Lab 6 – Sage Tutorial

1. Discrete (time) dynamical systems

Orbits of discrete dynamical systems defined by

$$x_{n+1}=f(x_n)$$

can be easily implemented in **Sage** by iteration, using a basic **for loop.**

It is, however, convenient to use a zero vector (or a list, but experts tend to favor vectors) of length equal to the total number of iterations pus one *(N+1)* as a starting point and iteratively change the *n*-th component of the vector. This way, all iterations are saved – or, in other words, the orbit is created. After *N* interations the original zero vector has been modified to

$$(x_0, x_1, \ldots, x_n, \ldots, x_N).$$

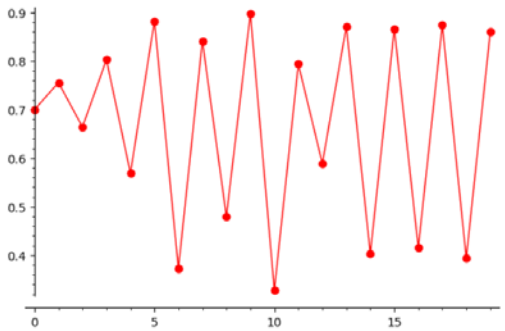
This can now be plotted using standard commands. (If the dynamic is complex, it might be convenient to connect orbit points by segments.)

For example

```
N=20
r = 3.6
x = zero_vector(RR, N+1)
x[0] = 0.7
for n in range(1,N):
        x[n] = r*x[n-1]*(1-x[n-1])
        p = line(Line(range(N),x,{}),color='red', marker = 'o')

show(p)
```

produces



2. Numerical methods for ODEs

Both numerical methods implemented from scratch by the user and those which are predefined in **Sage** are just discrete dynamical systems which iteratively generate orbits that approximate the continuous-time dynamics.

Obviously, a large variety of possible modifications/adaptations exists (including variable step size, etc.) but the principle remains the same.

One can generate the sequence of time steps in advance, i.e., outside the for loop, or inside the for loop. (The latter allows for variable or adaptive step sizes.)

Below, a built-in **Sage** numerical integrator ('desolve_odeint') is compared to two numerical schemes implemented from scratch (Euler and improved Euler methods)

```
h=0.1
x=srange(0,2,h)
ye =  desolve_odeint(f, 0, x, y) # the solution of the ODE IVP

N = 21

y = zero_vector(RR, N) # the Euler method
y[0] = 0
yy = zero_vector(RR, N) # improved Euler method
yy[0] = 0


for i in range(1,N-1):
    y[i] = y[i-1]+h*(y[i-1]^2 + x[i-1]^2) # the Euler method
    yy[i] = yy[i-1] + 0.5*h*(yy[i-1]^2 + x[i-1]^2) + 0.5*h*((yy[i-1]+h*(yy[i-1]^2 + x[i-1]^2))^2 + x[i]^2) # improved
```

and the plot commands (recall that 'line(zip(x,ye))'-like sequences have been already used in Lab 5 when plotting numerically generated solutions of ode/systems)

```
py = line(Line(x,y,{}),color='blue', marker='o')
pyy = line(Line(x,yy,{}),color='red', marker='o')

P_tot = line(zip(x,ye)) + py + pyy

show(P_tot)
```

then produce the desired graphical representation