파이썬을 이용한 과학계산
Final project

2022313047
김보연

Epidemic model

- SLIAR Model Equations and structure.

$$\begin{cases} S' &= -\beta(1-\sigma)S\Lambda - \nu S \\ L' &= \beta(1-\sigma)S\Lambda - \kappa L \\ I' &= p\kappa L - \alpha I - \tau I \\ A' &= (1-p)\kappa L - \eta A \end{cases} \qquad \text{with } \Lambda = \epsilon L + (1-q)I + \delta A$$

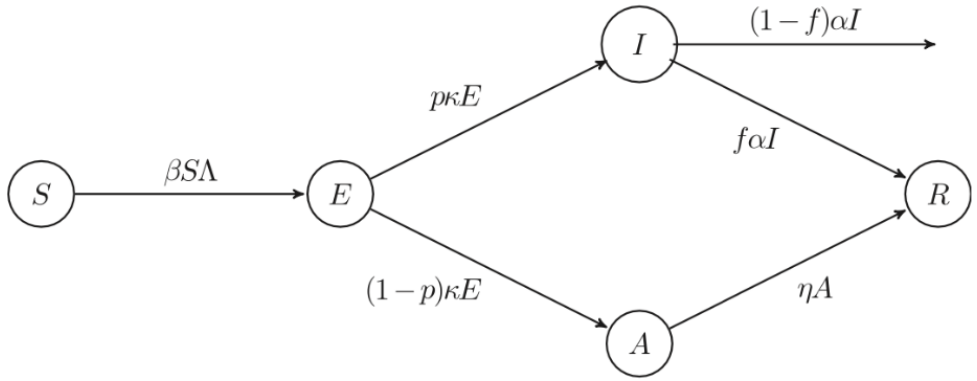


Fig. 1. SEIAR epidemic model.

Find the optimal vaccination control to minimize given objective function

$$\min_{u \in \mathcal{U}_{ad}} \int_0^T PI(t) + Q\nu^2(t)\, dt$$

subject to

$$\begin{cases} S' &= -\beta(1-\sigma)S\Lambda - \nu S \\ L' &= \beta(1-\sigma)S\Lambda - \kappa L \\ I' &= p\kappa L - \alpha I - \tau I \\ A' &= (1-p)\kappa L - \eta A \end{cases} \qquad with \quad \Lambda = \epsilon L + (1-q)I + \delta A$$

ν: vaccination

Pontryagin's Maximum Principle

- Hamilonian

$$
\begin{aligned}
H &= f + \lambda \cdot g \\
&= PI + Q\nu^2 + R\tau^2 + W\sigma^2 + \begin{bmatrix} \lambda_S \\ \lambda_L \\ \lambda_I \\ \lambda_A \end{bmatrix}' \cdot \begin{bmatrix} -\beta(1-\sigma)S\Lambda - \nu S \\ \beta(1-\sigma)S\Lambda - \kappa L \\ p\kappa L - \alpha I - \tau I \\ (1-p)\kappa L - \eta A \end{bmatrix} \\
\lambda' &= -\frac{\partial H}{\partial x}
\end{aligned}
$$

- Adjoint Equations

$$
\begin{cases}
\lambda_S' &= \nu\lambda_S + \beta(1-\sigma)\Lambda(\lambda_S - \lambda_L) \\
\lambda_L' &= \beta(1-\sigma)\epsilon S(\lambda_S - \lambda_L) - \kappa p(\lambda_I - \lambda_A) + \kappa(\lambda_L - \lambda_A) \\
\lambda_I' &= -P + \beta(1-q)(1-\sigma)S(\lambda_S - \lambda_L) + \lambda_I(\alpha + \tau) \\
\lambda_A' &= \beta(1-\sigma)\delta S(\lambda_S - \lambda_L) + \eta\lambda_A
\end{cases}
$$

Pontryagin's Maximum principle

$\frac{\partial H}{\partial u} = 0 \ \ at\ u^* \ \Rightarrow \ \ f_u + \lambda g_u = 0 \quad (optimality\ condition)$

$\lambda' = -\frac{\partial H}{\partial x} \ \Rightarrow \ \ \lambda' = -(f_x + \lambda g_x) \ \ (adjoint\ equation)$

$\lambda(t_1) = 0 \quad (transversality\ condition) \ \ and \ \ \lambda(t) \geq 0$

Then, for all controls u, we have $\boldsymbol{J(u^*) \geq J(u)}$.

Implementation functions

```
import sympy as sp
```

```
# State
S, L, I, A = sp.symbols('S L I A')
state = [S, L, I, A]

# Control
u = sp.symbols('u')

# Parmaeters
alpha, beta, sigma, epsilon, q, delta, kappa, p, tau, eta, P, Q = sp.symbols('alpha beta sigma epsilon q delta kappa p tau eta P Q')
params = [alpha, beta, sigma, epsilon, q, delta, kappa, p, tau, eta, P, Q]

# Adjoints
l_S, l_L, l_I, l_A = sp.symbols('lambda_S lambda_L lambda_I lambda_A')
adjoint = [l_S, l_L, l_I, l_A]
l = sp.Matrix([l_S, l_L, l_I, l_A])
```

Implementation functions

```
f = P*I + Q*(u**2)
g = sp.Matrix(
    [
    - beta * (1-sigma) * S * (epsilon * L + (1 - q) * I + delta * A) - u * S,
    beta * (1-sigma) * S * (epsilon * L + (1 - q) * I + delta * A) - kappa * L,
    p * kappa * L - alpha * I - tau * I,
    (1 - p) * kappa * L  - eta * A
    ]
)
H = sp.Matrix([f + l.dot(g)])
```

```
I*P + Q*u**2
Matrix([[-S*beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon) - S*u], [-L*kappa + S*beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon)], [-I*alpha
- I*tau + L*kappa*p], [-A*eta + L*kappa*(1 - p)]])
Matrix([[I*P + Q*u**2 + lambda_A*(-A*eta + L*kappa*(1 - p)) + lambda_I*(-I*alpha - I*tau + L*kappa*p) + lambda_L*(-L*kappa + S*beta*(1 - sigma)*
(A*delta + I*(1 - q) + L*epsilon)) + lambda_S*(-S*beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon) - S*u)]])
```

Implementation functions

```python
dHdx = H.jacobian(state)
dHdl = H.jacobian(adjoint)
dHdu = H.jacobian([u])

# Automation
cost_fn = sp.lambdify([*state, *params, u], f)
# adjoint method ode
dHdx_fn = sp.lambdify([*adjoint, *state, *params, u], dHdx)
# 기존 ODE
dHdl_fn = sp.lambdify([*adjoint, *state, *params, u], dHdl)
dHdu_fn = sp.lambdify([*adjoint, *state, *params, u], dHdu)
```

```
Matrix([[beta*lambda_L*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon) + lambda_S*(-beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon) - u), -
S*beta*epsilon*lambda_S*(1 - sigma) + kappa*lambda_A*(1 - p) + kappa*lambda_I*p + lambda_L*(S*beta*epsilon*(1 - sigma) - kappa), P +
S*beta*lambda_L*(1 - q)*(1 - sigma) - S*beta*lambda_S*(1 - q)*(1 - sigma) + lambda_I*(-alpha - tau), S*beta*delta*lambda_L*(1 - sigma) -
S*beta*delta*lambda_S*(1 - sigma) - eta*lambda_A]])
Matrix([[-S*beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon) - S*u, -L*kappa + S*beta*(1 - sigma)*(A*delta + I*(1 - q) + L*epsilon), -I*alpha -
I*tau + L*kappa*p, -A*eta + L*kappa*(1 - p)]])
Matrix([[2*Q*u - S*lambda_S]])
```

Simple gradient method

- step1. Set the initial ν_0
- step2. (Forward) Solve the State differential equation
- step3. (Backward) Solve the Adjoint differential equation
- step4. Update ν_0 (Using Gradient Method)
- step5. Check the convergence

Simple gradient method

- step1. Set the initial ν_0
- step2. (Forward) Solve the State differential equation

```
# State (Forward)
nu_intp = lambda tc: np.interp(tc, t, nu0)
sol = odeint(sliar, y0, t, args=(beta, sigma, kappa, alpha, tau, p, eta, epsilon, q, delta, P, Q, nu_intp))
```

- step3. (Backward) Solve the Adjoint differential equation
- step4. Update ν_0 (Using Gradient Method)
- step5. Check the convergence

Simple gradient method

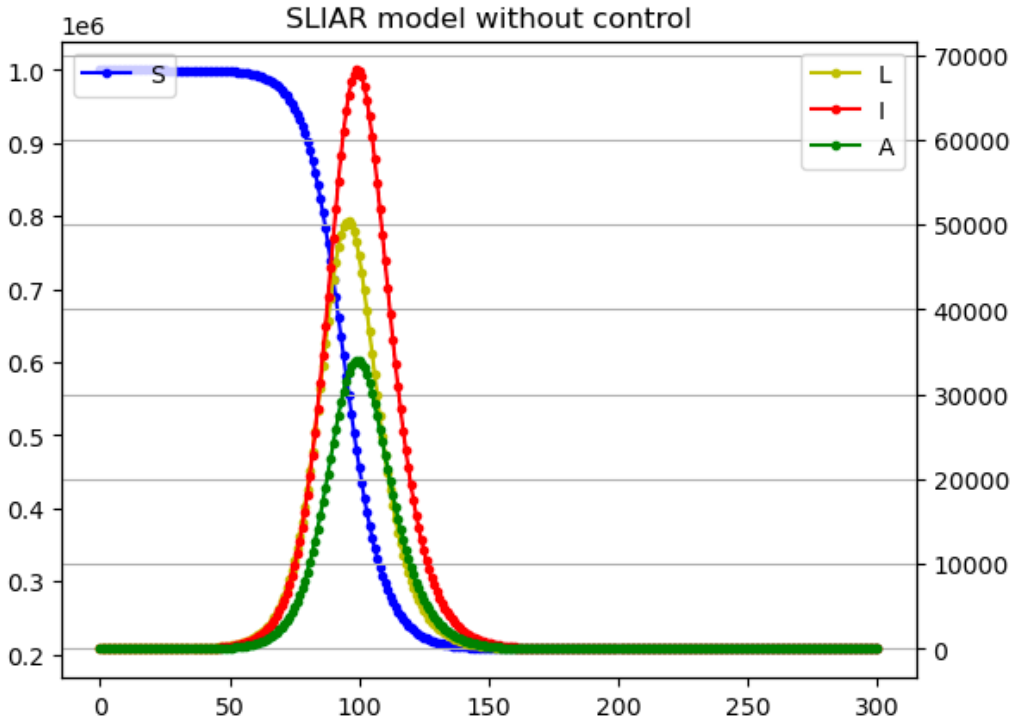
- step1. Set the initial ν_0
- step2. (Forward) Solve the State differential equation

```
# State (Forward)
nu_intp = lambda tc: np.interp(tc, t, nu0)
sol = odeint(sliar, y0, t, args=(beta, sigma, kappa, alpha, tau, p, eta, epsilon, q, delta, P, Q, nu_intp))
```

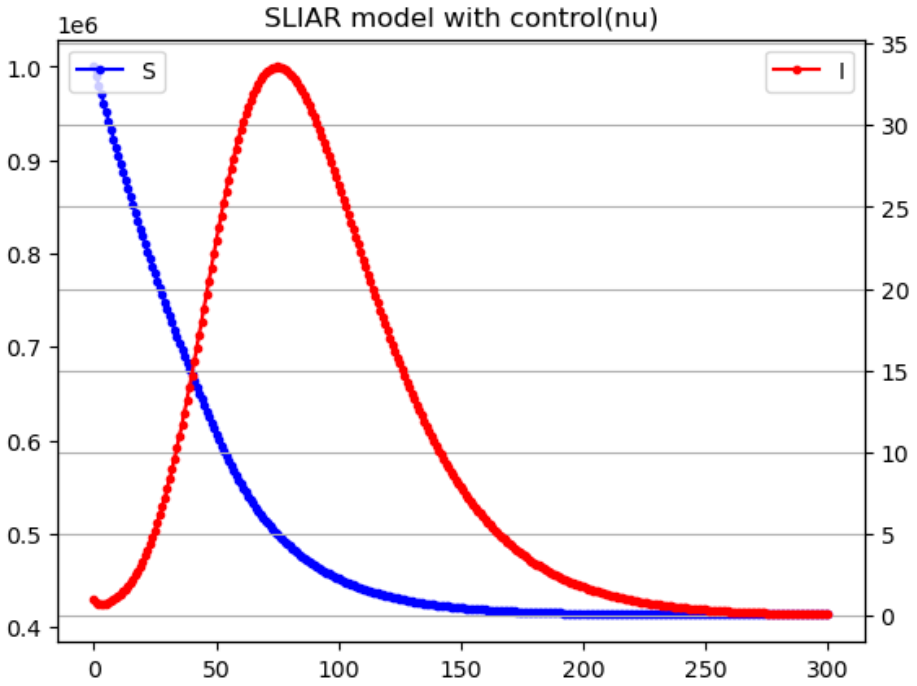
- step3. (Backward) Solve the Adjoint differential equation

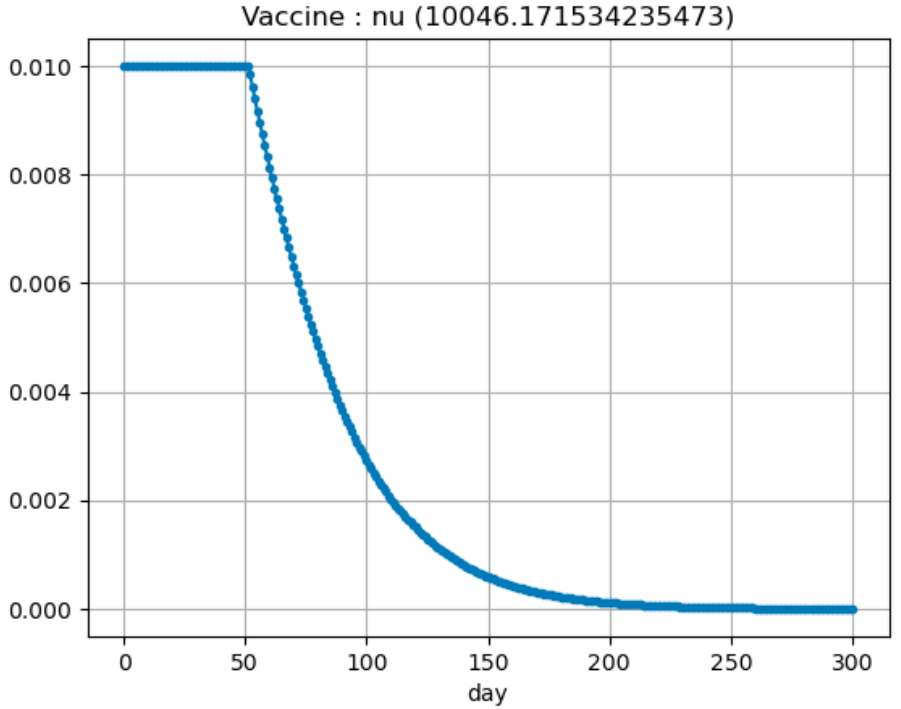
```
# Adjoint (Backward)
nu_intp = lambda tc: np.interp(tf - tc, t, nu0)
x_intp = lambda tc: np.array([np.interp(tf - tc, t, sol[:, 0]),
                              np.interp(tf - tc, t, sol[:, 1]),
                              np.interp(tf - tc, t, sol[:, 2]),
                              np.interp(tf - tc, t, sol[:, 3])])
# lambda_terminal = 0
y_T = np.array([0,0,0,0])
l_sol = odeint(adjoint_sliar, y_T, t, args=(x_intp, beta, sigma, kappa, alpha, tau, p, eta, epsilon, q, delta, P, Q, nu_intp))
l_sol = np.flipud(l_sol)
```

Result


SLIAR model without control
1e6
S
L
I
A
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
70000
60000
50000
40000
30000
20000
10000
0
0
50
100
150
200
250
300

Result


SLIAR model with control(nu)
1e6
S
I
1.0
0.9
0.8
0.7
0.6
0.5
0.4
35
30
25
20
15
10
5
0
0
50
100
150
200
250
300


Vaccine : nu (10046.171534235473)
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
300
day