

Prompts that I have used for answering HW11 - BMI500 - Kimia Ramezan

Part A:

Prompt :

Is this Python code correct for implementing the SIR model using an RK4 solver?

If not, list the exact issues (line-specific if possible).

Assume parameters: SIR with β, γ ; return arrays t, S, I, R over $[0, t_{\max}]$.

Check RK4 staging/weights, time grid coverage of `t_max`, and population conservation.

```
import numpy as np

def sir_derivatives(S, I, R, beta, gamma):
    """
    Returns:
    dSdt, dIdt, dRdt : tuple of floats
    """
    dSdt = -beta * S * I
    dIdt = beta * S * I - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt

def simulate_sir(S0, I0, R0, beta, gamma, t_max, dt=0.1):
    """
    Simulate the SIR model using a RK4 method.
    Returns:
    t - S - I - R
    """
    # (subtle) excludes the endpoint t_max
    t = np.arange(0, t_max, dt)
    n_steps = len(t)

    S = np.zeros(n_steps)
    I = np.zeros(n_steps)
    R = np.zeros(n_steps)

    # initial conditions
    S[0] = S0
    I[0] = I0
    R[0] = R0

    for k in range(n_steps - 1):
        Sk, Ik, Rk = S[k], I[k], R[k]

        # k1
        dS1, dI1, dR1 = sir_derivatives(Sk, Ik, Rk, beta, gamma)

        # k2
        dS2, dI2, dR2 = sir_derivatives(
            Sk + 0.5 * dt * dS1,
            Ik + 0.5 * dt * dI1,
```

```

        Rk + 0.5 * dt * dR1,
        beta, gamma
    )

# k3
dS3, dI3, dR3 = sir_derivatives(
    Sk + 0.5 * dt * dS2,
    Ik + 0.5 * dt * dI2,
    Rk + 0.5 * dt * dR2,
    beta, gamma
)

# k4
dS4, dI4, dR4 = sir_derivatives(
    Sk + dt * dS3,
    Ik + dt * dI2,
    Rk + dt * dR3,
    beta, gamma
)

S[k + 1] = Sk + (dt / 6.0) * (dS1 + dS2 + dS3 + dS4)
I[k + 1] = Ik + (dt / 6.0) * (dI1 + dI2 + dI3 + dI4)
R[k + 1] = Rk + (dt / 6.0) * (dR1 + dR2 + dR3 + dR4)

return t, S, I, R

```

GPT Answer:

Here's my review and the fixed code.

What's wrong (line-by-line highlights)

1. Time grid excludes the endpoint

```
t = np.arange(0, t_max, dt)
```

2.

This stops before `t_max`. For homework plots you usually want to include it: use `t = np.arange(0, t_max + dt, dt)`.

3. RK4 stage 4 uses the wrong intermediate for I

```
Ik + dt * dI2, # <- should use dI3 here
```

4.

In RK4, the 4th slope must be evaluated at the **k3-advanced** state (`Sk+dt*dS3`, `Ik+dt*dI3`, `Rk+dt*dR3`).

5. RK4 combination uses equal weights (wrong)

```
S[k+1] = Sk + (dt/6)*(dS1 + dS2 + dS3 + dS4)
```

6 .

Correct RK4 weights are **1, 2, 2, 1**: $(k_1 + 2*k_2 + 2*k_3 + k_4)$ for each variable.

Everything else (derivatives, loop structure, initial conditions) is fine.

Quick sanity checks you can mention/run

- **Population conservation:** `np.allclose(S+I+R, S0+I0+R0)` (small numerical drift is normal).
- **Peak detection:** `t[np.argmax(I)]` should be $\sim 38\text{--}39$ days with your parameters.
- **Endpoint included:** `t[-1] == t_max` should be `True` (up to floating error).

Part C:

Prompt :

Using the SIR model with parameters $S(0)=999$, $I(0)=1$, $R(0)=0$, $\beta=0.3\times10^{-3}$, $\gamma=0.1$ and a 150-day simulation, give a concise description of the general behavior of $S(t)$, $I(t)$, and $R(t)$. Explain (in simple terms) why each curve rises or falls using the terms $-\beta SI$, $\beta SI - \gamma I$, and γI .

GPT Answer:

Here's a clear, short description for your run ($S(0)=999$, $I(0)=1$, $R(0)=0$, $\beta=0.3\times10^{-3}$, $\gamma=0.1$):

- **Early phase:** Almost everyone is susceptible, so the infection term (βSI) is big; $(I(t))$ grows quickly, $(S(t))$ starts to fall, and $(R(t))$ is near zero.
- **Growth of infections:** As long as ($\beta SI > \gamma I$) (i.e., new infections outpace recoveries), $(I(t))$ keeps rising and $(S(t))$ drops faster.
- **Peak infections:** $(I(t))$ reaches a single peak (around day ~ 38 in your sim) when the **gain** (βSI) balances the **loss** (γI); after this, there aren't enough susceptibles left to sustain growth.
- **Decline phase:** Once ($\beta SI < \gamma I$), recoveries dominate— $(I(t))$ falls, $(R(t))$ rises steadily via (γI), and $(S(t))$ continues to decrease but much more slowly.

- **End state:** $(R(t))$ plateaus high (most people have been infected and recovered), $(S(t))$ settles to a small residual, and $(I(t))$ tends to near zero. The total population is conserved: $(S(t)+I(t)+R(t)) \approx 1000$ at all times.
- **One-line takeaway:** With $(R_0 = \frac{\beta S(0)}{\gamma} \approx \frac{0.0003 \times 999}{0.1} \approx 3! > 1)$, the model produces one large epidemic wave; as (S) is depleted, the effective reproduction number falls below 1, and the outbreak fades.

Part D:

Prompt :

Is the following Python implementation of the SEIR model with births/deaths (RK4) correct?

If not, list the specific issues (line-specific if possible) and provide a corrected version.

```
import numpy as np
```

```
def seir_derivatives(S, E, I, R, beta, gamma, sigma, mu):
```

```
    """
```

```
    Compute the derivatives dS/dt, dE/dt, dI/dt, dR/dt
```

```
    """
```

```
    N = S + E + I + R # total population
```

```
    dSdt = mu * N - beta * S * I - mu * S
```

```
    dEdt = beta * S * I - (sigma - mu) * E
```

```
    dIdt = sigma * E - (gamma + mu) * I
```

```
    dRdt = gamma * I - mu * R
```

```
    return dSdt, dEdt, dIdt, dRdt
```

```
def simulate_seir(S0, E0, I0, R0, beta, gamma, sigma, mu, t_max, dt=0.1):
```

```
    """
```

Simulate the SEIR model using a RK4 method.

"""

t = np.arange(0, t_max, dt)

n_steps = len(t)

S = np.zeros(n_steps)

E = np.zeros(n_steps)

I = np.zeros(n_steps)

R = np.zeros(n_steps)

S[0], E[0], I[0], R[0] = S0, E0, I0, R0

for k in range(n_steps - 1):

Sk, Ek, Ik, Rk = S[k], E[k], I[k], R[k]

k1

dS1, dE1, dI1, dR1 = seir_derivatives(Sk, Ek, Ik, Rk, beta, gamma, sigma, mu)

k2

dS2, dE2, dI2, dR2 = seir_derivatives(

Sk + 0.5 * dt * dS1,

Ek + 0.5 * dt * dE1,

Ik + 0.5 * dt * dI1,

Rk + 0.5 * dt * dR1,

beta, gamma, sigma, mu

)

k3

dS3, dE3, dI3, dR3 = seir_derivatives(

Sk + 0.5 * dt * dS2,

Ek + 0.5 * dt * dE2,

Ik + 0.5 * dt * dI2,

Rk + 0.5 * dt * dR2,

beta, gamma, sigma, mu

)

k4

dS4, dE4, dI4, dR4 = seir_derivatives(

Sk + dt * dS3,

Ek + dt * dE2, # <- should advance with dE3

Ik + dt * dI2, # <- should advance with dI3

Rk + dt * dR3,

beta, gamma, sigma, mu

)

S[k + 1] = Sk + (dt / 6.0) * (dS1 + dS2 + dS3 + dS4)

E[k + 1] = Ek + (dt / 6.0) * (dE1 + dE2 + dE3 + dE4)

I[k + 1] = Ik + (dt / 6.0) * (dI1 + dI2 + dI3 + dI4)

R[k + 1] = Rk + (dt / 6.0) * (dR1 + dR2 + dR3 + dR4)

```
return t, S, E, I, R
```

GPT Answer:

Here's my review and the fixed SEIR code.

What's wrong (specific issues)

1. E-equation sign (demography)

```
dEdt = beta * S * I - (sigma - mu) * E
```

2.

Should be **minus ($\sigma + \mu$)E**, not $(\sigma - \mu)E$.

3. Time grid excludes the endpoint

```
t = np.arange(0, t_max, dt)
```

4.

This stops before `t_max`. Use `np.arange(0, t_max + dt, dt)` to include it.

5. RK4 stage 4 uses wrong intermediates for E and I

```
Ek + dt * dE2 # should use dE3
```

6. Ik + dt * dI2 # should use dI3

7.

`k4` must be evaluated at the **k3-advanced** state.

8. RK4 combination weights are wrong

```
(dS1 + dS2 + dS3 + dS4)
```

9.

RK4 requires weights **1, 2, 2, 1**: $(k1 + 2*k2 + 2*k3 + k4)$ for each variable.

Quick sanity checks (optional but useful)

- **Endpoint included:** `abs(t[-1] - t_max) < 1e-12`.
- **Nonnegativity:** `np.all(S >= 0)`, etc. (small negatives $\sim 1e-12$ can happen from numerics; clip if needed).
- **Total population drift:** `N = S+E+I+R; np.allclose(N[1:] - N[:-1], 0, atol=1e-6)` (tiny drift OK).

- **Behavior:** With your parameters, you should see waves over longer horizons and an endemic level.