Dataset:

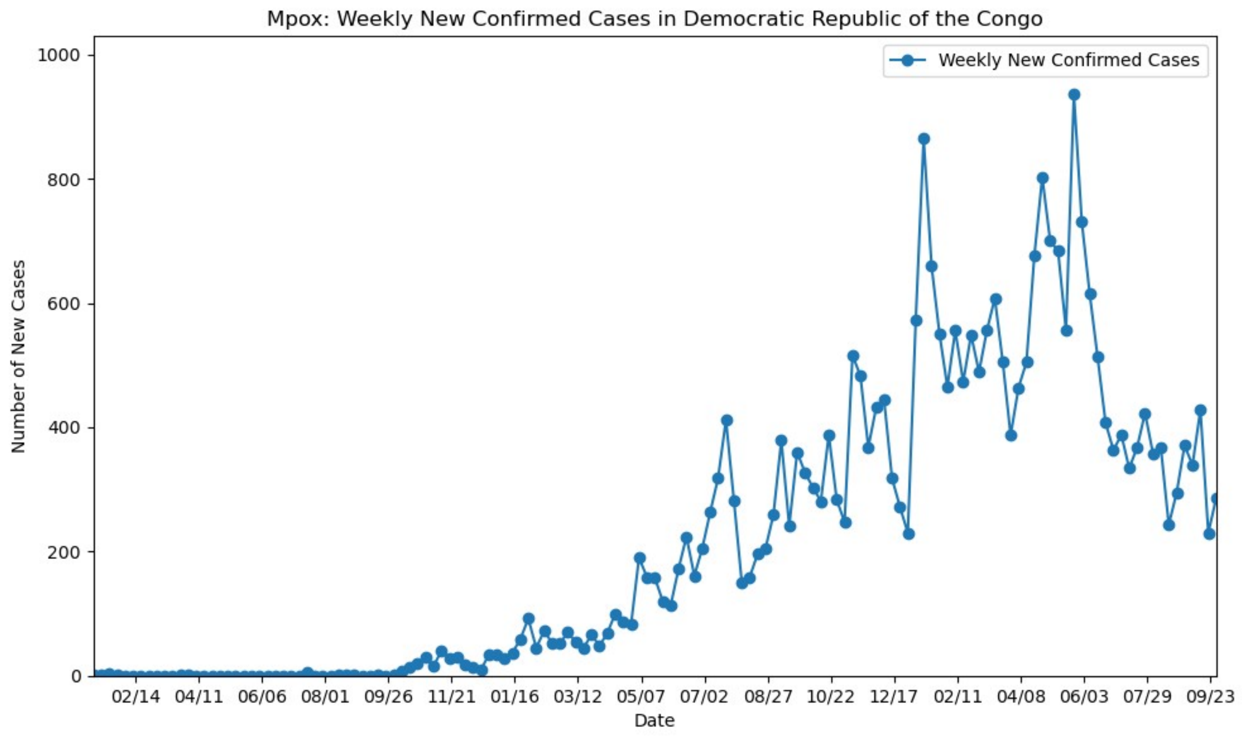
The data set, mpox_dem_rep_of_congo_data_2023_2025_new_cases.csv, is a time-series of weekly confirmed cases in the Democratic Republic of Congo from January 8, 2023 to September 28, 2025. Each row corresponds to a calendar week (ISO formatting) and the variable confirmed_cases gives the number of new labratory-confirmed monkeypox cases reported that week (cases = people). These counts are based on routine national monkeypox surveillance data, where suspected cases are investigated and confirmed via labratory testing. All confirmed cases are then reported to national health authorities and international agencies.

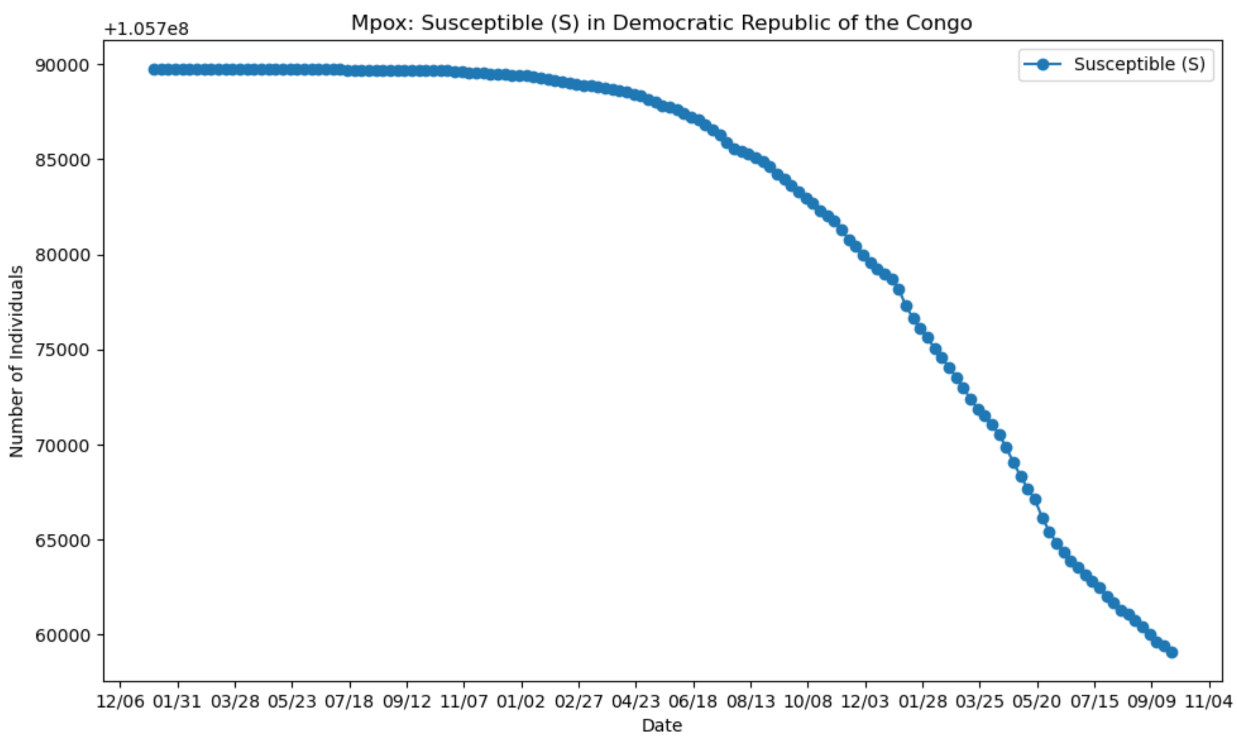
Data Analyis:

Methods

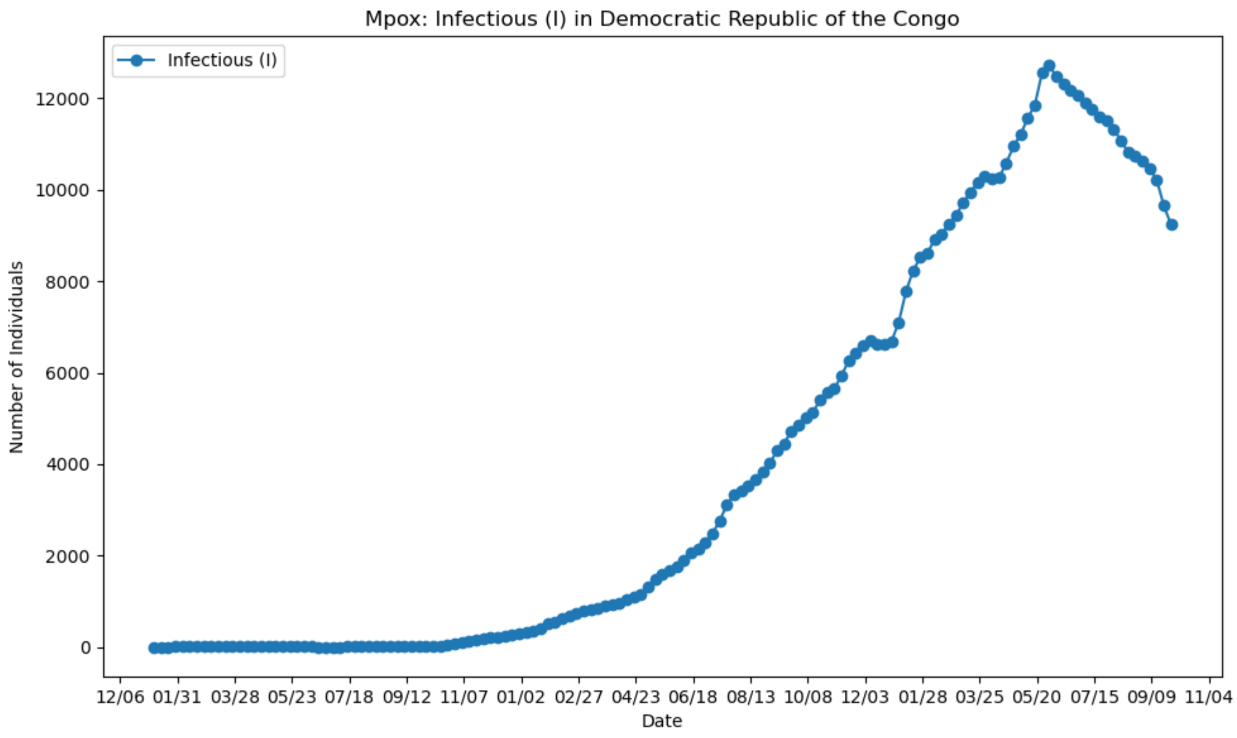
- Compared the curve of I(t) observed versus the predicted curves using Euler's and RK4 methods. These methods were then compared via their total SSE observed, verifyying the best model to predict the future infectious population.

```
## LOAD YOUR DATASET HERE.

# 1. Read in the csv file of cumulative cases.

from functions import convert_cumulative_to_SIR
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Load and Clean the Dataset
data = pd.read_csv('C:\\Users\\hudso\\OneDrive\\Desktop\\Module 4\\
mpox_dem_rep_of_congo_data_2023_2025_new_cases.csv')

data['date'] = pd.to_datetime(data['date'])
data = data.sort_values('date').reset_index(drop=True)
data['cumulative_cases'] = data['confirmed_cases'].cumsum()

plt.figure(figsize=(10, 6))
plt.plot(
    data['date'],
    data['confirmed_cases'],
    label = 'Weekly New Confirmed Cases',
    marker = "o"
)
plt.xlabel('Date')
plt.ylabel('Number of New Cases')
plt.xlim(data['date'].min(), data['date'].max())
plt.ylim(0, data['confirmed_cases'].max() * 1.1)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.title('Mpox: Weekly New Confirmed Cases in Democratic Republic of
the Congo')
plt.legend()
plt.tight_layout()
plt.show()

# 2. Use the convert_cumulative_to_SIR function to convert cumulative
cases to approximate S, I, R at any given time.

population_drc = 105_789_731  # Approximate Population for the
Democratic Republic of the Congo.
data_sir = convert_cumulative_to_SIR( # Convert Cumulative Reported
Cases into SIR Estimates
    df = data.copy(),
    date_col = 'date',
    cumulative_col = 'cumulative_cases',
    population = population_drc,
    infectious_period = 21,        # Infectious Period for Mpox
    new_case_col = 'new_cases',    # This Column will be Created by
the Function
    I_col = 'I_est',
    R_col = 'R_est',
    S_col = 'S_est')

# 3. Plot S, I, R over time.

plt.figure(figsize=(10, 6)) # Plot S_est Over Time
plt.plot(
    data_sir['date'],
    data_sir['S_est'],
    label = 'Susceptible (S)',
    marker = "o"
)
plt.xlabel('Date')
plt.ylabel('Number of Individuals')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d')) #
Format ticks as M/D
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8)) #
Put a tick approximately every 4 weeks
plt.title('Mpox: Susceptible (S) in Democratic Republic of the Congo')
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6)) # Plot I_est Over Time
plt.plot(
    data_sir['date'],
    data_sir['I_est'],
    label = 'Infectious (I)',
    marker = "o"
)
plt.xlabel('Date')
plt.ylabel('Number of Individuals')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.title('Mpox: Infectious (I) in Democratic Republic of the Congo')
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6)) # Plot R_est Over Time
plt.plot(
    data_sir['date'],
    data_sir['R_est'],
    label = 'Recovered (R)',
    marker = "o"
)
plt.xlabel('Date')
plt.ylabel('Number of Individuals')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.title('Mpox: Recovered (R) in Democratic Republic of the Congo')
plt.legend()
plt.tight_layout()
plt.show()
```

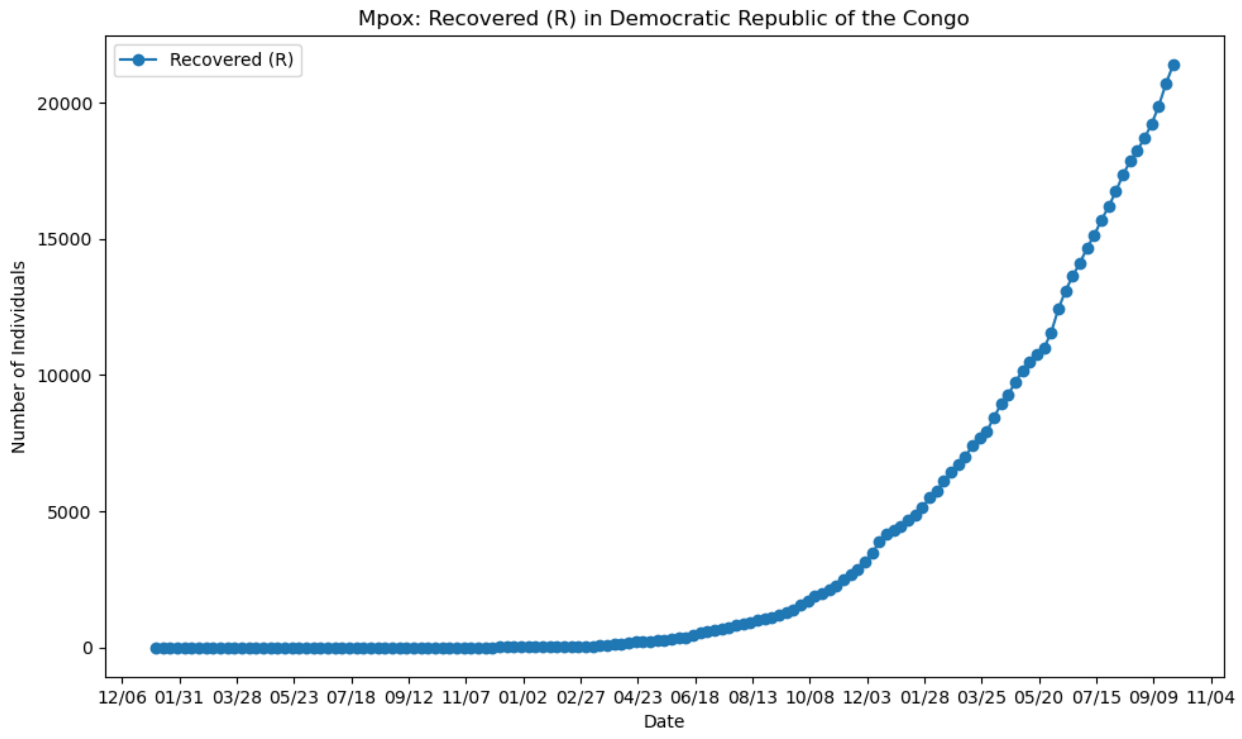




Mpox: Susceptible (S) in Democratic Republic of the Congo
+1.057e8
90000
85000
80000
75000
70000
65000
60000
Number of Individuals
Susceptible (S)
12/06
01/31
03/28
05/23
07/18
09/12
11/07
01/02
02/27
04/23
06/18
08/13
10/08
12/03
01/28
03/25
05/20
07/15
09/09
11/04
Date



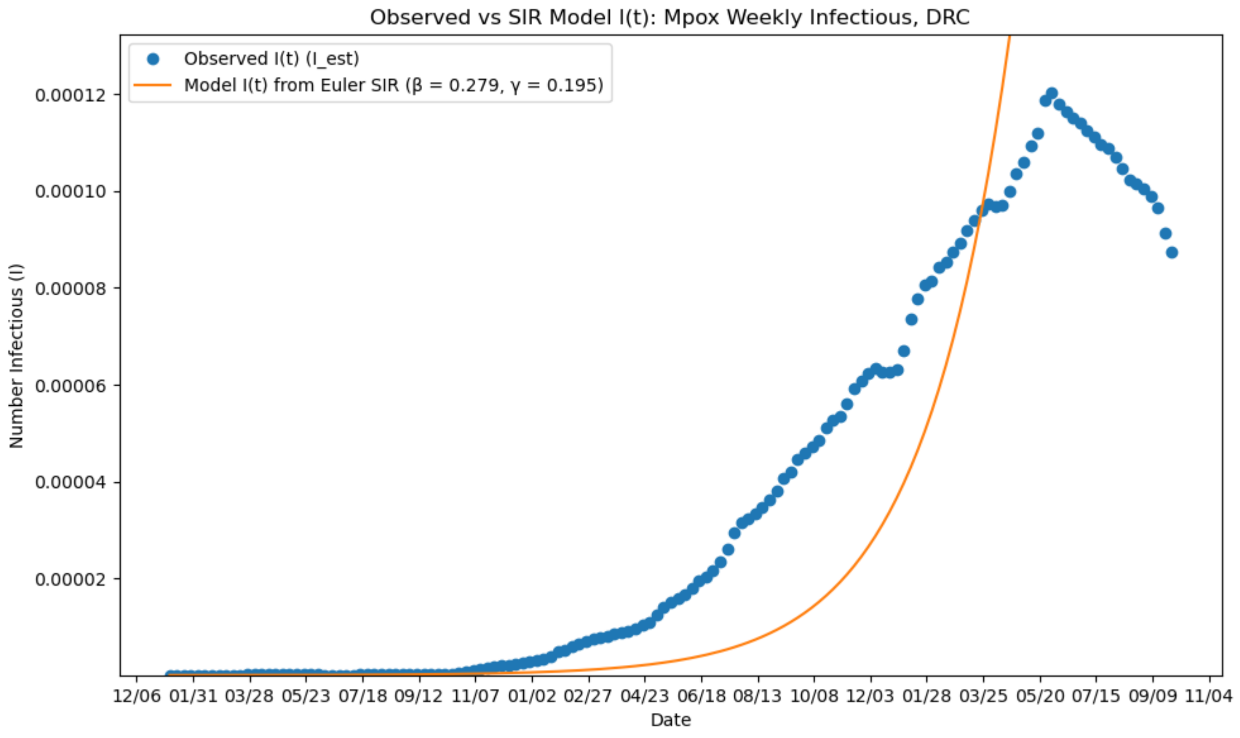
Mpox: Infectious (I) in Democratic Republic of the Congo
Infectious (I)
12000
10000
8000
6000
4000
2000
0
Number of Individuals
12/06
01/31
03/28
05/23
07/18
09/12
11/07
01/02
02/27
04/23
06/18
08/13
10/08
12/03
01/28
03/25
05/20
07/15
09/09
11/04
Date



1. Fitting the SIR Model

```
# Using the euler_SIR function defined earlier, we can simulate the
SIR model over time.

import numpy as np
from functions import euler_sir, euler_sir_normalized

# Plug in guesses for gamma and beta, plot I(t) model vs data, and
calculate SSE.

t = np.arange(len(data_sir))  # Time vector (one step per week)

beta = .2785 # Transmission rate
gamma = .195  # Recovery rate

# Initial conditions in ABSOLUTE COUNTS
S0 = data_sir['S_est'].iloc[0]
I0 = data_sir['I_est'].iloc[0]
R0 = data_sir['R_est'].iloc[0]

# Total population for SIR model
N = population_drc

# Run SIR model with Euler's method
# S_model, I_model, R_model = euler_sir(
#     beta  = beta,
#     gamma = gamma,
#     S0    = S0,
#     I0    = I0,
#     R0    = R0,
#     t     = t,
#     N     = N
# )
S_model, I_model, R_model = euler_sir_normalized(
    beta  = beta,
    gamma = gamma,
    S0    = S0,
    I0    = I0,
    R0    = R0,
    t     = t,
    N     = N
)

# Observed I(t) from data_sir
I_obs = data_sir['I_est'].values/N

# Calculate Sum of Squared Errors (SSE) for I(t)
SSE_I = np.sum((I_model - I_obs) ** 2)
print(f"SSE between model I(t) and observed I(t): {SSE_I:.3e}")

# Plot observed vs model I(t)
plt.figure(figsize=(10, 6))

plt.plot(
    data_sir['date'],
    I_obs,
    'o',
    label='Observed I(t) (I_est)'
)

plt.plot(
    data_sir['date'],
    I_model,
    '-',
    label=f'Model I(t) from Euler SIR (β = {beta:.3f}, γ =
{gamma:.3f})'
)

plt.xlabel('Date')
plt.ylabel('Number Infectious (I)')
plt.title('Observed vs SIR Model I(t): Mpox Weekly Infectious, DRC')
plt.ylim(I_obs.min()*0.9, I_obs.max()*1.1)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.legend()
plt.tight_layout()
plt.show()
```

```
SSE between model I(t) and observed I(t): 2.947e-06
```

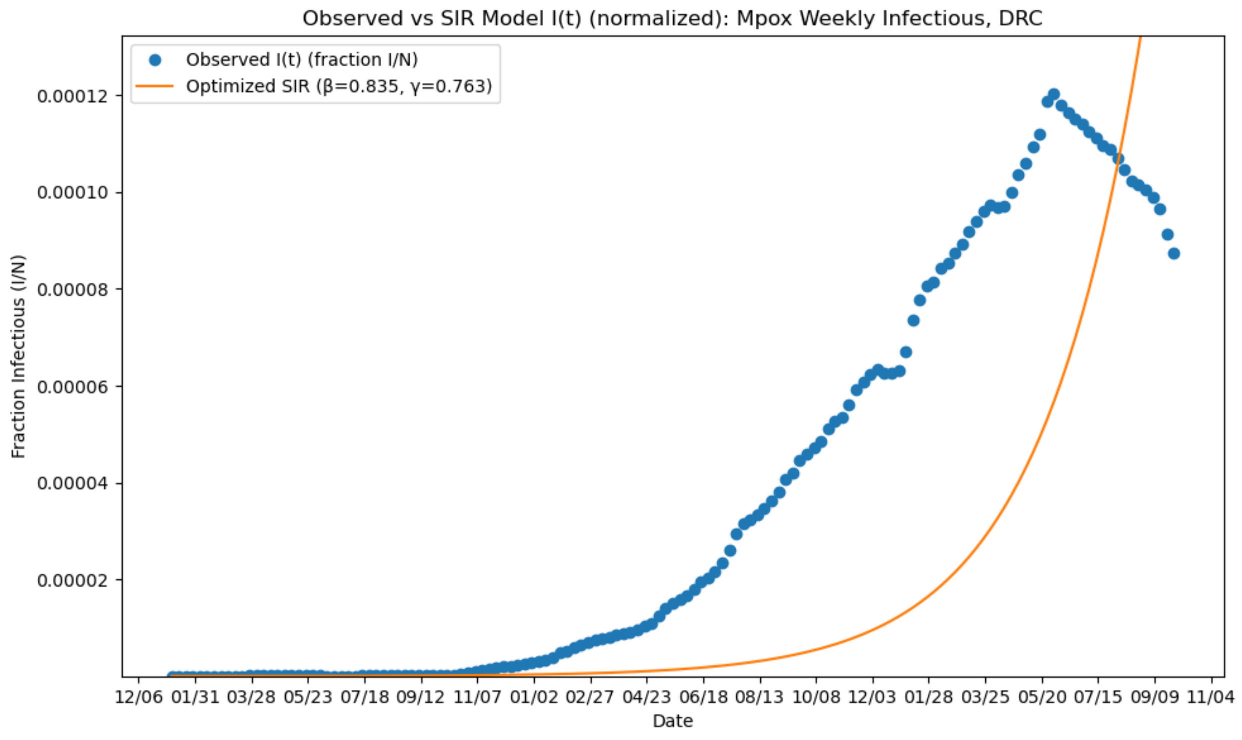


```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from scipy.optimize import minimize

# Time vector
t = np.arange(len(data_sir))

# Initial conditions in counts
S0 = float(data_sir['S_est'].iloc[0])
I0 = float(data_sir['I_est'].iloc[0])
R0 = float(data_sir['R_est'].iloc[0])

# Total population
N = S0 + I0 + R0

# Observed I in FRACTION form
I_obs = data_sir['I_est'].values.astype(float) / N

def sse_I(params):
    beta, gamma = params
    if beta <= 0 or gamma <= 0:
        return np.inf

    # Use the NORMALIZED SIR solver (returns fractions)
    S_model, I_model, R_model = euler_sir_normalized(
        beta=beta, gamma=gamma,
        S0=S0, I0=I0, R0=R0,
        t=t, N=N
    )

    if np.any(~np.isfinite(I_model)):
        return np.inf

    # Compare fraction vs fraction
    return np.sum((I_model - I_obs) ** 2)

# --- Coarse grid search (unchanged idea) ---
beta_grid  = np.linspace(0.05, 0.80, 25)
gamma_grid = np.linspace(0.05, 0.80, 25)

best_sse = np.inf
best_bg = None

for b in beta_grid:
    for g in gamma_grid:
        val = sse_I((b, g))
        if np.isfinite(val) and val < best_sse:
            best_sse = val
            best_bg = (b, g)

if best_bg is None:
    best_bg = (0.275687, 0.195)

# --- Powell with multiple starts ---
bounds = [(1e-8, 2.0), (1e-8, 2.0)]

starts = [np.array(best_bg)]
rng = np.random.default_rng(1)
for _ in range(8):
    starts.append(np.array([rng.uniform(0.05, 0.8), rng.uniform(0.05,
0.8)]))

best_res = None

for x0 in starts:
    res = minimize(
        sse_I,
        x0=x0,
        method="Powell",
        bounds=bounds,
        options={"maxiter": 2000, "xtol": 1e-6, "ftol": 1e-6}
    )
    if best_res is None or (res.fun < best_res.fun):
        best_res = res
```

```
beta_opt, gamma_opt = best_res.x

print("\nBest Powell result:")
print("  success:", best_res.success)
print("  message:", best_res.message)
print(f"  beta_opt  = {beta_opt:.6f}")
print(f"  gamma_opt = {gamma_opt:.6f}")
print(f"  R0 = beta/gamma = {beta_opt/gamma_opt:.3f}")
print(f"  SSE_opt = {best_res.fun:.3e}")

# --- Recompute model with OPTIMAL params ---
S_model, I_model, R_model = euler_sir_normalized(
    beta=beta_opt, gamma=gamma_opt,
    S0=S0, I0=I0, R0=R0,
    t=t, N=N
)

# --- Now compute stats using the OPTIMAL model ---
n = len(I_obs)
SSE = np.sum((I_model - I_obs)**2)
RMSE = np.sqrt(SSE / n)
NRMSE_mean = RMSE / np.mean(I_obs)
NRMSE_range = RMSE / (np.max(I_obs) - np.min(I_obs))
SS_tot = np.sum((I_obs - np.mean(I_obs))**2)
R2 = 1 - SSE / SS_tot

print("\nFit Statistics (normalized I):")
print(f"  SSE  = {SSE:.3e}")
print(f"  RMSE = {RMSE:.3e}")
print(f"  NRMSE (mean)  = {NRMSE_mean:.3%}")
print(f"  NRMSE (range) = {NRMSE_range:.3%}")
print(f"  R^2  = {R2:.4f}")

# --- Plot observed vs optimized I(t) in FRACTIONS ---
plt.figure(figsize=(10, 6))

plt.plot(
    data_sir['date'],
    I_obs,
    'o',
    label='Observed I(t) (fraction I/N)'
)
plt.plot(
    data_sir['date'],
    I_model,
    '-',
    label=f'Optimized SIR (β={beta_opt:.3f}, γ={gamma_opt:.3f})'
)
```

```python
plt.xlabel('Date')
plt.ylabel('Fraction Infectious (I/N)')
plt.title('Observed vs SIR Model I(t) (normalized): Mpox Weekly
Infectious, DRC')
plt.ylim(I_obs.min()*0.9, I_obs.max()*1.1)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.legend()
plt.tight_layout()
plt.show()
```

```
Best Powell result:
  success: True
  message: Optimization terminated successfully.
  beta_opt  = 0.834975
  gamma_opt = 0.762849
  R0 = beta/gamma = 1.095
  SSE_opt = 1.740e-07

Fit Statistics (normalized I):
  SSE  = 1.740e-07
  RMSE = 3.488e-05
  NRMSE (mean)  = 92.993%
  NRMSE (range) = 29.010%
  R^2  = 0.3099
```



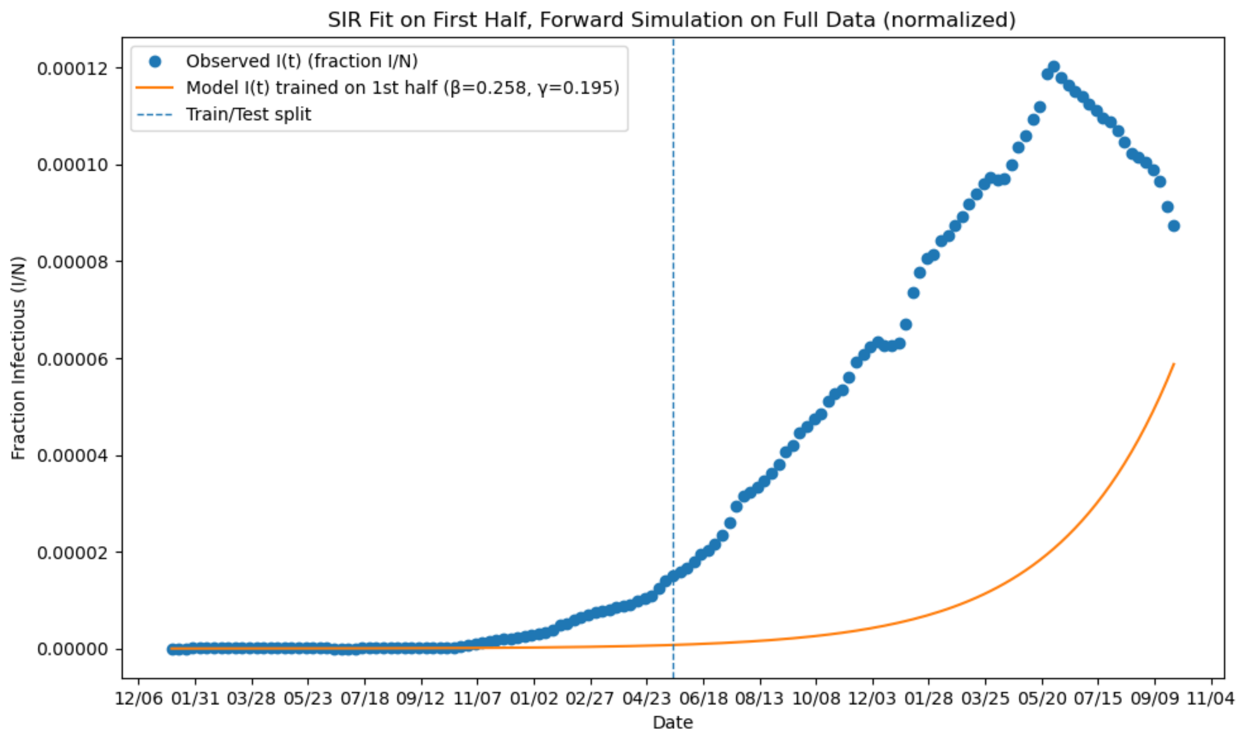
2. Predict "the future" with your fit SIR model

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from scipy.optimize import minimize

# Full time index
t_full = np.arange(len(data_sir))
n = len(data_sir)
mid = n // 2

# ----- Initial conditions in COUNTS -----
S0_counts = float(data_sir['S_est'].iloc[0])
I0_counts = float(data_sir['I_est'].iloc[0])
R0_counts = float(data_sir['R_est'].iloc[0])

# True population for this SIR system
N_true = S0_counts + I0_counts + R0_counts   # or population_drc if
preferred

# ----- Convert to FRACTIONS of population -----
S0 = S0_counts / N_true
I0 = I0_counts / N_true
R0 = R0_counts / N_true

# Training data: fractions I/N
t_train = np.arange(mid)
I_obs_train = data_sir['I_est'].iloc[:mid].values.astype(float) /
N_true

def sse_I_train_relative(params):
    beta, gamma = params

    if beta <= 0 or gamma <= 0:
        return 1e30

    # SIR in FRACTION space: N = 1
    S_model, I_model, R_model = euler_sir(
        beta=beta, gamma=gamma,
        S0=S0, I0=I0, R0=R0,
        t=t_train, N=1.0
    )

    if np.any(~np.isfinite(I_model)):
        return 1e30

    # Relative SSE on FRACTIONS
    eps = 1e-10
    return np.sum(((I_model - I_obs_train) / (I_obs_train + eps)) **
2)
```

```python
# Optimize first half
x0 = np.array([0.275687, 0.195])  # initial guess
bounds = [(1e-8, 2.0), (1e-8, 2.0)]

result = minimize(
    sse_I_train_relative,
    x0=x0,
    method="Powell",
    bounds=bounds,
    options={"maxiter": 2000, "xtol": 1e-6, "ftol": 1e-6}
)

beta_opt, gamma_opt = result.x

print("\nTrain on First Half (normalized)")
print("  success:", result.success)
print("  message:", result.message)
print(f"  beta_opt  = {beta_opt:.6f}")
print(f"  gamma_opt = {gamma_opt:.6f}")
print(f"  R0 = beta/gamma = {beta_opt/gamma_opt:.3f}")
print(f"  train relative SSE = {result.fun:.6e}")
print(f"  Train length = {mid} weeks, Full length = {n} weeks")
print(f"  Using N_true = {N_true:.0f}")

# ----- Simulate forward on FULL time in FRACTIONS -----
S_full, I_full, R_full = euler_sir(
    beta=beta_opt, gamma=gamma_opt,
    S0=S0, I0=I0, R0=R0,
    t=t_full, N=1.0
)

# Observed full I in FRACTIONS
I_obs_full = data_sir['I_est'].values.astype(float) / N_true

# Plot observed vs model with split marker
plt.figure(figsize=(10, 6))
plt.plot(
    data_sir['date'],
    I_obs_full,
    'o',
    label='Observed I(t) (fraction I/N)'
)
plt.plot(
    data_sir['date'],
    I_full,
    '-',
    label=f'Model I(t) trained on 1st half (β={beta_opt:.3f},
γ={gamma_opt:.3f})'
)
```

```
# Vertical line at split date
split_date = data_sir['date'].iloc[mid]
plt.axvline(split_date, linestyle='--', linewidth=1, label='Train/Test
split')

plt.xlabel('Date')
plt.ylabel('Fraction Infectious (I/N)')
plt.title('SIR Fit on First Half, Forward Simulation on Full Data
(normalized)')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.legend()
plt.tight_layout()
plt.show()
```

```
Train on First Half (normalized)
  success: True
  message: Optimization terminated successfully.
  beta_opt  = 0.258451
  gamma_opt = 0.195000
  R0 = beta/gamma = 1.325
  train relative SSE = 4.348544e+01
  Train length = 71 weeks, Full length = 143 weeks
  Using N_true = 105789731
```



Is the new gamma and beta close to what you found on the full dataset? Is the fit much worse? What is the SSE calculated for the second half of the data?

- The new gamma and beta found from half the data set differs from our gamma and beta found using the full data set. We suspect that the rapid increase in I(t) at approximately 71 weeks trains the Euler functions to significantly under-approximate the curve, giving a worse fit as the true curve is unaccounted for. Comparatively, the SSE for half the data set (2.972e-07) vs the SSE for the full data set (1.740e-07) is greater, verifying our reasoning above.

```
# --- Full-data SSE: trained-on-half (I_full) vs optimized-on-full
(I_model) ---

# Use the SAME denominator your cells used (N_true from your train
cell).
# If N_true isn't defined in this notebook state, fall back to
S0+I0+R0.
try:
    N_used = float(N_true)
except NameError:
    N_used = float(data_sir['S_est'].iloc[0] +
data_sir['I_est'].iloc[0] + data_sir['R_est'].iloc[0])

# Observed I over full time (fractions)
I_obs_full = data_sir['I_est'].values.astype(float) / N_used

# Trained model curve = I_full (fractions)
# Optimized model curve = I_model (fractions)
T = min(len(I_obs_full), len(I_full), len(I_model))

SSE_trained_full   = np.sum((np.asarray(I_full,  dtype=float)[:T] -
I_obs_full[:T])**2)
SSE_optimized_full = np.sum((np.asarray(I_model, dtype=float)[:T] -
I_obs_full[:T])**2)

print("\n--- Full SSE Comparison (I) ---")
print(f"SSE_trained_full   (I_full)  = {SSE_trained_full:.3e}")
print(f"SSE_optimized_full (I_model) = {SSE_optimized_full:.3e}")
print(f"Difference (trained - optimized) = {(SSE_trained_full -
SSE_optimized_full):.3e}")
```

```
--- Full SSE Comparison (I) ---
SSE_trained_full   (I_full)  = 2.927e-07
SSE_optimized_full (I_model) = 1.740e-07
Difference (trained - optimized) = 1.187e-07
```

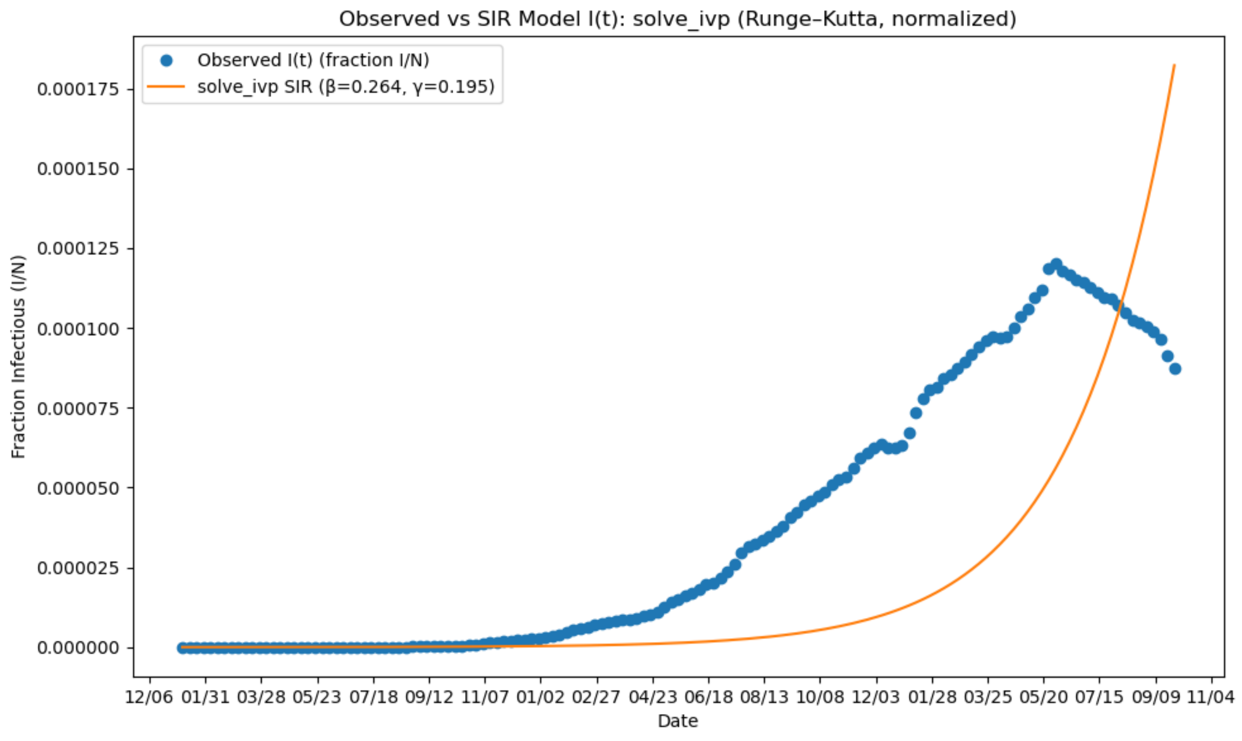
Describe how using a different method like the midpoint method might lower the numerical error.

- In comparison of methods (Euler vs Midpoint), the increase in equational order within the midpoint method more accuratley estimates the change of S, I, and R. Euler's method defines its "best" slope from the start of its interval, often under approxiamating the curve, especially when I(t) rises so quickly; however, the midpoint method estimates the slope at half of its step size and fits the slope depending on whats "best" midpoint. Typically, the higher order of equations used to predict the future curve the lower amount of truncation error accumulated.

3. Decreasing numerical error with the RK4 Method

```python
# === SIR with solve_ivp (Runge–Kutta), normalized ===
# Using scipy's solve_ivp function with the runge-kutta solver, re-
implement the SIR model simulation,
# find optimal gamma and beta again, and plot the results.

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from scipy.integrate import solve_ivp
from scipy.optimize import minimize

# Time grid: one step per week
t = np.arange(len(data_sir), dtype=float)
t_span = (t[0], t[-1])

# ----- Initial conditions in COUNTS -----
S0_counts = float(data_sir['S_est'].iloc[0])
I0_counts = float(data_sir['I_est'].iloc[0])
R0_counts = float(data_sir['R_est'].iloc[0])

# Total population for the SIR system
N_true = S0_counts + I0_counts + R0_counts

# ----- Convert to FRACTIONS of population -----
S0 = S0_counts / N_true
I0 = I0_counts / N_true
R0 = R0_counts / N_true
y0 = [S0, I0, R0]              # y0 from the slide

# Observed I in FRACTIONS
I_obs = data_sir['I_est'].values.astype(float) / N_true

# In normalized form we can take N = 1 in the ODEs
def sir(t, y, beta, gamma):
    S, I, R = y
    dS = -beta * S * I          # N=1 → no division needed
    dI =  beta * S * I - gamma * I
    dR =  gamma * I
    return [dS, dI, dR]

def simulate_ivp(beta, gamma):
    # solve_ivp(fun, t_span, y0, t_eval, args) just like in the slide
    sol = solve_ivp(
        fun=lambda tau, y: sir(tau, y, beta, gamma),
        t_span=t_span,
        y0=y0,
        t_eval=t,
        rtol=1e-9, atol=1e-12    # tighter tolerances = smoother curve
    )
    if not sol.success:
        return None
    return sol.y    # shape (3, len(t)) → [S(t), I(t), R(t)]

def obj(params):
    beta, gamma = params
    if beta <= 0 or gamma <= 0:
        return 1e30

    y = simulate_ivp(beta, gamma)
    if y is None:
        return 1e30

    I_model = y[1]              # FRACTION I/N_true
    if np.any(~np.isfinite(I_model)):
        return 1e30

    # SSE in FRACTIONS (same scale as I_obs)
    return np.sum((I_model - I_obs) ** 2)

# Initial guess (you can plug in your Euler-fit beta/gamma here if you
want)
x0 = np.array([0.275687, 0.195])
bounds = [(1e-8, 2.0), (1e-8, 2.0)]

res = minimize(obj, x0, method="Powell", bounds=bounds)
beta_opt, gamma_opt = res.x

print("solve_ivp fit:")
print("  beta_opt:", beta_opt)
print("  gamma_opt:", gamma_opt)
print("  R0 =", beta_opt / gamma_opt)

S_model, I_model, R_model = simulate_ivp(beta_opt, gamma_opt)

# ---- Plot observed vs model, both in FRACTIONS ----
plt.figure(figsize=(10, 6))
plt.plot(data_sir['date'], I_obs, 'o', label='Observed I(t) (fraction
I/N)')
plt.plot(data_sir['date'], I_model, '-', label=f'solve_ivp SIR
(β={beta_opt:.3f}, γ={gamma_opt:.3f})')
plt.xlabel('Date')
plt.ylabel('Fraction Infectious (I/N)')
plt.title('Observed vs SIR Model I(t): solve_ivp (Runge–Kutta, normalized)')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.legend()
plt.tight_layout()
plt.show()
```

```
solve_ivp fit:
  beta_opt: 0.26449317638993886
  gamma_opt: 0.19499135608709503
  R0 = 1.3564353912785758
```



Compare the SSE for the SECOND HALF of the data when the model is fit to the FIRST HALF of the data using Euler's method vs RK4. Did RK4 do a better job? Why or why not?

- In comparison of Euler (2.915e-07) vs RK4's (2.334e-07) SSE values, there is approximately a 20% decrease in error bewteen the two methods, RK4 being the more accurate method. This decrese in error is expected as Euler's method is a first-order equation and accumulates larger truncation error over its step size; whereas, RK4 is a fourth-order method that intergrates over the same SIR equations more precisely. The remaining error seen from both methods are most likely generated from model limitations, especially the abnormally large difference between our susceptible and infected populations.

```
# SSE comparison between Euler's method and RK4 on the SECOND HALF of
the data.

# RK4 / solve_ivp model over full time (fractions I/N_true)
S_full_RK4, I_full_RK4, R_full_RK4 = simulate_ivp(beta_opt, gamma_opt)

# Observed I over full time in FRACTIONS (same as used in the RK4 fit)
I_obs_full = data_sir['I_est'].values.astype(float) / N_true

# Second-half (test) slices
I_model_test_RK4 = I_full_RK4[mid:]
I_obs_test       = I_obs_full[mid:]   # same test segment used for
Euler

SSE_RK4 = np.sum((I_model_test_RK4 - I_obs_test) ** 2)

print("\n--- SSE comparison on second half ---")
print(f"SSE_Euler (test) = {SSE_SIR:.3e}")
print(f"SSE_RK4   (test) = {SSE_RK4:.3e}")
print(f"Difference in SSE (Euler - RK4): {SSE_SIR - SSE_RK4:.3e}")
```

```
--- SSE comparison on second half ---
SSE_Euler (test) = 2.915e-07
SSE_RK4   (test) = 1.744e-07
Difference in SSE (Euler - RK4): 1.171e-07
```

4. Improving model fit by overcoming model limitations

Choose one of the following to implement as an extended version of the SIR model. Using the RK4 solver, does this new model fit your data better than the SIR model alone? - Comparing the SIR RK4 model (1.744e-07) vs the SEIR RK4 model (3.757e-07), there is approximately a 115% increase in SSE values, with the SIR RK4 model being more accurate. Therefore from these values, it is clear that the SIER RK4 model is statistically a worse fit for the new infectious curve.

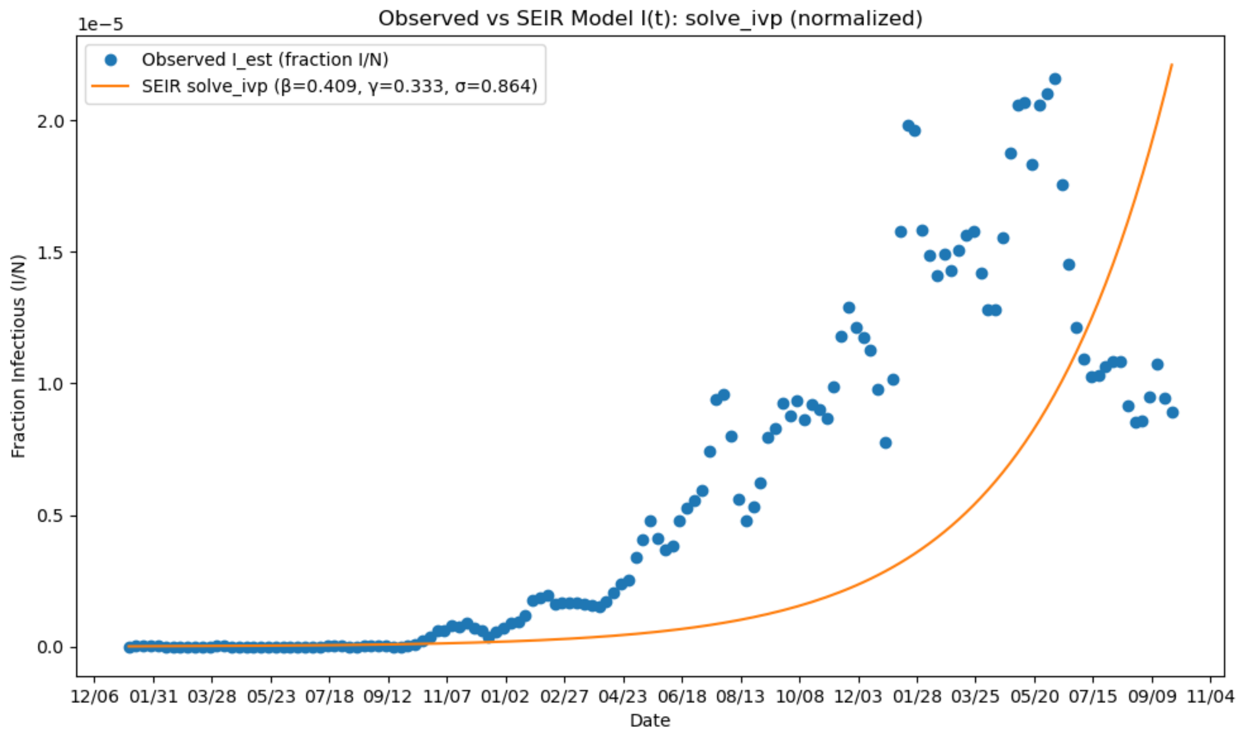
```
# === SEIR with solve_ivp (Runge–Kutta), normalized — evidence-based
durations ===

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
from functions import convert_cumulative_to_SEIR

# -----------------------
# 0) Evidence-based timing for mpox (convert to your data time-step)
# -----------------------
# Pooled mean incubation ~8.1 days (used as latent
proxy) :contentReference[oaicite:2]{index=2}
```

```python
latent_days = 8.1

# Infectious until lesions healed, usually 2–4
weeks :contentReference[oaicite:3]{index=3}
# Use midpoint 3 weeks = 21 days (you can test 14 and 28 for
sensitivity)
infectious_days = 21.0

# IMPORTANT: Your model time unit = 1 row of data.
# You said "one step per week", so assume weekly rows here.
days_per_step = 7.0

latent_steps = max(1, int(round(latent_days / days_per_step)))
# ~1 week
infectious_steps = max(1, int(round(infectious_days / days_per_step)))
# ~3 weeks

# Continuous-time rates in "per step" units (per week if weekly)
sigma = days_per_step / latent_days        # E -> I rate per week
gamma_guess = days_per_step / infectious_days  # I -> R rate per week
(good initial guess)

print("Using:")
print("  latent_days =", latent_days, "-> latent_steps =",
latent_steps, ", sigma =", sigma)
print("  infectious_days =", infectious_days, "-> infectious_steps =",
infectious_steps, ", gamma_guess =", gamma_guess)

# -----------------------
# 1) Build SEIR estimates from cumulative data (windows in STEPS)
# -----------------------
data_seir = convert_cumulative_to_SEIR(
    df=data.copy(),
    date_col='date',
    cumulative_col='cumulative_cases',
    population=population_drc,
    latent_period=latent_steps,
    infectious_period=infectious_steps,
    new_case_col='new_cases',
    S_col='S_est',
    E_col='E_est',
    I_col='I_est',
    R_col='R_est'
)

# -----------------------
# 2) Time grid (1 step per row)
# -----------------------
t = np.arange(len(data_seir), dtype=float)
t_span = (t[0], t[-1])
```

```
# -----------------------
# 3) Initial conditions in COUNTS (includes E) -> FRACTIONS
# -----------------------
S0_counts = float(data_seir['S_est'].iloc[0])
E0_counts = float(data_seir['E_est'].iloc[0])
I0_counts = float(data_seir['I_est'].iloc[0])
R0_counts = float(data_seir['R_est'].iloc[0])

N_true = S0_counts + E0_counts + I0_counts + R0_counts
if N_true <= 0:
    raise ValueError("N_true computed as 0 or negative. Check
population/estimates.")

S0 = S0_counts / N_true
E0 = E0_counts / N_true
I0 = I0_counts / N_true
R0 = R0_counts / N_true
y0 = [S0, E0, I0, R0]

# Observed I (fraction)
I_obs = data_seir['I_est'].values.astype(float) / N_true

# -----------------------
# 4) SEIR ODE system (normalized, N=1)
# -----------------------
def seir_rhs(tau, y, beta, gamma):
    S, E, I, R = y
    dS = -beta * S * I
    dE =  beta * S * I - sigma * E
    dI =  sigma * E - gamma * I
    dR =  gamma * I
    return [dS, dE, dI, dR]

def simulate_ivp_seir(beta, gamma):
    sol = solve_ivp(
        fun=lambda tau, y: seir_rhs(tau, y, beta, gamma),
        t_span=t_span,
        y0=y0,
        t_eval=t,
        rtol=1e-9,
        atol=1e-12
    )
    return None if not sol.success else sol.y   # (4, T): [S,E,I,R]

# -----------------------
# 5) Fit beta, gamma to I(t)
# -----------------------
def obj(params):
    beta, gamma = params
    if beta <= 0 or gamma <= 0:
        return 1e30

    y = simulate_ivp_seir(beta, gamma)
    if y is None:
        return 1e30

    I_model = y[2]
    if np.any(~np.isfinite(I_model)):
        return 1e30

    return np.sum((I_model - I_obs) ** 2)

# Initial guess: keep your beta guess, but use medical gamma guess
x0 = np.array([0.275687, gamma_guess])
bounds = [(1e-8, 2.0), (1e-8, 2.0)]

res = minimize(obj, x0, method="Powell", bounds=bounds)
beta_opt, gamma_opt = res.x

print("\nsolve_ivp SEIR fit (evidence-based sigma):")
print("  beta_opt :", beta_opt)
print("  gamma_opt:", gamma_opt)
print("  sigma    :", sigma)
print("  R0 approx beta/gamma:", beta_opt / gamma_opt)

S_model, E_model, I_model, R_model = simulate_ivp_seir(beta_opt,
gamma_opt)

# ------------------------
# 6) Plot observed vs model I(t) (fractions)
# ------------------------
plt.figure(figsize=(10, 6))
plt.plot(data_seir['date'], I_obs, 'o', label='Observed I_est
(fraction I/N)')
plt.plot(data_seir['date'], I_model, '-', label=f'SEIR solve_ivp
(β={beta_opt:.3f}, γ={gamma_opt:.3f}, σ={sigma:.3f})')
plt.xlabel('Date')
plt.ylabel('Fraction Infectious (I/N)')
plt.title('Observed vs SEIR Model I(t): solve_ivp (normalized)')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=8))
plt.legend()
plt.tight_layout()
plt.show()
```

```
Using:
  latent_days = 8.1 -> latent_steps = 1 , sigma = 0.8641975308641976
  infectious_days = 21.0 -> infectious_steps = 3 , gamma_guess =
0.3333333333333333
```

```
solve_ivp SEIR fit (evidence-based sigma):
  beta_opt : 0.40917211659358144
  gamma_opt: 0.33333606133452354
  sigma    : 0.8641975308641976
  R0 approx beta/gamma: 1.2275063038647704
```



```python
# SSE comparison between SIR RK4 and SEIR on the SECOND HALF of the
data.

# Observed I over full time in FRACTIONS (same scale as the models)
I_obs_full = data_sir['I_est'].values.astype(float) / N_true

# Second-half (test) slices
I_model_test_RK4  = I_full_RK4[mid:]   # from original RK4 model
I_model_test_SEIR = I_model[mid:]      # from SEIR solve_ivp model
I_obs_test        = I_obs_full[mid:]   # same test segment

# SSE on second half
SSE_RK4_test  = np.sum((I_model_test_RK4  - I_obs_test)**2)
SSE_SEIR_test = np.sum((I_model_test_SEIR - I_obs_test)**2)

print("\n--- SSE comparison on second half (normalized I) ---")
print(f"SSE_RK4  (test) = {SSE_RK4_test:.3e}")
print(f"SSE_SEIR (test) = {SSE_SEIR_test:.3e}")
print(f"Difference (RK4 - SEIR) = {-SSE_SEIR_test +
SSE_RK4_test:.3e}")
```

```
--- SSE comparison on second half (normalized I) ---
SSE_RK4  (test) = 1.744e-07
SSE_SEIR (test) = 3.757e-07
Difference (RK4 - SEIR) = -2.013e-07
```

Verify and validate your analysis:

(Describe how you checked to see that your analysis gave you an answer that you believe (verify). Describe how your determined if your analysis gave you an answer that is supported by other evidence (e.g., a published paper).

Conclusions and Ethical Implications:

(Think about the answer your analysis generated, draw conclusions related to your overarching question, and discuss the ethical implications of your conclusions.

Limitations and Future Work:

(Think about the answer your analysis generated, draw conclusions related to your overarching question, and discuss the ethical implications of your conclusions.

NOTES FROM YOUR TEAM:

This is where our team is taking notes and recording activity.

QUESTIONS FOR YOUR TA:

These are questions we have for our TA.