):
    print("iteration {iteration} is {name}".format(iteration=i, name=name))
```

Object oriented

Ease:

Easy to learn, change from IDL or Matlab.

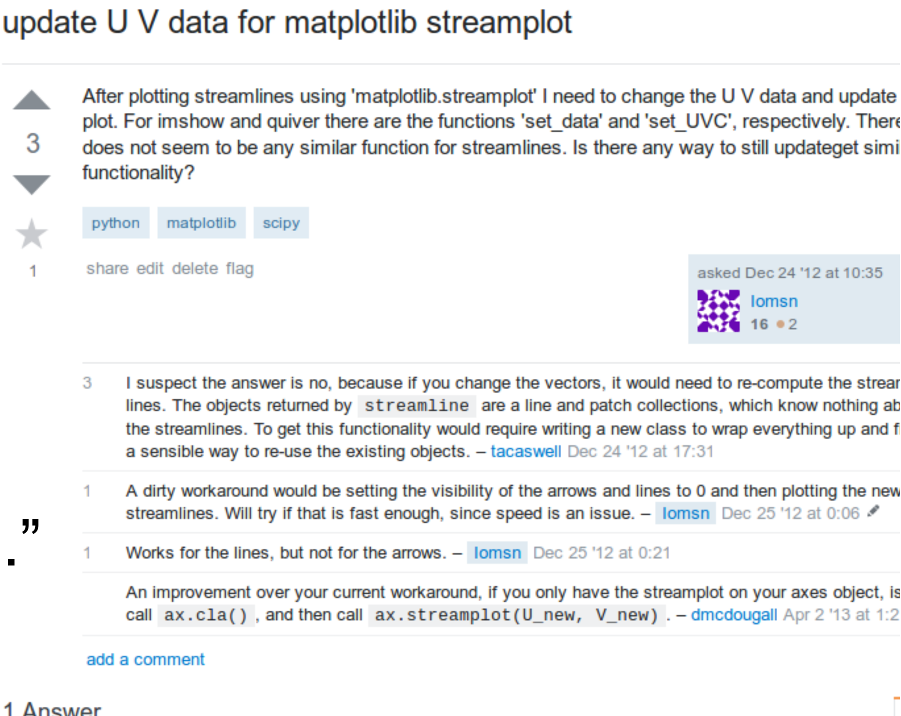
Very active community

Extendability:

libraries for numerics, data analysis, plotting, financing, ...

Interoperability:

Import IDL and Matlab code.
“But all my routines are written in IDL.”



update U V data for matplotlib streamplot

3

After plotting streamlines using 'matplotlib.streamplot' I need to change the U V data and update
plot. For imshow and quiver there are the functions 'set_data' and 'set_UVC', respectively. There
does not seem to be any similar function for streamlines. Is there any way to still updateget simi
functionality?

asked Dec 24 '12 at 10:35
lomsn
16 ●2

3 I suspect the answer is no, because if you change the vectors, it would need to re-compute the strear
lines. The objects returned by `streamline` are a line and patch collections, which know nothing ab
the streamlines. To get this functionality would require writing a new class to wrap everything up and f
a sensible way to re-use the existing objects. – tacaswell Dec 24 '12 at 17:31

1 A dirty workaround would be setting the visibility of the arrows and lines to 0 and then plotting the new
streamlines. Will try if that is fast enough, since speed is an issue. – lomsn Dec 25 '12 at 0:06

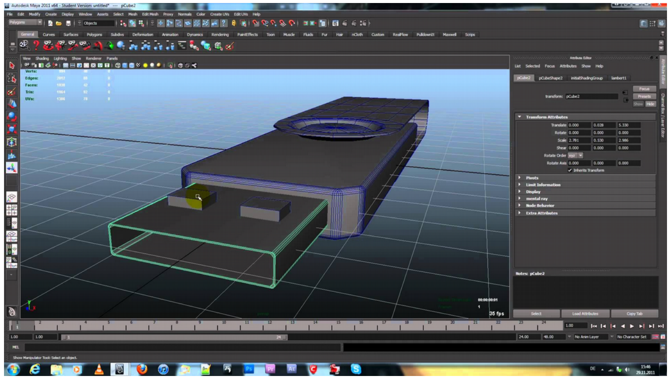
1 Works for the lines, but not for the arrows. – lomsn Dec 25 '12 at 0:21

An improvement over your current workaround, if you only have the streamplot on your axes object, is
call `ax.cla()` , and then call `ax.streamplot(U_new, V_new)` . – dmcdougall Apr 2 '13 at 1:2

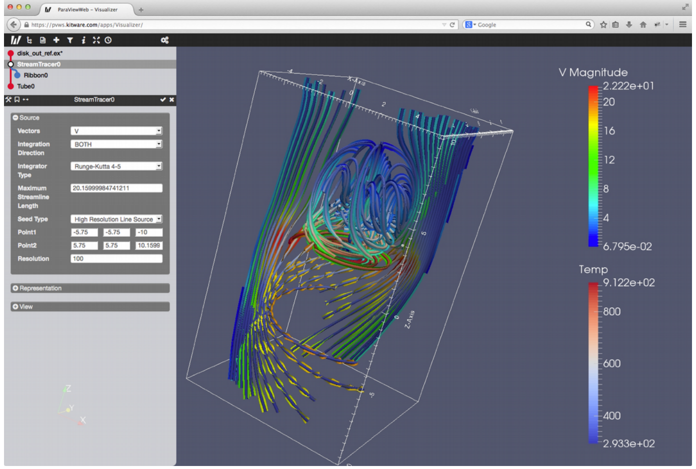
1 Answer

Why Python?

"Python is everywhere, it is all around us, even now in this very room."



3ds Max
Maya
Blender
Cinema 4D
...



ParaViewWeb – Visualizer

https://pvws.kitware.com/apps/Visualizer/

disk_out_ref.ex*
StreamTracer0
Ribbon0
Tube0

StreamTracer0

Source

Vectors	V		
Integration Direction	BOTH		
Integrator Type	Runge-Kutta 4-5		
Maximum Streamline Length	20.15999984741211		
Seed Type	High Resolution Line Source		
Point1	-5.75	-5.75	-10
Point2	5.75	5.75	10.1599
Resolution	100		

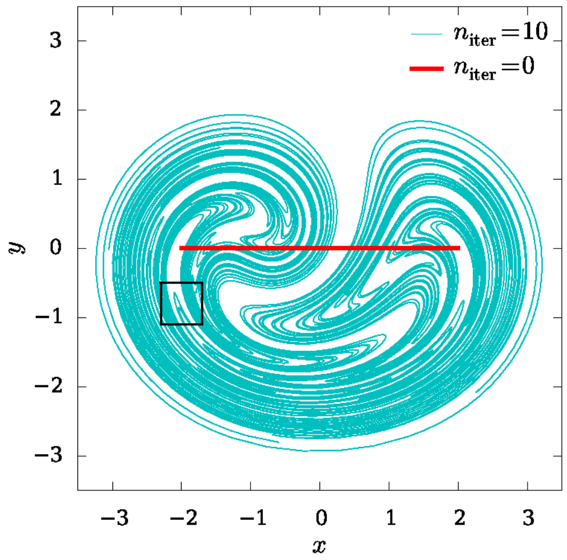
Representation

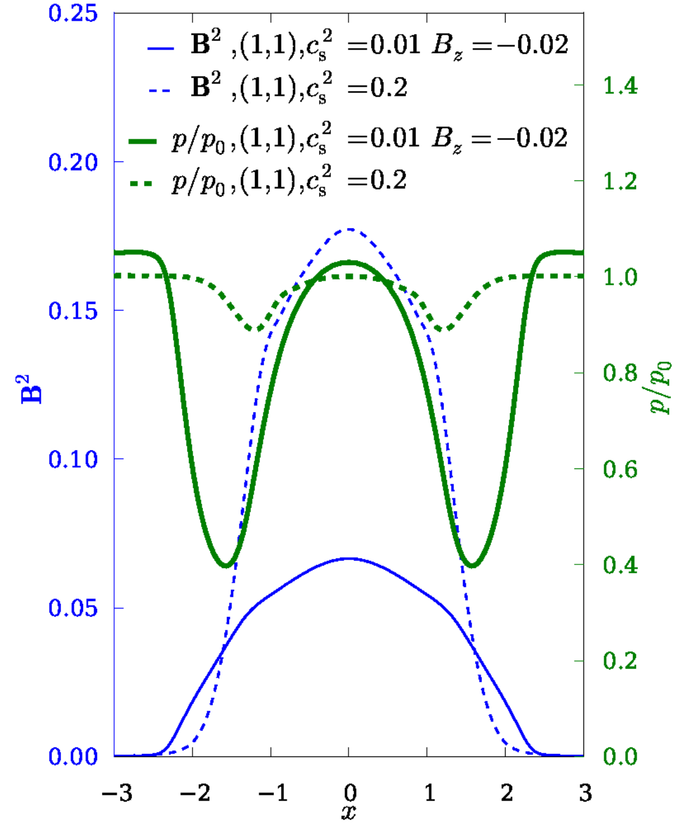
View

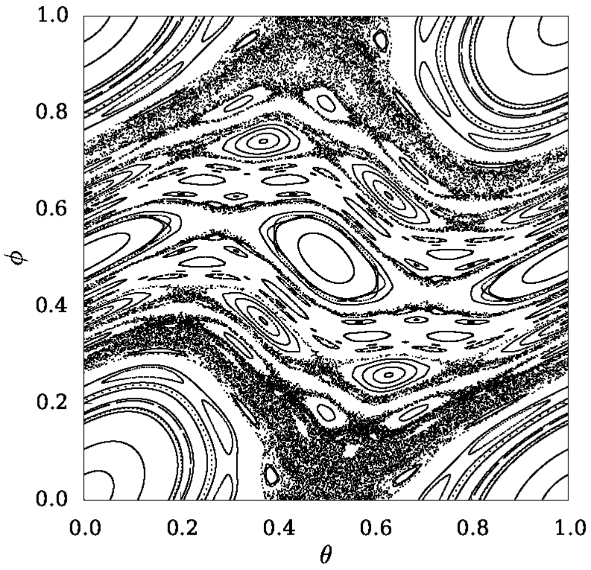
Paraview, Visit, Vapor, ...

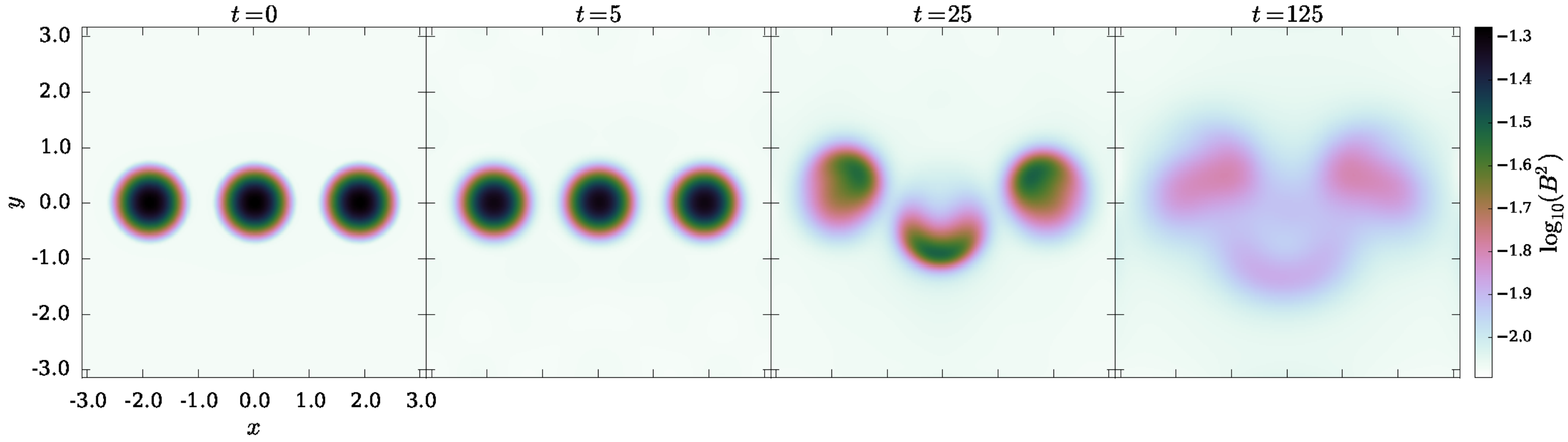
Civilization IV, Battlefield 2, World of Tanks, ...

Why Python?

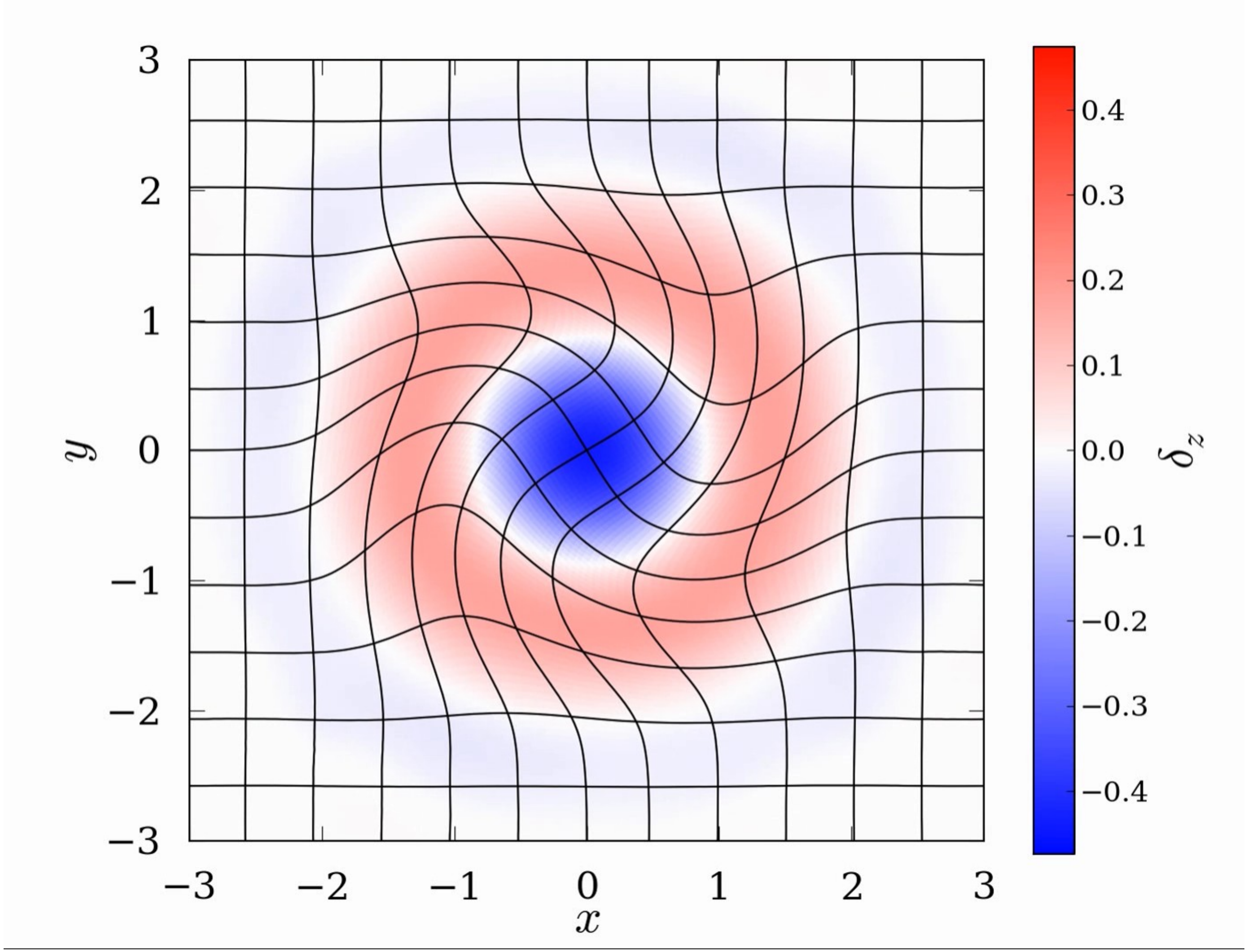

$n_{\rm iter}=10$
$n_{\rm iter}=0$
y
3
2
1
0
−1
−2
−3
x
−3
−2
−1
0
1
2
3


$\mathbf{B}^2$,(1,1),c_s^2 =0.01 B_z =−0.02
$\mathbf{B}^2$,(1,1),c_s^2 =0.2
p/p_0 ,(1,1),c_s^2 =0.01 B_z =−0.02
p/p_0 ,(1,1),c_s^2 =0.2
$\mathbf{B}^2$
0.25
0.20
0.15
0.10
0.05
0.00
p/p_0
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
x
−3
−2
−1
0
1
2
3

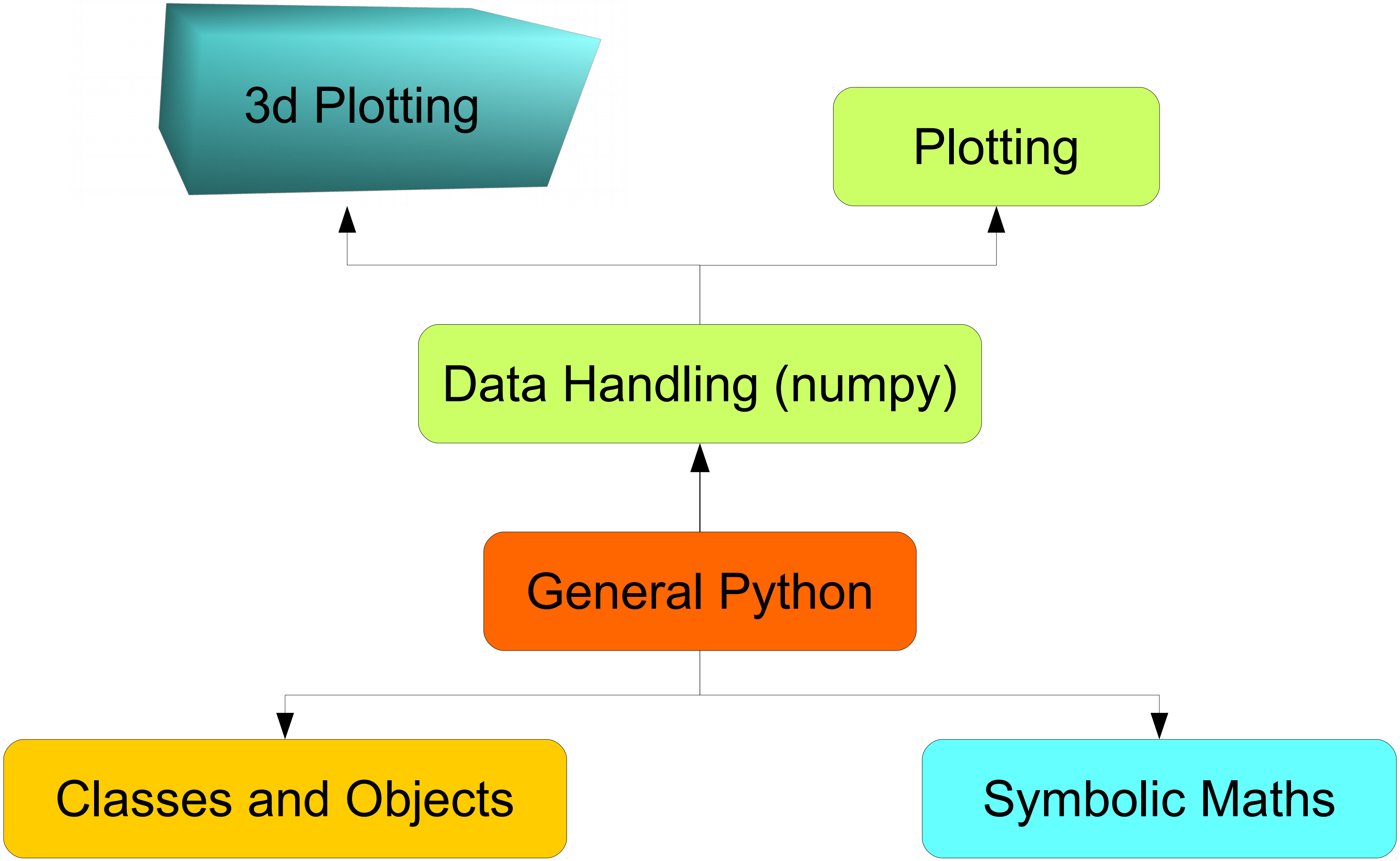

ϕ
1.0
0.8
0.6
0.4
0.2
0.0
θ
0.0
0.2
0.4
0.6
0.8
1.0


$t=0$
$t=5$
$t=25$
$t=125$
y
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
x
-3.0
-2.0
-1.0
0.0
1.0
2.0
3.0
$\log_{10}(B^2)$
−1.3
−1.4
−1.5
−1.6
−1.7
−1.8
−1.9
−2.0

Why Python?



Modules



3d Plotting
Plotting
Data Handling (numpy)
General Python
Classes and Objects
Symbolic Maths

History

1991, Guido van Rossum

Python 2.0
2000

Python 3.0
2008

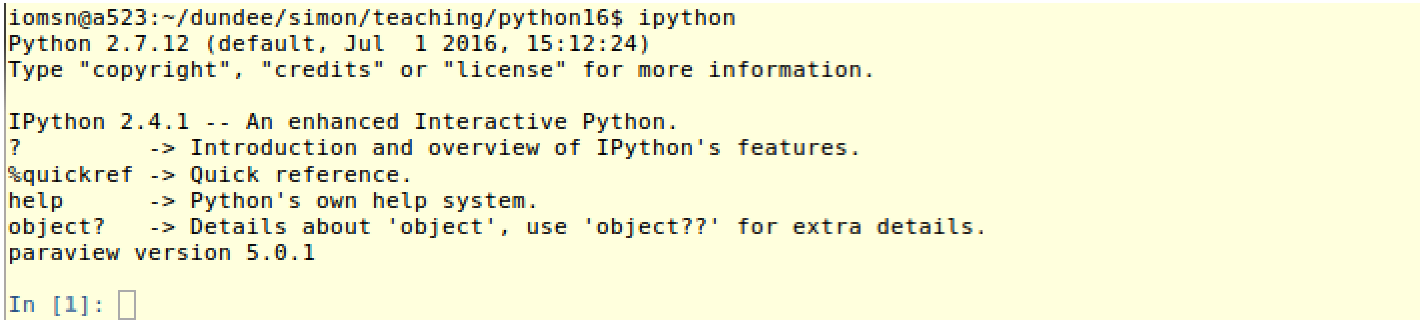
Python 3.52
2016

Python
Course

Namesake:

Ways of Using Python

python/ipython

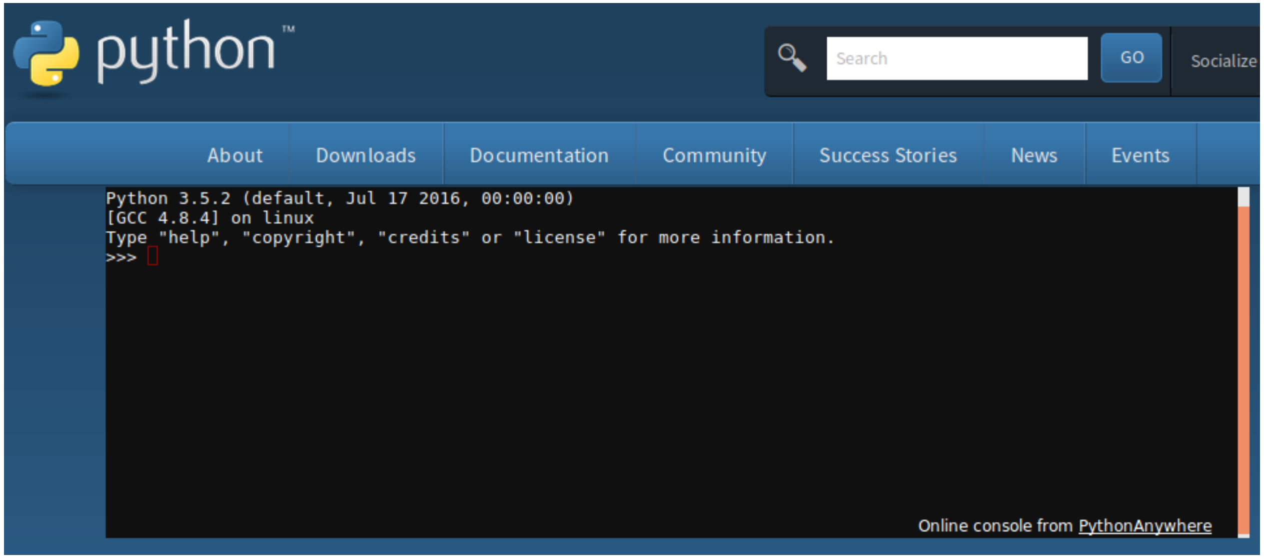


```
iomsn@a523:~/dundee/simon/teaching/python16$ ipython
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
paraview version 5.0.1

In [1]: 
```

https://www.python.org/

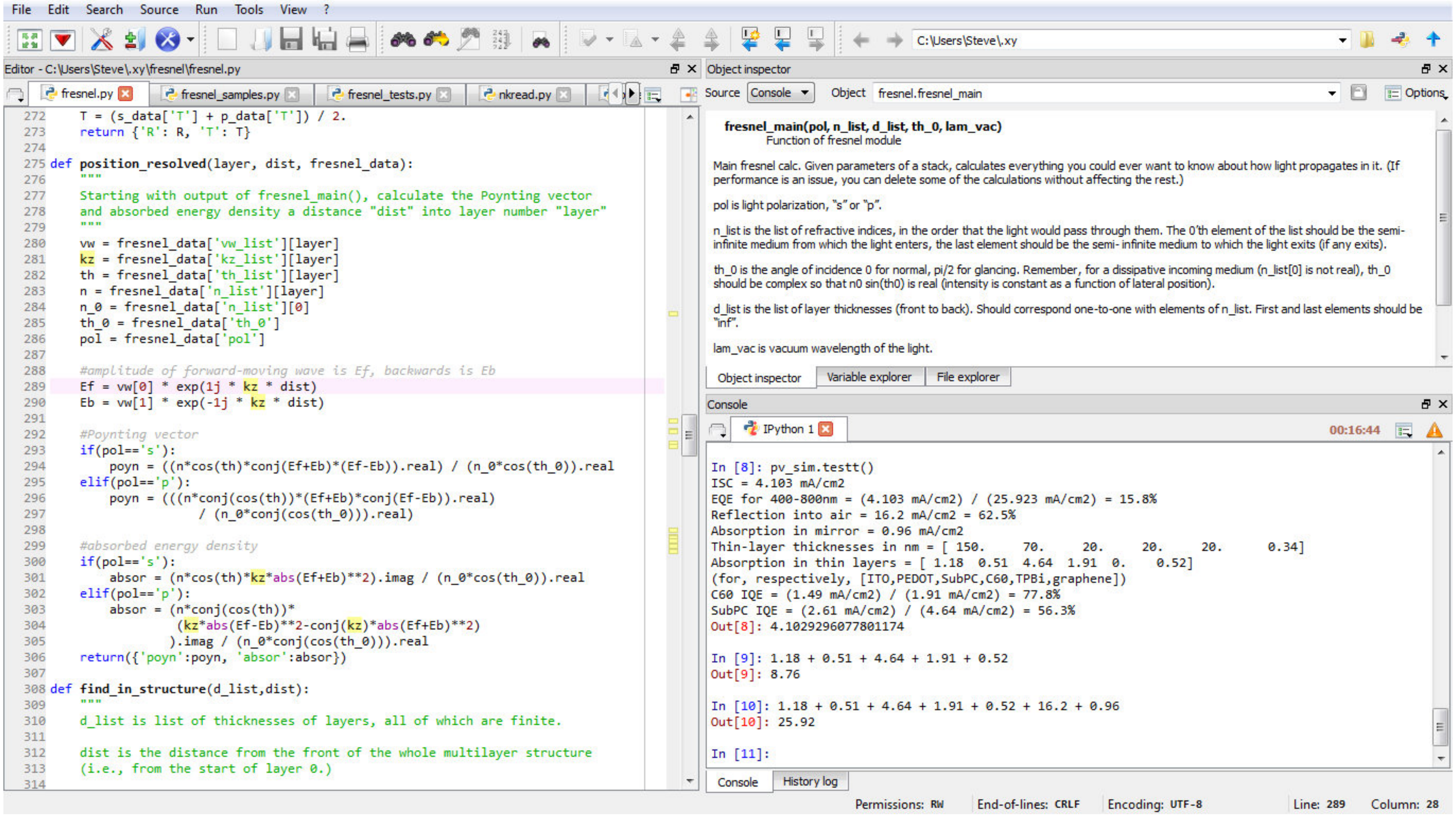


```
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Ways of Using Python

ANACONDA®

spyder



```python
    T = (s_data['T'] + p_data['T']) / 2.
    return {'R': R, 'T': T}

def position_resolved(layer, dist, fresnel_data):
    """
    Starting with output of fresnel_main(), calculate the Poynting vector
    and absorbed energy density a distance "dist" into layer number "layer"
    """
    vw = fresnel_data['vw_list'][layer]
    kz = fresnel_data['kz_list'][layer]
    th = fresnel_data['th_list'][layer]
    n = fresnel_data['n_list'][layer]
    n_0 = fresnel_data['n_list'][0]
    th_0 = fresnel_data['th_0']
    pol = fresnel_data['pol']

    #amplitude of forward-moving wave is Ef, backwards is Eb
    Ef = vw[0] * exp(1j * kz * dist)
    Eb = vw[1] * exp(-1j * kz * dist)

    #Poynting vector
    if(pol=='s'):
        poyn = ((n*cos(th)*conj(Ef+Eb)*(Ef-Eb)).real) / (n_0*cos(th_0)).real
    elif(pol=='p'):
        poyn = (((n*conj(cos(th))*(Ef+Eb)*conj(Ef-Eb)).real)
                    / (n_0*conj(cos(th_0))).real)

    #absorbed energy density
    if(pol=='s'):
        absor = (n*cos(th)*kz*abs(Ef+Eb)**2).imag / (n_0*cos(th_0)).real
    elif(pol=='p'):
        absor = (n*conj(cos(th))*
                 (kz*abs(Ef-Eb)**2-conj(kz)*abs(Ef+Eb)**2)
                ).imag / (n_0*conj(cos(th_0))).real
    return({'poyn':poyn, 'absor':absor})

def find_in_structure(d_list,dist):
    """
    d_list is list of thicknesses of layers, all of which are finite.

    dist is the distance from the front of the whole multilayer structure
    (i.e., from the start of layer 0.)

```

Object: fresnel.fresnel_main

fresnel_main(pol, n_list, d_list, th_0, lam_vac)
Function of fresnel module

Main fresnel calc. Given parameters of a stack, calculates everything you could ever want to know about how light propagates in it. (If performance is an issue, you can delete some of the calculations without affecting the rest.)

pol is light polarization, "s" or "p".

n_list is the list of refractive indices, in the order that the light would pass through them. The 0'th element of the list should be the semi-infinite medium from which the light enters, the last element should be the semi- infinite medium to which the light exits (if any exits).

th_0 is the angle of incidence 0 for normal, pi/2 for glancing. Remember, for a dissipative incoming medium (n_list[0] is not real), th_0 should be complex so that n0 sin(th0) is real (intensity is constant as a function of lateral position).

d_list is the list of layer thicknesses (front to back). Should correspond one-to-one with elements of n_list. First and last elements should be "inf".

lam_vac is vacuum wavelength of the light.

```
In [8]: pv_sim.testt()
ISC = 4.103 mA/cm2
EQE for 400-800nm = (4.103 mA/cm2) / (25.923 mA/cm2) = 15.8%
Reflection into air = 16.2 mA/cm2 = 62.5%
Absorption in mirror = 0.96 mA/cm2
Thin-layer thicknesses in nm = [ 150.     70.     20.     20.     20.      0.34]
Absorption in thin layers = [ 1.18  0.51  4.64  1.91  0.    0.52]
(for, respectively, [ITO,PEDOT,SubPC,C60,TPBi,graphene])
C60 IQE = (1.49 mA/cm2) / (1.91 mA/cm2) = 77.8%
SubPC IQE = (2.61 mA/cm2) / (4.64 mA/cm2) = 56.3%
Out[8]: 4.1029296077801174

In [9]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52
Out[9]: 8.76

In [10]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52 + 16.2 + 0.96
Out[10]: 25.92

In [11]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 289 Column: 28