04_calcium_2V_oscillations

November 14, 2025

1 Oscillator: Calcium-IP3 Oscillations

- Two variable set-up as in the X<->Y oscillator. X = cytosolic calcium; Y = IP3
- Steady state to oscillation transition as calcium supply is increased
- Basis of a spatio-temporal model

[1]:
```
from scipy.integrate import solve_ivp
from matplotlib.pyplot import subplots
from numpy import linspace, around, var, ndarray
from scipy.signal import find_peaks
```

[2]:
```
def plot_bifdiagram(results_min_f, results_max_f,
                    par_set):

    N = len(results_min_f)

    fig, ax = subplots(figsize=(6, 4))

    for xe, ye in zip(par_set, results_max_f[0]):

        if not isinstance(ye, ndarray):
            ax.scatter(xe, ye, c='k', s=6, marker='D')
        else:
            ax.scatter([xe] * len(ye), ye, s=3, c='r', marker='D')

    for xe, ye in zip(par_set, results_min_f[0]):

        if not isinstance(ye, ndarray):
            ax.scatter(xe, ye, c='gray', s=6, marker='d')
        else:
            ax.scatter([xe] * len(ye), ye, s=3, c='b', marker='d')


    ax.set_xticks(linspace(par_set[0], par_set[-1], 5));
    ax.set_xticklabels(around(linspace(par_set[0], par_set[-1], 5), 2),␣
↪fontsize=16);
    ax.set_xlabel('Parameter', fontsize=16)
```

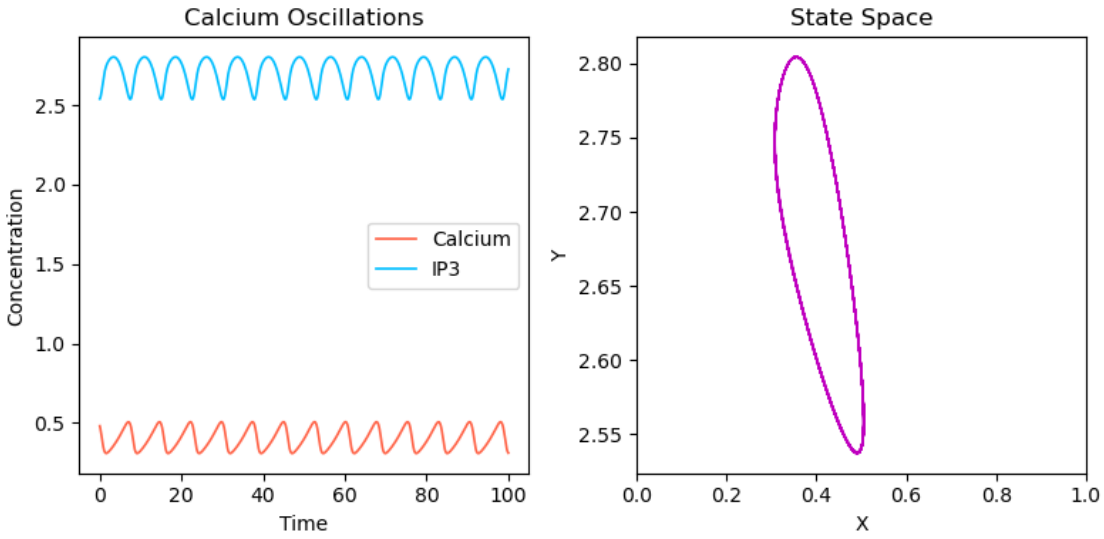
```
    ax.set_ylabel('EX', fontsize=14)

    y_min, y_max = ax.get_ylim()

    ax.set_yticks(linspace(y_min, y_max, 3));
    ax.set_yticklabels(around(linspace(y_min, y_max, 3),2), fontsize=14);

    fig.tight_layout()

    return fig, ax
```

2 Model of Calcium - IP3 interaction

2.1 Time Series

[3]:
```
def model(t, variables, a, m2, m3, ka, k, k1):
    """Coupled system with feedback inhibition"""
    X, Y = variables

    dXdt = a - m2*X/(1+X) + (m3*Y/(k1+Y))*X**2/(ka+X**2) + Y - k*X
    dYdt =     m2*X/(1+X) - (m3*Y/(k1+Y))*X**2/(ka+X**2) - Y

    return [dXdt, dYdt]

X_0 = 2.5
Y_0 = 1.0

y0 = [X_0, Y_0]
y0 = [0.48, 2.54]

a         = 0.32
m2, m3    = 20, 23
ka, k, k1 = 0.8, 0.8, 0.8

t_span = (0, 100)

solution = solve_ivp(model, t_span, y0, args=(a, m2, m3, ka, k, k1),␣
 ↪method='BDF', max_step=0.1)

t = solution.t

X = solution.y[0]
Y = solution.y[1]

fig, ax = subplots(ncols=2, figsize=(8, 4))

ax[0].plot(t, X, label='Calcium', linewidth=1.2, color='tomato')
```

```
ax[0].plot(t, Y, label='IP3', linewidth=1.2, color='deepskyblue')

ax[0].set_xlabel('Time')
ax[0].set_ylabel('Concentration')
ax[0].legend()
ax[0].set_title('Calcium Oscillations')

ax[1].plot(X, Y, linewidth=1, color='m');
# ax[1].plot(c2[:300], c3[:300], linewidth=1, color='k');
ax[1].set_xlabel('X')
ax[1].set_ylabel('Y')
ax[1].set_title('State Space')
ax[1].set_xlim(0, 1)
# ax[1].set_ylim(4, 14)


fig.tight_layout()
```



[4]:
```
around((X[-1], Y[-1]), 2)
```

[4]:
```
array([0.31, 2.73])
```

2.2 Bifurcation Diagram

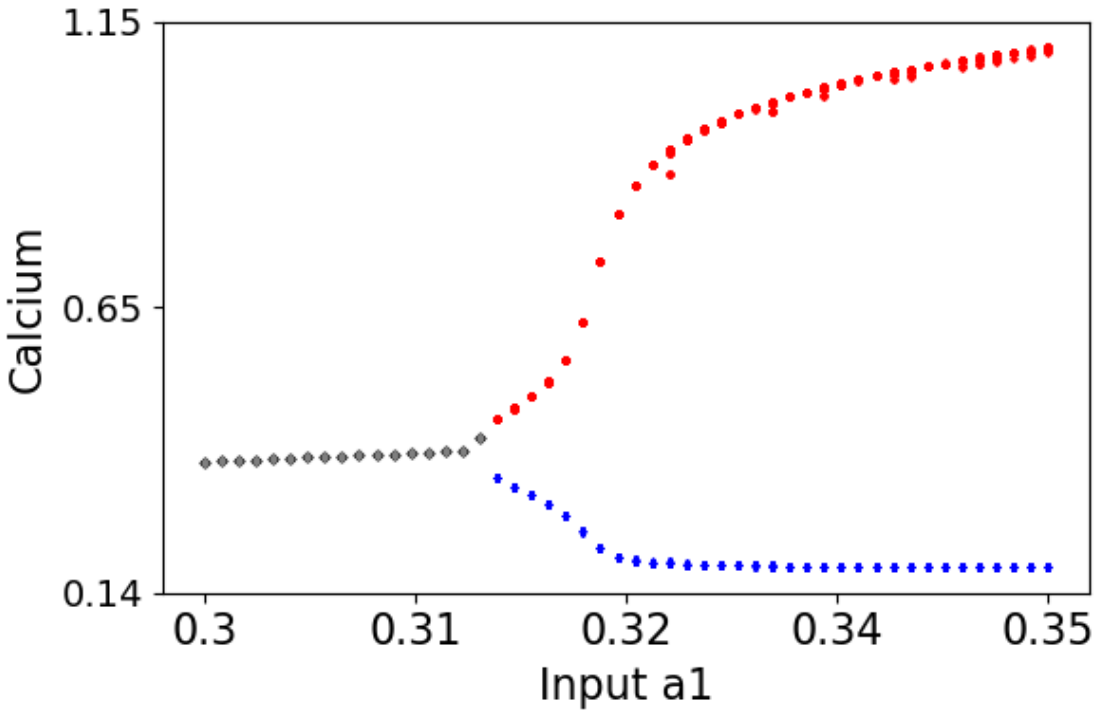
[5]:
```
# Bifurcation parameter set
par_min, par_max, steps = 0.3, 0.35, 50
# par_min, par_max, steps = 0.75, 0.63, 30
```

```
par_set = linspace(par_min, par_max, steps)

# Time array
t_span = (0, 500)

results_min_f      = dict()
results_min_inds_f = dict()
results_max_f      = dict()
results_max_inds_f = dict()

# Simulation "forward"
for par in par_set:

    solution = solve_ivp(model, t_span, y0, args=(par, m2, m3, ka, k, k1),␣
 ↪method='BDF', max_step=0.1)

    X = solution.y[0]
    Y = solution.y[1]

    rows = X.size//2

    series = X[rows//2:]

    num = 0

    if var(series) < 0.001:

        if num not in results_min_f:

            results_min_f[num]      = [series[-1]]
            results_min_inds_f[num] = [0]

        else:
            results_min_f[num].append(series[-1])
            results_min_inds_f[num].append(0)

        if num not in results_max_f:

            results_max_f[num]      = [series[-1]]
            results_max_inds_f[num] = [0]

        else:
            results_max_f[num].append(series[-1])
            results_max_inds_f[num].append(0)

    else:
```

```
        y_f_max_inds = find_peaks(series, distance=100)
        y_f_maxs     = series[y_f_max_inds[0]]

        y_f_min_inds = find_peaks(-series, distance=100)
        y_f_mins     = series[y_f_min_inds[0]]

        if num not in results_min_f:

            results_min_f[num]      = [y_f_mins]
            results_min_inds_f[num] = [y_f_min_inds]

            results_max_f[num]      = [y_f_maxs]
            results_max_inds_f[num] = [y_f_max_inds]

        else:

            results_min_f[num].append(y_f_mins)
            results_min_inds_f[num].append(y_f_min_inds)

            results_max_f[num].append(y_f_maxs)
            results_max_inds_f[num].append(y_f_max_inds)


    if par != par_set[-1]:

        y0 = solution.y[:, -1]

print('')
print('Scan complete!', list(around(solution.y[:, -1],3)))
print('')
```

```
Scan complete! [np.float64(0.29), np.float64(2.725)]
```

[8]:
```
fig, ax = plot_bifdiagram(results_min_f, results_max_f, par_set)

title_chars = 'Input a1'

ax.set_xlabel(title_chars, fontsize=16);
ax.set_ylabel('Calcium', fontsize=16);
```



[ ]:

[ ]:

[ ]:

[ ]:

3 Oscillation animation

[11]:
```
import numpy as np
import plotly.graph_objects as go
from scipy.integrate import solve_ivp

X_0 = 2.5
Y_0 = 1.0

y0 = [X_0, Y_0]
y0 = [0.48, 2.54]

a         = 0.32
m2, m3    = 20, 23
ka, k, k1 = 0.8, 0.8, 0.8
```

```
t_span = (0, 100)

solution = solve_ivp(model, t_span, y0, args=(a, m2, m3, ka, k, k1),␣
 ↪method='BDF', max_step=0.1)

X_sol = solution.y[0]
Y_sol = solution.y[1]
t_sol = solution.t


# Normalize for circle radii - using pixel units for shapes
min_radius, max_radius = 15, 40
X_radius = min_radius + (max_radius - min_radius) * (X_sol - np.min(X_sol)) /␣
 ↪(np.max(X_sol) - np.min(X_sol))
Y_radius = min_radius + (max_radius - min_radius) * (Y_sol - np.min(Y_sol)) /␣
 ↪(np.max(Y_sol) - np.min(Y_sol))

# Fixed positions
X_pos = [-4, 0]
Y_pos = [4, 0]

# Create the figure with shapes (much more stable than scatter plots)
fig = go.Figure()

# Create frames using shapes instead of scatter plots
frames = []
for i in range(len(t_sol)):
    frame_data = [
        # S circle as shape
        dict(
            type="circle",
            xref="x", yref="y",
            x0=X_pos[0] - X_radius[i]/20,  # Divide by scaling factor
            y0=X_pos[1] - X_radius[i]/20,
            x1=X_pos[0] + X_radius[i]/20,
            y1=X_pos[1] + X_radius[i]/20,
            line_color="blue",
            fillcolor="blue",
            line_width=2,
            opacity=0.7
        ),
        # P circle as shape
        dict(
            type="circle",
            xref="x", yref="y",
            x0=Y_pos[1] - Y_radius[i]/20,
```

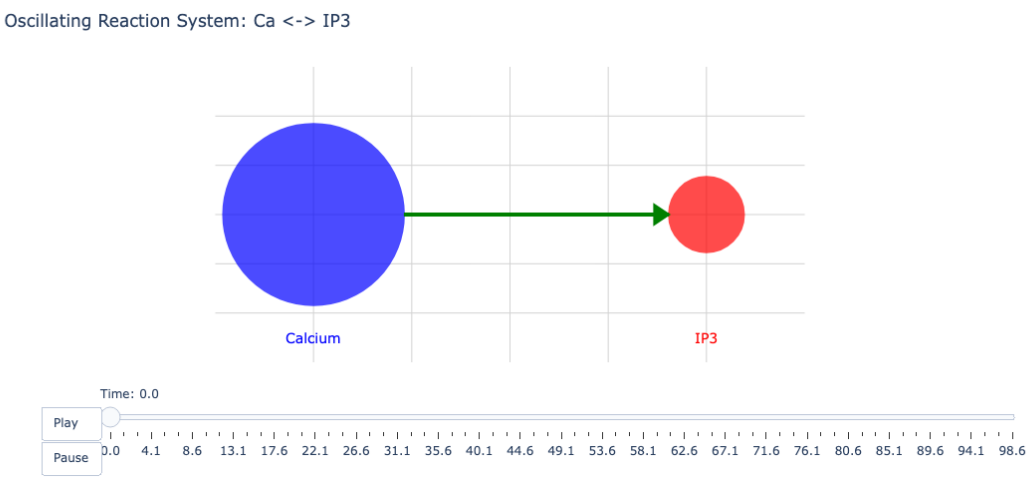
```
            x1=Y_pos[0] + Y_radius[i]/20,
            y1=Y_pos[1] + Y_radius[i]/20,
            line_color="red",
            fillcolor="red",
            line_width=2,
            opacity=0.7
        ),
        # Arrow line
        dict(
            type="line",
            xref="x", yref="y",
            x0=X_pos[0] + X_radius[i]/20,
            y0=X_pos[1],
            x1=Y_pos[0] - Y_radius[i]/20,
            y1=Y_pos[1],
            line=dict(color="green", width=4)
        ),
        # Arrow head
        # dict(
        #     type="path",
        #     xref="x", yref="y",
        #     path=f"M {Y_pos[0] - Y_radius[i]/20 - 0.3} {Y_pos[1] - 0.2} L␣
↪{Y_pos[0] - Y_radius[i]/20} {Y_pos[1]} L {Y_pos[0] - Y_radius[i]/20 - 0.3}␣
↪{Y_pos[1] + 0.2} Z",
        #     line_color="green",
        #     fillcolor="green"
        # )
    ]


    frame = go.Frame(
        data=[],  # No scatter data
        layout=dict(
            shapes=frame_data,
            # annotations=text_annotations
        ),
        name=f'frame_{i}'
    )
    frames.append(frame)

# Add initial shapes
fig.add_shape(
    type="circle",
    xref="x", yref="y",
    x0=X_pos[0] - X_radius[0]/20,
    y0=X_pos[1] - X_radius[0]/20,
    x1=X_pos[0] + X_radius[0]/20,
```

```
    y1=X_pos[1] + X_radius[0]/20,
    line_color="blue",
    fillcolor="blue",
    line_width=2,
    opacity=0.7
)

fig.add_shape(
    type="circle",
    xref="x", yref="y",
    x0=Y_pos[0] - Y_radius[0]/20,
    y0=Y_pos[1] - Y_radius[0]/20,
    x1=Y_pos[0] + Y_radius[0]/20,
    y1=Y_pos[1] + Y_radius[0]/20,
    line_color="red",
    fillcolor="red",
    line_width=2,
    opacity=0.7
)

fig.add_shape(
    type="line",
    xref="x", yref="y",
    x0=X_pos[0] + X_radius[0]/20,
    y0=X_pos[1],
    x1=Y_pos[0] - Y_radius[0]/20,
    y1=Y_pos[1],
    line=dict(color="green", width=4)
)

fig.add_shape(
    type="path",
    xref="x", yref="y",
    path=f"M {Y_pos[0] - Y_radius[0]/20 - 0.3} {Y_pos[1] - 0.2} L {Y_pos[0] -␣
 ↪Y_radius[0]/20} {Y_pos[1]} L {Y_pos[0] - Y_radius[0]/20 - 0.3} {Y_pos[1] + 0.
 ↪2} Z",
    line_color="green",
    fillcolor="green"
)

# Add initial annotations
fig.add_annotation(
    x=X_pos[0],
    y=X_pos[1] - max(X_radius[0], Y_radius[0])/15 - 0.8,
    # text=f"S: {S_sol[0]:.2f}",
    showarrow=False,
    font=dict(color="blue", size=12),
```

```
    xref="x",
    yref="y"
)

fig.add_annotation(
    x=Y_pos[0],
    y=Y_pos[1] - max(X_radius[0], Y_radius[0])/15 - 0.8,
    # text=f"P: {P_sol[0]:.2f}",
    showarrow=False,
    font=dict(color="red", size=12),
    xref="x",
    yref="y"
)

# Configure animation
fig.frames = frames

# Animation controls
sliders = [dict(
    steps=[dict(
        method="animate",
        args=[[f'frame_{k}'],
              dict(mode="immediate", frame=dict(duration=50, redraw=True),
                   transition=dict(duration=0))],
        label=f'{t_sol[k]:.1f}'
    ) for k in range(0, len(t_sol), 15)],
    active=0,
    currentvalue=dict(prefix="Time: ", visible=True),
    len=0.9,
    x=0.1,
    y=0
)]

updatemenus = [dict(
    type="buttons",
    showactive=False,
    buttons=[
        dict(label="Play", method="animate",
             args=[None, dict(frame=dict(duration=50, redraw=True),
                              fromcurrent=True, mode="immediate")]),
        dict(label="Pause", method="animate",
             args=[[None], dict(frame=dict(duration=0, redraw=False),
                                mode="immediate")])
    ],
    x=0.1, y=-0.15,
    xanchor="right",
    yanchor="top"
```

```
)]

# Update layout with FIXED ranges and no auto-scaling
fig.update_layout(
    title="Oscillating Reaction System: Ca <-> IP3",
    xaxis=dict(
        range=[-6, 6],  # Fixed range
        showticklabels=False,
        showgrid=True,
        gridcolor="lightgray",
        zeroline=False,
        fixedrange=True,  # Prevent any scaling
        constrain="domain"
    ),
    yaxis=dict(
        range=[-3, 3],  # Fixed range
        showticklabels=False,
        showgrid=True,
        gridcolor="lightgray",
        zeroline=False,
        fixedrange=True,  # Prevent any scaling
        scaleanchor="x",
        scaleratio=1
    ),
    plot_bgcolor="white",
    width=800,
    height=500,
    showlegend=False,
    sliders=sliders,
    updatemenus=updatemenus,
    # Disable auto-scaling and layout recalculations
    autosize=False,
    margin=dict(l=50, r=50, t=80, b=80)
)

# Add static labels
fig.add_annotation(
    x=X_pos[0], y=-2.5,
    text="Calcium",
    showarrow=False,
    font=dict(color="blue", size=14),
    xref="x",
    yref="y"
)

fig.add_annotation(
    x=Y_pos[0], y=-2.5,
```

```
    text="IP3",
    showarrow=False,
    font=dict(color="red", size=14),
    xref="x",
    yref="y"
)

fig.show()
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[9]:
```
import numpy as np
import plotly.graph_objects as go
from scipy.integrate import solve_ivp

# Your model definition
def model(t, variables, a1, b1, a2, b2, k_max, K_m, k_i, n, m, q):
    """Coupled system with feedback inhibition"""
    S, P = variables

    enzymatic_rate = (k_max * S**n) / (K_m**m + S**m) / (1 + k_i * P**q)

    dSdt = a1 - b1 * S - enzymatic_rate
    dPdt = a2 - b2 * P + enzymatic_rate

    return [dSdt, dPdt]

# Parameters
S_0 = 1.97
P_0 = 6.97
b1, b2 = 0.18, 0.05
a1, a2 = 0.7, 0.02
k_max, K_m, k_i = 25.0, 0.7, 0.06
n, m, q = 1, 3, 2.8
t_span = (0, 700)

# Solve the ODE
t_eval = np.linspace(t_span[0], t_span[1], 1000)  # Even fewer points for␣
 ↪stability
sol = solve_ivp(model, t_span, [S_0, P_0], args=(a1, b1, a2, b2, k_max, K_m,␣
 ↪k_i, n, m, q),
                t_eval=t_eval, method='RK45')

S_sol = sol.y[0]
P_sol = sol.y[1]
t_sol = sol.t

# Normalize for circle radii - using pixel units for shapes
min_radius, max_radius = 15, 40
S_radius = min_radius + (max_radius - min_radius) * (S_sol - np.min(S_sol)) /␣
 ↪(np.max(S_sol) - np.min(S_sol))
P_radius = min_radius + (max_radius - min_radius) * (P_sol - np.min(P_sol)) /␣
 ↪(np.max(P_sol) - np.min(P_sol))

# Fixed positions
S_pos = [-4, 0]
P_pos = [4, 0]
```

```
# Create the figure with shapes (much more stable than scatter plots)
fig = go.Figure()

# Create frames using shapes instead of scatter plots
frames = []
for i in range(len(t_sol)):
    frame_data = [
        # S circle as shape
        dict(
            type="circle",
            xref="x", yref="y",
            x0=S_pos[0] - S_radius[i]/20,  # Divide by scaling factor
            y0=S_pos[1] - S_radius[i]/20,
            x1=S_pos[0] + S_radius[i]/20,
            y1=S_pos[1] + S_radius[i]/20,
            line_color="blue",
            fillcolor="blue",
            line_width=2,
            opacity=0.7
        ),
        # P circle as shape
        dict(
            type="circle",
            xref="x", yref="y",
            x0=P_pos[0] - P_radius[i]/20,
            y0=P_pos[1] - P_radius[i]/20,
            x1=P_pos[0] + P_radius[i]/20,
            y1=P_pos[1] + P_radius[i]/20,
            line_color="red",
            fillcolor="red",
            line_width=2,
            opacity=0.7
        ),
        # Arrow line
        dict(
            type="line",
            xref="x", yref="y",
            x0=S_pos[0] + S_radius[i]/20,
            y0=S_pos[1],
            x1=P_pos[0] - P_radius[i]/20,
            y1=P_pos[1],
            line=dict(color="green", width=4)
        ),
        # Arrow head
        dict(
            type="path",
            xref="x", yref="y",
```

```
            path=f"M {P_pos[0] - P_radius[i]/20 - 0.3} {P_pos[1] - 0.2} L␣
↪{P_pos[0] - P_radius[i]/20} {P_pos[1]} L {P_pos[0] - P_radius[i]/20 - 0.3}␣
↪{P_pos[1] + 0.2} Z",
            line_color="green",
            fillcolor="green"
        )
    ]

    # Add text annotations
    # text_annotations = [
    #     dict(
    #         x=S_pos[0],
    #         y=S_pos[1] - max(S_radius[i], P_radius[i])/15 - 0.8,
    #         text=f"S: {S_sol[i]:.2f}",
    #         showarrow=False,
    #         font=dict(color="blue", size=12),
    #         xref="x",
    #         yref="y"
    #     ),
    #     dict(
    #         x=P_pos[0],
    #         y=P_pos[1] - max(S_radius[i], P_radius[i])/15 - 0.8,
    #         text=f"P: {P_sol[i]:.2f}",
    #         showarrow=False,
    #         font=dict(color="red", size=12),
    #         xref="x",
    #         yref="y"
    #     )
    # ]

    frame = go.Frame(
        data=[],  # No scatter data
        layout=dict(
            shapes=frame_data,
            # annotations=text_annotations
        ),
        name=f'frame_{i}'
    )
    frames.append(frame)

# Add initial shapes
fig.add_shape(
    type="circle",
    xref="x", yref="y",
    x0=S_pos[0] - S_radius[0]/20,
    y0=S_pos[1] - S_radius[0]/20,
    x1=S_pos[0] + S_radius[0]/20,
```

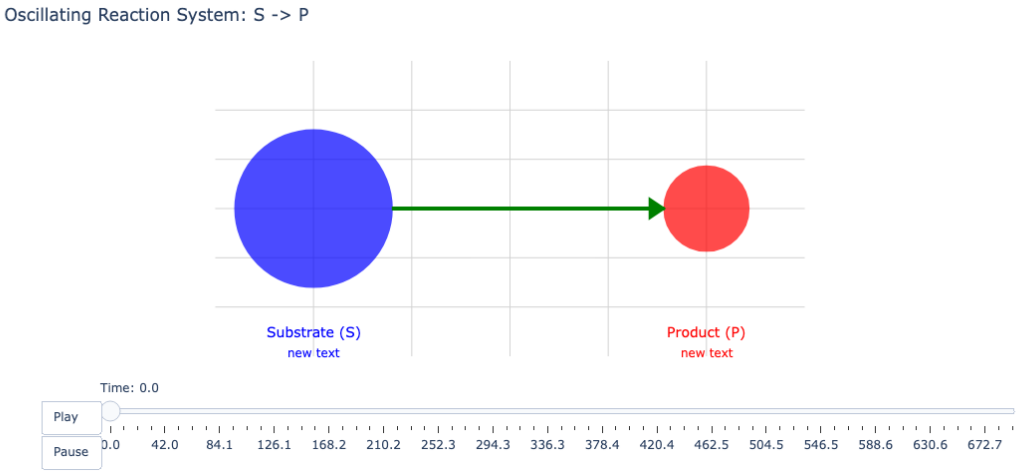
```
    y1=S_pos[1] + S_radius[0]/20,
    line_color="blue",
    fillcolor="blue",
    line_width=2,
    opacity=0.7
)

fig.add_shape(
    type="circle",
    xref="x", yref="y",
    x0=P_pos[0] - P_radius[0]/20,
    y0=P_pos[1] - P_radius[0]/20,
    x1=P_pos[0] + P_radius[0]/20,
    y1=P_pos[1] + P_radius[0]/20,
    line_color="red",
    fillcolor="red",
    line_width=2,
    opacity=0.7
)

fig.add_shape(
    type="line",
    xref="x", yref="y",
    x0=S_pos[0] + S_radius[0]/20,
    y0=S_pos[1],
    x1=P_pos[0] - P_radius[0]/20,
    y1=P_pos[1],
    line=dict(color="green", width=4)
)

fig.add_shape(
    type="path",
    xref="x", yref="y",
    path=f"M {P_pos[0] - P_radius[0]/20 - 0.3} {P_pos[1] - 0.2} L {P_pos[0] -␣
 ↪P_radius[0]/20} {P_pos[1]} L {P_pos[0] - P_radius[0]/20 - 0.3} {P_pos[1] + 0.
 ↪2} Z",
    line_color="green",
    fillcolor="green"
)

# Add initial annotations
fig.add_annotation(
    x=S_pos[0],
    y=S_pos[1] - max(S_radius[0], P_radius[0])/15 - 0.8,
    # text=f"S: {S_sol[0]:.2f}",
    showarrow=False,
    font=dict(color="blue", size=12),
```

```
    xref="x",
    yref="y"
)

fig.add_annotation(
    x=P_pos[0],
    y=P_pos[1] - max(S_radius[0], P_radius[0])/15 - 0.8,
    # text=f"P: {P_sol[0]:.2f}",
    showarrow=False,
    font=dict(color="red", size=12),
    xref="x",
    yref="y"
)

# Configure animation
fig.frames = frames

# Animation controls
sliders = [dict(
    steps=[dict(
        method="animate",
        args=[[f'frame_{k}'],
              dict(mode="immediate", frame=dict(duration=50, redraw=True),
                   transition=dict(duration=0))],
        label=f'{t_sol[k]:.1f}'
    ) for k in range(0, len(t_sol), 15)],
    active=0,
    currentvalue=dict(prefix="Time: ", visible=True),
    len=0.9,
    x=0.1,
    y=0
)]

updatemenus = [dict(
    type="buttons",
    showactive=False,
    buttons=[
        dict(label="Play", method="animate",
             args=[None, dict(frame=dict(duration=50, redraw=True),
                              fromcurrent=True, mode="immediate")]),
        dict(label="Pause", method="animate",
             args=[[None], dict(frame=dict(duration=0, redraw=False),
                                mode="immediate")])
    ],
    x=0.1, y=-0.15,
    xanchor="right",
    yanchor="top"
```

```
)]

# Update layout with FIXED ranges and no auto-scaling
fig.update_layout(
    title="Oscillating Reaction System: S -> P",
    xaxis=dict(
        range=[-6, 6],  # Fixed range
        showticklabels=False,
        showgrid=True,
        gridcolor="lightgray",
        zeroline=False,
        fixedrange=True,  # Prevent any scaling
        constrain="domain"
    ),
    yaxis=dict(
        range=[-3, 3],  # Fixed range
        showticklabels=False,
        showgrid=True,
        gridcolor="lightgray",
        zeroline=False,
        fixedrange=True,  # Prevent any scaling
        scaleanchor="x",
        scaleratio=1
    ),
    plot_bgcolor="white",
    width=800,
    height=500,
    showlegend=False,
    sliders=sliders,
    updatemenus=updatemenus,
    # Critical: disable all auto-scaling and layout recalculations
    autosize=False,
    margin=dict(l=50, r=50, t=80, b=80)
)

# Add static labels
fig.add_annotation(
    x=S_pos[0], y=-2.5,
    text="Substrate (S)",
    showarrow=False,
    font=dict(color="blue", size=14),
    xref="x",
    yref="y"
)

fig.add_annotation(
    x=P_pos[0], y=-2.5,
```

```
    text="Product (P)",
    showarrow=False,
    font=dict(color="red", size=14),
    xref="x",
    yref="y"
)

fig.show()
```

```
/var/folders/cs/lkcj7j890kv6kfxbkl56w9h80000gn/T/ipykernel_15293/409425694.py:10
: RuntimeWarning:

invalid value encountered in scalar power
```



[ ]: