Which ODE solver (Euler or Runge–Kutta) gives better stability and accuracy for SIR equations, and can you show how to implement it in Python?

Thought for 8s

Short answer: use Runge–Kutta (RK4 or adaptive RK45).
For SIR models (which are usually non-stiff), RK methods are far more accurate at the same step size and noticeably more stable than Forward Euler. Euler can work, but only with very small steps (which makes it slower and still less accurate).

Why:

- Euler: global error $O(h)$; tiny stability region → needs very small h (e.g., $h \lesssim 0.1$ days for typical $\beta \sim 0.3,\ \gamma \sim 0.1$).
- RK4: global error $O(h^4)$; much larger stability region → you can often take $h = 0.5{-}1$ day with good accuracy.
- Adaptive RK45 (Dormand–Prince) chooses the step automatically and is the most convenient/safe choice.

Below are two clean implementations you can paste into a script or notebook.

---

1) From-scratch implementations (Euler and RK4)

python Copy code

```python
import numpy as np

def sir_rhs(t, y, beta, gamma, N):
    S, I, R = y
    dS = -beta * S * I / N
    dI =  beta * S * I / N - gamma * I
    dR =  gamma * I
    return np.array([dS, dI, dR], dtype=float)
```

```
def integrate_euler(f, t0, y0, t_end, h, *fargs):
    ts = [t0]
    ys = [np.array(y0, dtype=float)]
    t = t0
    y = np.array(y0, dtype=float)
    while t < t_end - 1e-12:
        y = y + h * f(t, y, *fargs)
        t = t + h
        ts.append(t); ys.append(y.copy())
    return np.array(ts), np.vstack(ys)

def integrate_rk4(f, t0, y0, t_end, h, *fargs):
    ts = [t0]
    ys = [np.array(y0, dtype=float)]
    t = t0
    y = np.array(y0, dtype=float)
    while t < t_end - 1e-12:
        k1 = f(t, y, *fargs)
        k2 = f(t + 0.5*h, y + 0.5*h*k1, *fargs)
        k3 = f(t + 0.5*h, y + 0.5*h*k2, *fargs)
        k4 = f(t + h,     y + h*k3,     *fargs)
        y = y + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)
        t = t + h
        ts.append(t); ys.append(y.copy())
    return np.array(ts), np.vstack(ys)

# Example use
if __name__ == "__main__":
    # parameters
    N = 1_000_000
    beta = 0.30    # per day
    gamma = 0.10   # per day
    I0, R0 = 10, 0
    S0 = N - I0 - R0
    y0 = [S0, I0, R0]

    T = 160.0  # days

    # Euler (needs small step for stability)
    h_euler = 0.05
```

```
tE, yE = integrate_euler(sir_rhs, 0.0, y0, T, h_euler, beta, gamma, N)

# RK4 (can use a larger step while staying accurate)
h_rk4 = 0.5
t4, y4 = integrate_rk4(sir_rhs, 0.0, y0, T, h_rk4, beta, gamma, N)

# Optional sanity check: population conservation
print("Euler mass drift (max |S+I+R−N|):", np.max(np.abs(yE.sum(axis=1
print("RK4   mass drift (max |S+I+R−N|):", np.max(np.abs(y4.sum(axis=1
```

Tip: If Euler "rings/oscillates" or produces negative values, reduce `h`. RK4 is much more forgiving.

---

2) Adaptive step sizing with SciPy (RK45)

python Copy code

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

def sir_rhs(t, y, beta, gamma, N):
    S, I, R = y
    return [
        -beta * S * I / N,
         beta * S * I / N - gamma * I,
         gamma * I
    ]

# parameters
N = 1_000_000
beta, gamma = 0.30, 0.10
I0, R0 = 10, 0
S0 = N - I0 - R0
y0 = (S0, I0, R0)

T = 160.0
sol = solve_ivp(
    fun=lambda t, y: sir_rhs(t, y, beta, gamma, N),
```

```
    t_span=(0.0, T),
    y0=y0,
    method="RK45",            # Dormand–Prince (adaptive)
    rtol=1e-6, atol=1e-9,
    dense_output=True
)

t = np.linspace(0, T, 400)
S, I, R = sol.sol(t)
plt.figure()
plt.plot(t, S, label="S")
plt.plot(t, I, label="I")
plt.plot(t, R, label="R")
plt.xlabel("Days"); plt.ylabel("Individuals"); plt.legend(); plt.show()
```

What to choose

- Quick prototypes or teaching: RK4 with a moderate step (e.g., $h = 0.5$ day).
- Production/robustness: `solve_ivp(..., method="RK45")` (automatic step control).
- Only use Euler if you're explicitly studying numerical instability or you absolutely must have the simplest integrator and are willing to use very small steps.

If you want, I can adapt this to your exact SIR parameters/data and show a side-by-side Euler vs RK plot.