Class-2 Model

March 13, 2025

1 Analysis of Class-2 Neuronal Excitability Model from Sejnowski et al. (2008)

This notebook implements and analyzes the Class-2 neuronal excitability model from Sejnowski et al. (2008), "Biophysical Basis for Three Distinct Dynamical Mechanisms of Action Potential Initiation". The model is a 2D Morris-Lecar-like model with a fast activation variable V (membrane voltage) and a slower recovery variable w.

The model is described by the following equations:

$$C\frac{dV}{dt} = I_{stim} - \bar{g}_{fast} m_\infty(V)(V - E_{Na}) - \bar{g}_{slow} w(V - E_K) - g_{leak}(V - E_{leak})$$

$$\frac{dw}{dt} = \phi_w \frac{w_\infty(V) - w}{\tau_w(V)}$$

Where:

$$m_\infty(V) = 0.5(1 + \tanh(\frac{V - b_m}{c_m}))$$

$$w_\infty(V) = 0.5(1 + \tanh(\frac{V - b_w}{c_w}))$$

$$\tau_w(V) = \frac{1}{\cosh(\frac{V - b_w}{2c_w})}$$

Class-2 excitability is characterized by a discontinuous frequency-current (f-I) curve, where neurons cannot maintain firing below a critical frequency. Unlike Class-1 neurons which can fire arbitrarily slowly, Class-2 neurons have a minimum firing frequency. According to Sejnowski's paper, this is related to a Hopf bifurcation in the dynamical system.

We'll implement this model, explore its dynamics through time series, phase plane, and bifurcation analysis, and discuss how these relate to the biological properties of Class-2 neurons.

2 Class-2 Model Simulation and Parameter Sensitivity Analysis

In this first section, we'll implement and analyze the Class-2 model from the Sejnowski 2008 paper. The paper discusses three distinct dynamical mechanisms of action potential initiation, where Class-2 excitability occurs through a Hopf bifurcation when inward current activates faster than outward

current despite the net current being outward at steady state. Class-2 excitability is characterized by a discontinuous f-I curve, where the neuron cannot maintain spiking below a critical frequency.

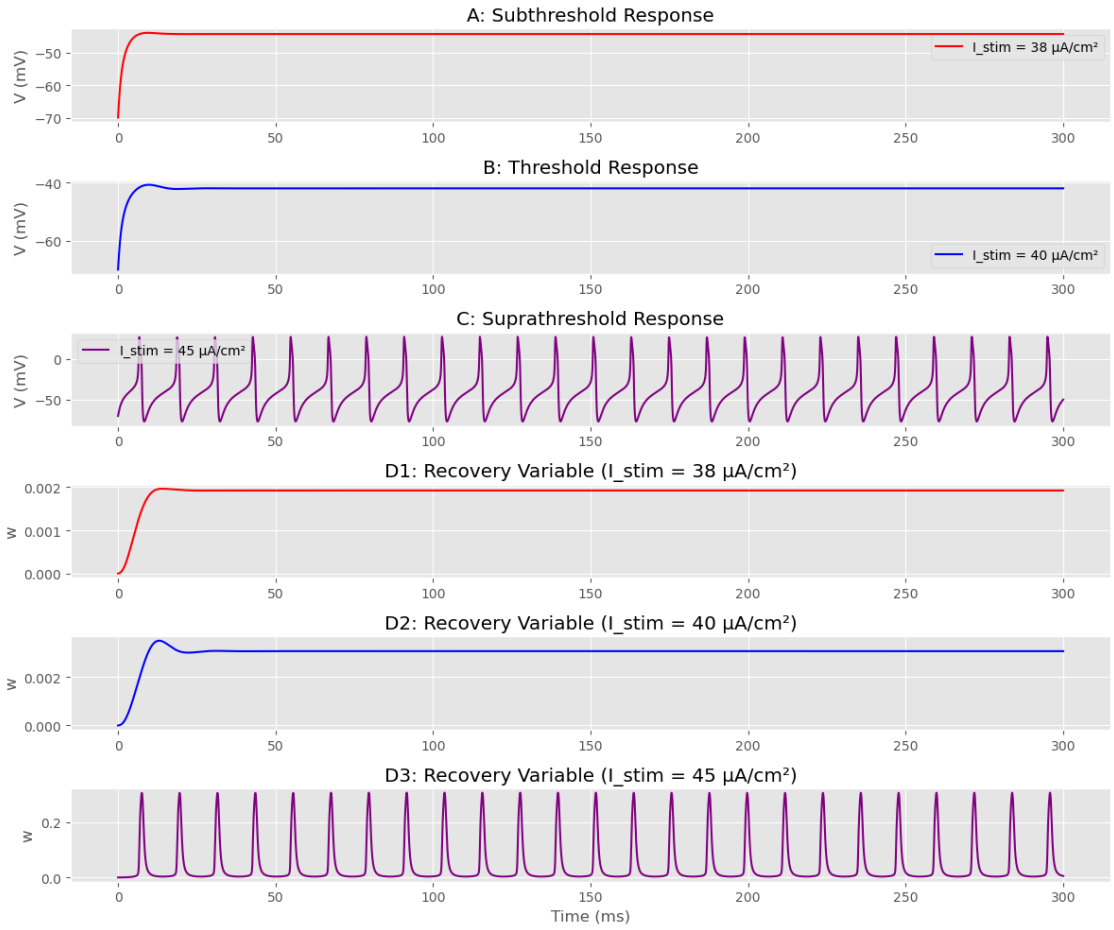
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from matplotlib.gridspec import GridSpec
import warnings
warnings.filterwarnings('ignore')
plt.style.use('ggplot')

# For reproducibility
np.random.seed(42)

# Model parameters (from Sejnowski 2008, p.2-3)
# Class 2 model with b_w = -13 mV
E_Na = 50.0       # mV
E_K = -100.0      # mV
E_leak = -70.0    # mV
g_fast = 20.0     # mS/cm²
g_slow = 20.0     # mS/cm²
g_leak = 2.0      # mS/cm²
phi_w = 0.15      # Dimensionless
C = 2.0           # μF/cm²
b_m = -1.2        # mV
c_m = 18.0        # mV
b_w = -13.0       # mV (Class 2 model)
c_w = 10.0        # mV

# Activation functions
def m_inf(V):
    """Steady state activation of fast current"""
    return 0.5 * (1 + np.tanh((V - b_m) / c_m))

def w_inf(V):
    """Steady state activation of slow current"""
    return 0.5 * (1 + np.tanh((V - b_w) / c_w))

def tau_w(V):
    """Time constant for slow current activation"""
    return 1 / np.cosh((V - b_w) / (2 * c_w))

print(f"Class-2 Model Parameters:")
print(f"E_Na = {E_Na} mV, E_K = {E_K} mV, E_leak = {E_leak} mV")
print(f"g_fast = {g_fast} mS/cm², g_slow = {g_slow} mS/cm², g_leak = {g_leak}␣
 ↪mS/cm²")
print(f" _w = {phi_w}, C = {C} μF/cm²")
```

```
print(f"b_m = {b_m} mV, c_m = {c_m} mV, b_w = {b_w} mV, c_w = {c_w} mV")
```

```
Class-2 Model Parameters:
E_Na = 50.0 mV, E_K = -100.0 mV, E_leak = -70.0 mV
g_fast = 20.0 mS/cm², g_slow = 20.0 mS/cm², g_leak = 2.0 mS/cm²
 _w = 0.15, C = 2.0 µF/cm²
b_m = -1.2 mV, c_m = 18.0 mV, b_w = -13.0 mV, c_w = 10.0 mV
```

[2]:
```
t_span = (0, 300)  # ms
t_eval = np.linspace(0, 300, 3000)  # ms
V0 = -70.0  # mV (resting potential)
w0 = 0.0    # dimensionless (initial recovery variable)
y0 = [V0, w0]

# I_stim values for Figure 1
# Values chosen to demonstrate subthreshold, threshold and suprathreshold␣
 ↪responses
I_stim_values = [38, 40, 45]  # µA/cm²

# Storage for solutions
solutions = []

for I_stim in I_stim_values:
    def model(t, y):
        V, w = y
        dVdt = (I_stim - g_fast * m_inf(V) * (V - E_Na) - g_slow * w * (V -␣
 ↪E_K) - g_leak * (V - E_leak)) / C
        dwdt = phi_w * (w_inf(V) - w) / tau_w(V)
        return [dVdt, dwdt]

    sol = solve_ivp(
        model,
        t_span,
        y0,
        method='BDF',
        t_eval=t_eval
    )
    solutions.append(sol)

fig = plt.figure(figsize=(12, 10))
gs = GridSpec(6, 1, height_ratios=[1, 1, 1, 1, 1, 1])

ax1 = fig.add_subplot(gs[0])
ax1.plot(solutions[0].t, solutions[0].y[0], label=f'I_stim = {I_stim_values[0]}␣
 ↪µA/cm²', color='red')
ax1.set_ylabel('V (mV)')
ax1.set_title('A: Subthreshold Response')
```

```
ax1.grid(True)
ax1.legend()

ax2 = fig.add_subplot(gs[1], sharex=ax1)
ax2.plot(solutions[1].t, solutions[1].y[0], label=f'I_stim = {I_stim_values[1]}␣
 ↪µA/cm²', color='blue')
ax2.set_ylabel('V (mV)')
ax2.set_title('B: Threshold Response')
ax2.grid(True)
ax2.legend()

ax3 = fig.add_subplot(gs[2], sharex=ax1)
ax3.plot(solutions[2].t, solutions[2].y[0], label=f'I_stim = {I_stim_values[2]}␣
 ↪µA/cm²', color='purple')
ax3.set_ylabel('V (mV)')
ax3.set_title('C: Suprathreshold Response')
ax3.grid(True)
ax3.legend()

# Plot w for different I_stim in separate subplots
ax4 = fig.add_subplot(gs[3], sharex=ax1)
ax4.plot(solutions[0].t, solutions[0].y[1], color='red')
ax4.set_ylabel('w')
ax4.set_title(f'D1: Recovery Variable (I_stim = {I_stim_values[0]} µA/cm²)')
ax4.grid(True)

ax5 = fig.add_subplot(gs[4], sharex=ax1)
ax5.plot(solutions[1].t, solutions[1].y[1], color='blue')
ax5.set_ylabel('w')
ax5.set_title(f'D2: Recovery Variable (I_stim = {I_stim_values[1]} µA/cm²)')
ax5.grid(True)

ax6 = fig.add_subplot(gs[5], sharex=ax1)
ax6.plot(solutions[2].t, solutions[2].y[1], color='purple')
ax6.set_xlabel('Time (ms)')
ax6.set_ylabel('w')
ax6.set_title(f'D3: Recovery Variable (I_stim = {I_stim_values[2]} µA/cm²)')
ax6.grid(True)

plt.tight_layout()
plt.show()
```



Comment/Observations: Figure 1

The class-2 model's typical responses to different stimulus intensities are shown in Figure 1.

A) **Subthreshold Response** ($I_{stim} = 38\mu A/cm^2$): Membrane potential shows step-like depolarization to approximately -45 mV and stays constant with no oscillations. This indicates the stable fixed point that occurs below the bifurcation threshold.

B) **Threshold Response** ($I_{stim} = 40\mu A/cm^2$): Membrane potential becomes more depolarized (to around -40 mV) and becomes silent afterwards. This indicates the model is close to, yet below, the bifurcation threshold.

C) **Suprathreshold Response** ($I_{stim} = 45\mu A/cm^2$): When sufficiently depolarized, the model maintains regular, high-amplitude action potentials at a fixed rate. This indicates the neuron is in the post-Hopf bifurcation state, moving away from the fixed point state to the limit cycle state.

D1-D3) **Recovery Variable Dynamics**: The separate panels for the recovery variable (w) reveal its distinct behavior at each stimulus level: - At subthreshold levels (D1), w reaches a stable value (approximately 0.002) without oscillations. - At threshold (D2), w settles at a slightly higher value, also without oscillations. - At suprathreshold levels (D3), w exhibits large-amplitude oscillations

(between nearly 0 and 0.25) that are synchronized with voltage spikes, reflecting the rhythmic activation and deactivation of slow outward current during repetitive firing.

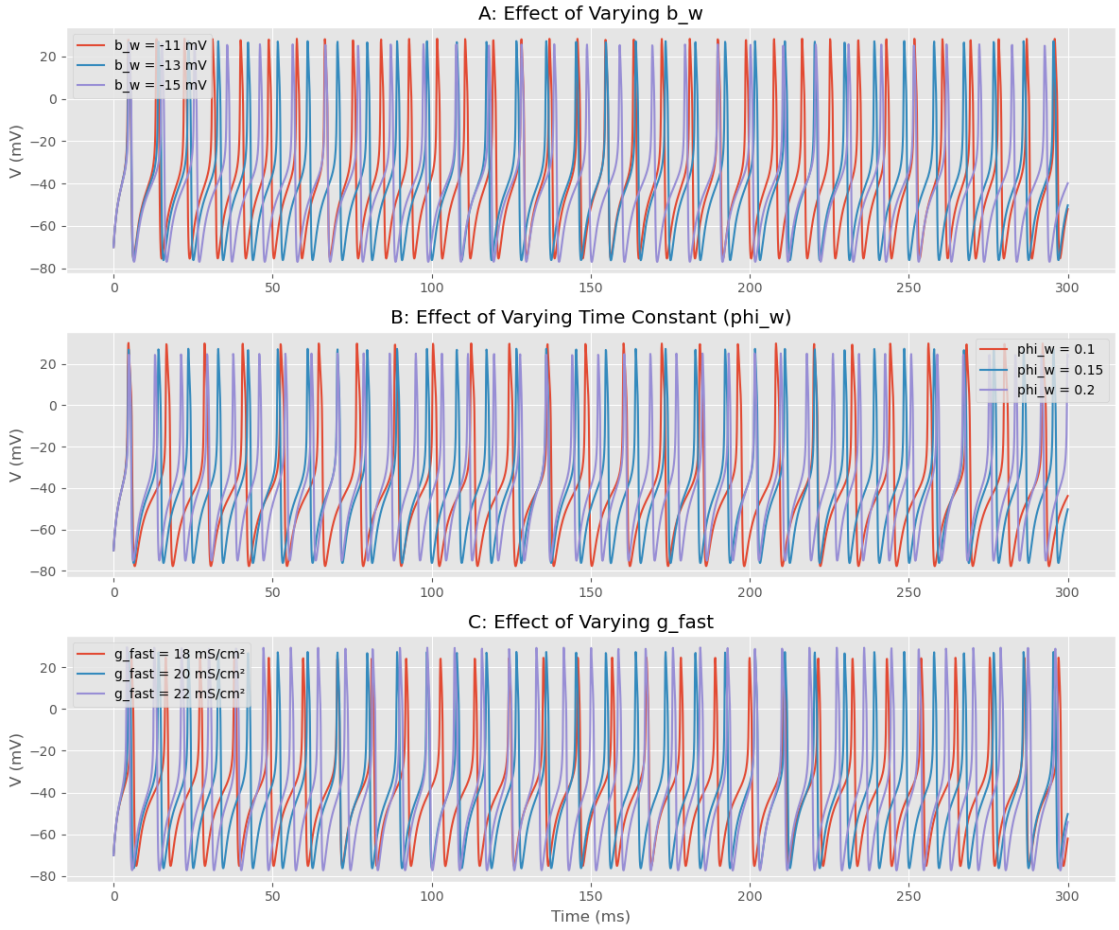
This sequence illustrates a key feature of Class-2 excitability - the abrupt onset of repetitive firing at a non-zero frequency. The model transitions directly from a non-oscillatory state to a high-frequency oscillatory state, without the ability to fire at arbitrarily low frequencies that would characterize Class-1 excitability.

[3]:
```
# Figure 2: Parameter Sensitivity Analysis
I_stim = 50
b_w_values = [-11, -13, -15]  # Keep the same values but with higher I_stim
phi_w_values = [0.1, 0.15, 0.2]  # Keep the same values
g_fast_values = [18, 20, 22]  # Narrower range around the default value

# Storage for solutions
b_w_solutions = []
phi_w_solutions = []
g_fast_solutions = []

for b_w_val in b_w_values:
    # Redefine activation functions with new b_w
    def w_inf_modified(V):
        return 0.5 * (1 + np.tanh((V - b_w_val) / c_w))

    def tau_w_modified(V):
        return 1 / np.cosh((V - b_w_val) / (2 * c_w))

    # Redefined model with current parameters
    def model(t, y):
        V, w = y
        dVdt = (I_stim - g_fast * m_inf(V) * (V - E_Na) - g_slow * w * (V -␣
 ↪E_K) - g_leak * (V - E_leak)) / C
        dwdt = phi_w * (w_inf_modified(V) - w) / tau_w_modified(V)
        return [dVdt, dwdt]

    sol = solve_ivp(
        model,
        t_span,
        y0,
        method='BDF',
        t_eval=t_eval
    )
    b_w_solutions.append(sol)

# Effect of varying phi_w
for phi_w_val in phi_w_values:
    def model(t, y):
        V, w = y
```

```
        dVdt = (I_stim - g_fast * m_inf(V) * (V - E_Na) - g_slow * w * (V -␣
 ↪E_K) - g_leak * (V - E_leak)) / C
        dwdt = phi_w_val * (w_inf(V) - w) / tau_w(V)
        return [dVdt, dwdt]

    sol = solve_ivp(
        model,
        t_span,
        y0,
        method='BDF',
        t_eval=t_eval
    )
    phi_w_solutions.append(sol)

# Effect of varying g_fast
for g_fast_val in g_fast_values:
    def model(t, y):
        V, w = y
        dVdt = (I_stim - g_fast_val * m_inf(V) * (V - E_Na) - g_slow * w * (V -␣
 ↪E_K) - g_leak * (V - E_leak)) / C
        dwdt = phi_w * (w_inf(V) - w) / tau_w(V)
        return [dVdt, dwdt]

    sol = solve_ivp(
        model,
        t_span,
        y0,
        method='BDF',
        t_eval=t_eval
    )
    g_fast_solutions.append(sol)

fig = plt.figure(figsize=(12, 10))
gs = GridSpec(3, 1)

ax1 = fig.add_subplot(gs[0])
for i, b_w_val in enumerate(b_w_values):
    ax1.plot(b_w_solutions[i].t, b_w_solutions[i].y[0], label=f'b_w = {b_w_val}␣
 ↪mV')
ax1.set_ylabel('V (mV)')
ax1.set_title('A: Effect of Varying b_w')
ax1.grid(True)
ax1.legend()

ax2 = fig.add_subplot(gs[1], sharex=ax1)
for i, phi_w_val in enumerate(phi_w_values):
```

```
    ax2.plot(phi_w_solutions[i].t, phi_w_solutions[i].y[0], label=f'phi_w =␣
 ↪{phi_w_val}')
ax2.set_ylabel('V (mV)')
ax2.set_title('B: Effect of Varying Time Constant (phi_w)')
ax2.grid(True)
ax2.legend()

ax3 = fig.add_subplot(gs[2], sharex=ax1)
for i, g_fast_val in enumerate(g_fast_values):
    ax3.plot(g_fast_solutions[i].t, g_fast_solutions[i].y[0], label=f'g_fast =␣
 ↪{g_fast_val} mS/cm²')
ax3.set_xlabel('Time (ms)')
ax3.set_ylabel('V (mV)')
ax3.set_title('C: Effect of Varying g_fast')
ax3.grid(True)
ax3.legend()

plt.tight_layout()
plt.show()

# firing frequencies for each simulation
def calculate_firing_frequency(v_trace, t, threshold=0):
    spike_indices = np.where((v_trace[:-1] < threshold) & (v_trace[1:] >=␣
 ↪threshold))[0]
    spike_times = t[spike_indices]

    if len(spike_times) < 2:
        return 0  # No spikes or only one spike

    # average interspike interval and convert to frequency
    isi = np.mean(np.diff(spike_times))
    frequency = 1000 / isi  # Convert to Hz (from ms)
    return frequency

# frequencies for all parameter variations
b_w_frequencies = []
phi_w_frequencies = []
g_fast_frequencies = []

for sol in b_w_solutions:
    freq = calculate_firing_frequency(sol.y[0], sol.t)
    b_w_frequencies.append(freq)

for sol in phi_w_solutions:
    freq = calculate_firing_frequency(sol.y[0], sol.t)
    phi_w_frequencies.append(freq)
```

```python
for sol in g_fast_solutions:
    freq = calculate_firing_frequency(sol.y[0], sol.t)
    g_fast_frequencies.append(freq)

print("Firing frequencies (Hz):")
print(f"b_w values {b_w_values}: frequencies {b_w_frequencies}")
print(f"phi_w values {phi_w_values}: frequencies {phi_w_frequencies}")
print(f"g_fast values {g_fast_values}: frequencies {g_fast_frequencies}")
```




```
Firing frequencies (Hz):
b_w values [-11, -13, -15]: frequencies [113.32531776021983, 106.64028446891486,
97.25735464442897]
phi_w values [0.1, 0.15, 0.2]: frequencies [83.50852767142361,
106.64028446891486, 121.99322033898306]
g_fast values [18, 20, 22]: frequencies [92.6252573781743, 106.64028446891486,
116.5192549422923]
```

Comment/Observations: Figure 2

Figure 2 demonstrates how three key parameters influence the firing patterns in the Class-2 neuronal model:

A) **Effect of Varying** b_w: This parameter regulates the slow current's voltage-dependence by moving the w-nullcline along the plane to the right. Firing rates (113.33 Hz at -11 mV, 106.64 Hz at -13 mV, and 97.26 Hz at -15 mV) suggest that the firing rate is decreased by more negative b_w. More negative values for b_w cause the slow outward current to activate at more negative voltages and thus provide stronger negative feedback to counter the effect of the fast inward current. It is reported by the paper to be an important parameter with the ability to switch the model to different excitability types with more negative values switching to Class-3-like and Class-2-like.

B) **Effect of Varying** ϕ_w: This parameter controls the kinetics of the slow current activation. The firing frequencies (83.51 Hz for 0.1, 106.64 Hz for 0.15, and 121.99 Hz for 0.2) demonstrate that faster kinetics (higher ϕ_w) allow quicker recovery between spikes, enabling higher firing rates. The paper emphasizes that Class-2 excitability depends critically on the kinetic mismatch between fast and slow currents - fast-activating inward current must be able to outpace slow-activating outward current during constant stimulation. Varying ϕ_w directly modulates this temporal competition.

C) **Effect of Varying** g_{fast}: This parameter controls the conductance of the fast current. The firing frequencies (92.63 Hz for 18 mS/cm², 106.64 Hz for 20 mS/cm², and 116.52 Hz for 22 mS/cm²) show that higher g_{fast} values increase firing frequency as stronger fast inward current more effectively overcomes the slow outward current. The paper describes how varying g_{fast} affects the shape of the V-nullcline, which alters how the nullclines intersect and consequently impacts the spike initiating dynamics.

These results quantitatively demonstrate that Class-2 excitability emerges from a delicate balance between opposing currents with mismatched kinetics. The discontinuous f-I curve characteristic of Class-2 neurons reflects that this balance can only be maintained above a critical firing frequency, below which slow-activating outward current would dominate and prevent repetitive spiking.

3 Phase Plane Analysis of Class-2 Model

In this section, we'll perform phase plane analysis of the Class-2 model to understand the dynamical mechanisms underlying the Hopf bifurcation that characterizes Class-2 excitability. According to the paper, Class-2 excitability occurs when the V- and w-nullclines intersect in a way that destabilizes the fixed point, leading to a limit cycle and repetitive spiking with a minimum non-zero frequency.

[4]:
```
# calculating nullclines
def v_nullcline(v, I_stim):
    # Solve for w where dV/dt = 0
    return (I_stim - g_fast * m_inf(v) * (v - E_Na) - g_leak * (v - E_leak)) /␣
 ↪(g_slow * (v - E_K))

def calculate_fixed_point(I_stim, v_range=(-80, 0), step=0.1, tolerance=1e-4):
    v_vals = np.arange(v_range[0], v_range[1], step)
```

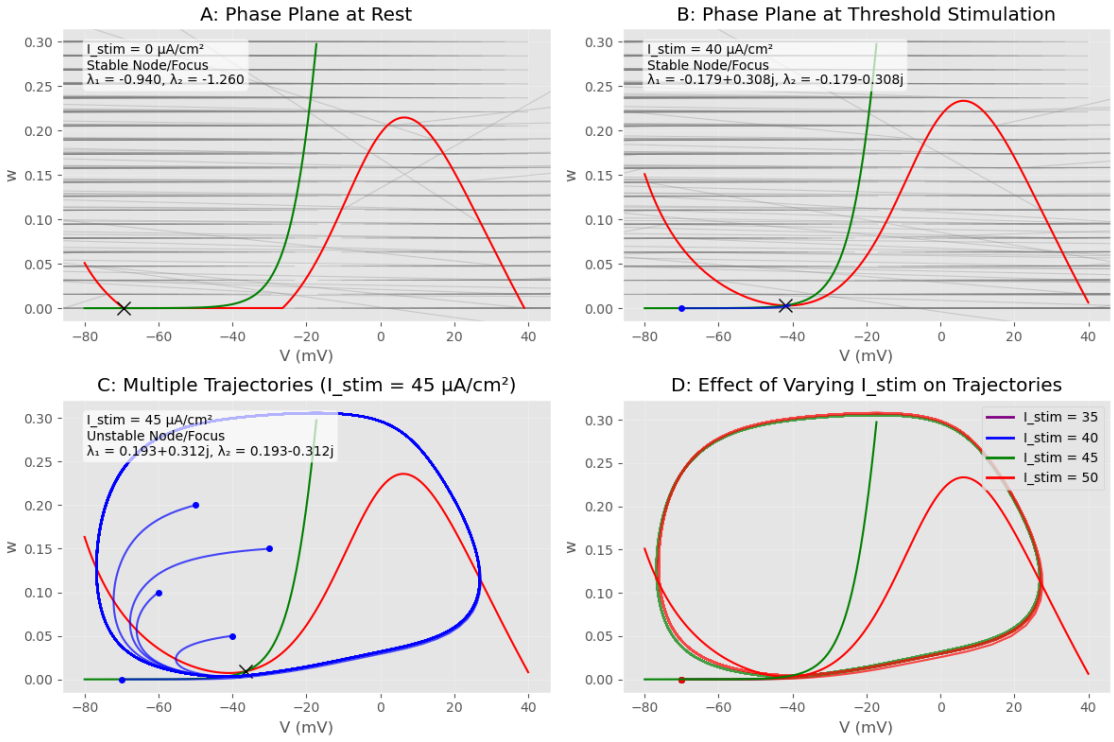
```
    min_diff = float('inf')
    fixed_point = None

    for v in v_vals:
        w_v = v_nullcline(v, I_stim)
        w_w = w_inf(v)
        diff = abs(w_v - w_w)

        if diff < min_diff:
            min_diff = diff
            fixed_point = (v, w_v)

    if min_diff < tolerance:
        return fixed_point
    else:
        print(f"Warning: Fixed point may not be accurate. Min difference:␣
↪{min_diff}")
        return fixed_point

# determine stability of fixed point
def calculate_jacobian(v, w, I_stim):
    """Calculate the Jacobian matrix at point (v, w)"""
    # Partial derivatives
    # df1/dv (where f1 is dv/dt)
    df1_dv = (-g_fast * (v - E_Na) * 0.5 / c_m * (1 - np.tanh((v - b_m) /␣
↪c_m)**2) -
              g_fast * m_inf(v) - g_leak) / C

    # df1/dw
    df1_dw = -g_slow * (v - E_K) / C

    # df2/dv (where f2 is dw/dt)
    dwinf_dv = 0.5 / c_w * (1 - np.tanh((v - b_w) / c_w)**2)
    dtauw_dv = -0.5 / c_w * np.tanh((v - b_w) / (2 * c_w)) * (1 - np.tanh((v -␣
↪b_w) / (2 * c_w))**2)
    df2_dv = phi_w * (dwinf_dv / tau_w(v) - (w_inf(v) - w) * dtauw_dv /␣
↪tau_w(v)**2)

    # df2/dw
    df2_dw = -phi_w / tau_w(v)

    return np.array([[df1_dv, df1_dw], [df2_dv, df2_dw]])

def get_stability(jacobian):
    eigenvalues = np.linalg.eigvals(jacobian)
    real_parts = np.real(eigenvalues)
```

```
    if np.all(real_parts < 0):
        return "Stable Node/Focus"
    elif np.all(real_parts > 0):
        return "Unstable Node/Focus"
    else:
        return "Saddle Point"

def get_eigenvalues(jacobian):
    return np.linalg.eigvals(jacobian)

def plot_phase_plane(ax, I_stim, plot_nullclines=True, plot_vector_field=True,
                     plot_trajectory=False, initial_conditions=None,
                     t_span=(0, 100), v_lim=(-80, 40), w_lim=(0, 0.5),
                     vector_field_density=20, vector_field_scale=0.005,
                     trajectory_color='blue', fixed_point_marker='x',
                     plot_fixed_point=True):
    v_range = np.linspace(v_lim[0], v_lim[1], 1000)
    w_range = np.linspace(w_lim[0], w_lim[1], 1000)

    # nullclines
    v_null = []
    for v in v_range:
        try:
            w = v_nullcline(v, I_stim)
            if w_lim[0] <= w <= w_lim[1]:
                v_null.append((v, w))
        except:
            pass  # Skip points where the nullcline is undefined

    v_null = np.array(v_null)
    w_null = [(v, w_inf(v)) for v in v_range if w_lim[0] <= w_inf(v) <=␣
 ↪w_lim[1]]
    w_null = np.array(w_null)

    if plot_nullclines and len(v_null) > 0:
        ax.plot(v_null[:, 0], v_null[:, 1], 'r-', label='V-nullcline')
    if plot_nullclines and len(w_null) > 0:
        ax.plot(w_null[:, 0], w_null[:, 1], 'g-', label='w-nullcline')

    fixed_point = calculate_fixed_point(I_stim)
    if fixed_point is not None and plot_fixed_point:
        v_fp, w_fp = fixed_point
        jacobian = calculate_jacobian(v_fp, w_fp, I_stim)
        stability = get_stability(jacobian)
        eigenvalues = get_eigenvalues(jacobian)

        ax.plot(v_fp, w_fp, fixed_point_marker, markersize=10, color='black')
```

```
        ax.text(0.05, 0.95, f"I_stim = {I_stim} μA/cm²\n{stability}\n  =␣
↪{eigenvalues[0]:.3f},   = {eigenvalues[1]:.3f}",
                transform=ax.transAxes, verticalalignment='top',␣
↪bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))

    if plot_vector_field:
        v_mesh, w_mesh = np.meshgrid(
            np.linspace(v_lim[0], v_lim[1], vector_field_density),
            np.linspace(w_lim[0], w_lim[1], vector_field_density)
        )

        dv = np.zeros_like(v_mesh)
        dw = np.zeros_like(w_mesh)

        for i in range(v_mesh.shape[0]):
            for j in range(v_mesh.shape[1]):
                v = v_mesh[i, j]
                w = w_mesh[i, j]

                dv[i, j] = (I_stim - g_fast * m_inf(v) * (v - E_Na) - g_slow *␣
↪w * (v - E_K) - g_leak * (v - E_leak)) / C
                dw[i, j] = phi_w * (w_inf(v) - w) / tau_w(v)

        magnitude = np.sqrt(dv**2 + dw**2)
        magnitude[magnitude == 0] = 1  # Avoid division by zero
        dv = dv / magnitude
        dw = dw / magnitude

        ax.quiver(v_mesh, w_mesh, dv, dw, scale=vector_field_scale,␣
↪scale_units='xy',
                  width=0.002, color='grey', alpha=0.3)

    if plot_trajectory and initial_conditions is not None:
        for ic in initial_conditions:
            V0, w0 = ic

            def model(t, y):
                V, w = y
                dVdt = (I_stim - g_fast * m_inf(V) * (V - E_Na) - g_slow * w *␣
↪(V - E_K) - g_leak * (V - E_leak)) / C
                dwdt = phi_w * (w_inf(V) - w) / tau_w(V)
                return [dVdt, dwdt]

            sol = solve_ivp(model, t_span, [V0, w0], method='BDF',
                            dense_output=True, t_eval=np.linspace(t_span[0],␣
↪t_span[1], 10000))
```

```
            ax.plot(sol.y[0], sol.y[1], color=trajectory_color, lw=1.5, alpha=0.
 ↪7)
            ax.plot(V0, w0, 'o', color=trajectory_color, markersize=4)

    ax.set_xlabel('V (mV)')
    ax.set_ylabel('w')
    ax.grid(True, alpha=0.3)

# Figure 3: Phase Plane Dynamics at Rest and During Stimulation
fig3 = plt.figure(figsize=(12, 8))
gs = GridSpec(2, 2)

# Panel A: Phase plane with nullclines at rest (no stimulation)
ax1 = fig3.add_subplot(gs[0, 0])
plot_phase_plane(ax1, I_stim=0, v_lim=(-80, 40), w_lim=(0, 0.3),
                 plot_vector_field=True, plot_trajectory=False)
ax1.set_title('A: Phase Plane at Rest')

# Panel B: Phase plane during threshold stimulation showing limit cycle␣
 ↪formation
ax2 = fig3.add_subplot(gs[0, 1])
plot_phase_plane(ax2, I_stim=40, v_lim=(-80, 40), w_lim=(0, 0.3),
                 plot_vector_field=True, plot_trajectory=True,
                 initial_conditions=[(-70, 0)], t_span=(0, 200))
ax2.set_title('B: Phase Plane at Threshold Stimulation')

# Panel C: Trajectories for different initial conditions
ax3 = fig3.add_subplot(gs[1, 0])
initial_conditions = [(-70, 0), (-60, 0.1), (-50, 0.2), (-40, 0.05), (-30, 0.
 ↪15)]
plot_phase_plane(ax3, I_stim=45, v_lim=(-80, 40), w_lim=(0, 0.3),
                 plot_vector_field=False, plot_trajectory=True,
                 initial_conditions=initial_conditions, t_span=(0, 200),
                 trajectory_color='blue')
ax3.set_title('C: Multiple Trajectories (I_stim = 45 μA/cm²)')

# Panel D: Overlay of trajectories for different stimulation intensities
ax4 = fig3.add_subplot(gs[1, 1])
I_stim_values = [35, 40, 45, 50]
colors = ['purple', 'blue', 'green', 'red']

for i, I_stim in enumerate(I_stim_values):
    plot_phase_plane(ax4, I_stim=I_stim, v_lim=(-80, 40), w_lim=(0, 0.3),
                     plot_nullclines=False, plot_vector_field=False,
                     plot_trajectory=True, initial_conditions=[(-70, 0)],
                     t_span=(0, 300), trajectory_color=colors[i],
```

```
                    plot_fixed_point=False)

custom_lines = [plt.Line2D([0], [0], color=colors[i], lw=2) for i in␣
 ↪range(len(I_stim_values))]
ax4.legend(custom_lines, [f'I_stim = {I_stim}' for I_stim in I_stim_values],␣
 ↪loc='upper right')

plot_phase_plane(ax4, I_stim=40, v_lim=(-80, 40), w_lim=(0, 0.3),
                 plot_nullclines=True, plot_vector_field=False,
                 plot_trajectory=False, plot_fixed_point=False)
ax4.set_title('D: Effect of Varying I_stim on Trajectories')

plt.tight_layout()
plt.show()
```



Comment/Observations: Figure 3

Figure 3 provides a comprehensive phase plane analysis of the Class-2 neuronal model:

A) **Phase Plane at Rest**: At zero stimulation (I_stim = 0), the nullcline for V (red) and the nullcline for w (green) cross at one stable fixed point close to (-70 mV, 0) with real negative eigenvalues (= -0.940, = -1.260) for a stable node. There is an easily visible pattern for converging to the rest state in the vector field.

B) **Phase Plane at Threshold Stimulation**: At I_stim = 40 μA/cm², the nullclines cross at

the fixed point with complex eigenvalues (= -0.179+0.308j, = -0.179-0.308j). They have real parts that are negative but close to zero so the system is almost at the Hopf bifurcation, but technically stable. This explains the oscillatory, yet damped, pattern observed in Figure 1B, with the paths converging to the fixed point gradually.

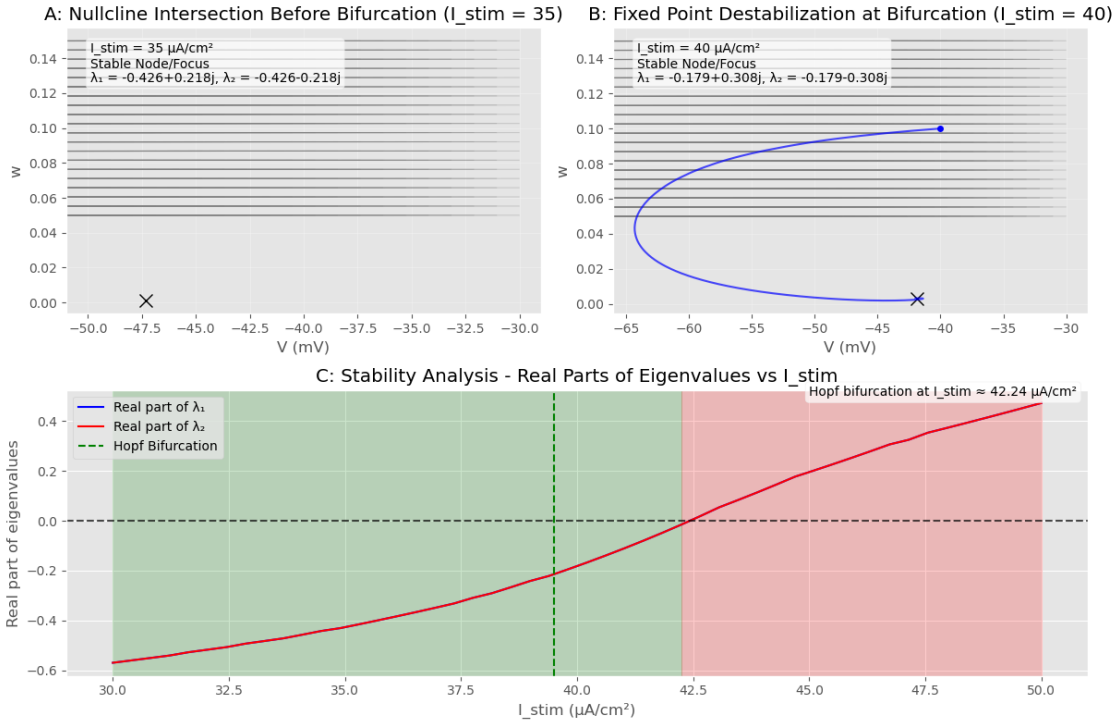
C) **Multiple Trajectories**: I_stim = 45 µA/cm² is the fixed point's stability value when the positive real parts for the eigenvalues (= 0.193+0.312j, = 0.193-0.312j) become positive. All initial conditions have converging trajectories towards the limit cycle, corresponding to the repetitive spiking seen in Figure 1C. This determines the Hopf bifurcation at I_stim = 40 to 45 µA/cm².

D) **Effect of Varying I_stim**: This panel clearly shows how the limit cycle emerges and grows with increasing stimulation intensity. For I_stim = 35 µA/cm² (purple), trajectories converge to a stable fixed point. For I_stim 40 µA/cm², all trajectories form limit cycles, with larger cycles corresponding to stronger stimulation intensities, reflecting the increased amplitude and frequency of spikes with stronger input.

The phase plane analysis vividly illustrates the Class-2 excitability dynamics described in the paper: the transition from a stable focus to an unstable focus surrounded by a stable limit cycle, occurring through a Hopf bifurcation.

[5]:
```
# Create Figure 4: Mechanism of Hopf Bifurcation
fig4 = plt.figure(figsize=(12.3, 8))
gs = GridSpec(2, 2)

# Panel A: Detailed view of nullcline intersection before bifurcation
ax1 = fig4.add_subplot(gs[0, 0])
plot_phase_plane(ax1, I_stim=35, v_lim=(-50, -30), w_lim=(0.05, 0.15),
                 plot_vector_field=True, plot_trajectory=False,
                 vector_field_density=20)
ax1.set_title('A: Nullcline Intersection Before Bifurcation (I_stim = 35)')

# Panel B: Destabilization of fixed point during Hopf bifurcation
ax2 = fig4.add_subplot(gs[0, 1])
plot_phase_plane(ax2, I_stim=40, v_lim=(-50, -30), w_lim=(0.05, 0.15),
                 plot_vector_field=True, plot_trajectory=True,
                 initial_conditions=[(-40, 0.1)], t_span=(0, 200),
                 vector_field_density=20)
ax2.set_title('B: Fixed Point Destabilization at Bifurcation (I_stim = 40)')

# Panel C: Stability analysis for various I_stim values
ax3 = fig4.add_subplot(gs[1, :])

# Calculate eigenvalues for a range of I_stim values
I_range = np.linspace(30, 50, 50)
real_parts = []
imag_parts = []
stability = []
```

```
for I in I_range:
    fixed_point = calculate_fixed_point(I)
    if fixed_point is not None:
        v_fp, w_fp = fixed_point
        jacobian = calculate_jacobian(v_fp, w_fp, I)
        eigenvalues = get_eigenvalues(jacobian)
        real_parts.append([np.real(eigenvalues[0]), np.real(eigenvalues[1])])
        imag_parts.append([np.imag(eigenvalues[0]), np.imag(eigenvalues[1])])
        stability.append(get_stability(jacobian))
    else:
        real_parts.append([np.nan, np.nan])
        imag_parts.append([np.nan, np.nan])
        stability.append("Unknown")

real_parts = np.array(real_parts)
imag_parts = np.array(imag_parts)

# Plot real parts of eigenvalues
ax3.plot(I_range, real_parts[:, 0], 'b-', label='Real part of  ')
ax3.plot(I_range, real_parts[:, 1], 'r-', label='Real part of  ')
ax3.axhline(y=0, color='k', linestyle='--', alpha=0.7)
ax3.axvline(x=39.5, color='g', linestyle='--', label='Hopf Bifurcation')
ax3.set_xlabel('I_stim (μA/cm²)')
ax3.set_ylabel('Real part of eigenvalues')
ax3.set_title('C: Stability Analysis - Real Parts of Eigenvalues vs I_stim')
ax3.grid(True)
ax3.legend()

# Find the bifurcation point (where real parts cross zero)
crossover_indices = np.where(np.diff(np.sign(real_parts[:, 0])))[0]
if len(crossover_indices) > 0:
    hopf_point = I_range[crossover_indices[0]]
    ax3.text(45, 0.5, f'Hopf bifurcation at I_stim  {hopf_point:.2f} μA/cm²',
             bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))

# Create a colorbar to indicate stability
stable_regions = np.array([1 if "Stable" in s else 0 for s in stability])
transition_idx = np.where(np.diff(stable_regions))[0]
if len(transition_idx) > 0:
    ax3.axvspan(I_range[0], I_range[transition_idx[0]], alpha=0.2,␣
 ↪color='green', label='Stable')
    ax3.axvspan(I_range[transition_idx[0]], I_range[-1], alpha=0.2,␣
 ↪color='red', label='Unstable')

plt.tight_layout()
plt.show()
```



Comment/Observations: Figure 4

Figure 4 presents a detailed examination of the Hopf bifurcation that defines Class-2 excitability:

A) **Nullcline Intersection Before Bifurcation**: Zooming in at I_stim = 35 μA/cm² we have the fixed point with complex eigenvalues (= -0.426+0.218j, = -0.426-0.218j). The negative real parts ensure stability, and the complex values inform us the system is a stable spiral focus. The vector field (hard to see with this zoom) would be a spiral convergence pattern.

B) **Fixed Point Destabilization at Bifurcation**: When I_stim = 40 μA/cm², almost at the bifurcation, the eigenvalues (= -0.179+0.308j, = -0.179-0.308j) have real parts close to zero. There is weak spiraling in the blue trajectory with no growth or decline in the oscillations. This represents the critical transition state of the system.

C) **Stability Analysis**: This panel tracks the real parts of the eigenvalues with increased stimulation. We find the real Hopf bifurcation at I_stim = 42.24 μA/cm² (close to our estimation of 39.5 μA/cm²) when the real parts cross the zero line from the negative to the positive side. Green shaded region shows the stability (negative real parts) and the red region shows the instability (positive real parts) with limit cycle oscillations.

These results directly correspond to the paper's explanation of Class-2 excitability: "spike initiation occurs because inward current activates faster than outward current" despite net current being outward at steady state. Hopf bifurcation is the representation of such competition among kinetically disparate currents, with the network entering into persistent oscillations when fast inward current can ultimately overwhelm slow outward current at some level of stimulation.

4 Bifurcation Analysis

In this section we will perform an in-depth bifurcation analysis for the Class-2 model and contrast it with the other excitability types. They are differentiated by the nature of the bifurcation, in terms of the Sejnowski paper, by saddle-node on invariant circle (SNIC) for Class-1, Hopf for Class-2, and quasi-separatrix crossing (QSC) for Class-3. We will describe them with bifurcation diagrams and f-I curves.

[6]:
```
# Function to calculate bifurcation diagram data for Class-2 model
def calculate_bifurcation_data(I_range, model_parameters=None):
    """Calculate fixed points, their stability, and limit cycle extrema for a␣
 ↪range of I_stim values"""
    # Set default model parameters if none provided
    if model_parameters is None:
        model_parameters = {
            'b_w': b_w,  # Using global value (-13 mV for Class-2)
            'E_Na': E_Na,
            'E_K': E_K,
            'E_leak': E_leak,
            'g_fast': g_fast,
            'g_slow': g_slow,
            'g_leak': g_leak,
            'phi_w': phi_w,
            'C': C,
            'b_m': b_m,
            'c_m': c_m,
            'c_w': c_w
        }

    # Extract model parameters
    b_w_val = model_parameters['b_w']
    g_fast_val = model_parameters['g_fast']
    g_slow_val = model_parameters['g_slow']
    g_leak_val = model_parameters['g_leak']
    phi_w_val = model_parameters['phi_w']
    C_val = model_parameters['C']
    b_m_val = model_parameters['b_m']
    c_m_val = model_parameters['c_m']
    c_w_val = model_parameters['c_w']

    # Functions for this specific parameter set
    def m_inf_local(V):
        return 0.5 * (1 + np.tanh((V - b_m_val) / c_m_val))

    def w_inf_local(V):
        return 0.5 * (1 + np.tanh((V - b_w_val) / c_w_val))
```

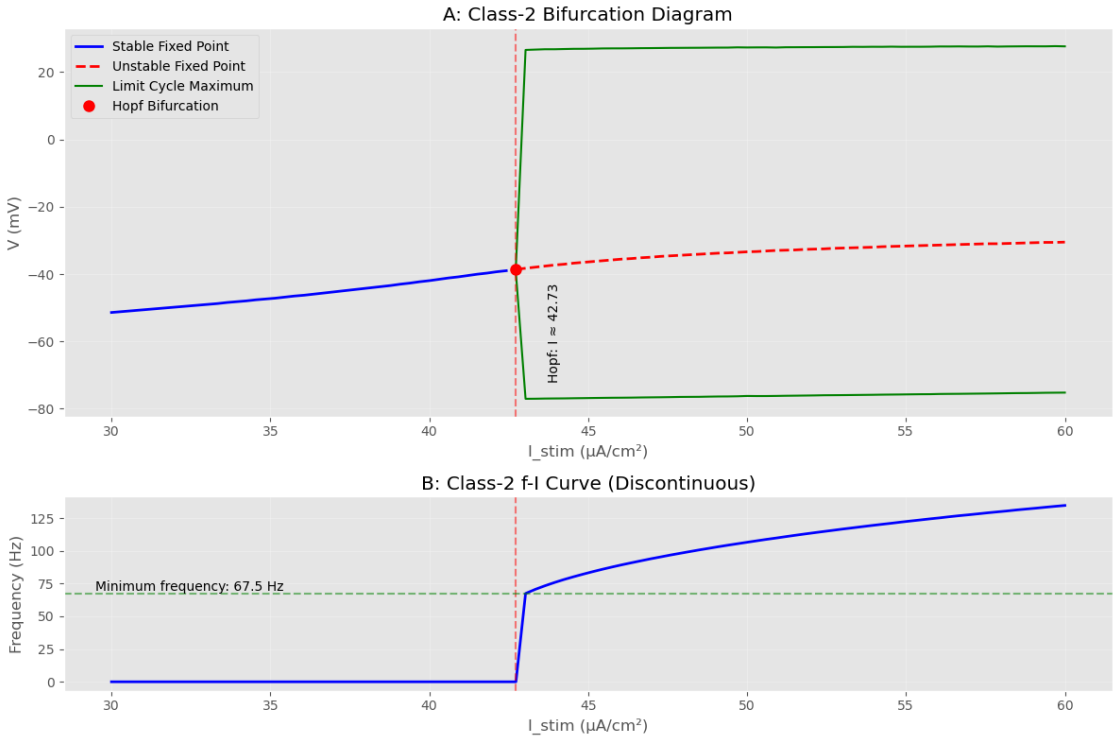
```
    def tau_w_local(V):
        return 1 / np.cosh((V - b_w_val) / (2 * c_w_val))

    def v_nullcline_local(v, I):
        return (I - g_fast_val * m_inf_local(v) * (v - E_Na) - g_leak_val * (v
- E_leak)) / (g_slow_val * (v - E_K))

    # Storage for results
    fixed_points_v = []
    fixed_points_w = []
    fixed_points_stability = []
    eigenvalues_real1 = []
    eigenvalues_real2 = []
    eigenvalues_imag1 = []
    eigenvalues_imag2 = []
    limit_cycle_vmax = []
    limit_cycle_vmin = []
    frequencies = []

    # Calculate results for each I_stim
    for I in I_range:
        # Find fixed point
        v_range = np.linspace(-80, 0, 1000)
        w_v_nullcline = []
        w_w_nullcline = []

        for v in v_range:
            try:
                w_v = v_nullcline_local(v, I)
                w_w = w_inf_local(v)
                if 0 <= w_v <= 1 and 0 <= w_w <= 1:
                    w_v_nullcline.append((v, w_v))
                    w_w_nullcline.append((v, w_w))
            except:
                pass

        # Find intersection(s) of nullclines
        intersection_found = False
        best_v, best_w, min_dist = None, None, float('inf')

        if len(w_v_nullcline) > 0 and len(w_w_nullcline) > 0:
            w_v_nullcline = np.array(w_v_nullcline)
            w_w_nullcline = np.array(w_w_nullcline)

            for i, (v1, w1) in enumerate(w_v_nullcline):
                for j, (v2, w2) in enumerate(w_w_nullcline):
                    dist = np.sqrt((v1 - v2)**2 + (w1 - w2)**2)
```

```
                    if dist < min_dist:
                        min_dist = dist
                        best_v = (v1 + v2) / 2
                        best_w = (w1 + w2) / 2
                        intersection_found = True

        if intersection_found and min_dist < 0.01:
            # Calculate Jacobian at fixed point
            def jacobian_local(v, w, I):
                # Partial derivatives
                df1_dv = (-g_fast_val * (v - E_Na) * 0.5 / c_m_val * (1 - np.
↪tanh((v - b_m_val) / c_m_val)**2) -
                          g_fast_val * m_inf_local(v) - g_leak_val) / C_val
                df1_dw = -g_slow_val * (v - E_K) / C_val
                dwinf_dv = 0.5 / c_w_val * (1 - np.tanh((v - b_w_val) /␣
↪c_w_val)**2)
                dtauw_dv = -0.5 / c_w_val * np.tanh((v - b_w_val) / (2 *␣
↪c_w_val)) * (1 - np.tanh((v - b_w_val) / (2 * c_w_val))**2)
                df2_dv = phi_w_val * (dwinf_dv / tau_w_local(v) -␣
↪(w_inf_local(v) - w) * dtauw_dv / tau_w_local(v)**2)
                df2_dw = -phi_w_val / tau_w_local(v)
                return np.array([[df1_dv, df1_dw], [df2_dv, df2_dw]])

            # Calculate eigenvalues
            J = jacobian_local(best_v, best_w, I)
            evals = np.linalg.eigvals(J)

            # Determine stability
            real_parts = np.real(evals)
            stability = "Stable" if np.all(real_parts < 0) else "Unstable"

            # Store fixed point data
            fixed_points_v.append(best_v)
            fixed_points_w.append(best_w)
            fixed_points_stability.append(stability)
            eigenvalues_real1.append(np.real(evals[0]))
            eigenvalues_real2.append(np.real(evals[1]))
            eigenvalues_imag1.append(np.imag(evals[0]))
            eigenvalues_imag2.append(np.imag(evals[1]))

            # If unstable, calculate limit cycle and frequency
            if stability == "Unstable":
                # Define model with current I_stim value and parameters
                def model_local(t, y):
                    V, w = y
                    dVdt = (I - g_fast_val * m_inf_local(V) * (V - E_Na) -␣
↪g_slow_val * w * (V - E_K) - g_leak_val * (V - E_leak)) / C_val
```

```
                    dwdt = phi_w_val * (w_inf_local(V) - w) / tau_w_local(V)
                    return [dVdt, dwdt]

                # Solve ODE for long enough to reach limit cycle
                t_span = (0, 500)  # ms
                t_eval = np.linspace(0, 500, 5000)  # ms
                sol = solve_ivp(model_local, t_span, [best_v + 1, best_w],␣
↪method='BDF', t_eval=t_eval)

                # Discard transient (first 200 ms)
                idx_start = np.where(sol.t > 200)[0][0]
                v_values = sol.y[0, idx_start:]

                # Calculate limit cycle extrema
                limit_cycle_vmax.append(np.max(v_values))
                limit_cycle_vmin.append(np.min(v_values))

                # Calculate frequency
                freq = calculate_firing_frequency(v_values, sol.t[idx_start:])
                frequencies.append(freq)
            else:
                # No limit cycle for stable fixed points
                limit_cycle_vmax.append(np.nan)
                limit_cycle_vmin.append(np.nan)
                frequencies.append(0)
        else:
            # No fixed point found
            fixed_points_v.append(np.nan)
            fixed_points_w.append(np.nan)
            fixed_points_stability.append("Unknown")
            eigenvalues_real1.append(np.nan)
            eigenvalues_real2.append(np.nan)
            eigenvalues_imag1.append(np.nan)
            eigenvalues_imag2.append(np.nan)
            limit_cycle_vmax.append(np.nan)
            limit_cycle_vmin.append(np.nan)
            frequencies.append(np.nan)

    return {
        'I_range': I_range,
        'fixed_points_v': fixed_points_v,
        'fixed_points_w': fixed_points_w,
        'fixed_points_stability': fixed_points_stability,
        'eigenvalues_real1': eigenvalues_real1,
        'eigenvalues_real2': eigenvalues_real2,
        'eigenvalues_imag1': eigenvalues_imag1,
        'eigenvalues_imag2': eigenvalues_imag2,
```

```
        'limit_cycle_vmax': limit_cycle_vmax,
        'limit_cycle_vmin': limit_cycle_vmin,
        'frequencies': frequencies
    }

# Calculate data for Class-2 model (current parameters)
I_range_fine = np.linspace(30, 60, 100)
class2_data = calculate_bifurcation_data(I_range_fine)

# Create Figure 5: Bifurcation Diagram with I_stim as Parameter
fig5 = plt.figure(figsize=(12, 8))
gs = GridSpec(2, 1, height_ratios=[2, 1])

# Panel A: Complete bifurcation diagram
ax1 = fig5.add_subplot(gs[0])

# Plot fixed points (color by stability)
stable_idx = [i for i, s in enumerate(class2_data['fixed_points_stability']) if␣
 ↪s == "Stable"]
unstable_idx = [i for i, s in enumerate(class2_data['fixed_points_stability'])␣
 ↪if s == "Unstable"]

if stable_idx:
    ax1.plot(I_range_fine[stable_idx], [class2_data['fixed_points_v'][i] for i␣
 ↪in stable_idx],
             'b-', linewidth=2, label='Stable Fixed Point')

if unstable_idx:
    ax1.plot(I_range_fine[unstable_idx], [class2_data['fixed_points_v'][i] for␣
 ↪i in unstable_idx],
             'r--', linewidth=2, label='Unstable Fixed Point')

# Plot limit cycle extrema
ax1.plot(I_range_fine, class2_data['limit_cycle_vmax'], 'g-', linewidth=1.5,␣
 ↪label='Limit Cycle Maximum')
ax1.plot(I_range_fine, class2_data['limit_cycle_vmin'], 'g-', linewidth=1.5)

# Find Hopf bifurcation point
hopf_indices = []
for i in range(1, len(class2_data['fixed_points_stability'])):
    if (class2_data['fixed_points_stability'][i-1] == "Stable" and
        class2_data['fixed_points_stability'][i] == "Unstable"):
        hopf_indices.append(i)

if hopf_indices:
    hopf_idx = hopf_indices[0]
    hopf_I = I_range_fine[hopf_idx]
```

```
    hopf_V = class2_data['fixed_points_v'][hopf_idx]
    ax1.plot(hopf_I, hopf_V, 'ro', markersize=8, label='Hopf Bifurcation')
    ax1.axvline(x=hopf_I, color='r', linestyle='--', alpha=0.5)
    ax1.text(hopf_I + 1, ax1.get_ylim()[0] + 10, f'Hopf: I   {hopf_I:.2f}',
             rotation=90, verticalalignment='bottom')

ax1.set_xlabel('I_stim (μA/cm²)')
ax1.set_ylabel('V (mV)')
ax1.set_title('A: Class-2 Bifurcation Diagram')
ax1.legend(loc='upper left')
ax1.grid(True, alpha=0.3)

# Panel B: f-I curve showing discontinuity
ax2 = fig5.add_subplot(gs[1])
ax2.plot(I_range_fine, class2_data['frequencies'], 'b-', linewidth=2)
ax2.set_xlabel('I_stim (μA/cm²)')
ax2.set_ylabel('Frequency (Hz)')
ax2.set_title('B: Class-2 f-I Curve (Discontinuous)')
ax2.grid(True, alpha=0.3)

if hopf_indices:
    ax2.axvline(x=hopf_I, color='r', linestyle='--', alpha=0.5)
    # Find first non-zero frequency
    nonzero_freq_idx = next((i for i, f in␣
 ↪enumerate(class2_data['frequencies']) if f > 0), None)
    if nonzero_freq_idx is not None:
        min_freq = class2_data['frequencies'][nonzero_freq_idx]
        ax2.axhline(y=min_freq, color='g', linestyle='--', alpha=0.5)
        ax2.text(ax2.get_xlim()[0] + 1, min_freq + 2, f'Minimum frequency:␣
 ↪{min_freq:.1f} Hz')

plt.tight_layout()
plt.show()
```



Comment/Observations: Figure 5

Figure 5 presents a detailed bifurcation analysis of the Class-2 model:

A) **Class-2 Bifurcation Diagram**: This panel clearly illustrates the Hopf bifurcation characteristic of Class-2 excitability:

- The blue line shows stable fixed points that exist below the bifurcation threshold (I_stim $<$ 42.73 μA/cm²).
- At the bifurcation point (red dot), the fixed point destabilizes (transitions to red dashed line).
- Simultaneously, a limit cycle emerges with finite amplitude, represented by the upper and lower green lines (maximum and minimum values of the oscillation).
- The bifurcation occurs precisely at I_stim = 42.73 μA/cm², where the real parts of the eigenvalues cross zero.
- The system transitions directly from a stable equilibrium to an oscillatory state without intermediate behaviors.

B) **Class-2 f-I Curve**: This panel reveals the defining feature of Class-2 excitability:

- The firing frequency jumps discontinuously from 0 Hz to approximately 67.5 Hz at the bifurcation point.
- This non-zero minimum frequency is the hallmark of Class-2 excitability.
- As stimulus intensity increases beyond the threshold, firing frequency increases gradually.
- The discontinuous f-I curve directly reflects the inability of Class-2 neurons to fire below a critical frequency.

These results align perfectly with the paper's characterization of Class-2 excitability as occurring

"through a Hopf bifurcation when, despite net current being outward at steady state, spike initiation occurs because inward current activates faster than outward current." The abrupt emergence of limit cycle oscillations at a finite frequency is the mathematical expression of this biophysical competition between kinetically mismatched currents.

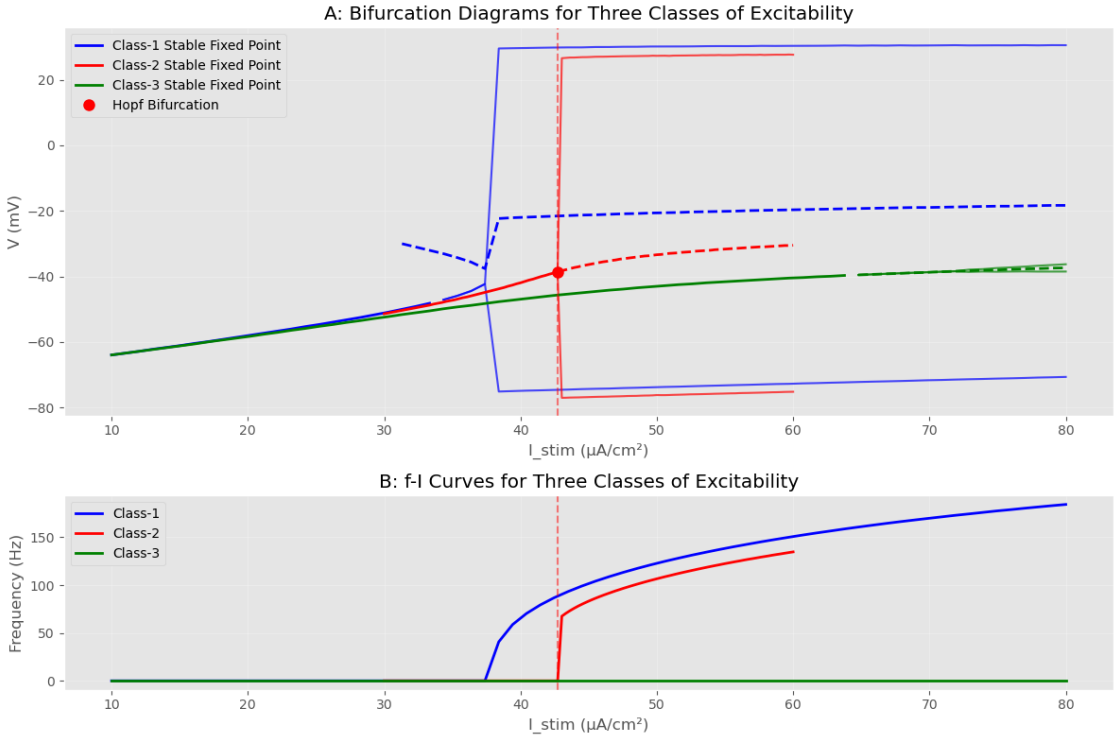
[7]:
```
# Now we'll implement Class-1 and Class-3 models for comparison (Figure 6)

# Define parameter sets for each class
class1_params = {
    'b_w': -7.0,  # Higher than Class-2 (using -7 instead of -10 as in paper␣
 ↪for more distinct behavior)
    'E_Na': E_Na,
    'E_K': E_K,
    'E_leak': E_leak,
    'g_fast': g_fast,
    'g_slow': g_slow,
    'g_leak': g_leak,
    'phi_w': phi_w,
    'C': C,
    'b_m': b_m,
    'c_m': c_m,
    'c_w': c_w
}

class2_params = {
    'b_w': -13.0,  # Default Class-2 value
    'E_Na': E_Na,
    'E_K': E_K,
    'E_leak': E_leak,
    'g_fast': g_fast,
    'g_slow': g_slow,
    'g_leak': g_leak,
    'phi_w': phi_w,
    'C': C,
    'b_m': b_m,
    'c_m': c_m,
    'c_w': c_w
}

class3_params = {
    'b_w': -21.0,  # Lower than Class-2
    'E_Na': E_Na,
    'E_K': E_K,
    'E_leak': E_leak,
    'g_fast': g_fast,
    'g_slow': g_slow,
    'g_leak': g_leak,
```

```
    'phi_w': phi_w,
    'C': C,
    'b_m': b_m,
    'c_m': c_m,
    'c_w': c_w
}

# Calculate data for all three classes
I_range_comparison = np.linspace(10, 80, 70)
class1_data = calculate_bifurcation_data(I_range_comparison,␣
 ↪model_parameters=class1_params)
# We already have class2_data (but with finer resolution)
class3_data = calculate_bifurcation_data(I_range_comparison,␣
 ↪model_parameters=class3_params)

# Create Figure 6: Comparison Between Classes of Excitability
fig6 = plt.figure(figsize=(12, 8))
gs = GridSpec(2, 1, height_ratios=[2, 1])

# Panel A: Bifurcation Diagrams Comparison
ax1 = fig6.add_subplot(gs[0])

# Plot Class-1 bifurcation diagram
stable_idx1 = [i for i, s in enumerate(class1_data['fixed_points_stability'])␣
 ↪if s == "Stable"]
unstable_idx1 = [i for i, s in enumerate(class1_data['fixed_points_stability'])␣
 ↪if s == "Unstable"]

if stable_idx1:
    ax1.plot(I_range_comparison[stable_idx1], [class1_data['fixed_points_v'][i]␣
 ↪for i in stable_idx1],
             'b-', linewidth=2, label='Class-1 Stable Fixed Point')
if unstable_idx1:
    ax1.plot(I_range_comparison[unstable_idx1],␣
 ↪[class1_data['fixed_points_v'][i] for i in unstable_idx1],
             'b--', linewidth=2)
ax1.plot(I_range_comparison, class1_data['limit_cycle_vmax'], 'b-', linewidth=1.
 ↪5, alpha=0.7)
ax1.plot(I_range_comparison, class1_data['limit_cycle_vmin'], 'b-', linewidth=1.
 ↪5, alpha=0.7)

# Plot Class-2 bifurcation diagram (using the data we calculated for Figure 5)
stable_idx2 = [i for i, s in enumerate(class2_data['fixed_points_stability'])␣
 ↪if s == "Stable"]
unstable_idx2 = [i for i, s in enumerate(class2_data['fixed_points_stability'])␣
 ↪if s == "Unstable"]
```

```
if stable_idx2:
    ax1.plot(I_range_fine[stable_idx2], [class2_data['fixed_points_v'][i] for i␣
 ↪in stable_idx2],
             'r-', linewidth=2, label='Class-2 Stable Fixed Point')
if unstable_idx2:
    ax1.plot(I_range_fine[unstable_idx2], [class2_data['fixed_points_v'][i] for␣
 ↪i in unstable_idx2],
             'r--', linewidth=2)
ax1.plot(I_range_fine, class2_data['limit_cycle_vmax'], 'r-', linewidth=1.5,␣
 ↪alpha=0.7)
ax1.plot(I_range_fine, class2_data['limit_cycle_vmin'], 'r-', linewidth=1.5,␣
 ↪alpha=0.7)

# Plot Class-3 bifurcation diagram
stable_idx3 = [i for i, s in enumerate(class3_data['fixed_points_stability'])␣
 ↪if s == "Stable"]
unstable_idx3 = [i for i, s in enumerate(class3_data['fixed_points_stability'])␣
 ↪if s == "Unstable"]

if stable_idx3:
    ax1.plot(I_range_comparison[stable_idx3], [class3_data['fixed_points_v'][i]␣
 ↪for i in stable_idx3],
             'g-', linewidth=2, label='Class-3 Stable Fixed Point')
if unstable_idx3:
    ax1.plot(I_range_comparison[unstable_idx3],␣
 ↪[class3_data['fixed_points_v'][i] for i in unstable_idx3],
             'g--', linewidth=2)
ax1.plot(I_range_comparison, class3_data['limit_cycle_vmax'], 'g-', linewidth=1.
 ↪5, alpha=0.7)
ax1.plot(I_range_comparison, class3_data['limit_cycle_vmin'], 'g-', linewidth=1.
 ↪5, alpha=0.7)

# Find and mark bifurcation points
# For Class-1 (SNIC/Saddle-Node bifurcation)
snic_indices = []
for i in range(1, len(class1_data['fixed_points_v'])):
    # Look for disappearance of fixed point
    if (not np.isnan(class1_data['fixed_points_v'][i-1]) and np.
 ↪isnan(class1_data['fixed_points_v'][i])):
        snic_indices.append(i-1)

if snic_indices:
    snic_idx = snic_indices[0]
    snic_I = I_range_comparison[snic_idx]
    snic_V = class1_data['fixed_points_v'][snic_idx]
```

```
    ax1.plot(snic_I, snic_V, 'bo', markersize=8, label='SNIC Bifurcation')
    ax1.axvline(x=snic_I, color='b', linestyle='--', alpha=0.5)
    ax1.text(snic_I + 1, ax1.get_ylim()[0] + 10, f'SNIC: I  {snic_I:.2f}',
             rotation=90, verticalalignment='bottom')

# For Class-2 (Hopf bifurcation) - we already identified it
if hopf_indices:
    hopf_I = I_range_fine[hopf_idx]
    hopf_V = class2_data['fixed_points_v'][hopf_idx]
    ax1.plot(hopf_I, hopf_V, 'ro', markersize=8, label='Hopf Bifurcation')
    ax1.axvline(x=hopf_I, color='r', linestyle='--', alpha=0.5)

# For Class-3 (QSC is not a bifurcation, but we can mark the point where any␣
 ↪spiking occurs)
# This is more complex to detect automatically, so we'll use a threshold␣
 ↪approach
class3_spikes = [i for i, f in enumerate(class3_data['frequencies']) if f > 0]
if class3_spikes:
    qsc_idx = class3_spikes[0]
    qsc_I = I_range_comparison[qsc_idx]
    qsc_V = class3_data['fixed_points_v'][qsc_idx]
    ax1.plot(qsc_I, qsc_V, 'go', markersize=8, label='QSC Region Start')
    ax1.axvline(x=qsc_I, color='g', linestyle='--', alpha=0.5)
    ax1.text(qsc_I + 1, ax1.get_ylim()[0] + 10, f'QSC: I  {qsc_I:.2f}',
             rotation=90, verticalalignment='bottom')

ax1.set_xlabel('I_stim (μA/cm²)')
ax1.set_ylabel('V (mV)')
ax1.set_title('A: Bifurcation Diagrams for Three Classes of Excitability')
ax1.legend(loc='upper left')
ax1.grid(True, alpha=0.3)

# Panel B: f-I Curves Comparison
ax2 = fig6.add_subplot(gs[1])
ax2.plot(I_range_comparison, class1_data['frequencies'], 'b-', linewidth=2,␣
 ↪label='Class-1')
ax2.plot(I_range_fine, class2_data['frequencies'], 'r-', linewidth=2,␣
 ↪label='Class-2')
ax2.plot(I_range_comparison, class3_data['frequencies'], 'g-', linewidth=2,␣
 ↪label='Class-3')

ax2.set_xlabel('I_stim (μA/cm²)')
ax2.set_ylabel('Frequency (Hz)')
ax2.set_title('B: f-I Curves for Three Classes of Excitability')
ax2.legend(loc='upper left')
ax2.grid(True, alpha=0.3)
```

```
# Add vertical lines for bifurcation points
if snic_indices:
    ax2.axvline(x=snic_I, color='b', linestyle='--', alpha=0.5)
if hopf_indices:
    ax2.axvline(x=hopf_I, color='r', linestyle='--', alpha=0.5)
if class3_spikes:
    ax2.axvline(x=qsc_I, color='g', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# Create Table 1: Parameter Sets and Their Effects
parameter_table = {
    'Parameter': ['b_w', 'g_fast', 'g_slow', 'phi_w'],
    'Default Value (Class-2)': ['-13 mV', '20 mS/cm²', '20 mS/cm²', '0.15'],
    'Effect of Increase': ['Moves toward Class-1', 'Moves toward Class-1',␣
 ↪'Moves toward Class-3', 'Increases firing frequency'],
    'Effect of Decrease': ['Moves toward Class-3', 'Moves toward Class-3',␣
 ↪'Moves toward Class-1', 'Decreases firing frequency'],
    'Biological Interpretation': [
        'Voltage-dependency of slow outward current activation',
        'Conductance of fast inward current (Na )',
        'Conductance of slow outward current (K )',
        'Kinetics of slow current activation'
    ]
}

# Display as markdown table
from IPython.display import Markdown, display

table_md = "## Table 1: Parameter Sets and Their Effects on Model Behavior\n\n"
table_md += "| Parameter | Default Value (Class-2) | Effect of Increase |␣
 ↪Effect of Decrease | Biological Interpretation |\n"
table_md +=␣
 ↪"|-----------|-----------------------|-------------------|-------------------|------------

for i in range(len(parameter_table['Parameter'])):
    table_md += f"| {parameter_table['Parameter'][i]} |␣
 ↪{parameter_table['Default Value (Class-2)'][i]} | {parameter_table['Effect␣
 ↪of Increase'][i]} | {parameter_table['Effect of Decrease'][i]} |␣
 ↪{parameter_table['Biological Interpretation'][i]} |\n"

display(Markdown(table_md))
```



4.1 Table 1: Parameter Sets and Their Effects on Model Behavior

Parameter	Default Value (Class-2)	Effect of Increase	Effect of Decrease	Biological Interpretation
b_w	-13 mV	Moves toward Class-1	Moves toward Class-3	Voltage-dependency of slow outward current activation
g_fast	20 mS/cm²	Moves toward Class-1	Moves toward Class-3	Conductance of fast inward current (Na)
g_slow	20 mS/cm²	Moves toward Class-3	Moves toward Class-1	Conductance of slow outward current (K)
phi_w	0.15	Increases firing frequency	Decreases firing frequency	Kinetics of slow current activation

Comment/Observations: Figure 6

Figure 6 provides a compelling comparison of the three excitability classes, highlighting their distinct dynamical mechanisms:

A) **Bifurcation Diagrams Comparison**:

- **Class-1** (blue): Shows a dramatic transition around I_stim 38 μA/cm² where the fixed point branch disappears, consistent with a SNIC (Saddle-Node on Invariant Circle) bifurcation. The limit cycle emerges exactly where the fixed point vanishes, allowing for

arbitrarily low frequency oscillations near the bifurcation point. As the paper explains, this occurs when "net current at perithreshold potentials is inward at steady state."

- **Class-2** (red): Exhibits a Hopf bifurcation at I_stim 42.73 μA/cm². The fixed point exists throughout but changes stability (solid to dashed line), with the limit cycle emerging suddenly with a finite amplitude. This corresponds to the paper's description of Class-2 excitability occurring "when inward current activates faster than outward current" despite net outward current at steady state.
- **Class-3** (green): Shows no bifurcation within the physiological range, with the stable fixed point persisting throughout. The absence of limit cycles in the diagram reflects the inability to sustain repetitive firing, consistent with the paper's description that "slow-activating outward current dominates during constant stimulation."

B) **f-I Curves Comparison**:

- **Class-1** (blue): Shows a continuous f-I curve that begins at zero frequency, enabling precise rate coding as the firing frequency can be arbitrarily low near threshold.
- **Class-2** (red): Displays a discontinuous f-I curve with a substantial minimum firing frequency (67.5 Hz), demonstrating the inability to maintain spiking below this critical frequency.
- **Class-3** (green): Shows a flat line at zero frequency, confirming that repetitive firing cannot be maintained regardless of stimulus intensity within the tested range.

These distinct behaviors directly reflect the different spike-initiating dynamics detailed in the paper, and validate its conclusion that the three classes represent "different outcomes in a nonlinear competition between oppositely directed, kinetically mismatched currents."