Advanced Dynamical Systems CW1

CID: 02044064

January 23, 2025

Q1 Flow properties

a) Let $c_1(x) = \frac{x_1+x_2}{2}, \ c_2(x) = \frac{x_1-x_2}{2}$ and define

$$\varphi(t,x) = \begin{pmatrix} c_1(x)e^t + c_2(x)e^{-t} \\ c_1(x)e^t - c_2(x)e^{-t} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} (x_1+x_2)\,e^t + (x_1-x_2)\,e^{-t} \\ (x_1+x_2)\,e^t - (x_1-x_2)\,e^{-t} \end{pmatrix}$$

Check initial condition:

$$\varphi(0,x) = \begin{pmatrix} c_1(x) + c_2(x) \\ c_1(x) - c_2(x) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} (x_1+x_2) + (x_1-x_2) \\ (x_1+x_2) - (x_1-x_2) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x \ \ \forall \ x \in R^2$$

Check composition condition:

$$\begin{aligned} \varphi\left(t_2, \varphi\left(t_1, x\right)\right. &= \varphi\left(t_2, \frac{1}{2}\begin{pmatrix} (x_1+x_2)\,e^{t_1} + (x_1-x_2)\,e^{-t_1} \\ (x_1+x_2)\,e^{t_1} - (x_1-x_2)\,e^{-t_1}) \end{pmatrix}\right) \\ &= \frac{1}{2}\begin{pmatrix} \frac{1}{2}\cdot 2\,(x_1+x_2)\,e^{t_1}e^{t_2} + \frac{1}{2}\cdot 2\,(x_1-x_2)\,e^{-t_1}e^{-t_2} \\ \frac{1}{2}\cdot 2\,(x_1-x_2)\,e^{t_1}e^{t_2} - \frac{1}{2}\cdot 2\,(x_1-x_2)\,e^{-t_1}e^{-t_2} \end{pmatrix} \\ &= \frac{1}{2}\begin{pmatrix} (x_1+x_2)\,e^{t_1+t_2} + (x_1-x_2)\,e^{-(t_1+t_2)} \\ (x_1+x_2)\,e^{t_1+t_2} - (x_1-x_2)\,e^{-(t_1+t_2)}) \end{pmatrix} \\ &= \varphi\left(t_1+t_2, x\right) \end{aligned}$$

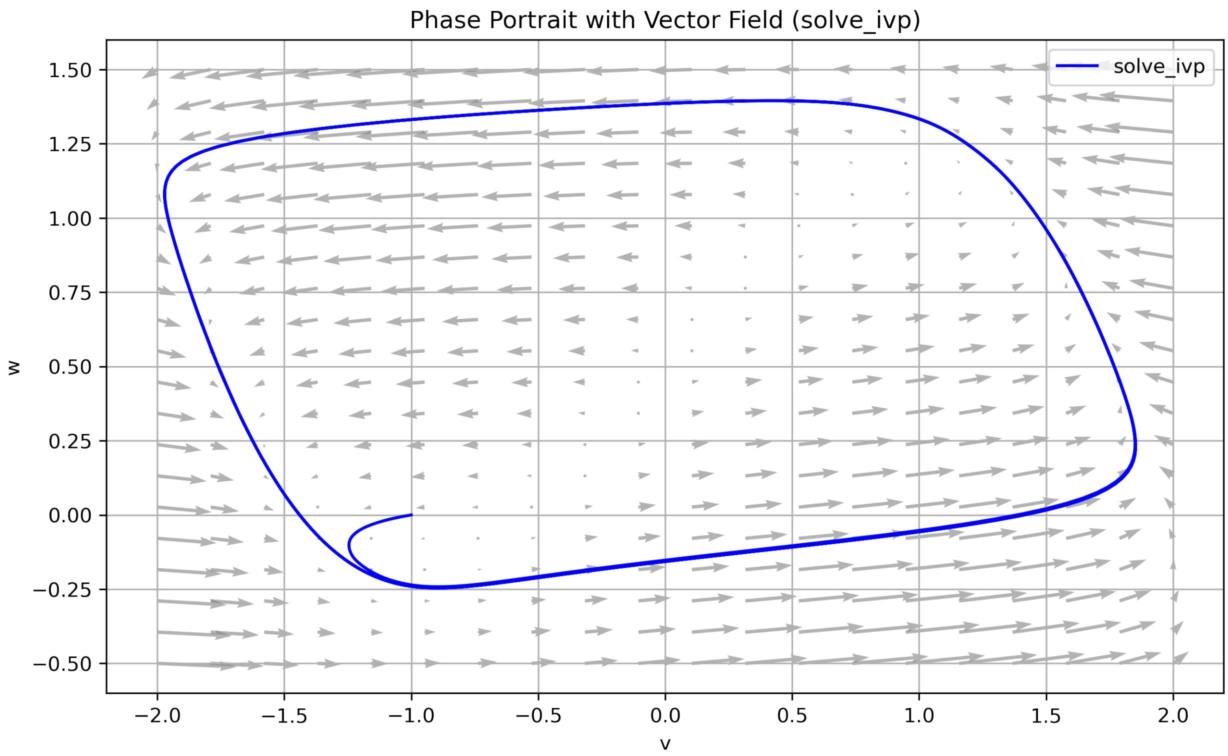
As both flow conditions are satisfied, this is a valid flow.

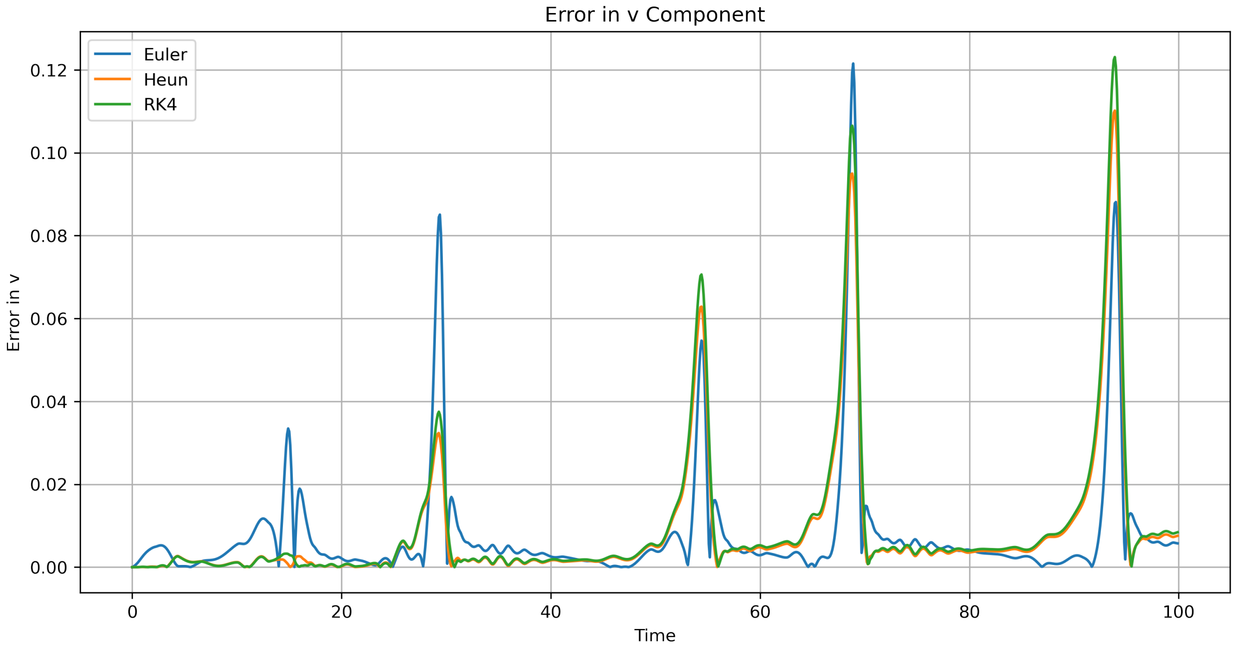
b) This is not a valid flow because the initial condition is not satisfied: there do not exist $c_1(x), c_2(x)$ such that

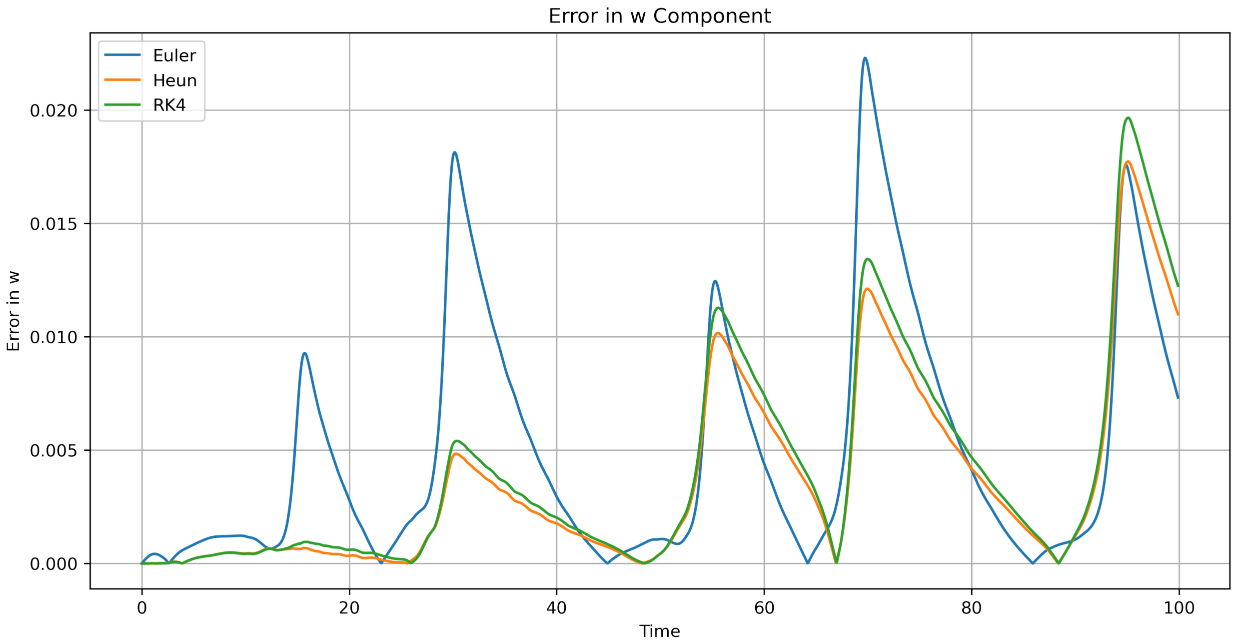
$$\varphi(0,x) = \left(\frac{1}{c_1(x)+c_2(x)}, \frac{c_1(x)-c_2(x)}{c_1(x)+c_1(x)}\right) = (0,0)$$

as $c_1(x)+c_2(x)$ is always finite.

Q2 Comparing numerical integration methods for the FitzHugh-Nagumo model







We notice that both error graphs have five peaks. This makes sense because these occur when the "true" (solve_ivp) trajectory changes direction abruptly, which happens five times up to t = 100. The other numerical solvers, Euler, Heun and RK4, produce piecewise linear approximations of the true trajectory that are less accurate than solve_ivp, and the greater the true trajectory's curvature, the more the line segment approximations of Euler, Heun and RK4 deviate from it, and so the larger the errors.

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


# FitzHugh-Nagumo model
def fitzhugh_nagumo(t, y, epsilon, a, b, I):
    v, w = y
    dv_dt = v - (v**3) / 3 - w + I
    dw_dt = epsilon * (v + a - b * w)
    return [dv_dt, dw_dt]


# Euler method
def euler_method(f, y0, t_span, h):
    t_vals = np.arange(t_span[0], t_span[1], h)
    y_vals = np.zeros((len(t_vals), len(y0)))
    y_vals[0] = y0
    for i in range(1, len(t_vals)):
        y_vals[i] = y_vals[i - 1] + h * np.array(f(t_vals[i - 1], y_vals[i - 1]))
    return t_vals, y_vals


# Heun method
def heun_method(f, y0, t_span, h):
    t_vals = np.arange(t_span[0], t_span[1], h)
    y_vals = np.zeros((len(t_vals), len(y0)))
    y_vals[0] = y0
    for i in range(1, len(t_vals)):
        k1 = np.array(f(t_vals[i - 1], y_vals[i - 1]))
        k2 = np.array(f(t_vals[i - 1] + h, y_vals[i - 1] + h * k1))
        y_vals[i] = y_vals[i - 1] + (h / 2) * (k1 + k2)
    return t_vals, y_vals


# Runge-Kutta method
def runge_kutta_method(f, y0, t_span, h):
    t_vals = np.arange(t_span[0], t_span[1], h)
    y_vals = np.zeros((len(t_vals), len(y0)))
    y_vals[0] = y0
    for i in range(1, len(t_vals)):
        k1 = np.array(f(t_vals[i - 1], y_vals[i - 1]))
        k2 = np.array(f(t_vals[i - 1] + h / 2, y_vals[i - 1] + h * k1 / 2))
        k3 = np.array(f(t_vals[i - 1] + h / 2, y_vals[i - 1] + h * k2 / 2))
        k4 = np.array(f(t_vals[i - 1] + h, y_vals[i - 1] + h * k3))
        y_vals[i] = y_vals[i - 1] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
    return t_vals, y_vals


# Parameters
epsilon = 0.08
```

```python
a = 0.7
b = 0.8
I = 0.5
t_span = (0, 100)
h = 0.1
initial_conditions = [-1.0, 0.0]

# Solve with numerical methods
f = lambda t, y: fitzhugh_nagumo(t, y, epsilon, a, b, I)

# Euler
t_euler, sol_euler = euler_method(f, initial_conditions, t_span, h)

# Heun
t_heun, sol_heun = heun_method(f, initial_conditions, t_span, h)

# Runge-Kutta
t_rk4, sol_rk4 = runge_kutta_method(f, initial_conditions, t_span, h)

# solve_ivp
t_eval = np.arange(t_span[0], t_span[1], h)
ivp_solution = solve_ivp(fitzhugh_nagumo, t_span, initial_conditions, t_eval=t_eval,
    args=(epsilon, a, b, I))
v_ivp, w_ivp = ivp_solution.y  # Extract v and w components from the solution

# Compute errors
errors_v = {
    "Euler": np.abs(sol_euler[:, 0] - v_ivp),
    "Heun": np.abs(sol_heun[:, 0] - v_ivp),
    "RK4": np.abs(sol_rk4[:, 0] - v_ivp),
}

errors_w = {
    "Euler": np.abs(sol_euler[:, 1] - w_ivp),
    "Heun": np.abs(sol_heun[:, 1] - w_ivp),
    "RK4": np.abs(sol_rk4[:, 1] - w_ivp),
}


# Create vector field
def compute_vector_field(v_range, w_range, epsilon, a, b, I, density=20):
    v_vals = np.linspace(*v_range, density)
    w_vals = np.linspace(*w_range, density)
    V, W = np.meshgrid(v_vals, w_vals)
    dV = V - (V**3) / 3 - W + I
    dW = epsilon * (V + a - b * W)
    return V, W, dV, dW


v_range = (-2, 2)
w_range = (-0.5, 1.5)
```

```
V, W, dV, dW = compute_vector_field(v_range, w_range, epsilon, a, b, I)

# Plot phase portrait with vector field
plt.figure(figsize=(10, 6), dpi=300)
plt.quiver(V, W, dV, dW, color='gray', alpha=0.6)
plt.plot(v_ivp, w_ivp, label="solve_ivp", color="blue")
plt.xlabel("v")
plt.ylabel("w")
plt.title("Phase Portrait with Vector Field (solve_ivp)")
plt.legend()
plt.grid()
plt.show()

# Plot errors
# Error in v
plt.figure(figsize=(12, 6), dpi=300)
for method, error in errors_v.items():
    plt.plot(t_eval, error, label=f"{method}")
plt.xlabel("Time")
plt.ylabel("Error in v")
plt.title("Error in v Component")
plt.legend()
plt.grid()
plt.show()

# Error in w
plt.figure(figsize=(12, 6), dpi=300)
for method, error in errors_w.items():
    plt.plot(t_eval, error, label=f"{method}")
plt.xlabel("Time")
plt.ylabel("Error in w")
plt.title("Error in w Component")
plt.legend()
plt.grid()
plt.show()
```