SIR Modelling

The *SIR* Model for disease propagation consists on a system of simulatneous differential equations:

$$\begin{cases} S'(t) = -\lambda \dfrac{I(t)S(t)}{N} \\[2ex] I'(t) = \lambda \dfrac{I(t)S(t)}{N} - \mu I(t) \\[2ex] R'(t) = \mu I(t) \end{cases}$$

where *S(t)* is the amount of individuals suceptible to getting infected, *I(t)* the amount of infected and *R(t)* the amount of recovered in the instant t for a contant population.

The purpose for this investigation is to analyze the effect of the contagion parameter (or infection rate) λ in the 'curve flattening' phenomenon.

Resolution The initial conditions set for this model are the following:

$I(0) = 0$, $S(0) = 990$, $R(0) = 0$,
$\lambda = 1{,}0\ day^{-1}$, $\mu = 0{,}3\ day^{-1}$ and $N = 1000$ (population).

In order to solve this, we are required to apply a numerical method: 2^{nd} Order Runge-Kutta Method.

The following curves for $I(t)$, $S(t)$ and $R(t)$ where obtained with the code attached in the annex, using the following integration parameters:

- $t_0 = 0$
- $t_f = 30$
- $h = 0{,}1$

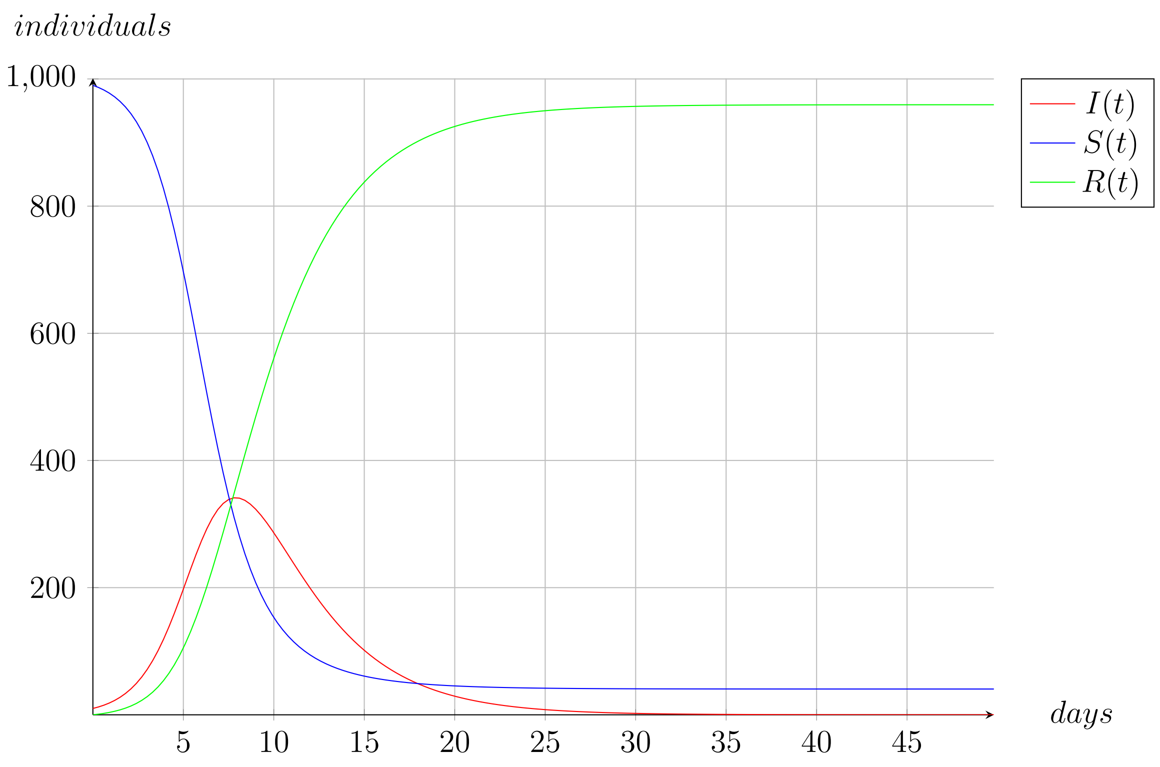

Figura 1: $I(t)$, $S(t)$ y $R(t)$

To investigate the duration of the epidemic, even if the time taken for the amount of infected to reach its Maximum is less than the amount of time taken for the disease to disappear from the population, $I(t) = 0$, as the amount of infected decreases, so does the amount of individuals susceptible to the disease. Thus, instead of considering the duration of the epidemic to be the period of time in which $S(t) > I(t)$, we can estimate a duration of $t_e = 2 \times t_{max}$.

With this criterion in mind, a $\{\lambda_i; t_{e,i}\}$ data set was created, which can be visualized in the following plot:

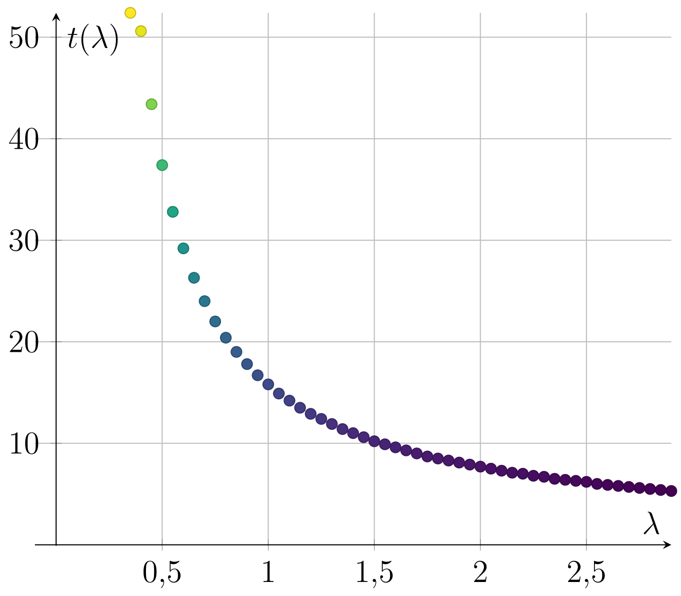

Figura 2: $t(\lambda)$. As the value of λ increases, t_{max} happens faster.

The following link contains an animation showing the behaviour of $I(t)$, $S(t)$ y $R(t)$ as the value of λ changes: Epidemic

The curve flattening effect is obtained when the value of λ is smaller, since the value of the t_{max} increases, thus, delaying the infection peak.

This is one type of model that can be used to analyze the behaviour of an epidemic. Several similar models also consider changes in the population, by including deaths and births during the epidemic duration, and also, some, consider vaccination and re-infection, make those models more complex, but also, more representative of what happens in real-life disease spreading. Understanding the values of parameters such as the infection rate, the recovery rate and also, additionally, vaccination rates can help comprehend the measurements that need to be taken in order to control and prevent a disease from spreading.

Annex

SIR Modelling code for lambda variation

```python
import numpy as np
import matplotlib.pyplot as plt

def f(t,x1,x2,x3):
    '''
    Funcion: f(t,x)
    Proposito: establece la EDO-2D x'=f(t,x)
    Arg-in: t(float)
            x (list)
    Arg-out: f(list)
    '''
    return [l * x1 * x2/N - u * x1, -l * x1 * x2/N, u * x1]

##Lambda
l0 = 0.3
lf = 3.3
s = 0.05
m = np.int((lf-l0)/s)
l_matriz = np.zeros(m+1)
l = l0

l_matriz[0] = l0

##Initial conditions
t0 = 0.0
x10 = 10.0
x20 = 990.0
x30 = 0.0

##Constants
u = 0.3
N = 1000


h = 0.1
tf = 30
n = int((tf-t0)/h)
t = np.zeros(n+1)
x = np.zeros((n+1, 3))  #matriz de n+1 x 3 llena de ceros


t[0] = t0
x[0] = [x10, x20, x30]


t_pico = 0
matriz_duracion_epid = np.zeros(m+1)

for j in range(m):
    l = l_matriz[j]
    print("iterating with lambda = ", l_matriz[j])
    for i in range(n):
        t[i+1] = t[i] + h
        p = np.multiply(f(t[i], x[i,0], x[i,1], x[i,2]),h)
        k1 = p[0]
        l1 = p[1]
        j1 = p[2]
        q = np.multiply(f(t[i+1], x[i,0] + k1, x[i,1] + l1, x[i,2] + j1),h)
        k2 = q[0]
        l2 = q[1]
        j2 = q[2]
        x[i+1,0] = x[i,0] + (k1 + k2)/2
        x[i+1,1] = x[i,1] + (l1 + l2)/2
        x[i+1,2] = x[i,2] + (j1 + j2)/2


        #For I(t) Max
        if (x[i-1,0] < x[i,0] and x[i,0] > x[i+1,0]):
            t_pico = t[i]
            matriz_duracion_epid[j] = t_pico*2
            print("max time = ", t_pico)

        #For t where I(t) = S(t)
        if (x[i-1,0] > x[i-1,1] and x[i+1,0] < x[i+1,1]):
            print("2nd time when I(t) = S(t) = ", t[i])

        #For t where R(t) = I(t)
        if (x[i-1,0] > x[i-1,2] and x[i+1,0] < x[i+1,2]):
            print("time when I(t) = R(t) = ", t[i])

        #For t where S(t) = R(t)
        if (x[i-1,1] > x[i-1,2] and x[i+1,1] < x[i+1,2]):
            print("time when S(t) = R(t) = ", t[i])

    l_matriz[j+1] = l_matriz[j] + s


    plt.plot(t,x[:,:1], label='i(t)')
    plt.plot(t,x[:,1:2], label='s(t)')
    plt.plot(t,x[:,2:3], label='r(t)')
    plt.legend()
    plt.xlabel('$days$')
    plt.ylabel('$individuals$')
    plt.savefig('grafico.jpg')
    plt.show()

#print(x)

matriz_duracion_epid[m] = t_pico*2
plt.scatter(l_matriz,matriz_duracion_epid)
plt.xlabel('$\lambda$')
plt.ylabel('$t(\lambda)$')
plt.savefig('grafico.jpg')
plt.show()

for i in range(m+1):
    print('%f'% l_matriz[i], '%f' % matriz_duracion_epid[i])
```