Plotting of the state variables

```
#Define Parameters here!!!
##############################
# This defines a functions that will be used in the ODEs

def alpha(t):
    return (e-(m*t)*(alpha_1+alpha_1*alpha_2*np.sin((2*np.pi*t/182.5))))

def epsilon_a(t):
    return ((epsilon_a1+epsilon_a1*epsilon_a2*np.sin((2*np.pi*t/182.5))))

def epsilon_s(t):
    return ((epsilon_s1 + epsilon_s1* epsilon_s2*np.sin((2*np.pi*t/182.5))))
#####################################################################
# Give your Initial conditions
##################################
# Your initial conditions should be written in a vector, because your results will be written in
this form
Y0 = [ S0, E0, I_a0, I_s0, R0, C_e0 ]
# Define your maximum time for your simulations
# Time vector for solution
T = np.arange(0, tMax, 1)
# This defines a function that is the right-hand side of the ODEs
def rhs(Y, t):
    '''
    SEIR model.

    This function gives the right-hand sides of the ODEs.
    '''

    # Convert vector to meaningful component vectors
    # Note: Indices start with index 0, not 1!
    S = Y[0]
    E = Y[1]
    I_a = Y[2]
    I_s = Y[3]
    R = Y[4]
    C_e = Y[5]
    # The right-hand sides
    dS =  L -  alpha(t)*(I_a /N + e*I_s /N + (C_e/(C_e + K)))*S - mu*S
    dE =  alpha(t)*(I_a /N + e*I_s /N + (C_e/(C_e + K)))*S - (mu + kappa)*E
    dI_a = kappa*p*E - (mu + a + gamma_a )*I_a
    dI_s = kappa*(1 - p)*E + a*I_a - (mu + x + gamma_s )*I_s
    dR = gamma_a*I_a + gamma_s*I_s - mu*R
    dC_e = epsilon_a(t) + epsilon_s(t) - sigma*C_e
```

```
    # Convert meaningful component vectors into a single vector
    dY = [ dS, dE, dI_a, dI_s, dR, dC_e ]


    return dY
# Integrate the ODE
# Warning!  The ODE solver over-writes the initial value.
# This can be a pain in the ass if you run the ODE multiple times.
# Also, 'args' passes parameters to right-hand-side function.
solution = scipy.integrate.odeint(rhs,Y0,T)
S = solution[:, 0]
E = solution[:, 1]
I_a = solution[:,2]
I_s = solution[:,3]
R = solution[:,4]
C_e = solution[:,5]
# Make plots

# Load a plotting package
#plt.figure() # calls the figure environment
#plt.plot(T, I_s,'b',linewidth=2, label = 'm = 0.2')
#plt.plot(T, C_e,'b',linewidth=2, label = 'C_e')
#plt.scatter(xdata, ydata, linewidth=2, label = 'data')
#plt.xlabel('Time(days)')
#plt.ylabel('Infected Population (I_s)')
#plt.legend(loc='best')
#plt.xlim()
#plt.ylim()
#plt.show()
```

Fitting using Python

```
ydata =np.array([new_cases_data_here])
tmax = len(ydata)
xdata=  np.arange(0, tmax, 1)
plt.scatter(xdata, ydata)
######################Defining parameters and model equation################
def seir_model(y, x, L, beta, beta_1, t, e, mu, kappa, p, a, gamma_a, xi, gamma_s,
epsilion_a, epsilion_s, sigma):
    S = L - (beta * (1 + beta_1*np.sin(2*np.pi*t/182.5)))*((y[2] + e*y[3] + y[5])/N)*y[0] - mu*y[0]
    E = (beta * (1 + beta_1*np.sin(2*np.pi*t/182.5)))*((y[2] + e*y[3] + y[5])/N)*y[0] - (mu +
kappa)*y[1]
    I_a = kappa*p*y[1] - (mu + a + gamma_a)*y[2]
    I_s = kappa*(1 - p)*y[1] + a*y[2] - (mu + xi + gamma_s)*y[3]
    R =  gamma_a*y[2] + gamma_s*y[3] - mu*y[4]
    C_e = (epsilion_a*(1 + epsilion_a * np.sin(2*np.pi*t/182.5)))*y[2] + (epsilion_s*(1 +
epsilion_s * np.sin(2*np.pi*t/182.5)))* y[3] - sigma*y[5]
```

```
    return S, E, I_a, I_s, R, C_e

##########################Solving ode##############
def fit_odeint(x, L, beta, beta_1, t, e, mu, kappa, p, a, gamma_a, xi, gamma_s, epsilion_a,
epsilion_s, sigma):
    return integrate.odeint(seir_model, (S0, E0, I_a0, I_s0, R0, C_e0), x, args=(L, beta,
beta_1, t, e, mu, kappa, p, a, gamma_a, xi, gamma_s, epsilion_a, epsilion_s, sigma))[:,3]
######################
Initial_condition_for_state_variable
######################
N
E0
I_a0
I_s0
R0
C_e0
S0 = N - (E0 + I_a0 + I_s0 + R0 + C_e0 )
##############finding the optimized values#######you can do this for all the waves##
popt, pcov = optimize.curve_fit(fit_odeint, xdata, ydata,
        p0=(Initial_condition_parameter))
fitted = fit_odeint(xdata, *popt)
#########Plotting############
plt.figure()
plt.plot(xdata, ydata, 'o', label = 'Data')
plt.plot(xdata, fitted, linewidth=4, color = 'r', label = 'Model')
plt.legend(loc='best')
plt.xlim(xmin=0)
plt.ylim(ymin=0)
plt.xlabel('Days')
plt.show()
```