In [1]:

```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
from GetModelParameters import *
from scipy.integrate import solve_ivp
from geneticalgorithm import geneticalgorithm as ga
```

In [2]:

```
def thetaModel(t, y, N, gammas, p):
S, E, I, I_u, Hr, Hd = y
actual = math.floor(t)
gamma_E, gamma_I, gamma_Iu, gamma_Hr, gamma_Hd, gamma_Q = gammas.__fields__()
omega, tau1, tau2, omega_u = p[actual,0], p[actual,1], p[actual,2], p[actual,3]
theta, rho = p[actual,4], p[actual,5]
beta_e, beta_I, beta_Iu, beta_hr, beta_hd = p[actual,6], p[actual,7], p[actual,8], p[ac
betas_mul = beta_e*E + beta_I*I + beta_Iu*I_u + beta_hr*Hr + beta_hd*Hd
newE = E/gamma_E
newI = I/gamma_I
newIu = I_u/gamma_Iu
newHr = Hr/gamma_Hr
newHd = Hd/gamma_Hd
dSdt = -(S/N) * betas_mul
dEdt = (S/N) * betas_mul - newE + tau1 - tau2
dIdt = newE - newI
dIudt = (1 - theta - omega_u) * newI - newIu
dHrdt = rho*(theta - omega) * newI - newHr
dHddt = omega * newI - newHd
return dSdt, dEdt, dIdt, dIudt, dHrdt, dHddt
```

In [3]:

```
class OptimizeModel:
def __init__(self, path, t0, tMAX, lambdas, ms_val):
self.data = Data(path)
self.tspan = np.arange(t0, tMAX)
self.dates = get_lambdas(lambdas, self.data)
self.N = self.data.population[t0]
self.u0 = [
self.data.susceptible[t0],
self.data.exposed[t0],
self.data.infec[t0],
self.data.infec_u[t0],
self.data.hospitalized[t0],
self.data.hospitalized[t0]*0.3,
]
self.ms_val = ms_val
def distance(self, X):
gamma_E, gamma_I, gamma_Iu, gamma_Hr, gamma_Hd, gamma_Q, c3, c5, omega_u0, max_omega, m
beta_I0_min, beta_e0 = c_E * beta_I0, c_u * beta_I0
gammas = Gammas(gamma_E, gamma_I, gamma_Iu, gamma_Hr, gamma_Hd, gamma_Q)
delays = Delays(gammas)
times = Times(self.tspan, self.data, delays)
ms = get_Ms(
self.tspan[0], self.dates, self.ms_val,
np.array([k2]),
np.array([c3, c5])
)
p = parameters_list(times, self.data, gammas, delays, ms, self.dates, max_omega, min_om
```

```
sol = solve_ivp(thetaModel, self.tspan, self.u0, args=(self.N, gammas, p), dense_output
u = sol.sol(self.tspan)
distance = math.sqrt(np.sum(list(
map(
lambda x: x**2,
u.T[:,2] - self.data.infec
)
)))
#print(distance)
return distance
def genetic_alg(self, bounds, params, timeOut=10.0):
model = ga(
function=self.distance,
dimension=bounds.shape[0],
variable_type='real',
variable_boundaries=bounds,
algorithm_parameters=params,
function_timeout=timeOut
)
return model
```