06.22.24

DATA ASSIMILATION

SIR epidemiologic model with lock-down effect

Azganush Sargsyan
Davit Kobaidze

INTRODUCTION

- Extend the SIR epidemiologic model
- To analyze how a lock-down, affects the spread of the disease
- To estimate the reduction factor
- To implement numerical simulations
- To employ the Gauss-Newton algorithm for parameter estimation.

THE SIR MODEL

The SIR model is a set of ordinary differential equations (ODEs) used to describe the spread of infectious diseases. It divides the population into three compartments:

- $S(t)$: Susceptible individuals
- $I(t)$: Infected individuals
- $R(t)$: Recovered individuals

The model equations are:

$$\frac{dS(t)}{dt} = -\tau S(t)I(t)$$

$$\frac{dI(t)}{dt} = \tau S(t)I(t) - \nu I(t)$$

$$\frac{dR(t)}{dt} = \nu I(t)$$

LOCK-DOWN EFFECT

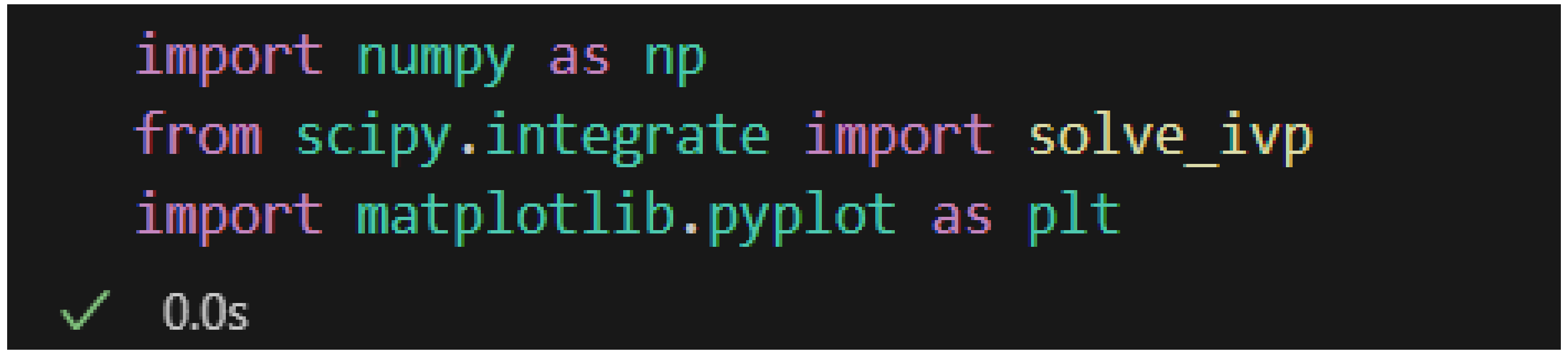
MODIFICATION OF THE MODEL

A lock-down affects the transmission coefficient τ. We introduce a time-dependent transmission coefficient $\tau(t)$:

$$\tau(t) = \begin{cases} \tau & \text{if } t \in [0, t_1) \\ f\tau & \text{if } t \in [t_1, t_2) \\ \tau & \text{if } t \in [t_2, T] \end{cases}$$

where t_1 and t_2 are the start and end of the lock-down period, and f is the reduction factor (0 < f < 1).

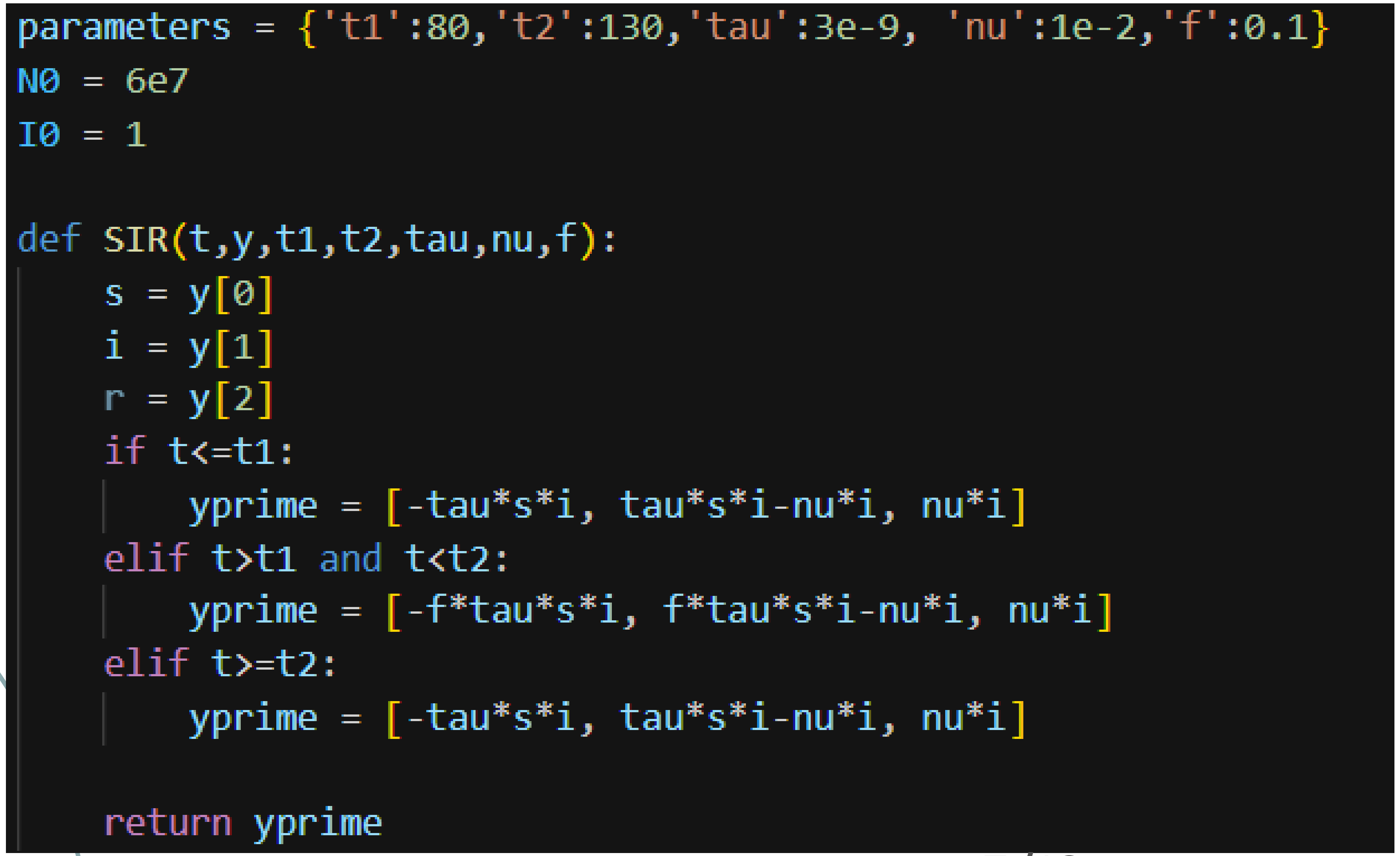
CODE IMPLEMENTATION



```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
```

✓ 0.0s

We implement the SIR model with lock-down effect using Python's solve_ivp function from the scipy.integrate module. The function defining the SIR model is



```
parameters = {'t1':80,'t2':130,'tau':3e-9, 'nu':1e-2,'f':0.1}
N0 = 6e7
I0 = 1

def SIR(t,y,t1,t2,tau,nu,f):
    s = y[0]
    i = y[1]
    r = y[2]
    if t<=t1:
        yprime = [-tau*s*i, tau*s*i-nu*i, nu*i]
    elif t>t1 and t<t2:
        yprime = [-f*tau*s*i, f*tau*s*i-nu*i, nu*i]
    elif t>=t2:
        yprime = [-tau*s*i, tau*s*i-nu*i, nu*i]

    return yprime
```

DERIVATIVE CALCULATIONS

Determine Effective Transmission Rate (τ_{eff}):

- If $t \leq t1$:
  - $\tau_{\text{eff}} = \tau$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial f} = 0$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial \tau} = 1$
- If $t1 < t < t2$:
  - $\tau_{\text{eff}} = f\tau$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial f} = \tau$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial \tau} = f$
- If $t \geq t2$:
  - $\tau_{\text{eff}} = \tau$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial f} = 0$
  - $\frac{\partial(\tau_{\text{eff}})}{\partial \tau} = 1$

Derivatives of State Variables:

- $\frac{dS(t)}{dt} = -\tau_{\text{eff}} s i$
- $\frac{dI(t)}{dt} = \tau_{\text{eff}} s i - \nu i$
- $\frac{dR(t)}{dt} = \nu i$

Derivatives with Respect to τ:

- $\frac{d}{d\tau}\left(\frac{dS(t)}{dt}\right) = -\frac{\partial(\tau_{\text{eff}})}{\partial \tau} s i - \tau_{\text{eff}} \frac{\partial S(t)}{\partial \tau} i - \tau_{\text{eff}} s \frac{\partial I(t)}{\partial \tau}$
- $\frac{d}{d\tau}\left(\frac{dI(t)}{dt}\right) = \frac{\partial(\tau_{\text{eff}})}{\partial \tau} s i + \tau_{\text{eff}} \frac{\partial S(t)}{\partial \tau} i + \tau_{\text{eff}} s \frac{\partial I(t)}{\partial \tau} - \nu \frac{\partial I(t)}{\partial \tau}$
- $\frac{d}{d\tau}\left(\frac{dR(t)}{dt}\right) = \nu \frac{\partial I(t)}{\partial \tau}$

Derivatives with Respect to ν:

- $\frac{d}{d\nu}\left(\frac{dS(t)}{dt}\right) = -\tau_{\text{eff}} \frac{\partial S(t)}{\partial \nu} i - \tau_{\text{eff}} s \frac{\partial I(t)}{\partial \nu}$
- $\frac{d}{d\nu}\left(\frac{dI(t)}{dt}\right) = -i + \tau_{\text{eff}} \frac{\partial S(t)}{\partial \nu} i + \tau_{\text{eff}} s \frac{\partial I(t)}{\partial \nu} - \nu \frac{\partial I(t)}{\partial \nu}$
- $\frac{d}{d\nu}\left(\frac{dR(t)}{dt}\right) = i + \nu \frac{\partial I(t)}{\partial \nu}$

Derivatives with Respect to f (During Lock-Down):

- $\frac{d}{df}\left(\frac{dS(t)}{dt}\right) = -\frac{\partial(\tau_{\text{eff}})}{\partial f} s i - \tau_{\text{eff}} \frac{\partial S(t)}{\partial f} i - \tau_{\text{eff}} s \frac{\partial I(t)}{\partial f}$
- $\frac{d}{df}\left(\frac{dI(t)}{dt}\right) = \frac{\partial(\tau_{\text{eff}})}{\partial f} s i + \tau_{\text{eff}} \frac{\partial S(t)}{\partial f} i + \tau_{\text{eff}} s \frac{\partial I(t)}{\partial f} - \nu \frac{\partial I(t)}{\partial f}$
- $\frac{d}{df}\left(\frac{dR(t)}{dt}\right) = \nu \frac{\partial I(t)}{\partial f}$

CODE IMPLEMENTATION

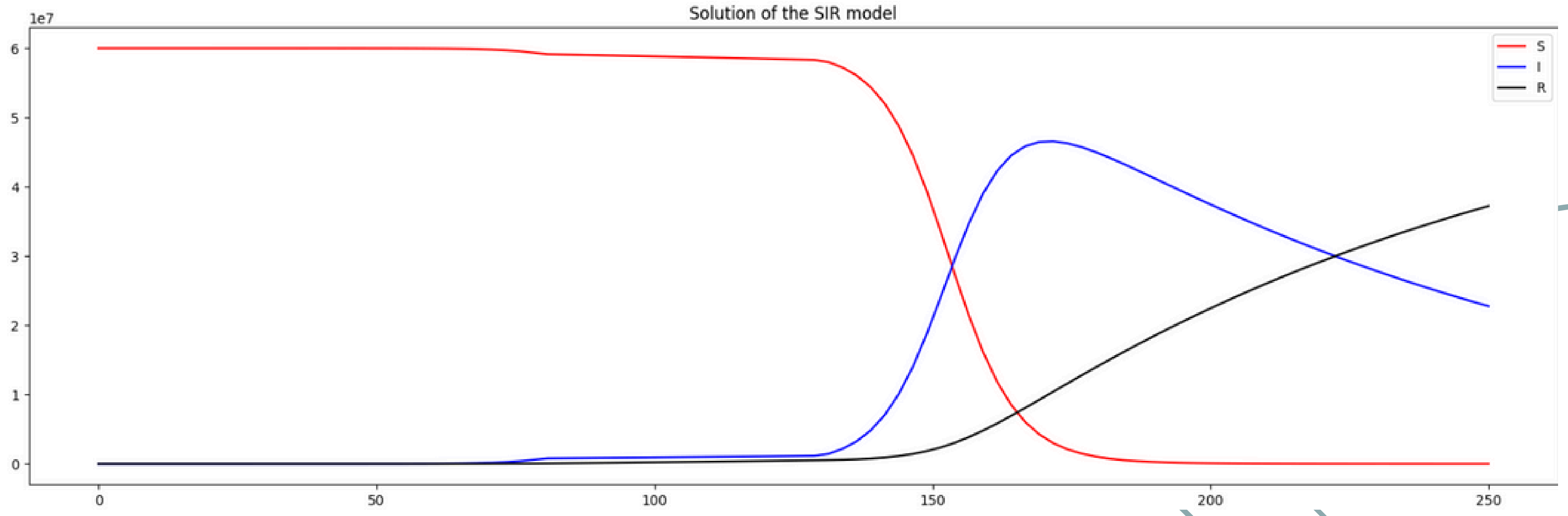
```
def SIRDSIR(t,Y,t1,t2,tau,nu,f):
    # d1 : derivative wrt tau ; d2 : derivative wrt nu, d3 : derivative wrt f
    s = Y[0]
    i = Y[1]
    r = Y[2]
    if t<=t1:
        taueff = tau
        dftaueff = 0
        dtautaueff = 1
    elif t>t1 and t<t2:
        taueff = f*tau
        dftaueff = tau
        dtautaueff = f
    elif t>=t2:
       taueff = tau
       dftaueff = 0
       dtautaueff = 1
    d1s = Y[3]
    d1i = Y[4]
    d1r = Y[5]
    d2s = Y[6]
    d2i = Y[7]
    d2r = Y[8]
    d3s = Y[9]
    d3i = Y[10]
    d3r = Y[11]
    dts = -taueff*s*i
    dti = taueff*s*i-nu*i
    dtr = nu*i
    dtd1s = -dtautaueff*s*i -taueff*d1s*i -taueff*s*d1i
    dtd1i = dtautaueff*s*i + taueff*d1s*i + taueff*s*d1i-nu*d1i
    dtd1r = nu*d1i
    dtd2s = -taueff*d2s*i -taueff*s*d2i
    dtd2i = -i + taueff*d2s*i + taueff*s*d2i-nu*d2i
    dtd2r = i + nu*d2i
    dtd3s = -dftaueff*s*i -taueff*d3s*i -taueff*s*d3i
    dtd3i = dftaueff*s*i +taueff*d3s*i + taueff*s*d3i-nu*d3i
    dtd3r = nu*d3i
    return np.array([dts,dti,dtr,dtd1s,dtd1i,dtd1r,dtd2s,dtd2i,dtd2r,dtd3s,dtd3i,dtd3r])
```

$$N = 6 \times 10^7, I_0 = 1,$$

$$S_0 = N - I_0, R_0 = 0,$$

```
T = 250
numpoints = 100
timepoints = np.linspace(0,T,numpoints)
def yDy(tau,nu,f):
    sol = solve_ivp(SIRDSIR,[0,T],[N0-I0,I0,0,0,0,0,0,0,0,0,0,0],t_eval=timepoints,args=(parameters['t1'],parameters['t2'],tau,nu,f),rtol=1e-13,atol=1e-5)
    return sol.y

Y_true=yDy(parameters['tau'],parameters['nu'],parameters['f'])

plt.figure(figsize=(20,6))
plt.plot(timepoints,Y_true[0,:].T,'r',label='S')
plt.plot(timepoints,Y_true[1,:].T,'b',label='I')
plt.plot(timepoints,Y_true[2,:].T,'k',label='R')
plt.legend()
plt.title('Solution of the SIR model');
```

OUTPUT



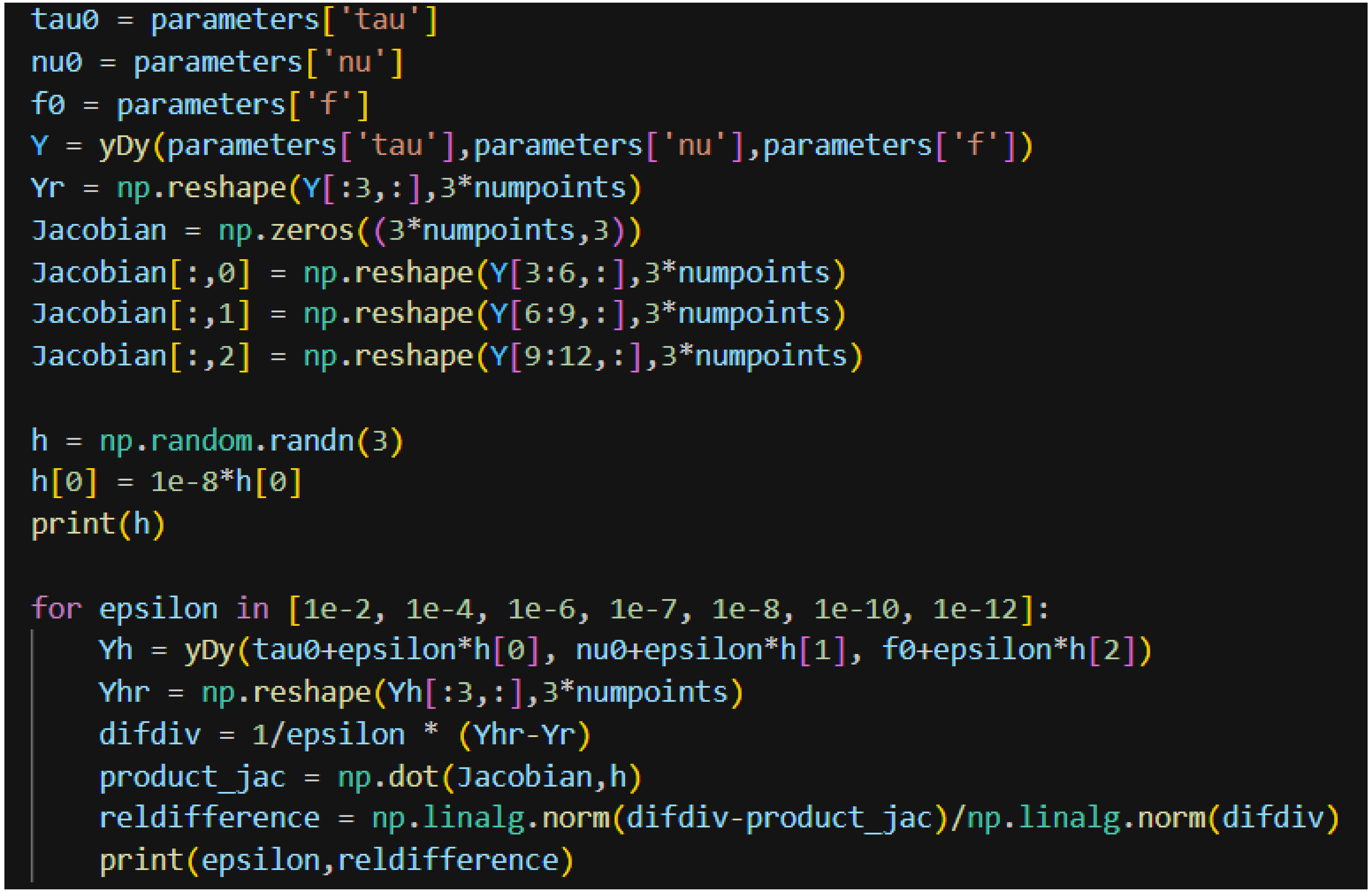
PARAMETER ESTIMATION

GAUSS-NEWTON ALGORITHM

To estimate the parameters τ, ν, and f, we use the Gauss-Newton algorithm.

1. Initialize the parameters.
2. Compute the Jacobian matrix using the derivative function.
3. Iterate to minimize the cost function $J(\theta)$.
4. Update the parameters using the computed derivatives.

CHECK THE DERIVATIVE



```
tau0 = parameters['tau']
nu0 = parameters['nu']
f0 = parameters['f']
Y = yDy(parameters['tau'],parameters['nu'],parameters['f'])
Yr = np.reshape(Y[:3,:],3*numpoints)
Jacobian = np.zeros((3*numpoints,3))
Jacobian[:,0] = np.reshape(Y[3:6,:],3*numpoints)
Jacobian[:,1] = np.reshape(Y[6:9,:],3*numpoints)
Jacobian[:,2] = np.reshape(Y[9:12,:],3*numpoints)

h = np.random.randn(3)
h[0] = 1e-8*h[0]
print(h)

for epsilon in [1e-2, 1e-4, 1e-6, 1e-7, 1e-8, 1e-10, 1e-12]:
    Yh = yDy(tau0+epsilon*h[0], nu0+epsilon*h[1], f0+epsilon*h[2])
    Yhr = np.reshape(Yh[:3,:],3*numpoints)
    difdiv = 1/epsilon * (Yhr-Yr)
    product_jac = np.dot(Jacobian,h)
    reldifference = np.linalg.norm(difdiv-product_jac)/np.linalg.norm(difdiv)
    print(epsilon,reldifference)
```

RESULTS WITH NOISE

- True Parameters Initialization
- Generate True and Noisy Observed Data
- True data

$$Y_{\text{true}} = \text{yDy}(\tau_{\text{true}}, \nu_{\text{true}}, f_{\text{true}})$$

Computes the true solution of the SIR model using adjusted true parameters τ_{true}, ν_{true}, and f_{true}.

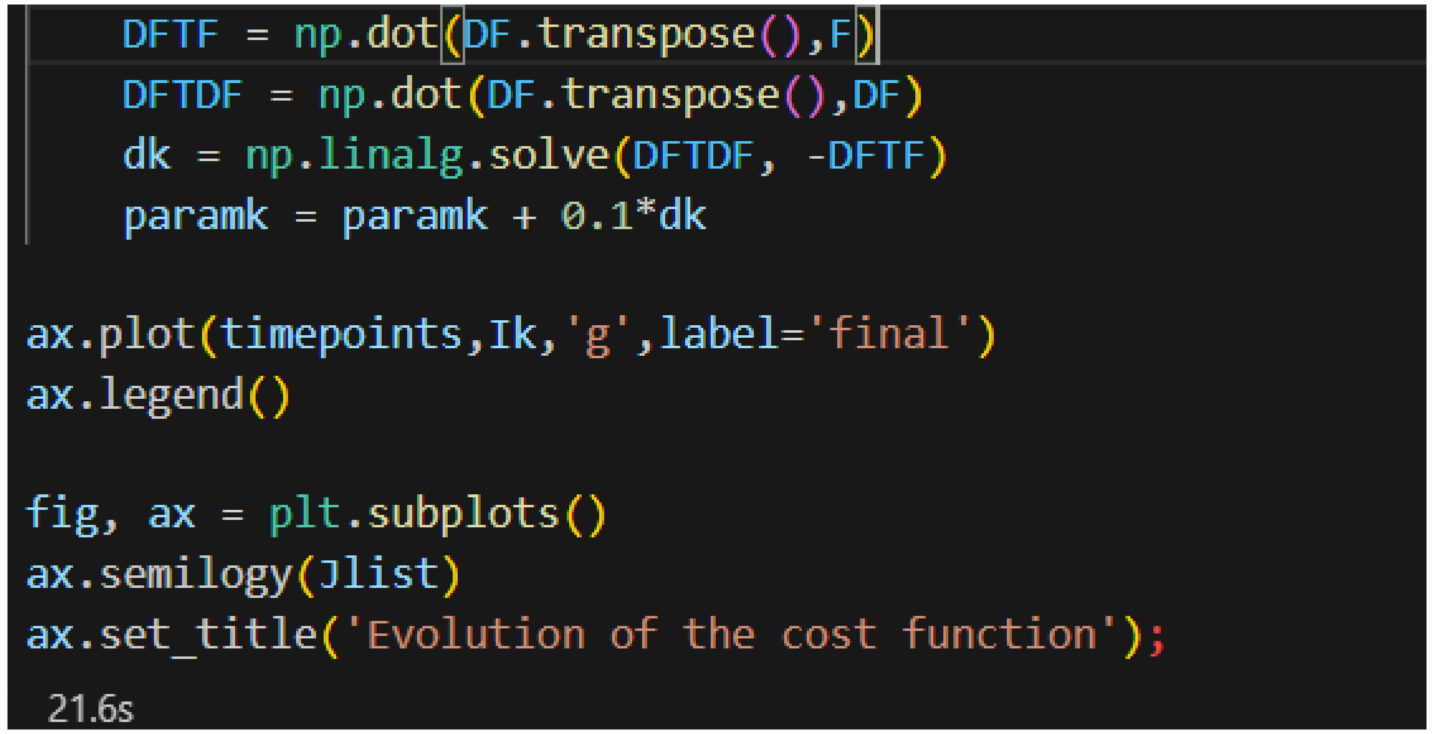
- Noisy Observed Data

$$I_{\text{obs}} = Y_{\text{true}}[1, :] + \text{noise level} \times \text{random noise}$$

Adds Gaussian noise to the true infected data $Y_{\text{true}}[1, :]$ to simulate observed (noisy) infection data I_{obs}.

CODE IMPLEMENTATION

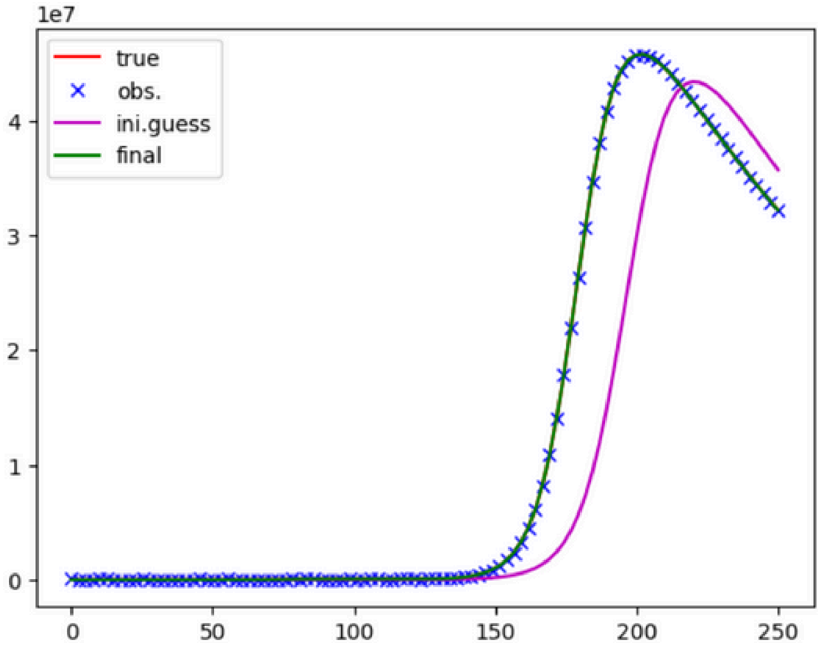
```
#parameters estimation
tautrue = 0.8*parameters['tau']
nutrue = 0.9*parameters['nu']
ftrue = 1.1*parameters['f']

Ytrue = yDy(tautrue, nutrue,ftrue)
noiselevel = 5e4
Iobs = Ytrue[1,:] + noiselevel*np.random.randn(numpoints)

fig, ax = plt.subplots()
ax.plot(timepoints,Ytrue[1,:],'r',label='true')
ax.plot(timepoints,Iobs,'bx',label='obs.')

precision = 0.2
tauini = tautrue*(1-precision+2*precision*np.random.rand())
nuini = nutrue*(1-precision+2*precision*np.random.rand())
fini = ftrue*(1-precision+2*precision*np.random.rand())

paramk = np.array([tauini, nuini,fini])
nbit = 80
Jlist = []

for it in range(nbit):
    print(it, paramk)
    Y = yDy(paramk[0], paramk[1], paramk[2])
    Ik = Y[1,:]
    if it==0:
        ax.plot(timepoints,Ik,'m',label='ini.guess')
    F = Ik - Iobs
    Jlist.append(0.5*np.linalg.norm(Ik-Iobs)**2)
    DF = np.zeros((numpoints,3))
    DF[:,0] = np.reshape(Y[4,:],numpoints)
    DF[:,1] = np.reshape(Y[7,:],numpoints)
    DF[:,2] = np.reshape(Y[10,:],numpoints)
```

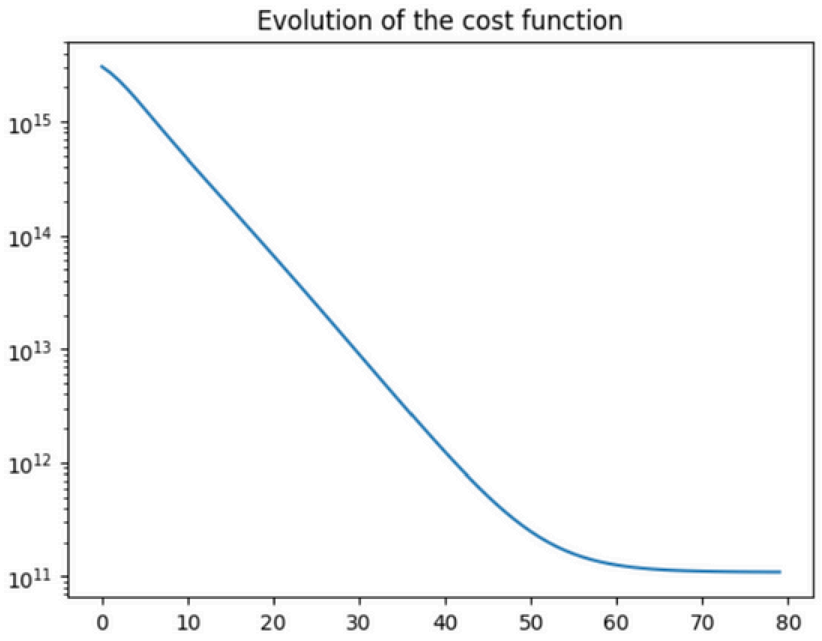


```
    DFTF = np.dot(DF.transpose(),F)
    DFTDF = np.dot(DF.transpose(),DF)
    dk = np.linalg.solve(DFTDF, -DFTF)
    paramk = paramk + 0.1*dk

ax.plot(timepoints,Ik,'g',label='final')
ax.legend()

fig, ax = plt.subplots()
ax.semilogy(Jlist)
ax.set_title('Evolution of the cost function');
```

21.6s

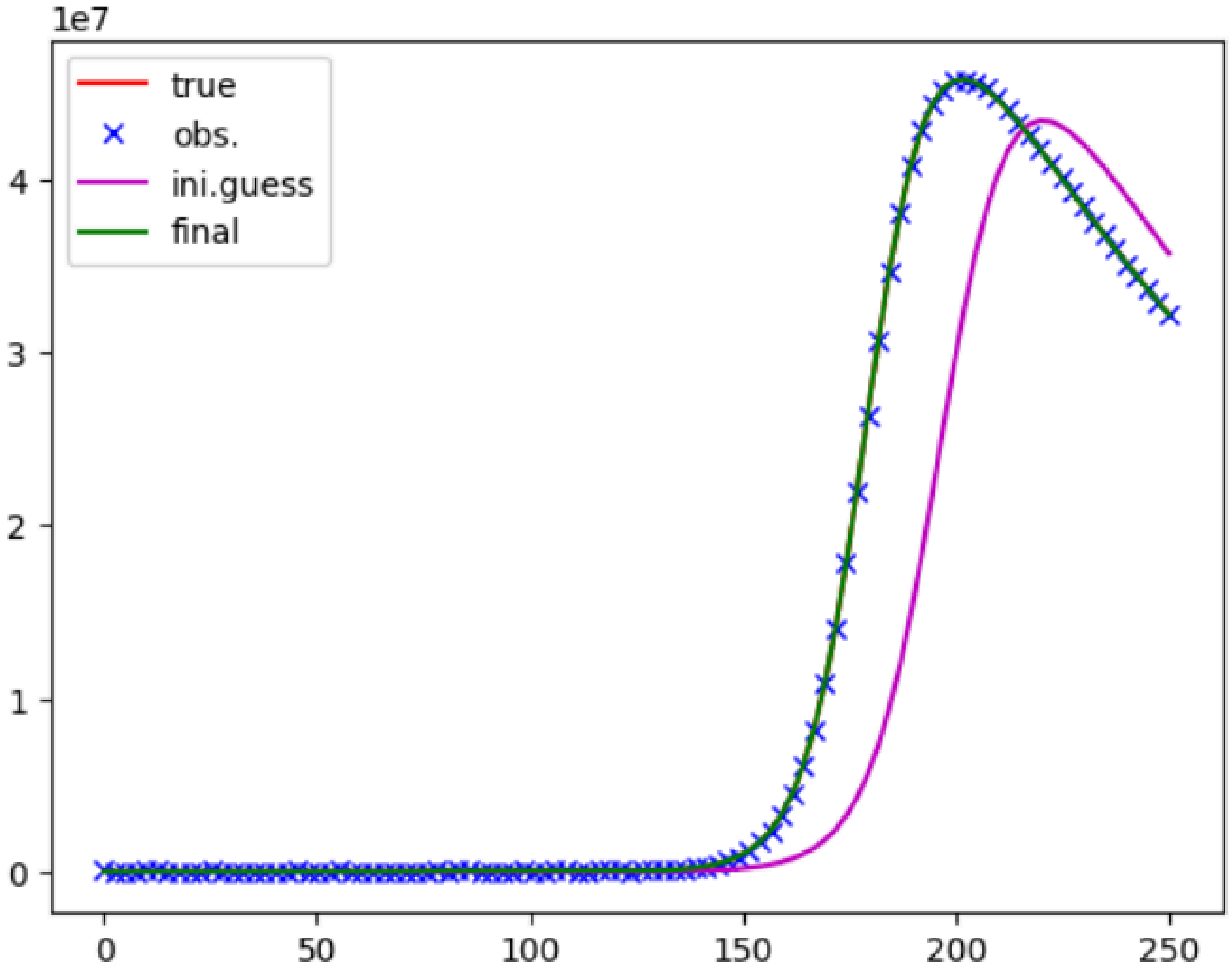
OUTPUT





RESULTS WITH NOISE

Our analysis shows that the lock-down significantly reduces the spread of the epidemic by lowering the transmission rate τ\tauτ. The Gauss-Newton algorithm successfully estimates the parameters even with noisy data, demonstrating its robustness. We also explored the influence of different time intervals [t1,t2] on the identifiability of the reduction factor f



THANK YOU!