Exercise No. 9

David Bubeck, Pascal Becht, Patrick Nisblè

June 23, 2017

1 2 - The Lorenz attractor

The Lorenz attractor problem is given by the following coupled set of differentail equations:

$$\dot{x} = -\sigma(x - y) \tag{1}$$
$$\dot{y} = rx - y - xz \tag{2}$$
$$\dot{z} = xy - bz \tag{3}$$

Our fixed points are $(0, 0, 0)$ for all r and $C_\pm = (\pm a_0, \pm a_0, r - 1)$ with $a_0 = \sqrt{b(r - 1}$ for $r > 1$. For this exercise we use $\sigma = 10$ and $b = \frac{8}{3}$.

a)

Using rk4 we solve the set of equations above numerically for $r = 0.5, 1.15, 1.3456, 24$ and 30.

Following is the code for rk4 calculation:

```python
#! /usr/bin/python3

from typing import List, Callable
import matplotlib.pyplot as plt
import numpy as np

def rKN(x: np.ndarray, fx: List[Callable], hs: float):
    k1 = np.array(x)
    k2 = np.array(x)
    k3 = np.array(x)
    k4 = np.array(x)

    l = len(fx)

    for i in range(l):
        k1[i] = fx[i](x)*hs

    for i in range(l):
        k2[i] = fx[i](x + k1*.5)*hs

    for i in range(l):
        k3[i] = fx[i](x + k2*.5)*hs

    for i in range(l):
        k4[i] = fx[i](x + k3)*hs

    return x + (k1 + 2*(k2 + k3) + k4)/6

if __name__ == '__main__':

    def diffeq(x: np.ndarray):
        return -1*x

    nsteps = 10
    maxx = 10.
    ndata = np.linspace(0, maxx, nsteps+1)
    xlist = np.array(ndata, ndmin=2).T
    xlist[0][0] = 1

    for i in range(0,xlist.shape[0]-1):
        out = rKN(xlist[i], [diffeq], maxx/nsteps)
        xlist[i+1] = out

    plt.yscale('log')
    plt.plot(ndata, np.array(xlist))
    plt.savefig('rk4-test.png')
```

The problem will be solved with:

```python
import numpy as np
from rk4 import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

sigma = 10
b = 8/3
nsteps = 1000
stepsize = .01

for u,r in enumerate([.5,1.15,1.3456,24,30]):

    a0 = np.sqrt(b*(r-1))

    fsys = [
        lambda x: -sigma*(x[0] - x[1]), # x
        lambda x: r*x[0] - x[1] - x[0]*x[2], # y
        lambda x: x[0]*x[1] - b*x[2] # z
    ]

    xlist = np.ndarray(shape=(nsteps+1,3), dtype=float)
    if r == .5:
        xlist[0] = np.array([[
            .1,.1,.1
        ]])
    else:
        xlist[0] = np.array([[
            a0+.1*r,
            a0-.5*r,
            r-1
        ]])

    for i in range(nsteps):
        xlist[i+1] = rKN(xlist[i],fsys, stepsize)

    fig = plt.figure(figsize=(15, 10))
    ax = fig.gca(projection='3d')
    ax.plot(xlist[:,0], xlist[:,1], xlist[:,2])
    ax.scatter([a0,-a0,0],[a0,-a0,0],[r-1,r-1,0], color='r')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.savefig('plot_{}.png'.format(u))
```

Here are the plots for the solutions given with the code above

In figure 1 you can see that the function converge fast against point $(0, 0, 0)$ in which lies our fix point. In figure 2 our function converge also against a fix point, but here it is the C_+ fix point. All following values lies on this point, so it is stabel. Starting at figure 4 the function forms butterfly. The function circulates around the fix points but never converge. So for large r we don't have stable fix points.

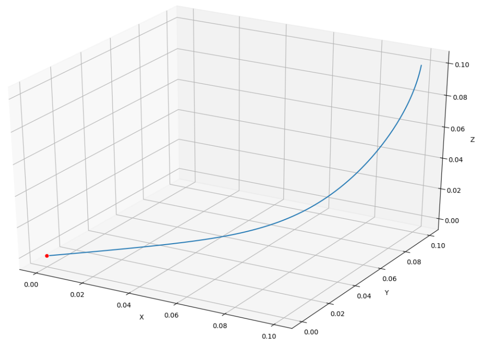

Figure 1: r = 0.5

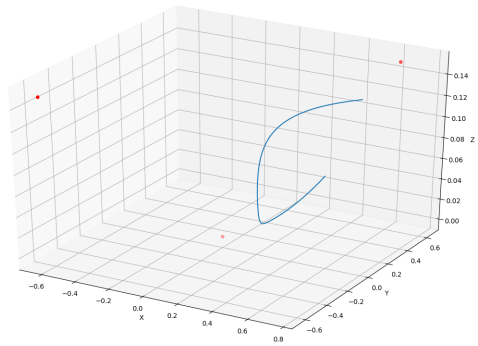

Figure 2: r = 1.15

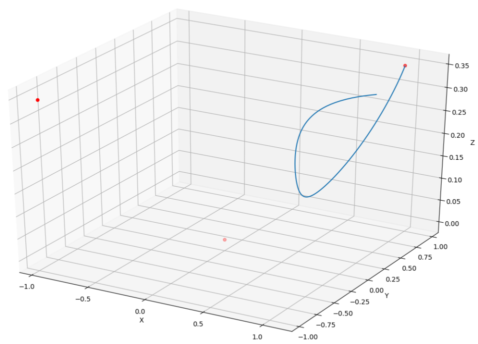

Figure 3: r = 1.3456

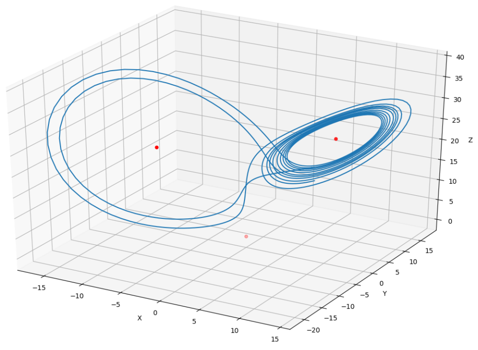

Figure 4: r = 24

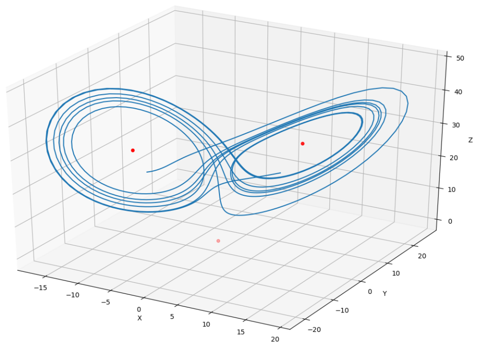

Figure 5: r = 26

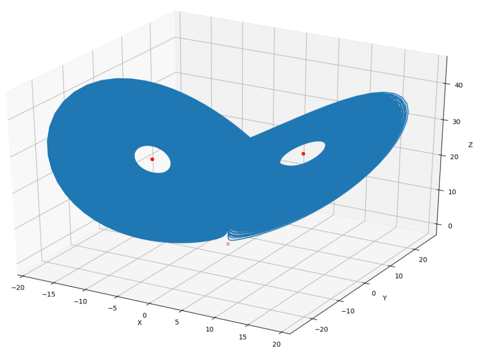

Figure 6: r = 30

b)

For the second part we determine for $r = 26$ the local maximum in z and plot z_{k+1} as a function of z_k

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#runge kutta 4 for the functions of x, y and z with defined functions x, y, z
def rk4x(x_dot, x, y, sig, h):
    k1 = x_dot(x, y, sig)
    k2 = x_dot(x + (h / 2) * k1, y, sig)
    k3 = x_dot(x + (h / 2) * k2, y, sig)
    k4 = x_dot(x + h * k3, y, sig)

    return( 1 / 6 * (k1 + (2 * k2) + (2 * k3) + k4))

def x_dot(x = 1, y = 1, sig = 10):
    return(-sig * (x - y))

def rk4y(y_dot, x, y, z, r, h):
    k1 = y_dot(x, y, z, r)
    k2 = y_dot(x, y + (h / 2) * k1, z, r)
    k3 = y_dot(x, y + (h / 2) * k2, z, r)
    k4 = y_dot(x, y + h * k3, z, r)

    return( 1 / 6 * (k1 + (2 * k2) + (2 * k3) + k4))

def y_dot(x = 1, y = 1, z = 1, r = 1):
    return((r * x) - y - (x * z))

def rk4z(z_dot, x, y, z, b, h):
    k1 = z_dot(x, y, z, b)
    k2 = z_dot(x, y, z + (h / 2) * k1, b)
    k3 = z_dot(x, y, z + (h / 2) * k2, b)
    k4 = z_dot(x + h * k3, y, z, b)

    return( 1 / 6 * (k1 + (2 * k2) + (2 * k3) + k4))

def z_dot(x = 1, y = 1, z = 1, b = 8 / 3):
    return((x * y) - (b * z))

#function to find maximum of z
def maxfinder(l = []):
    zk = []
    for k in range(1, len(l) - 1):
        if(l[k] > l[k - 1] and l[k] > l[k + 1]):
            zk.append(l[k])
    return zk

x_list = []
y_list = []
z_list = []
t_list = []
```

```python
#starting values
r = 26
sig = 10
b = 8 / 3
h = 0.01
a = np.sqrt(b * (r - 1))

#fix points
c_plus = [a - 0.5, a, r - 1]
c_minus = [-a, -a, r - 1]

#starting values in space of fix point
x_list.append(c_plus[0])
y_list.append(c_plus[1])
z_list.append(c_plus[2])
t_list.append(0)

#calculating values
for i in range(100000):
    x_list.append(x_list[i] + h * rk4x(x_dot, x_list[i], y_list[i], sig, h))
    y_list.append(y_list[i] + h * rk4y(y_dot, x_list[i], y_list[i], z_list[i], r,
        h))
    z_list.append(z_list[i] + h * rk4z(z_dot, x_list[i], y_list[i], z_list[i], b,
        h))
    t_list.append((i + 1) * h)

zk = maxfinder(z_list)
del zk[len(maxfinder(z_list)) - 1]
zk1 = maxfinder(z_list)
del zk1[0]

plt.plot(zk, zk1, '*b')
plt.plot(zk, zk, '-r')
plt.xlabel('z_k')
plt.ylabel('z_k+1')
plt.savefig('z_k.png', dpi=300)
plt.show()
```

We can see that the slope at the fixed point is approximately -1, which is the limit for a pereodic solution.

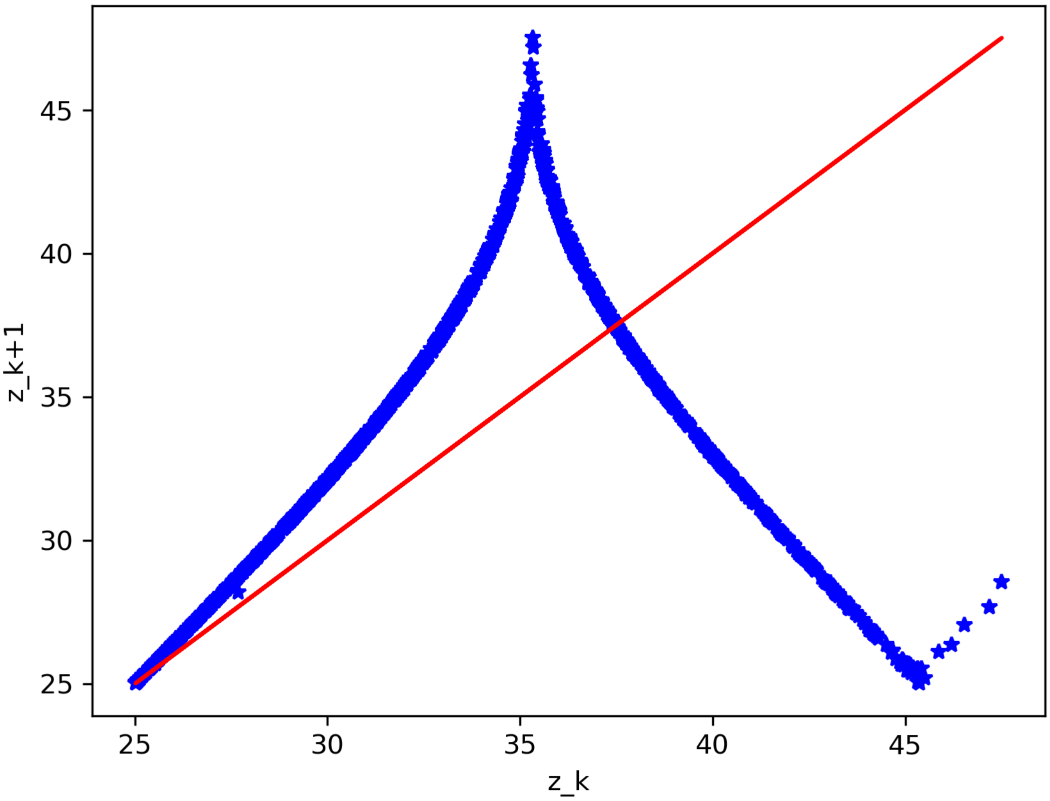

Figure 7: r = 26