Python for Silicon Photoncis

edX Phot1x by: Dr. Lukas Chrostowski
Prepared by: Huixu (Chandler) Deng

Feb, 2017

Python Installation

Platform

https://www.continuum.io/downloads

Python versions:
Python has developed into two major branches: Python 2.x.x and Python 3.x.x. Eventually, the two major branches will merge into one.

Python packages:
numpy
matplotlib.pyplot
scipy

IDE

```python
from __future__ import print_function
# make 'print' compatible in Python 2X and 3X
import matplotlib.pyplot as plt
import numpy

a = 1
b = 2
c = a + b

print ('a=', a)
print ('b=', b)
print ('c=', c)

# Practice figures:
x = numpy.arange(1,10.1,0.1)

plt.figure()
plt.plot(x, numpy.sin(x))
plt.title('The First figure')

plt.figure()
plt.plot(x, numpy.exp(x))
plt.title('The Second figure')
```

Numpy: Operation on Array(Matrix) in Python

Numpy is:

- extension package to Python for multidimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)

Numpy array operations:

```
import numpy as np

a = np.array([0, 1, 2, 3])
array([0, 1, 2, 3])

a = np.arange(10) # 0 .. n-1 (!)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

a = np.linspace(0, 1, 6) # start, end, num-points
array([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

```
a = np.ones((3, 3))
# reminder: (3, 3) is a tuple
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])

a = np.zeros((2, 2))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

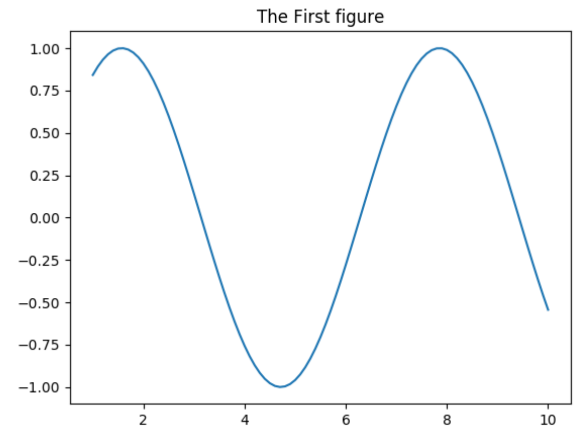
More indroduction can be found in the following references:

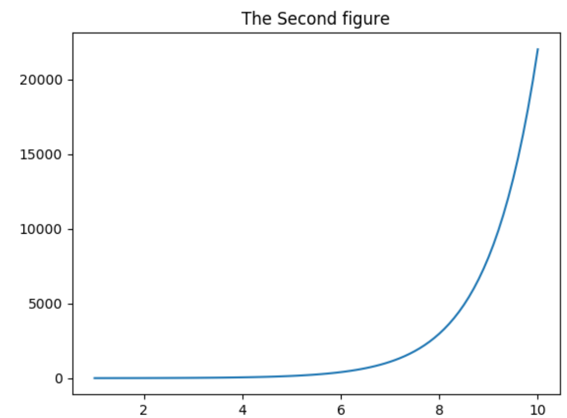
- http://www.scipy-lectures.org/intro/numpy/numpy.html
- https://www.datacamp.com/community/tutorials/python-numpy-tutorial#gs.8c8eKAE
- http://cs231n.github.io/python-numpy-tutorial/

Matplotlib: Figure Plot as Matlab

- Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

```
from __future__ import print_function
# make 'print' compatible in Python 2X and 3X
import matplotlib.pyplot as plt
import numpy

a = 1
b = 2
c = a + b

print ('a=', a)
print ('b=', b)
print ('c=', c)

# Practice figures:
x = numpy.arange(1,10.1,0.1)

plt.figure()
plt.plot(x, numpy.sin(x))
plt.title('The First figure')

plt.figure()
plt.plot(x, numpy.exp(x))
plt.title('The Second figure')
```

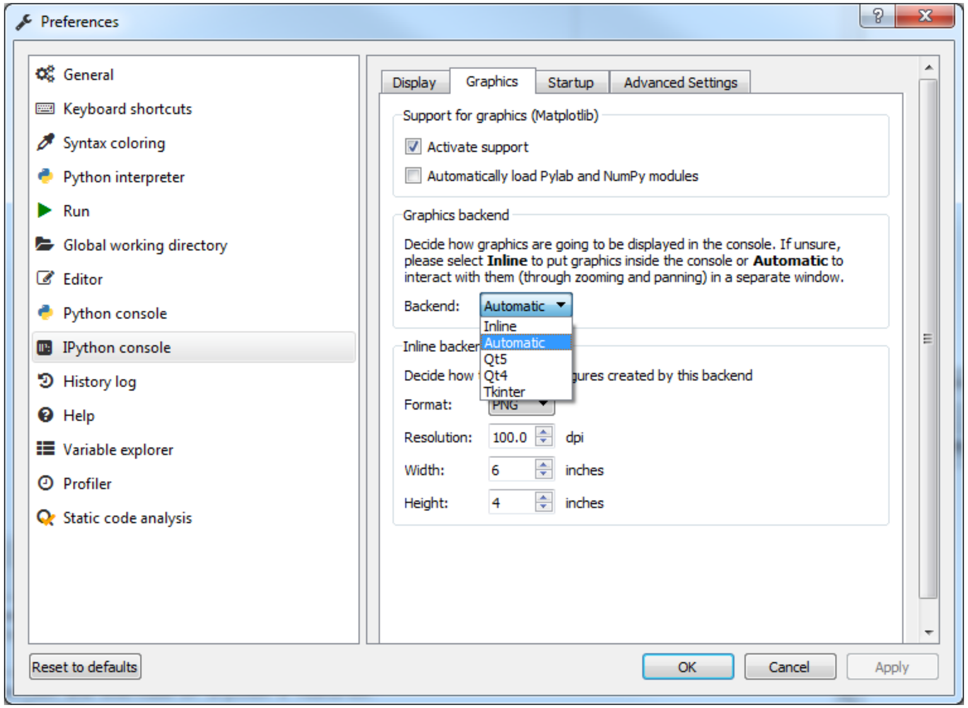


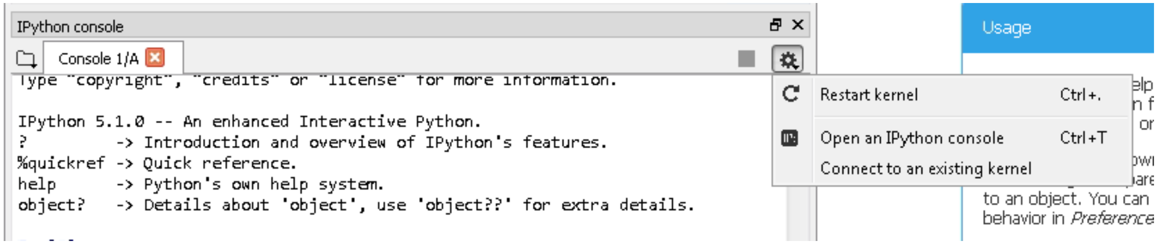


- http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html
- http://matplotlib.org/

Ipython Graphics

- To import the matplotlib.pyplot module correctly , the graphics in Ipython console can be set as ‘Automatic’, then restart the Ipython kernel by Ctrl + .
- It will also make the figures plotted in a new window as Matlab.





Python 2.x vs. Python 3.x

Differences between Python 2.x and 3.x include:

- Print function
- attribute names, like urllib.request

Under Python 2.6 there is a __future__ import to make print into a function. So to avoid any syntax errors and other differences you should start any file where you use print() with:

```
from __future__ import print_function
```

```
>>> print("This works in all versions of Python!")
This works in all versions of Python!
```

- http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html
- http://python3porting.com/noconv.html

In Python 3.x, the urlretrieve function is located in the urllib.request module

Python 2.x

```
# Download the file from Dropbox.  Dropbox requires that you have a ?dl=1 at the e
# Store the file in the local directory
url = "https://www.dropbox.com/s/1rvjfef4jqybc12/ZiheGao_MZI2_271_Scan1.mat?dl=1"
FileName = os.path.split(os.path.splitext(url)[0]+'.mat')[1]
print (FileName)
urllib.urlretrieve (url, FileName)

```

Python 3.x

```
# Download the file from Dropbox.  Dropbox requires that you have a ?dl=1 at the er
# Store the file in the local directory
url = "https://www.dropbox.com/s/1rvjfef4jqybc12/ZiheGao_MZI2_271_Scan1.mat?dl=1"
FileName = os.path.split(os.path.splitext(url)[0]+'.mat')[1]
print (FileName)
urllib.request.urlretrieve (url, FileName)

```

- https://docs.python.org/3.0/library/urllib.request.html

fsolve: wg_1D_slab

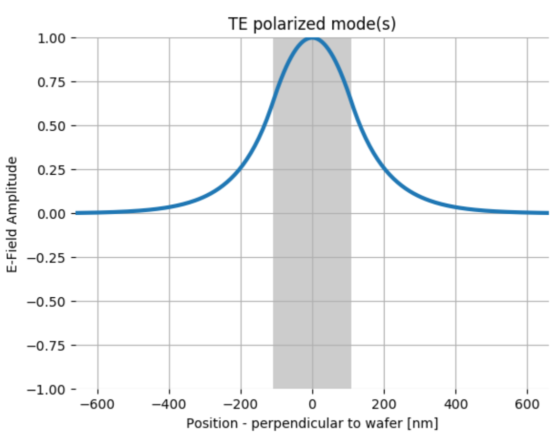
scipy.optimize.fsolve

scipy.optimize.fsolve(*func, x0, args=(), fprime=None, full_output=0, col_deriv=0, xtol=1.49012e-08, maxfev=0, band=None, epsfcn=None, factor=100, diag=None*) [source]

Find the roots of a function.

Return the roots of the (non-linear) equations defined by `func(x) = 0` given a starting estimate.

```
from scipy.optimize import fsolve
…
 nTE[i] = fsolve(lambda x: TE_eq(x,k0,n1,n2,n3,t)[0], X0[i,0]) / k0
```



In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Effective index value(s) of the TE mode(s): [2.84184958]
Effective index value(s) of the TM mode(s): [2.04888884]

Waveguide 2D Eigensolver

https://github.com/lbolla/EMpy

EMpy - ElectroMagnetic Python

build passing codacy A

EMpy - Electromagnetic Python is a suite of algorithms widely known and used in electromagnetic problems and optics: the transfer matrix algorithm, the rigorous coupled wave analysis algorithm and more.

Run the examples in examples/* to have an idea how EMpy works.

Visit http://lbolla.github.io/EMpy/ for more information.

Installation

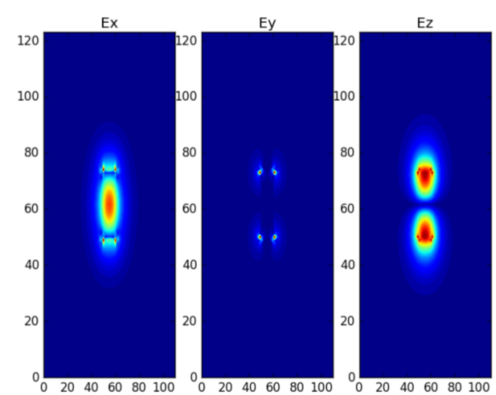
```
$> pip install ElectromagneticPython
```

Optionally, install bvp:

```
$> pip install scikits.bvp1lg
```

The ‘future’ package should be installed before installing EMpy.

- Run the ‘Anaconda Prompt’ as adminstrator,
- Type: pip install future
- Type: pip install ElectromagneticPython
- The 2D Eigensolver can be found in lbolla-EMpy-8628b19>>examples>>ex_modesolver.py



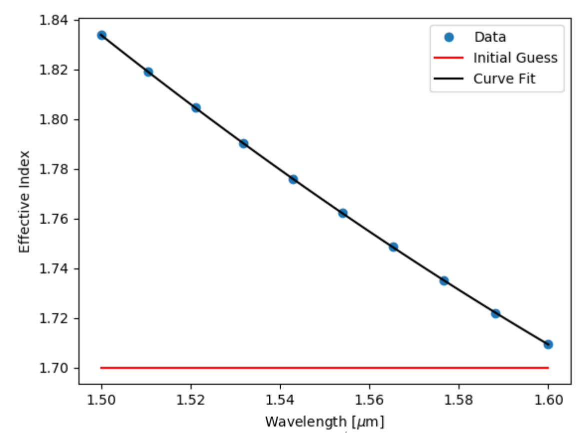
Least squares curve fitting: Phot1x_fit_wg_compactmodel

Matlab

```matlab
% curve fit to find expression for neff.
format long
X = lsqcurvefit (neff_eq, X, lambdas, neff)

r=corrcoef(neff,neff_eq(X, lambdas));
r2=r(1,2).^2;
disp (['Goodness of fit, r^2 value: ' num2str(r2) ])
```

Python: a function of residuals is created for least squares curve fitting

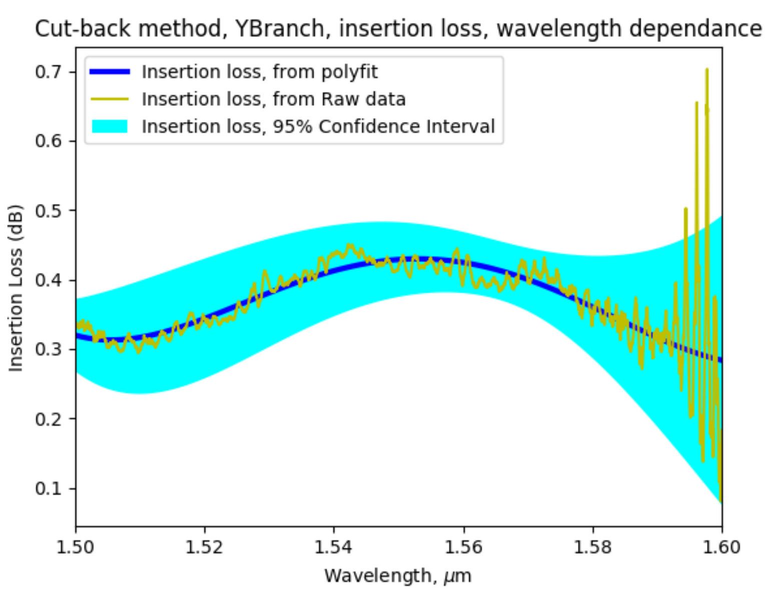
```python
# function for the effective index expression:
def neff_eq(nx, lam):
    return nx[0] + nx[1]*(lam-lam0) + nx[2]*(lam-lam0)**2

# In Python, to do the curve fitting,
# the leastsq function is used and the residuals between the data and the model sh
# function for residuals between the data and the model
def residuals(nx, y, lam):
    return y - neff_eq(nx, lam)

# initial guess
nx0=np.array([1.7,0,0])

# curve fit to find expression for neff.

nx, flag = opt.leastsq(residuals, nx0, args=(neff, lams))

r=np.corrcoef(neff,neff_eq(nx, lams))
r2=r[0,1]**2
print 'Goodness of fit, r^2 value: ', r2
print 'n1 = ', nx[0], 'n2 =', nx[1], 'n3 =', nx[2]

```

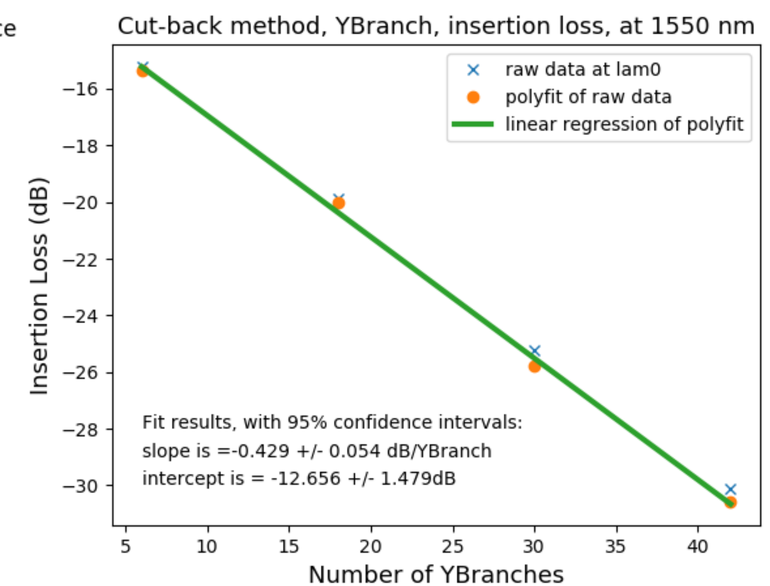


Regression interval: lukasc_YBranch_TM

import statsmodels.api as sm

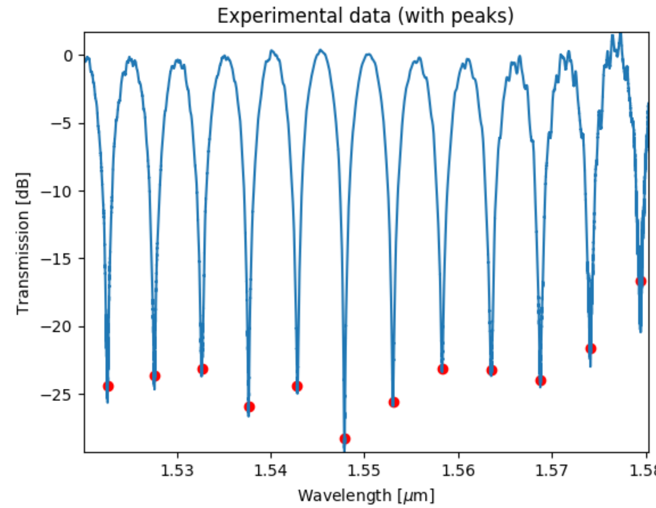
```python
# Calculate the slope error, +/- dB, with a 95% confidence interval
# using the StatsModels' Regression Results
y_index = np.array([amplitude_poly[i][index] for i in np.arange(len(Num))])
x_index = np.hstack((Num[:,np.newaxis], np.ones(len(Num))[:,np.newaxis]))
mod = sm.OLS(y_index, x_index)
res = mod.fit()
b = res.params
bint =  res.conf_int(0.05)

SlopeError95CI = np.diff(bint[0,:])/2
InterceptError95CI = np.diff(bint [1,:])/2
plt.annotate('Fit results, with 95% confidence intervals:', xy=(6, -28))
plt.annotate('slope is ='+'%.3f' %A[1] + ' +/- ' + '%.3f' %SlopeError95CI + ' dB/YBranch', xy=(6, -29))
plt.annotate('intercept is = '+'%.3f' %A[0] + ' +/- ' + '%.3f' %InterceptError95CI + 'dB', xy=(6, -30))
plt.show()
print 'Cut-back method, YBranch (TM) insertion loss, at', lam0*1e9, 'nm is =', -A[1], 'dB/YBranch'
```





Find peaks: Fitting_MZI_findpeaks

Algorithm	Integration	Filters	MatLab `findpeaks` -like?
scipy.signal.find_peaks_cwt	Included in Scipy	?	✗
detect_peaks	Single file source Depends on Numpy	Minimum distance Minimum height Relative threshold	✓
peakutils.peak.indexes	PyPI package PeakUtils Depends on Scipy	Amplitude threshold Minimum distance	✓
peakdetect	Single file source Depends on Scipy	Minimum peak distance	✗
Octave-Forge findpeaks	Requires an Octave-Forge distribution + PyPI package oct2py Depends on Scipy	Minimum distance Minimum height Minimum peak width	✗
Janko Slavic findpeaks	Single function Depends on Numpy	Minimum distance Minimum height	✗
Tony Beltramelli detect_peaks	Single function Depends on Numpy	Amplitude threshold	✗



https://github.com/MonsieurV/py-findpeaks

```python
#==========================================================================
# Find peaks, extract FSR and ng, and neff
# as initial parameters
#==========================================================================
# smooth the data to find peaks accurately
amplitude_smooth=savgol_filter(amplitude1, 2001, 5)
from scipy.signal import argrelmax
indexes = argrelmax(-amplitude_smooth)

# peak amplitude above half the whole amplitude is incorrect and removed
amplitude_half = (np.max(amplitude_smooth) + np.min(amplitude_smooth))/2
index_remove = np.nonzero(amplitude_smooth[indexes] > amplitude_half)
indexes = np.delete(indexes, index_remove)
x_values = lam1[indexes]

```

Cross-correlation: Fitting_MZI_autocorrelation

numpy.correlate

numpy.correlate(*a*, *v*, *mode='valid'*) [source]

Cross-correlation of two 1-dimensional sequences.

This function computes the correlation as generally defined in signal processing texts:

```
c_{av}[k] = sum_n a[n+k] * conj(v[n])
```

with a and v sequences being zero-padded where necessary and conj being the conjugate.

Parameters:

a, v : *array_like*
Input sequences.

mode : *{'valid', 'same', 'full'}, optional*
Refer to the convolve docstring. Note that the default is 'valid', unlike convolve, which uses 'full'.

old_behavior : *bool*
old_behavior was removed in NumPy 1.10. If you need the old behavior, use *multiarray.correlate*.

Returns:

out : *ndarray*
Discrete cross-correlation of *a* and *v*.