Introduction to Computational Physics – Exercise 9

Simon Groß-Bölting, Lorenz Vogel, Sebastian Willenberg

June 26, 2020

The Lorenz Attractor

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

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

As discussed in the lecture, the fixed points are $(0\,,\,0\,,\,0)$ for all r, and (for $r>1$) the points $C_\pm = (\pm a_0\,,\,\pm a_0\,,\,r-1)$ with $a_0 = \sqrt{b(r-1)}$. For the entire exercise, please use $\sigma = 10$ and $b = 8/3$. The value of r can be experimented with. When you create numerical solutions you can make plots in 2-D projection (e.g. in the (x,y)- or (x,z)-plane). You can also try a full 3-D plot.

Task: Solve numerically, using `rk4`, the above coupled set of equations for the values $r = 0.5$, 1.17, 1.3456, 25.0 and 29.0. Choose the initial conditions near one of the fixed points: $C_\pm$ for $r>1$ and $(0\,,\,0\,,\,0)$ for $r<1$. Explain the behavior, as much as possible, with the stability properties of the fixed points.

To solve the problem numerically we can use the Runge-Kutta Algorithm to approximate the functions x, y and z. It is important to note that $x(t)$, $y(t)$ and $z(t)$ are time-dependent. We will start by importing the packages that we will need for this exercise:

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

The `numpy` package allows us to work with arrays while the `matplotpib.pyplot` and `mplot3d` packages will allow us to plot our data in a three-dimensional graph. The next step is to define our parameters:

```python
# set initial conditions
sigma = 10
b = 8/3
r = np.array([0.5,1.17,1.3456,25.0,29.0])
a0 = np.sqrt(b*(r[r>1.]-1))
```

The coupled set of differential equations are defined in the following manner:

```python
# Lorenz system: coupled set of equations
def dx(x,y,z,sigma):
    return -sigma*(x-y)
def dy(x,y,z,r):
    return r*x-y-x*z
def dz(x,y,z,b):
    return x*y-b*z
```

All the differential equations in our Lorenz system are dependent of the functions `x`, `y` and `z` and a variable σ, r or b. That means that we have to rewrite our `rk4` algorithm in such a way that we will iterate over `x`, `y` and `z` instead of iterating over `y` and `t` (or rather `x`). The `rk4_step` function looks as follows:

```python
def rk4_step(x0, y0, z0, fx, fy, fz, h, fx_args={}, fy_args={}, fz_args={}):
    ''' Simple python implementation for one RK4 step.
        Inputs: (i=x,y,z)
            i0 - M x 1 numpy arrays specifying all variables of the three
                 ODE's at the current time step
            f  - function that calculates the derivates of all variables of the ODE
            h  - step size
            fi_args - dictionary of additional arguments to be passed to the function fi
        Output: (i=x,y,z)
            ip1 - M x 1 numpy array of variables of the first ODE at time step i0+h '''
    k1_x = h*fx(x0, y0, z0, **fx_args)
    k1_y = h*fy(x0, y0, z0, **fy_args)
    k1_z = h*fz(x0, y0, z0, **fz_args)

    k2_x = h*fx(x0+k1_x/2., y0+k1_x/2., z0+k1_x/2., **fx_args)
    k2_y = h*fy(x0+k1_y/2., y0+k1_y/2., z0+k1_y/2., **fy_args)
    k2_z = h*fz(x0+k1_z/2., y0+k1_z/2., z0+k1_z/2., **fz_args)

    k3_x = h*fx(x0+k2_x/2., y0+k2_x/2., z0+k2_x/2, **fx_args)
    k3_y = h*fy(x0+k2_y/2., y0+k2_y/2., z0+k2_y/2, **fy_args)
    k3_z = h*fz(x0+k2_z/2., y0+k2_z/2., z0+k2_z/2, **fz_args)

    k4_x = h*fx(x0+k3_x/2., y0+k3_x/2., z0+k3_x/2, **fx_args)
    k4_y = h*fy(x0+k3_y/2., y0+k3_y/2., z0+k3_y/2, **fy_args)
    k4_z = h*fz(x0+k3_z/2., y0+k3_z/2., z0+k3_z/2, **fz_args)

    xp1 = x0+1./6.*(k1_x+2.*k2_x+2.*k3_x+k4_x)
    yp1 = y0+1./6.*(k1_y+2.*k2_y+2.*k3_y+k4_y)
    zp1 = z0+1./6.*(k1_z+2.*k2_z+2.*k3_z+k4_z)
    return (xp1,yp1,zp1)

def rk4(x0, y0, z0, fx, fy, fz, h, n, fx_args={}, fy_args={}, fz_args={}):
    ''' Simple implementation of RK4
        Inputs: (i=x,y,z)
            i0 - M x 1 numpy arrays specifying all variables of the three
                 ODE's at the current time step
            f  - function that calculates the derivates of all variables of the ODE
            h  - step size
            fi_args - dictionary of additional arguments to be passed to the function fi
        Output: (i=x,y,z)
            in - N+1 x M numpy array with the results of the integration for
                 every time step (includes i0) '''
    xn = np.zeros(n+1); yn = np.zeros(n+1); zn = np.zeros(n+1)
    xn[0] = x0; yn[0] = y0; zn[0] = z0

    for n in np.arange(1,n+1,1):
        xn[n], yn[n], zn[n] = rk4_step(x0=xn[n-1], y0=yn[n-1], z0=zn[n-1],
                                       fx=fx, fy=fy, fz=fz, h=h, fx_args=fx_args,
                                       fy_args=fy_args, fz_args=fz_args)
    return (xn,yn,zn)
```

The only thing that is left to get the results is to plot our values:

```
N = 10000   # number of steps
h = 0.001   # step size

for i in range(0,len(r)):
    # choose the initial conditions near one of the fixed points
    if r[i] < 1: x0 = 1. ; y0 = 1. ; z0 = 1.
    else: x0 = a0[i-1]+1.; y0 = a0[i-1]+1. ; z0 = (r[i]-1.)+1.

    # solve numerically the above coupled set of equations (using Runge-Kutta-4)
    xn, yn, zn = rk4(x0=x0, y0=y0, z0=z0, fx=dx, fy=dy, fz=dz, h=h, n=N,
                     fx_args={'sigma':sigma}, fy_args={'r':r[i]}, fz_args={'b':b})

    # plot the solution (3-D plot)
    fig = plt.figure(); ax = fig.add_subplot(111, projection='3d')
    ax.set_title(r'Lorenz Attractor for $r=${}'.format(r[i]))
    ax.scatter(xn,yn,zn, marker='.', s=1, c='blue')
    ax.scatter([x0],[y0],[z0], marker='x', s=25, c='green')
    ax.scatter([x0-1.],[y0-1.],[z0-1.], marker='x', s=25, c='red')
    ax.set_xlabel(r'$x$-axis'); ax.set_ylabel(r'$y$-axis'); ax.set_zlabel(r'$z$-axis')
    fig.savefig('figures/3D-Plot_Lorenz-Attractor_r={}.pdf'.format(r[i]), format='pdf')

plt.show(); plt.clf(); plt.close()
```

It is important to note here that our initial conditions were set to $(x_0\,,\,y_0\,,\,z_0) = (1\,,\,1\,,\,1)$ in the case that $r < 1$. That means that our initial conditons are always real.

Task: Determine the sequence z_k for $r = 26.5$, where z_k is a local maximum in $z$ on the solution curve after k periods. Plot z_{k+1} as a function of z_k. When sufficient points are there, connect the points.

The resulting function $z_{k+1} = f(z_k)$ has an intersection with the diagonal $z_{k+1} = z_k$. It is a fixed point of the function $f(z_k)$. Is the slope m of this function > 1, < -1 or between -1 and $+1$? Notice: The theory of discrete maps says that there is no periodic solution if $|m| > 1$. So, in such a case we can deduce that this solution of the Lorenz system is not periodic.