Python course, 91017, tikka

**Spyder 3.6 is good for doing python coding.**

In the code, F9(+fn) or paint control+enter execute the code in one line

%reset -f -> variable explorer deleted

%clear –f -> scroll the console to the beginning

Array is matrix, for with you need numpy, tab get the codes

Broadcasting happens to vector and scalar, so vector is preferred

magic command in windows:

<http://ipython.readthedocs.io/en/stable/interactive/magics.html>

Comments for the coding session

‘for and in’ are the special words in python; white space matters in code (e.g. four spaces before print)! There has to be colon also.

Counting starts at zero if you use range(N).

Check: ?range -> start stop step. No help for python keywords though.

Use expressive names, like: underscores\_can\_help=4, but not: not8goodidea=3, and yes-no=8;

*Code testing at Spyder console*

The first loop example prints the defined values:

for i in (0, 1, 2, 3, 4):

print(i)

*Ranges*

range(3)

Out[24]: range(0, 3)

tuple(range(3))

Out[25]: (0, 1, 2)

for i in range(4):

print(i)

i = 'r'

print(i)

for i in range(4):

print(i)

i = 'r'

print(i)

0

r

1

r

2

r

3

r

%pwd

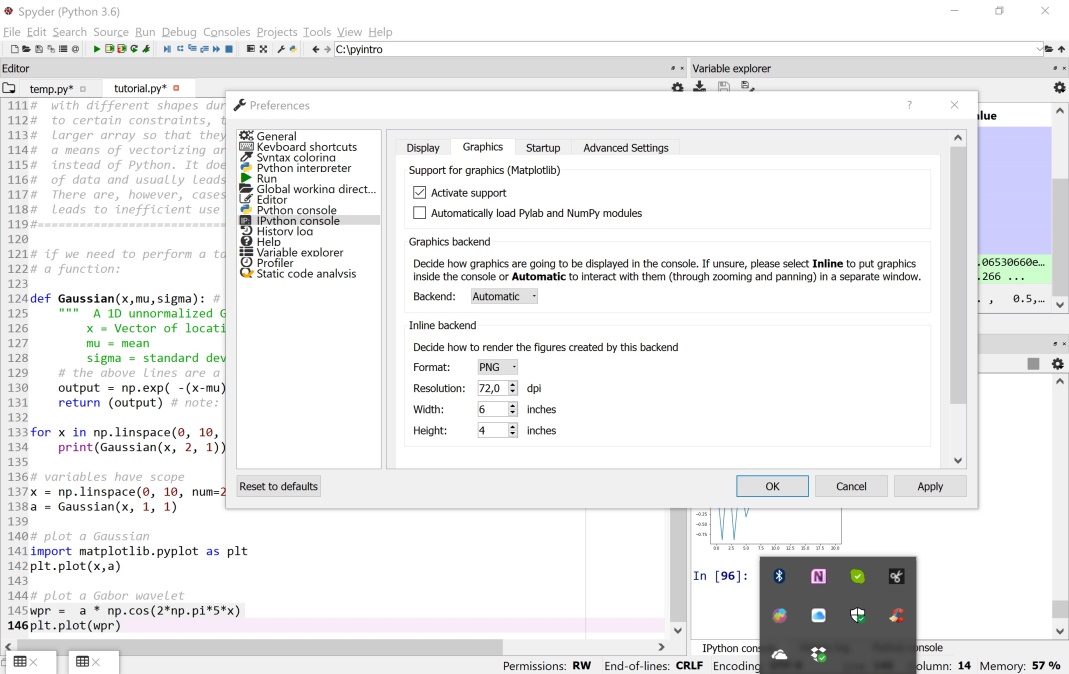
Out[38]: 'C:\\pyintro'

Name spaces help to avoid conflicts;

import numpy

numpy.ndarray

Out[43]: numpy.ndarray

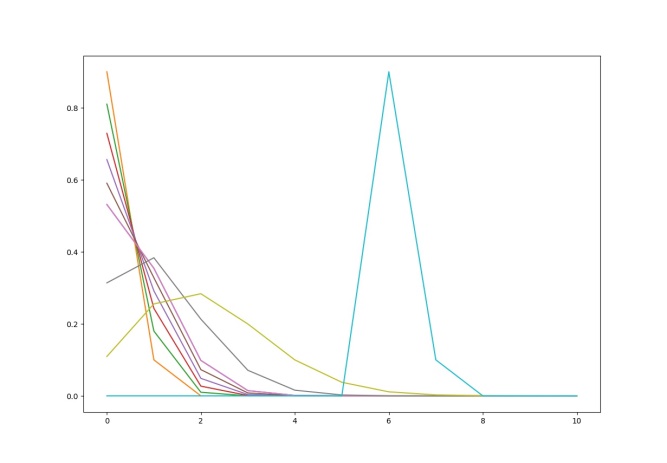


get new image, click away the old ones

‘’ to the file

ax = np.load('bc.npy')

10.10.2017



Got it! ☺ The random walker distribution plots. What was needed: right array types:

#%% The randomwalker function:

def rand(f,g,t): # arguments are functions

""" Convolution function.

f = define signal function

g = define other, less noisy, signal function """

# the above lines are a so-called docstring; try ?fg

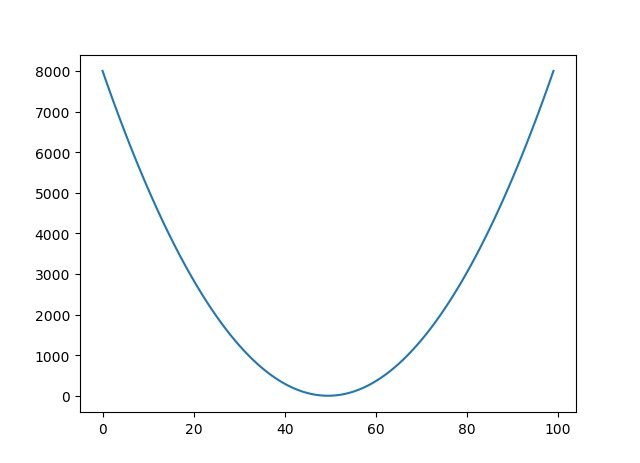
a=convl(f,g)

for i in range(t):

a=convl(a,g) **#HERE USE JUST VECTOR a, NOT MATRIX a[i] etc.**

return a

Doing a computational solution for the Shcroedinger equation:



def parab(alpha,vec):

""" A parabola function. “””

y=np.zeros([len(vec)])

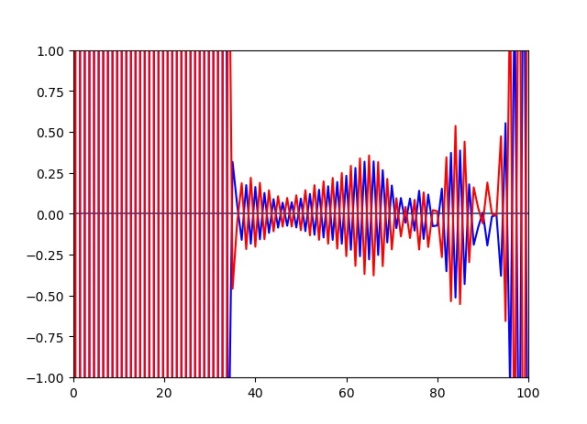
x=vec

y=alpha\*(x)\*\*2

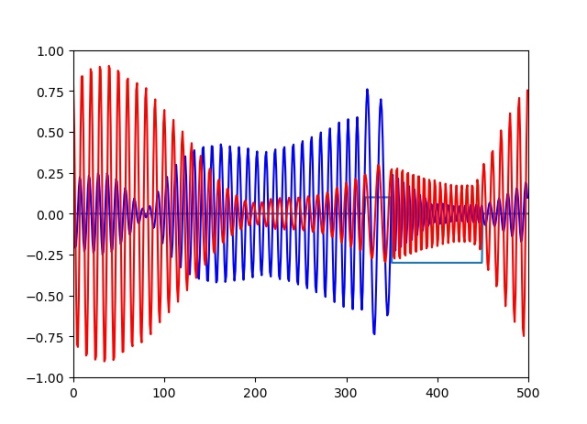
return y

11.10.17

Wow, first one went bad!

☺

And the second better one:



And the code for that:

# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 10 10:43:31 2017

@author: Pauli

"""

# Integrating 1D Schroedinger equation over time:

#%% The initial setpus:

import numpy as np

import scipy.ndimage

import matplotlib.pyplot as plt

vec=np.linspace(-10,10,100)

alpha=80

# functions for integrating Schroedinger equation over time and space:

# A function that returns a vector (a 1D numpy array) holding a parabola

#%%

def parab(alpha,vec):

""" A parabola function.

a = seed for function

t = time """

# the above lines are a so-called docstring; try ?parab

y=np.zeros([len(vec)])

x=vec

y=alpha\*(x)\*\*2

return y

#%%

plt.plot(parab(alpha,vec))

#

#%% A function that returns a parametric potential landscape

def landscape(x):

""" A Potential landscape function

x = spatial coordinate

"""

V=np.zeros(x)

V[0:320]=0

V[321:350]=0.1

V[350:450]=-0.3

V[451:500]=0

return V

#%%

plt.plot(landscape(100))

#%% The wave packet function

def wave(k,x,m,s):

"""

k = wave number

x = position

m = myy value

s = standard deviation value

"""

real=np.cos(2\*np.pi\*k\*x)\*np.exp(-(x-m)\*\*2/2/s\*\*2)

img=-np.sin(2\*np.pi\*k\*x)\*np.exp(-(x-m)\*\*2/2/s\*\*2)

return real, img

#%%

#%% **The final Schroedinger function**

def shcr\_fun(landscape, img\_res, real\_res, t, dt):

""" t = time change indicator, vec = initial vector, V = potential """

# Final functions

y\_1=img\_res

y\_2=real\_res

for i in range(t):

y\_1=y\_1+dt\*(0.5\*scipy.ndimage.filters.convolve1d(y\_2,(1,-2,1), mode='wrap')-landscape\*y\_2)

y\_2=y\_2+dt\*(-0.5\*scipy.ndimage.filters.convolve1d(y\_1,(1,-2,1), mode='wrap')+landscape\*y\_1)

return y\_1, y\_2

#%%

#The initial variables and test before animation of the Schrödinger equation

# Wave packet Variables

k = 5

x = np.linspace(1,11,500)

m = 5

s = 1

# Test the wave equation output

ri=wave(k,x,m,s)

plt.plot(ri[0])

plt.plot(ri[1])

#time steps

dt=0.1

#Landscape

ls=landscape(len(x))

#The function with one drive

a=shcr\_fun(ls,ri[0],ri[1],1,dt)

#a[0]

plt.plot(a[0])

plt.plot(a[1])

plt.plot(landscape(500))

#Number of steps

number\_of\_steps=500

#%% Plotting the Shcroedinger equation:

#def ShcrPlt(number\_of\_steps,x):

fig = plt.figure()

ax = fig.add\_subplot(111)

for t in range(number\_of\_steps):

print(t)

ax.clear()

ax.set\_xlim(0, 500)

ax.set\_ylim(-1, 1)

ax.plot(ls)

a=shcr\_fun(ls, a[0], a[1],50, dt)

ax.plot(a[0], '-b')

ax.plot(a[1], '-r')

ax.figure.canvas.draw()

plt.pause(0.001) #

Good ways to understand and analyse the legendary code script ‘for loop something with if’:

<https://stackoverflow.com/questions/19666626/replace-all-elements-of-python-numpy-array-that-are-greater-than-some-value>

For example for making a heat plot:

#%% Convolving:

#Initial values

t=1000

dt=0.1

d=[]

d=np.where(init\_cond ==1)

# The function

def somel\_fun(init\_cond,t,dt):

u=init\_cond.copy()

shape = init\_cond.shape

for i in range(t):

u=u+dt\*(scipy.ndimage.convolve(u,sten,mode='wrap'))

#see comment below for this for loop…

for x in range(0, shape[0]):

for y in range(0, shape[1]):

if init\_cond[x, y] == 1:

u[x, y] = 1

elif init\_cond[x, y] == -1:

u[x, y] = 1

# One can replace this nested loop by… tadaa:

# u[init\_cond != 0] = init\_cond[init\_cond != 0]

return u

#The end result of the heat equation over time:

con\_u=somel\_fun(init\_cond,t,dt)

#plt.imshow(init\_cond, interpolation = 'catrom')

plt.imshow(con\_u, interpolation = 'catrom')

About the loop…, better to make your code work rather than fancy!

13.10.17

Do your tasks one by one. And start from the one that you can think you can produce a better outcome first within a day or two. Just start, you do not need to hesitate unless your landlord/lady/similar is *literally* screaming to your ear at the moment. In that case, good luck! ☺

2D Dirichlet problem code:

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 11 13:44:37 2017

@author: Pauli

"""

# Solving a Dirichlet problem in 2D at Python

#%% Initial conditions for the Dirichlet problem

#Importing libraries

import numpy as np

import scipy.ndimage

import matplotlib.pyplot as plt

#Making the steps

init\_cond=np.load('bc.npy')

init\_cond[np.isnan(init\_cond)]=0

# Figure for the initial condition

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.set\_xlim(0, 30)

ax.set\_ylim(0, 30)

ax.imshow(init\_cond, interpolation = 'catrom')

ax.set\_title('Initial condition')

sten=np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])

#%% Integrating 2D heat eqution over time:

# Initial conditions

t=500

dt=0.01

# Integrating 2D heat eqution over time:

def somel\_fun(init\_cond,t,dt):

u=init\_cond.copy()

shape = init\_cond.shape

for i in range(t):

u=u+dt\*(scipy.ndimage.convolve(u,sten,mode='wrap'))

# Alternatively:

# u[init\_cond != 0] = init\_cond[init\_cond != 0]

for x in range(0, shape[0]):

for y in range(0, shape[1]):

if init\_cond[x, y] == 1:

u[x, y] = 1

elif init\_cond[x, y] == -1:

u[x, y] = -1

return u

#The end result of the heat equation over time:

con\_u=somel\_fun(init\_cond,t,dt)

#Plotting the iteration

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.set\_xlim(0, 30)

ax.set\_ylim(0, 30)

ax.imshow(con\_u, interpolation = 'catrom')

ax.set\_title('Iteration after 500 steps')

#%% Image flow for the Dirichlet problem

#Number of steps

number\_of\_steps=500

dt=0.1

#%% Plotting the numerical result of the Dirichlet problem

fig = plt.figure()

ax = fig.add\_subplot(111)

a=init\_cond.copy()

for t in range(number\_of\_steps):

ax.clear()

ax.set\_xlim(0, 30)

ax.set\_ylim(0, 30)

a=somel\_fun(a,1,dt)

ax.imshow(a, interpolation='catrom')

ax.set\_title('Iteration')

ax.figure.canvas.draw()

plt.pause(0.001)

#%% Calculating the exact solution of the heat equation over time with linear solver

#%% Functions for making L and r for solving laplace function u:

def LR\_fun(init\_cond, sten):

""" init\_cond = init\_cond matrix

sten = the seed stencil matrix """

# Easy part of the hard zone;

# initial fixings of the matrix L

r = init\_cond.copy()

L = np.zeros((1024,1024))

# Hard zone starts;

# filling the L rows with a vector of seeded or just one values

for i in range(32):

for j in range(32):

Lrow = np.zeros((32,32))

if r[i,j] == 0:

# hard case

Lrow[i,j] = 1

L[32\*i+j,:] = (scipy.ndimage.convolve(Lrow,sten,mode='wrap')).flatten()

elif r[i,j] != 0:

# easy case

Lrow[i,j] = 1

L[32\*i+j,:] = Lrow.flatten()

*#****The ‘Lrow’ matrix is inserted to a correct position of a row of L by using math:* 32\*i+j !!!!**

return L

#%% Ploting the boundary conditions, and evolution over time of u

# initial values

L = LR\_fun(init\_cond,sten)

r = init\_cond.copy()

r = r.flatten()

#%% Solving the system of linear equations for -Lu = r

def lin\_solv(L,r):

# The return matrix

om=np.linalg.solve(L,r) #minus not needed, it was on stencil

return om

#%% Ploting the steady state solution or condition for solving u

solved=lin\_solv(L,r)

#Image for the exact solution called 'solved'

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.set\_xlim(0, 30)

ax.set\_ylim(0, 30)

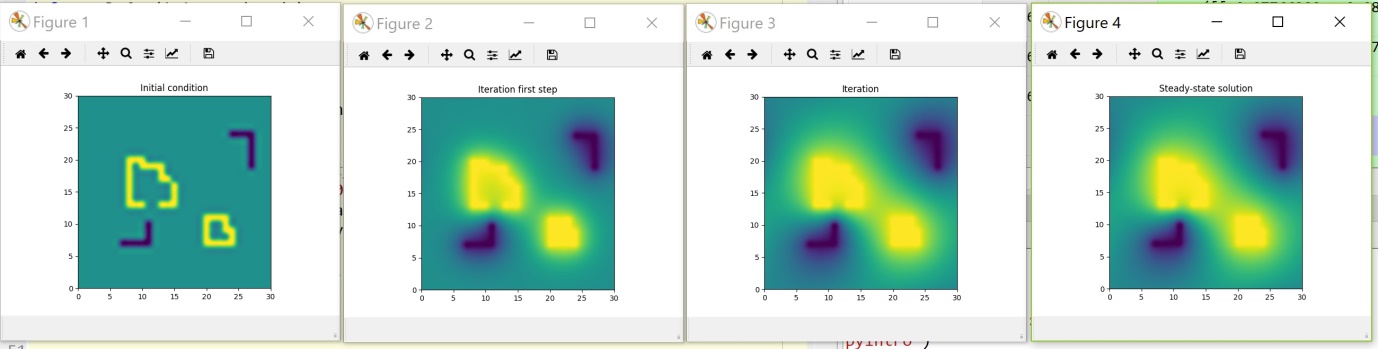
ax.set\_title('Steady-state solution')

ax.imshow(solved.reshape((32,32)), interpolation='catrom')

ax.figure.canvas.draw()

plt.pause(0.001)

# AND the results are:



Results of the codings, ☺

The last tasks: machine learning:

<http://www.scipy-lectures.org/advanced/scikit-learn/>

The code as it is:

#Machine learning

#%% Initial starting for random forest

import matplotlib

import numpy as np

import scipy.ndimage

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

features=np.zeros((100,10))

labels=np.zeros((100,1)).ravel()

#%% Defining the random forest system with features and labels

# rf = RandomForestClassifier() # create Random Forest

# Loading initial image and features with 'matplotlib.pyplot.imread'

neuron\_img=matplotlib.pyplot.imread('rawEMslice.tif')

neuron\_ann=matplotlib.pyplot.imread('labels.tif')

neuron\_ann2=np.load('features.npy')

# Printing random forest plot, starting with fitting samples with these features and labels

rf.fit(neuron\_img[neuron\_ann!=0].reshape((-1,1)), neuron\_ann[neuron\_ann!=0])

# Labels for neuron with random forest function

neuron\_labels = rf.predict(neuron\_img.flatten().reshape((-1,1)))

neuron\_labels = neuron\_labels.reshape((1000,1000))

# Defining more variables for the plot

colors = np.array([[1,0,0], [0, 1, 0], [0, 0, 1]])

colormap = matplotlib.colors.ListedColormap(colors)

##Plotting the neuron image

#Direct

plt.imshow(neuron\_labels)

##Grey

plt.imshow(neuron\_labels, cmap='gray')

#Transparent

plt.imshow(neuron\_labels, alpha=0.5)