Scientific Computing using python - mini project Documentation

Release

Juan de Dios Flores Mendez

Jun 20, 2017

CONTENTS:

CHAPTER ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER TWO

LORENZ SYSTEM

The Lorenz system is a system of ordinary differential equations first studied by Edward Lorenz. It has chaotic solutions for certain parameter values and initial conditions. The Lorenz attractor is a set of chaotic solutions of the Lorenz system which, when plotted, resemble a butterfly or figure eight.

The lorenz equations are as follows

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x),\\ \frac{dy}{dt} &= x(\rho - z) - y,\\ \frac{dz}{dt} &= xy - \beta z\end{aligned}$$

In order to solve the differential equations it is necessary to implement a solver for the differential equations, for which the Euler approximation for first order differential equations is selected.

It is formulated as

$$\begin{aligned}\frac{x_{n+1} - x_n}{t_d} &= f(x, y, z)\\ x_{n+1} &= f(x, y, z) \cdot t_d + x_n\end{aligned}$$

The same formulation is applied for the rest of the differential equations.

It should be noticed that the solution for a period of time of the lorenz attractor is very sensitive to initial conditions.

CHAPTER

THREE

LAUNCHING THE TEST CASES FOR DIFFERENT PARAMETERS

Locate the terminal/ipython in the folder of cases. When running the script it will plot the different solutions and save all the figures, data and solutions to pdf and a hdf5 file.

You can run the test case 1 as

```
>>> python case1.py
```

And for test case 2

```
>>> python case2.py
```

and so on

```
>>> python case3.py
>>> python case4.py
>>> python case5.py
```

CHAPTER

FOUR

TESTING THE CODE

Locate the terminal/ipython in the folder of test. You can run the tests and it will be compared to numpy ODE solver. It should be noticed that the more chaotic the system it will fail the test.

It was observed that the Euler solver carry delays to the solution than the ODE solver inside numpy.

To run the tests you can type

```
>>> python test.py
>>> ...FF
```

It will pass the first three cases and fail the last two cases, which are compared to the ODE solver of numpy. The output of dots mean that the case passed the test and the last two Fs means that it failed the case.

CHAPTER

FIVE

THE LORENZ PACKAGE

The package consists of the following modules and functions

The modules and functions in the lorenz package are listed here

Solver: lorenz attractor solver

Intro

It includes a basic solver for the lorenz attractor.

Functions

This is the solver for the differential equations of Lorenz attractor.

`lorenz.solver.lorenz_solver(state, parameters, t_d)`

Returns the array of x,y,z value after solving the ODE of the lorenz attractor using the Euler approach.

INPUT:

```
state:    x,y,z state
parameters:    sigma, rho, beta parameters
t_d:    the time step for the discrete integration
```

OUTPUT:

```
[x+1, y+1, z+1]:    The next x,y,z values
```

It is more accurate when it is close to the convergence point of the attractor.

Example:

```
>>> state = [1.0, 1.0, 1.0]
>>> parameters = [1.0, 1.0, 1.0]
>>> t_d = 0.001
>>> lorenz_solver(state, parameters, t_d)
(1.0, 0.999, 1.0)
```

Details

Solver .- It solves the differential equations with a first order euler approach which explicit math is depicted below.

$$[x_{n+1}, y_{n+1}, z_{n+1}] = [t_d\sigma(y_n - x_n) + x_n, t_d(x_n(\rho - z_n) - y_n) + y_n, t_d(x_n y_n - \beta z_n) + z_n]$$

Code

```
"""
This is the solver for the differential equations of Lorenz attractor.
"""

def lorenz_solver(state, parameters, t_d):
    """

    Returns the array of x,y,z value after solving the ODE of the lorenz
    attractor using the Euler approach.

    INPUT::

     state:    x,y,z state
     parameters:    sigma, rho, beta parameters
     t_d:    the time step for the discrete integration

    OUTPUT::

     [x+1, y+1, z+1]:    The next x,y,z values

    It is more accurate when it is close to the convergence point of the
    attractor.

    Example:

    >>> state = [1.0, 1.0, 1.0]
    >>> parameters = [1.0, 1.0, 1.0]
    >>> t_d = 0.001
    >>> lorenz_solver(state, parameters, t_d)
    (1.0, 0.999, 1.0)

    """
    x, y, z = state # get the state
    sigma, rho, beta = parameters # get the parameters
    return t_d * sigma * (y - x) + x, t_d * ( x * (rho - z) - y) + y, \
           t_d * (x * y - beta * z ) + z
```

Plot: functions for plotting

Functions

This file may contain functionalities for plotting

`lorenz.plot.`**`plot_2d`**(*selection*, *states*, *save=False*, *fname='experimental'*, *directory=None*)

Plot the x,y and x,z and y,z coordinates of the lorenz attractor

INPUT:

```
selection:    which to plot
states:    array of [x,y,z]
save:    bool value
fname:    name of file
directory: route or name of folder
```

OUTPUT:

```
Plots and pdf figures
```

Example

```
>>> plot_2d("xy", states)
```

`lorenz.plot.`**`plot_3d_states`**(*states*, *save=False*, *fname='experimental'*, *directory=None*)

Plot the x,y,z coordinates of the lorenz attractor

INPUT:

```
states:    array of [x,y,z]
save:    bool value
fname:    name of file
directory: route or name of folder
```

OUTPUT:

```
Plots and pdf figures
```

Example

```
>>> plot_3d_states(states)
```

Code

```
"""
This file may contain functionalities for plotting
"""
import os
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D

def plot_3d_states(states, save = False, fname = 'experimental',
                   directory = None):
    """
    Plot the x,y,z coordinates of the lorenz attractor

    INPUT::

     states:    array of [x,y,z]
     save:    bool value
     fname:    name of file
     directory: route or name of folder

    OUTPUT::

     Plots and pdf figures
```

```
    Example

    >>> plot_3d_states(states)

    """
    fig = plt.figure()
    ax = Axes3D(fig)
    #ax = fig.add_subplot(111, projection='3d')
    #ax = fig.gca(projection='3d')
    #ax = fig.add_subplot(111, projection = '3d')
    ax.plot(states[:,0], states[:,1], states[:,2])
    plt.title('Lorenz Attractor')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    if save:
        if directory != None:
            if not os.path.exists(directory):
                os.makedirs(directory)
            fig.savefig( directory + '/' + fname + '_3d.pdf' )
        else:
            fig.savefig( fname + '_3d.pdf' )
    plt.show()
    return

def plot_2d(selection, states, save = False, fname = 'experimental',
            directory = None):
    """
    Plot the x,y and x,z and y,z coordinates of the lorenz attractor

    INPUT::

     selection:    which to plot
     states:    array of [x,y,z]
     save:    bool value
     fname:    name of file
     directory: route or name of folder

    OUTPUT::

     Plots and pdf figures

    Example

    >>> plot_2d("xy", states)

    """
    plt.figure()
    if selection == "xy":
        """
        PLot xy graph
        """
        plt.plot(states[:,0],states[:,1])
        plt.title("Lorenz Attractor XY")
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid()
```

```
        if save:
            if directory != None:
                if not os.path.exists(directory):
                    os.makedirs(directory)
                plt.savefig( directory + '/' + fname + '_xy.pdf' )
            else:
                plt.savefig( fname + '_xy.pdf' )
        plt.show()
    elif selection == "xz":
        """
        Plot xz graph
        """
        plt.plot(states[:,0],states[:,2])
        plt.title("Lorenz Attractor XZ")
        plt.xlabel('X')
        plt.ylabel('Z')
        plt.grid()
        if save:
            if directory != None:
                if not os.path.exists(directory):
                    os.makedirs(directory)
                plt.savefig( directory + '/' + fname + '_xz.pdf' )
            else:
                plt.savefig( fname + '_xz.pdf' )
        plt.show()
    elif selection == "yz":
        """
        Plot yz graph
        """
        plt.plot(states[:,1],states[:,2])
        plt.title("Lorenz Attractor YZ")
        plt.xlabel('Y')
        plt.ylabel('Z')
        plt.grid()
        if save:
            if directory != None:
                if not os.path.exists(directory):
                    os.makedirs(directory)
                plt.savefig( directory + '/' + fname + '_yz.pdf' )
            else:
                plt.savefig( fname + '_yz.pdf' )
        plt.show()
    else:
        print ("The selection was not correct")
    return
```

File Handling: File handling for saving data for reproducible research

Functions

This file can contain functionalities for saving/loading data

`lorenz.filehandling.`**`load_all`**(*fname*)

Save the variables, parameters and results in a hdf5 file

INPUT:

```
fname:    name of file
```

OUTPUT:

```
sigma:    sigma parameter
rho:    rho parameter
beta:    beta parameter
x:    x initial state
y:    y initial state
z:    z initial state
t_d:    time differential
N:    number of samples
states:    array of states
```

Example:

```
>>> fname = "experimental"
>>> [sigma, rho, beta, x, y, z, t_d, N, states] = load_all(fname)
```

`lorenz.filehandling.`**`save_all`**(*fname*, *sigma*, *rho*, *beta*, *x*, *y*, *z*, *t_d*, *N*, *states*, *directory=None*)

Save the variables, parameters and results in a hdf5 file

INPUT:

```
fname:    name of file
sigma:    sigma parameter
rho:    rho parameter
beta:    beta parameter
x:    x initial state
y:    y initial state
z:    z initial state
t_d:    time differential
N:    number of samples
states:    array of states
directory:    directory for saving
```

OUTPUT:

```
A file with all the variables in it.
```

Example:

```
>>> save_all(fname, sigma, rho, beta, x, y, z, t_d, N, states)
```

Code

```
"""
This file can contain functionalities for saving/loading data
"""

import h5py, os

def save_all(fname, sigma, rho, beta, x, y, z, t_d, N, states, directory = None):
    """
    Save the variables, parameters and results in a hdf5 file
```

```
    INPUT::

     fname:    name of file
     sigma:    sigma parameter
     rho:    rho parameter
     beta:    beta parameter
     x:    x initial state
     y:    y initial state
     z:    z initial state
     t_d:    time differential
     N:    number of samples
     states:    array of states
     directory:    directory for saving

    OUTPUT::

     A file with all the variables in it.

    Example:

    >>> save_all(fname, sigma, rho, beta, x, y, z, t_d, N, states)

    """
    if directory != None:
        if not os.path.exists(directory):
            os.makedirs(directory)
        f = h5py.File( directory + '/' + fname + '.hdf5', 'w')
    else:
        f = h5py.File( fname + '.hdf5', 'w')

    f.create_dataset( 'sigma', data = sigma)
    f.create_dataset( 'rho', data = rho)
    f.create_dataset( 'beta', data = beta)
    f.create_dataset( 'x', data = x)
    f.create_dataset( 'y', data = y)
    f.create_dataset( 'z', data = z)
    f.create_dataset( 't_d', data = t_d)
    f.create_dataset( 'N', data = N)
    f.create_dataset( 'states', data = states)
    f.close()

def load_all(fname):
    """

    Save the variables, parameters and results in a hdf5 file

    INPUT::

     fname:    name of file

    OUTPUT::

     sigma:    sigma parameter
     rho:    rho parameter
     beta:    beta parameter
     x:    x initial state
     y:    y initial state
     z:    z initial state
```

```
    t_d:    time differential
    N:    number of samples
    states:    array of states

  Example:

  >>> fname = "experimental"
  >>> [sigma, rho, beta, x, y, z, t_d, N, states] = load_all(fname)


  """
  f = h5py.File( fname + '.hdf5', 'r')
  sigma = f [ 'sigma' ]
  sigma = sigma [...]
  rho = f [ 'rho' ]
  rho = rho [...]
  beta = f [ 'beta' ]
  beta = beta [...]
  x = f [ 'x' ]
  x = x [...]
  y = f [ 'y' ]
  y = y [...]
  z = f [ 'z' ]
  z = z [...]
  t_d = f [ 't_d' ]
  t_d = t_d [...]
  N = f [ 'N' ]
  N = N [...]
  states = f [ 'states' ]
  states = states [...]
  return sigma,rho,beta,x,y,z,t_d,N,states
```

Run: Code for running all together

Functions

This file may contain a convenient interface/function for

1: computing a trajectory using an ODE solver from solver.py 2: save data to file 3: plot data and possible another function that

2: load data from file 3: plot data

`lorenz.run.`**`load_lorenz`** (*fname='Test'*, *plot=True*)

This function will load the lorenz and plot it

INPUT:

```
fname:    the name of the file to open
plot:    bool to plot or not
```

OUTPUT:

```
Nothing, just plots
```

Example:

```
>>> load_lorenz('Test', True)
```

`lorenz.run.`**`run_lorenz`**(*parameters, ini_state=[0.1, 0.1, 0.1], t_d=0.001, N=50000, plot=False, save=False, fname='Test', directory=None*)

Return the states, save the variables, plots and save the plots using the auxiliary functions provided.

INPUT:

```
parameters:    the parameters of the lorenz attractor
ini_state:    initial state of the lorenz attractor
t_d:    the time differential
N:    number of samples
plot:    bool to plot
save:    bool for saving
fname:    name of file
directory:    directory to save
```

OUTPUT:

```
states:    the states of the lorenz attractor in the span of
           the differential time and number of samples
```

Example:

```
>>> param = [10, 2, 6]
>>> run_lorenz(param)
```

Code

```
"""
This file may contain a convenient interface/function for

1: computing a trajectory using an ODE solver from solver.py
2: save data to file
3: plot data
and possible another function that

2: load data from file
3: plot data

"""
import numpy as np
import sys, os
sys.path.append(os.path.join(os.path.dirname(__file__), ".."))
import lorenz.plot as pl
import lorenz.solver as sol
import lorenz.filehandling as fh
import lorenz.util as ut
import matplotlib.pyplot as plt



def run_lorenz(parameters, ini_state = [0.1, 0.1, 0.1], t_d = 1e-3,
               N = 50000, plot = False, save = False, fname = 'Test',
               directory = None):
```

```
    """
    Return the states, save the variables, plots and save the plots
    using the auxiliary functions provided.

    INPUT::

     parameters:    the parameters of the lorenz attractor
     ini_state:    initial state of the lorenz attractor
     t_d:    the time differential
     N:    number of samples
     plot:    bool to plot
     save:    bool for saving
     fname:    name of file
     directory:    directory to save

    OUTPUT::

     states:    the states of the lorenz attractor in the span of
                the differential time and number of samples

    Example:

    >>> param = [10, 2, 6]
    >>> run_lorenz(param)

    """
    sigma, rho, beta = parameters
    x, y, z = ini_state
    #t_d = t/N
    #time = [i*t_d for i in range(N)] #get each discrete time
    states = np.array([[x,y,z]]) #create the array
    t = t_d * N
    ti = np.arange(0.0, t, t_d)
    cntr = 1
    for i in ti[:-1]:
        states = np.concatenate((states,
             [sol.lorenz_solver(states[cntr-1,:], parameters, t_d)]))
        cntr = cntr + 1
        #states = np.concatenate((states,
        #    [sol.lorenz_solver(states[i-1,:], parameters, t_d)]))
    if plot:
        pl.plot_3d_states(states, save, fname, directory) #plot x,y,z
        pl.plot_2d("xy", states, save, fname, directory) #plot xy
        pl.plot_2d("xz", states, save, fname, directory) #plot xz
        pl.plot_2d("yz", states, save, fname, directory) #plot yz
    if save:
        fh.save_all(fname, sigma, rho, beta, x, y, z, t_d, N, states,
                    directory)
    return states

def load_lorenz(fname = 'Test', plot = True):
    """
    This function will load the lorenz and plot it

    INPUT::

     fname:    the name of the file to open
     plot:    bool to plot or not
```

```
    OUTPUT::

     Nothing, just plots

    Example:

    >>> load_lorenz('Test', True)
    """
    [s2,r3,b2,x2,y2,z2,t_d2,N2,st2] = fh.load_all(fname)
    if plot:
        pl.plot_3d_states(st2)
    return

if __name__ == '__main__':
    """
    This is just debugging code
    """
#print("This is a convenient interface for running the simulation \
#      of a lorenz attractor")
    #sigma = ut.my_input_float("sigma") #input all parameters
    #rho = ut.my_input_float("rho")
    #beta = ut.my_input_float("beta")
    #x = ut.my_input_float("x") #input initial conditions
    #y = ut.my_input_float("y")
    #z = ut.my_input_float("z")
    #ini_state = x, y, z
    sigma = 10
    rho = 6
    beta = 8/3
    ini_state = [1.0, 1.0, 1.0]
    t_d = 0.01
    #t = ut.my_input_int("time in seconds")
    parameters = (sigma, rho, beta)
    N = 80000 #assign 50000 steps as suggested
    t = t_d * N #timestep calculated from simulation time
    #states = np.array([[x,y,z]]) #create the array
    states = np.array([ini_state])
    for i in range(0,N-1):
        states = np.concatenate((states,
            [sol.lorenz_solver(states[i-1,:], parameters, t_d)]))

    pl.plot_3d_states(states) #plot x,y,z
    #pl.plot_2d("xy", states) #plot xy
    #pl.plot_2d("xz", states) #plot xz
    #pl.plot_2d("yz", states) #plot yz

    #fh.save_all('testo',sigma,rho,beta,x,y,z,t,t_d,states)
    #[s2,r3,b2,x2,y2,z2,t2,t_d2,st2] = fh.load_all('testo')

    #pl.plot_3d_states(st2)

    """
    This is for testing. For testing the function.
    """
    w_states = ut.wikipedia_lorenz([sigma, rho, beta], ini_state, t_d, N, plot = True)

    fig = plt.figure()
```

```
    ax = fig.gca(projection='3d')
    ax.plot(w_states[0000:70000,0], w_states[0000:70000,1], w_states[0000:70000,2])
    plt.legend(['wiki'])
    plt.title('Wikipedia')
    plt.show()

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot(states[0000:70000,0], states[0000:70000,1], states[0000:70000,2])
    plt.legend(['my'])
    plt.title(' mine')
    plt.show()

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot(states[:,0], states[:,1], states[:,2])
    ax.plot(w_states[:,0], w_states[:,1], w_states[:,2])
    plt.legend(['my','wiki'])
    plt.title('Wikipedia vs mine')
    plt.show()


    ss = states - w_states
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot(ss[:,0], ss[:,1], ss[:,2])
    plt.legend(['my'])
    plt.title('diff Wikipedia vs mine')
    plt.show()
#

#
#     import numpy as np
#     import matplotlib.pyplot as plt
#     from scipy.integrate import odeint
#     from mpl_toolkits.mplot3d import Axes3D
#
#     rho = 28.0
#     sigma = 10.0
#     beta = 8.0 / 3.0
#
#     def f(state, t):
#         x, y, z = state  # unpack the state vector
#         return sigma * (y - x), x * (rho - z) - y, x * y - beta * z  # derivatives
#
#     state0 = [1.0, 1.0, 1.0]
#     t = np.arange(0.0, 40.0, 0.01)
#
#     states = odeint(f, state0, t)
#
#     fig = plt.figure()
#     ax = fig.gca(projection='3d')
#     ax.plot(states[:,0], states[:,1], states[:,2])
#     plt.show()
```

Util: Utility code

Functions

This file may contain utility functionalities.

lorenz.util.**my_input_float**(*var*)

This function is for validation of entering a float value.

INPUT:

```
var:    name of variable to display.
```

OUTPUT:

```
variable:    float input variable
```

Example:

```
>>> a = my_input_float("a")
```

lorenz.util.**my_input_int**(*var*)

This function is for validation of entering an int value.

INPUT:

```
var:    name of variable to display.
```

OUTPUT:

```
variable:    int input variable
```

Example:

```
>>> a = my_input_int("a")
```

lorenz.util.**wikipedia_lorenz**(*parameters, ini_state=[1.0, 1.0, 1.0], t_d=0.0008, N=50000, plot=False*)

This is a Lorenz attractor taken from the wikipedia page It will solve the lorenz attractor with a built-in solver odeint of numpy

INPUT:

```
parameters:    array of parameters of the lorenz attractor
ini_state:    initial state
t_d:    time difference
N:    number of samples
plot:    bool value for ploting
```

OUTPUT:

```
states:    array of states
```

Example:

```
>>> state = [1.0, 1.0, 1.0]
>>> parameters = [1.0, 1.0, 1.0]
>>> t_d = 0.001
>>> states = lorenz_solver(parameters, state, t_d, plot = True)
```

Code

```
"""
This file may contain utility functionalities.
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D

def my_input_float(var):
    """

    This function is for validation of entering a float value.

    INPUT::

     var:    name of variable to display.

    OUTPUT::

     variable:    float input variable

    Example:

    >>> a = my_input_float("a")

    """
    while True:
        try:
            variable = float(input("Please enter "+ var +": "))
        except ValueError:
            print("Sorry, I didn't understand that.")
            continue
        else:
            break
    return variable

def my_input_int(var):
    """

    This function is for validation of entering an int value.

    INPUT::

     var:    name of variable to display.

    OUTPUT::

     variable:    int input variable

    Example:

    >>> a = my_input_int("a")

    """
    while True:
```

```
        try:
            variable = int(input("Please enter "+ var +": "))
        except ValueError:
            print("Sorry, I didn't understand that.")
            continue
        else:
            break
    return variable

def wikipedia_lorenz(parameters, ini_state = [1.0, 1.0, 1.0],
                     t_d = 8e-4, N = 50000, plot = False):
    """
    This is a Lorenz attractor taken from the wikipedia page
    It will solve the lorenz attractor with a built-in solver
    odeint of numpy

    INPUT::

     parameters:    array of parameters of the lorenz attractor
     ini_state:    initial state
     t_d:    time difference
     N:    number of samples
     plot:    bool value for ploting

    OUTPUT::

     states:    array of states

    Example:

    >>> state = [1.0, 1.0, 1.0]
    >>> parameters = [1.0, 1.0, 1.0]
    >>> t_d = 0.001
    >>> states = lorenz_solver(parameters, state, t_d, plot = True)

    """
    sigma, rho, beta = parameters
    x, y, z = ini_state
    #t_d = t/N

    def f(state, t):
        x, y, z = state  # unpack the state vector
        return sigma * (y - x), x * (rho - z) - y, x * y - beta * z  # derivatives


    t_t = t_d * N
    t = np.arange(0.0, t_t, t_d)

    states = odeint(f, ini_state, t)
    if plot:
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot(states[:,0], states[:,1], states[:,2])
        plt.title('Wikipedia Lorenz Attractor')
        plt.show()
    return states
```

PYTHON MODULE INDEX

l

INDEX