

# Introducción EDO con Python

Luis E. Ascencio G.  
CIMAT  
luis.ascencio@cimat.mx

**Abstract** En este NoteBook encontraras código para manejar EDO con el lenguaje de programación Python.

quarto preview /home/leag555/Escritorio/py-jl-Differential-Equations/Introduction-py.ipynb –  
thtml –no-browser –no-watch-inputs –host 0.0.0.0

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

# =====
# 1. User Definitions & Configuration
# =====

# Configuration Parameters
CONFIG = {
    'x_range': (-3, 3),      # Domain for x
    'y_range': (-3, 3),      # Domain for y
    'grid_density': 25,       # Density of arrows in vector field
    'num_integral_curves': 25, # Number of curves for Figure 2 pip install myst-
                                # parser sphinx

    'specific_ics': [-3,-2,-1,-0.5,0,0.5,1,2,3], # Specific initial conditions
    y(x_start) for Figure 3
    'resolution': 500          # Resolution for numerical integration steps
}

# =====
# 2. Visualization Functions
# =====

def plot_vector_field(f, x_lim, y_lim, density=20):
    """
```

Figure 1: Generates the Vector Field (Slope Field).

```

Mathematical Context:
At every point  $(x, y)$ , the ODE defines a slope  $m = f(x, y)$ .
The vector at that point is  $\langle 1, f(x, y) \rangle$ .
We normalize these vectors to show direction without magnitude distortion.
"""

fig = plt.figure(figsize=(10, 8))

# Create a grid of points
x = np.linspace(x_lim[0], x_lim[1], density)
y = np.linspace(y_lim[0], y_lim[1], density)
X, Y = np.meshgrid(x, y)

# Calculate vector components
# dx is constant (1), dy is the function output
U = np.ones_like(X)
V = f(X, Y)

# Normalize the arrows (make them all unit length for clarity)
# Magnitude N = sqrt(U^2 + V^2)
N = np.sqrt(U**2 + V**2)
U = U / N
V = V / N

# Plot Quiver
plt.quiver(X, Y, U, V, angles='xy', scale_units='xy', scale=3,
color='#555555', width=0.003)

# Styling
plt.title("Figure 1: Vector Field (Direction Field)", fontsize=16)
plt.xlabel("x", fontsize=14)
plt.ylabel("y", fontsize=14)
plt.xlim(x_lim)
plt.ylim(y_lim)
plt.grid(True, linestyle='--', alpha=0.6)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)

# Add equation text
plt.text(x_lim[0] + 0.5, y_lim[1] - 0.5, r"\frac{dy}{dx} = f(x, y)",
         fontsize=14, bbox=dict(facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()

def plot_integral_curves(f, x_lim, y_lim, num_curves=20):
"""
"""

Figure 2: Plots a dense set of integral curves to show flow.

```

```

Method:
We define a range of initial conditions along the left boundary ( $x_{\min}$ )
and integrate forward to  $x_{\max}$  using Runge-Kutta 4(5).
"""

fig = plt.figure(figsize=(10, 8))

# Generate initial conditions along the left edge
y0_values = np.linspace(y_lim[0], y_lim[1], num_curves)
x_span = x_lim

# Evaluation points for smooth curves
t_eval = np.linspace(x_lim[0], x_lim[1], CONFIG['resolution'])

# Plot background vector field faintly for context
x_grid = np.linspace(x_lim[0], x_lim[1], 20)
y_grid = np.linspace(y_lim[0], y_lim[1], 20)
X, Y = np.meshgrid(x_grid, y_grid)
U = np.ones_like(X)
V = f(X, Y)
N = np.sqrt(U**2 + V**2)
plt.quiver(X, Y, U/N, V/N, alpha=0.2, color='gray')

# Solve and plot each curve
print(f"Generating {num_curves} integral curves...")
for y0 in y0_values:
    # solve_ivp requires function signature fun(t, y)
    sol = solve_ivp(f, x_span, [y0], t_eval=t_eval, method='RK45')

    if sol.success:
        plt.plot(sol.t, sol.y[0], '--', color='teal', alpha=0.6, linewidth=1.5)

# Styling
plt.title("Figure 2: Integral Curves (General Flow)", fontsize=16)
plt.xlabel("x", fontsize=14)
plt.ylabel("y", fontsize=14)
plt.xlim(x_lim)
plt.ylim(y_lim)
plt.grid(True, linestyle='--', alpha=0.6)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)

plt.tight_layout()
plt.show()

def plot_specific_solutions(f, x_lim, y_lim, initial_conditions):
"""
Figure 3: Plots specific, labeled numerical solutions.

```

```

Use Case:
Highlighting specific behaviors based on exact starting points.
"""

fig = plt.figure(figsize=(10, 8))

x_span = x_lim
t_eval = np.linspace(x_lim[0], x_lim[1], CONFIG['resolution'])

colors = plt.cm.viridis(np.linspace(0, 0.9, len(initial_conditions)))

print("Generating specific solutions...")
for i, y0 in enumerate(initial_conditions):
    sol = solve_ivp(f, x_span, [y0], t_eval=t_eval, method='RK45')

    if sol.success:
        label_text = f"$y({x_lim[0]}) = {y0}$"
        plt.plot(sol.t, sol.y[0], linewidth=2.5, color=colors[i],
label=label_text)

        # Mark the initial condition point
        plt.scatter([x_lim[0]], [y0], color=colors[i], s=50, zorder=5)

# Styling
plt.title("Figure 3: Specific Numerical Solutions", fontsize=16)
plt.xlabel("x", fontsize=14)
plt.ylabel("y", fontsize=14)
plt.xlim(x_lim)
plt.ylim(y_lim)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(title="Initial Conditions", loc='best', frameon=True, shadow=True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)

plt.tight_layout()
plt.show()

# -----
# 3. Main Execution Block
# -----


def plots_ODE(ode_f):
    print("--- Differential Equation Visualizer ---")
    print(f"ODE: dy/dx = x - y")
    print(f"Domain: x in {CONFIG['x_range']}, y in {CONFIG['y_range']}")

    # 1. Plot Vector Field
    plot_vector_field(
        ode_f,

```

```

        CONFIG['x_range'],
        CONFIG['y_range'],
        density=CONFIG['grid_density']
    )

# 2. Plot Integral Curves (Flow)
plot_integral_curves(
    ode_f,
    CONFIG['x_range'],
    CONFIG['y_range'],
    num_curves=CONFIG['num_integral_curves']
)

# 3. Plot Specific Solutions
plot_specific_solutions(
    ode_f,
    CONFIG['x_range'],
    CONFIG['y_range'],
    initial_conditions=CONFIG['specific_ics']
)

```

```

def ode_f1(x, y):
    """
    Defines the First-Order ODE: dy/dx = f(x, y).

    Example: dy/dx = x - y

    Parameters:
        x (float): Independent variable (often time).
        y (float): Dependent variable.

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return -0.5*y

```

```

def ode_f2(x, y):
    """
    Defines the First-Order ODE: dy/dx = f(x, y).

    Example: dy/dx = y^2

    Parameters:
        x (float): Independent variable (often time).
        y (float): Dependent variable.
    """

```

```

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return y**2

```

```

def ode_f3(x, y):
    """
    Defines the First-Order ODE: dy/dx = f(x, y).

    Example: dy/dx = y - sin(x)

    Parameters:
        x (float): Independent variable (often time).
        y (float): Dependent variable.

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return y-np.sin(x)

```

```

def ode_f4(x, y):
    """
    Defines the First-Order ODE: dy/dx = f(x, y).

    Example: dy/dx = -x**2+np.sin(y)

    Parameters:
        x (float): Independent variable (often time).
        y (float): Dependent variable.

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return -x**2+np.sin(y)

```

```

def ode_f5(x, y):
    """
    Defines the First-Order ODE: dy/dx = f(x, y).

    Example: dy/dx = x**2-y

    Parameters:
        x (float): Independent variable (often time).
        y (float): Dependent variable.

```

```

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return x**2-y

```

```

def ode_f6(x, y):
"""
Defines the First-Order ODE: dy/dx = f(x, y).

Example: dy/dx = np.sin(x-y)

Parameters:
    x (float): Independent variable (often time).
    y (float): Dependent variable.

Returns:
    float: The derivative dy/dx at (x, y).
"""
return np.sin(x-y)

```

```

def ode_f7(x, y):
"""
Defines the First-Order ODE: dy/dx = f(x, y).

Example: dy/dx = -6*x*y

Parameters:
    x (float): Independent variable (often time).
    y (float): Dependent variable.

Returns:
    float: The derivative dy/dx at (x, y).
"""
return -6*x*y

```

```

def ode_f8(x, y):
"""
Defines the First-Order ODE: dy/dx = f(x, y).

Example: dy/dx = (4-2*x)/(3*y**2-5)

Parameters:
    x (float): Independent variable (often time).
    y (float): Dependent variable.

```

```

    Returns:
        float: The derivative dy/dx at (x, y).
    """
    return (4-2*x)/(3*y**2-5)

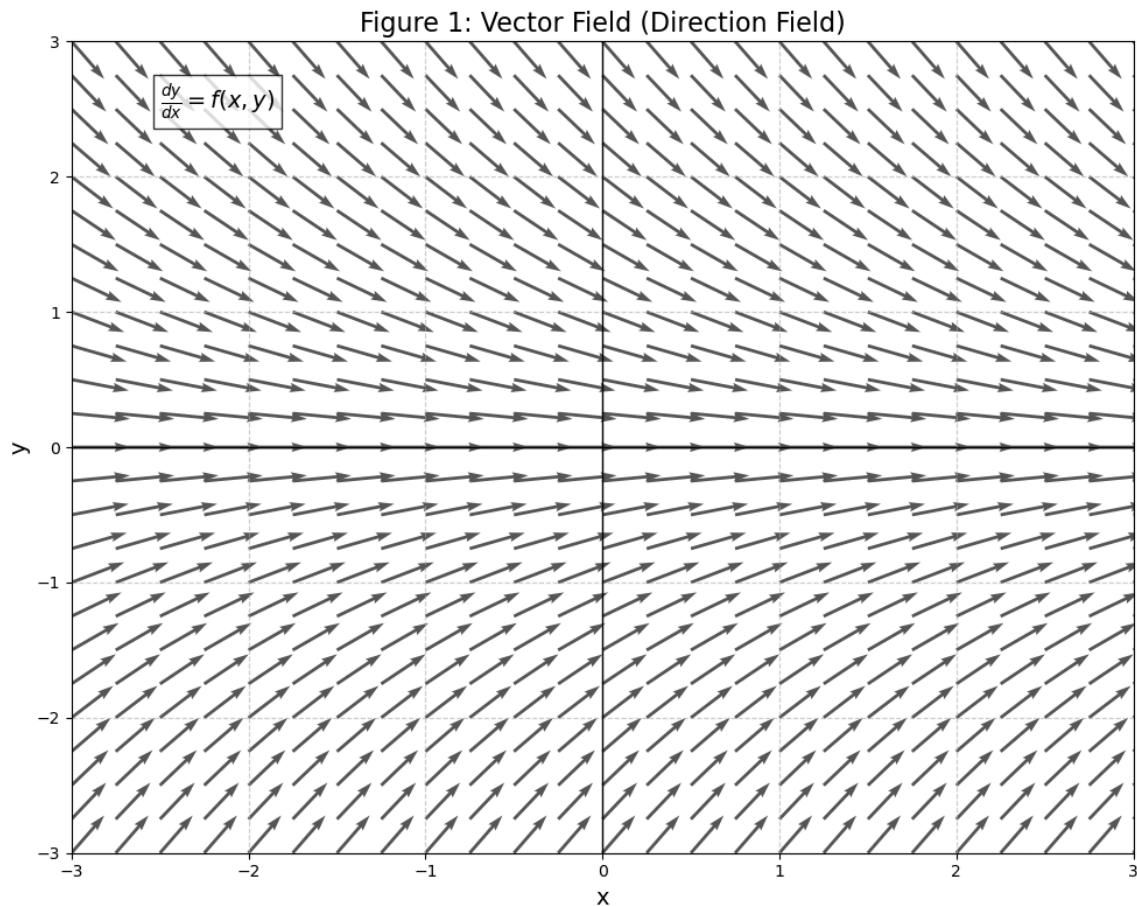
```

```
plots_ODE(ode_f1)
```

```

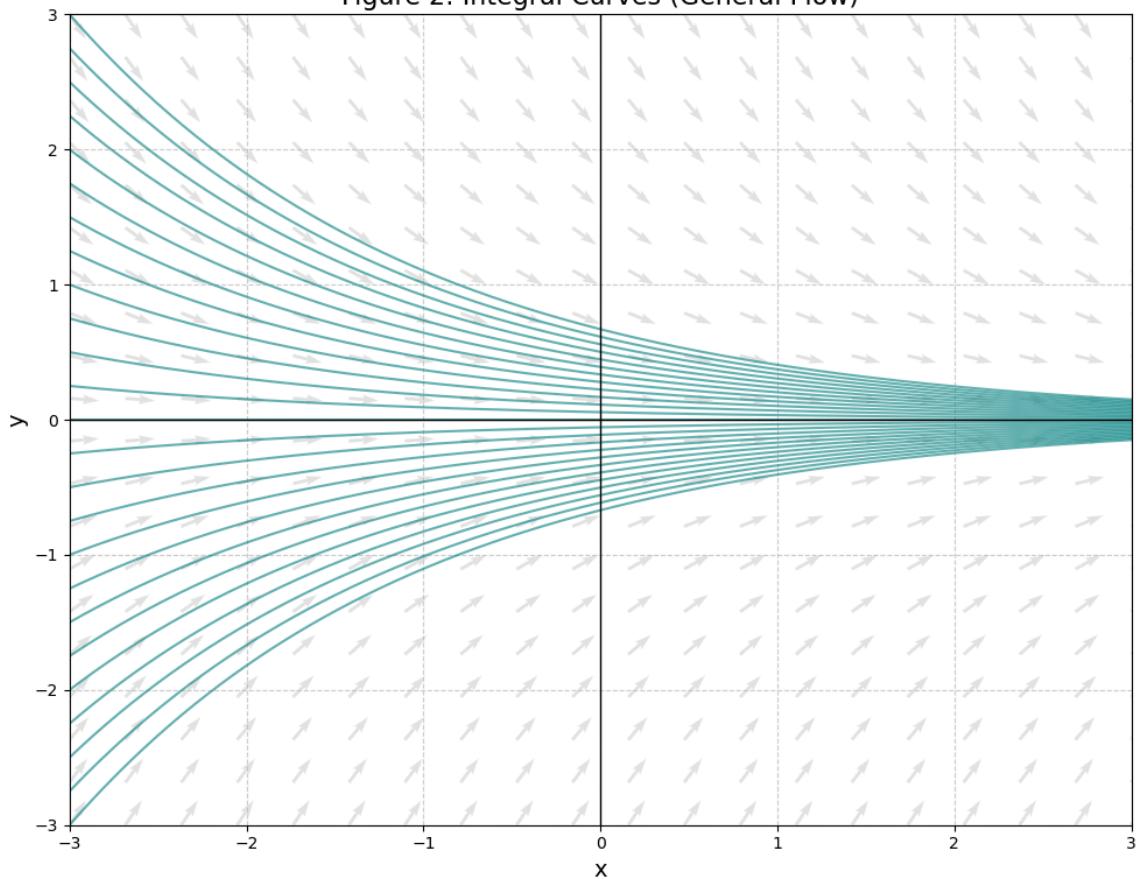
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)

```



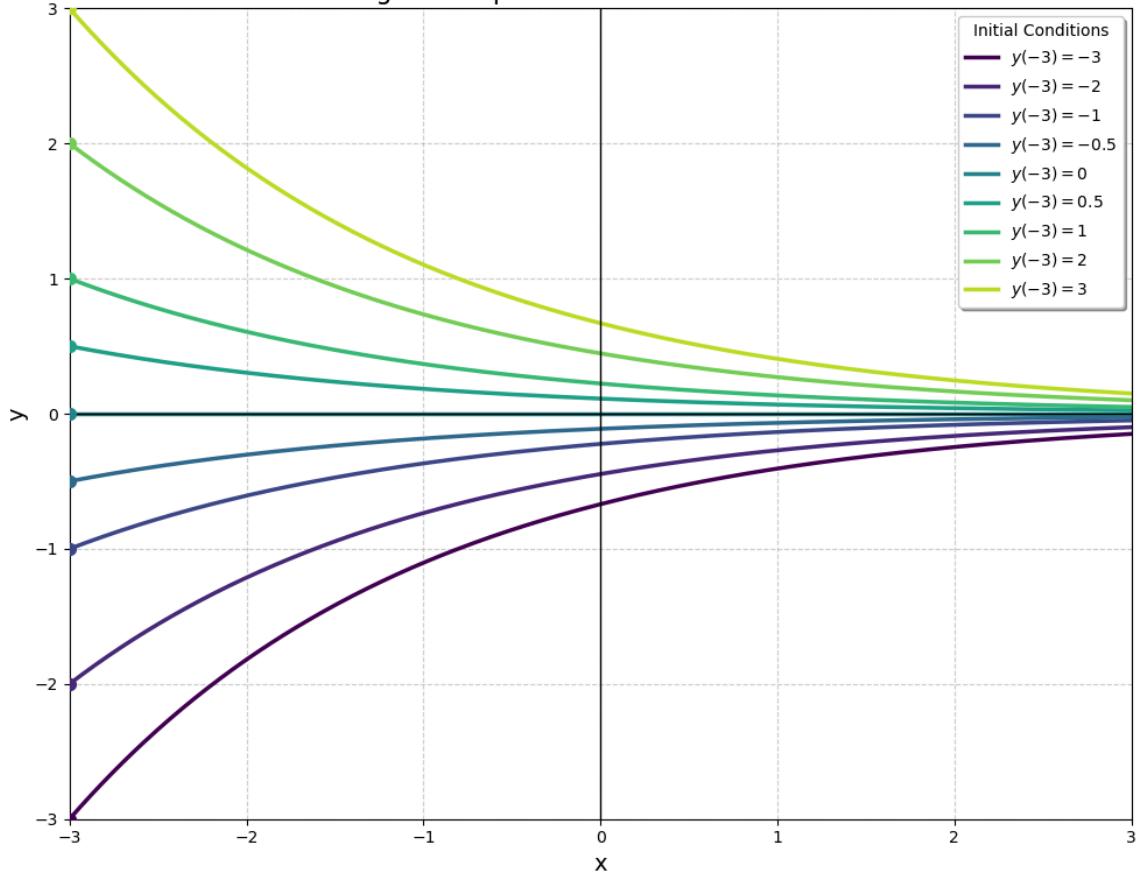
```
Generating 25 integral curves...
```

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

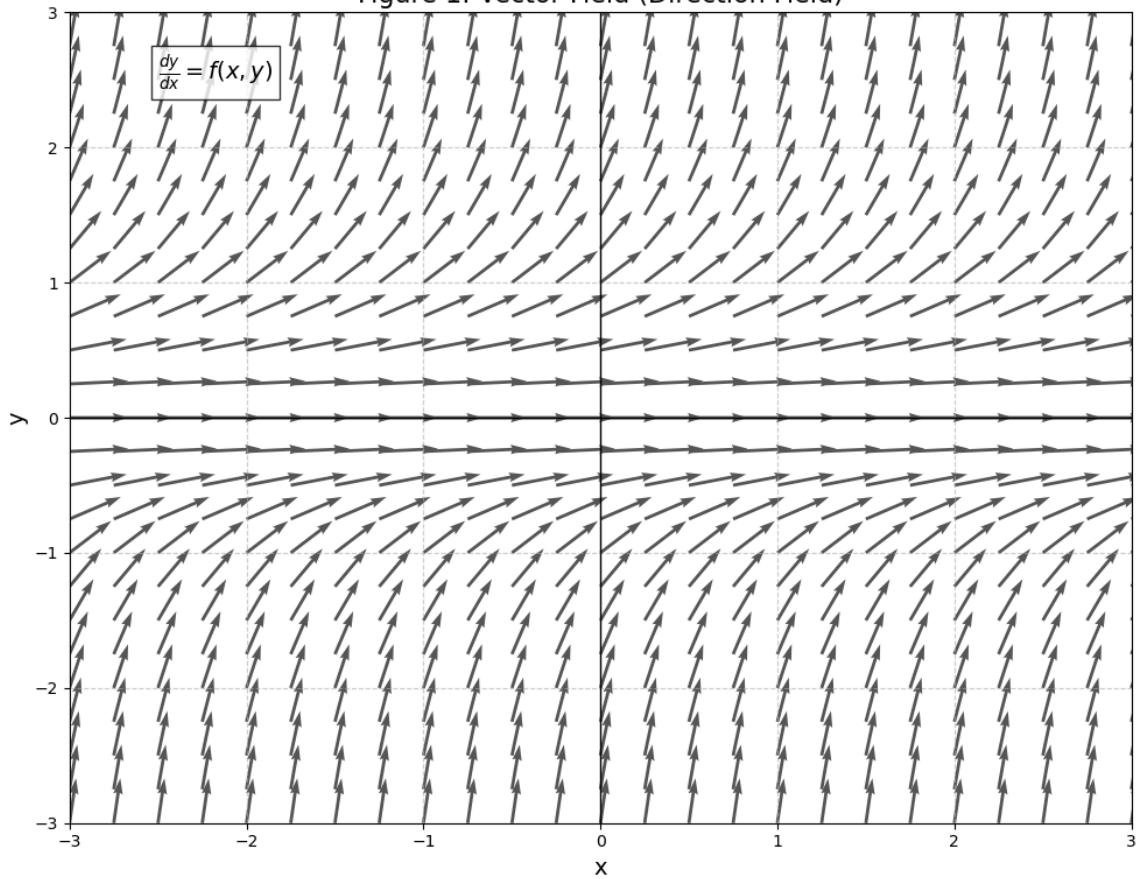
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f2)
```

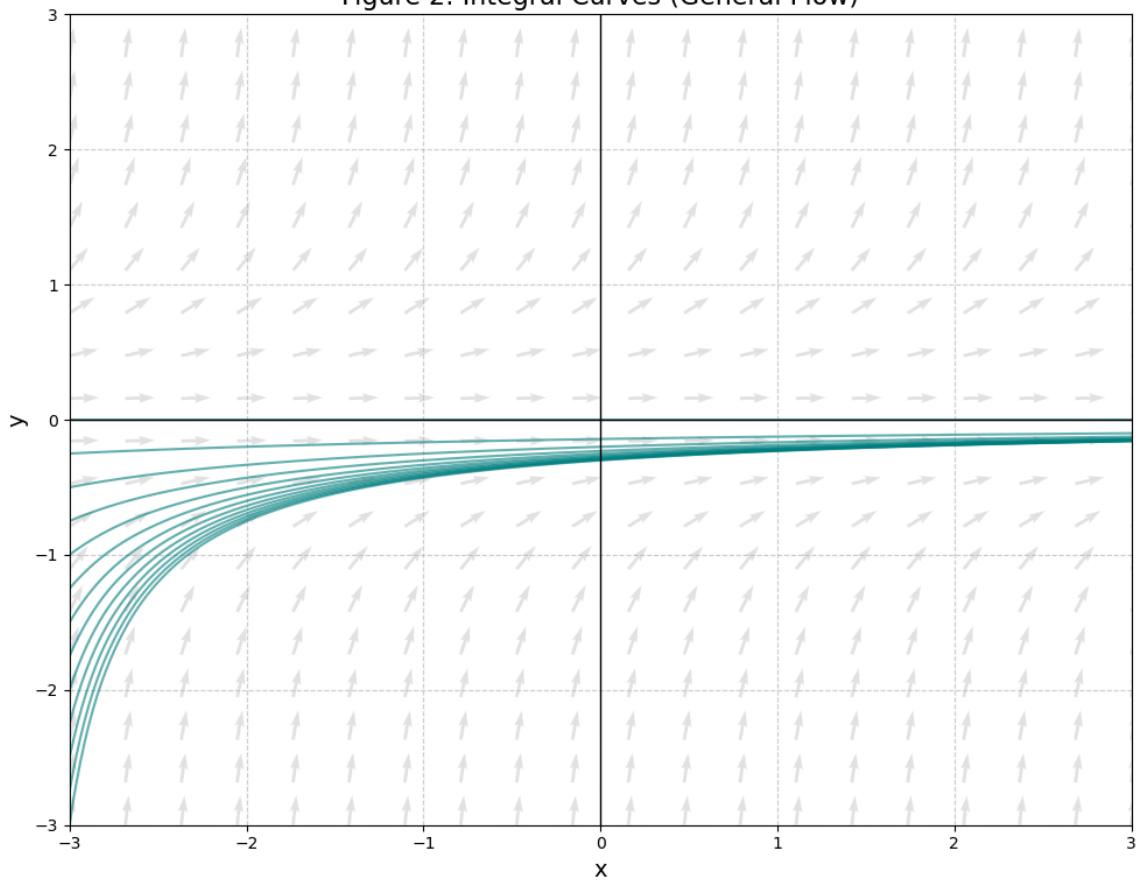
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



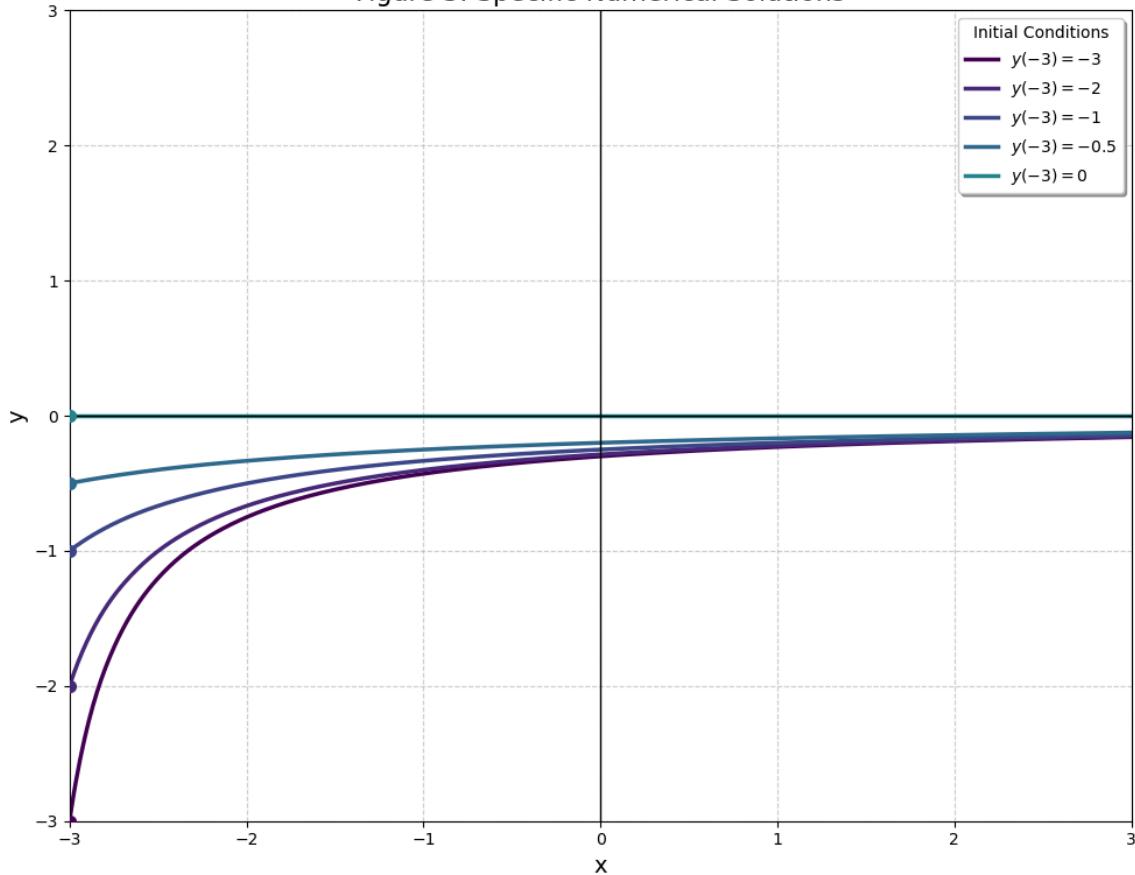
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

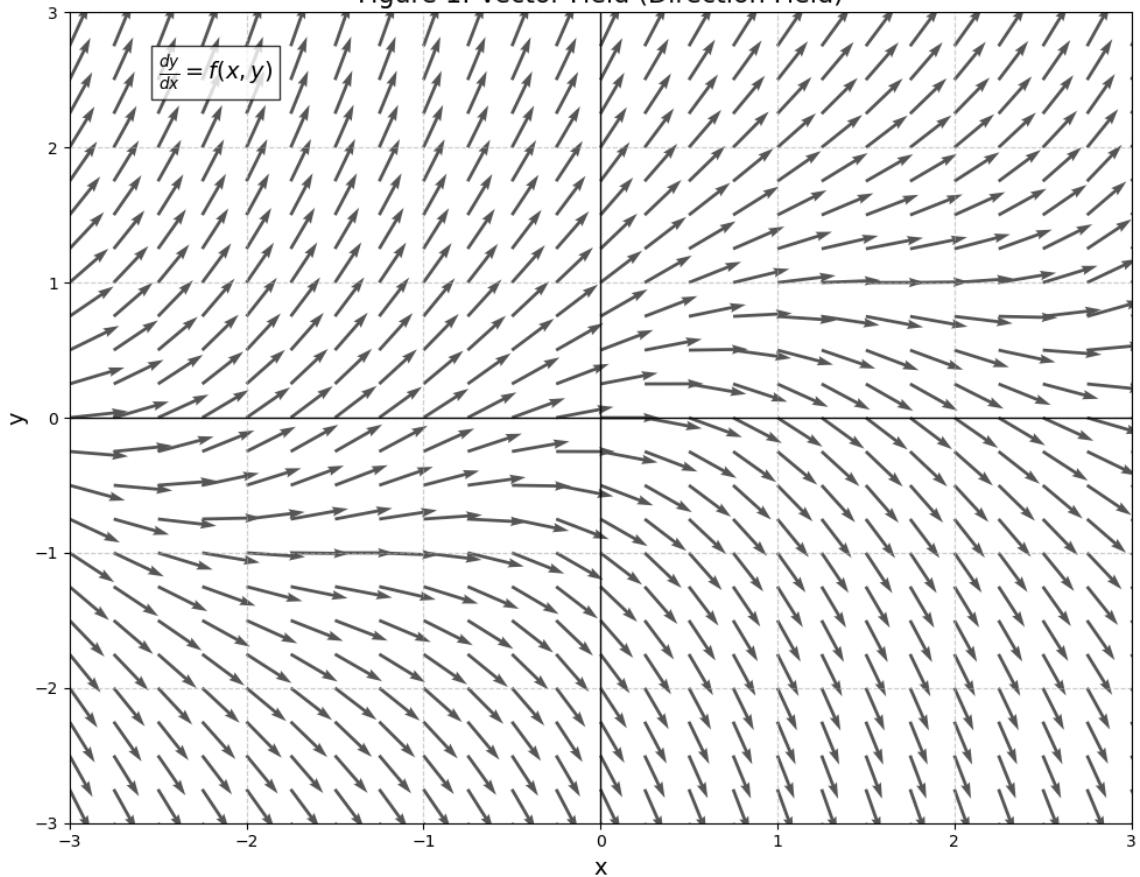
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f3)
```

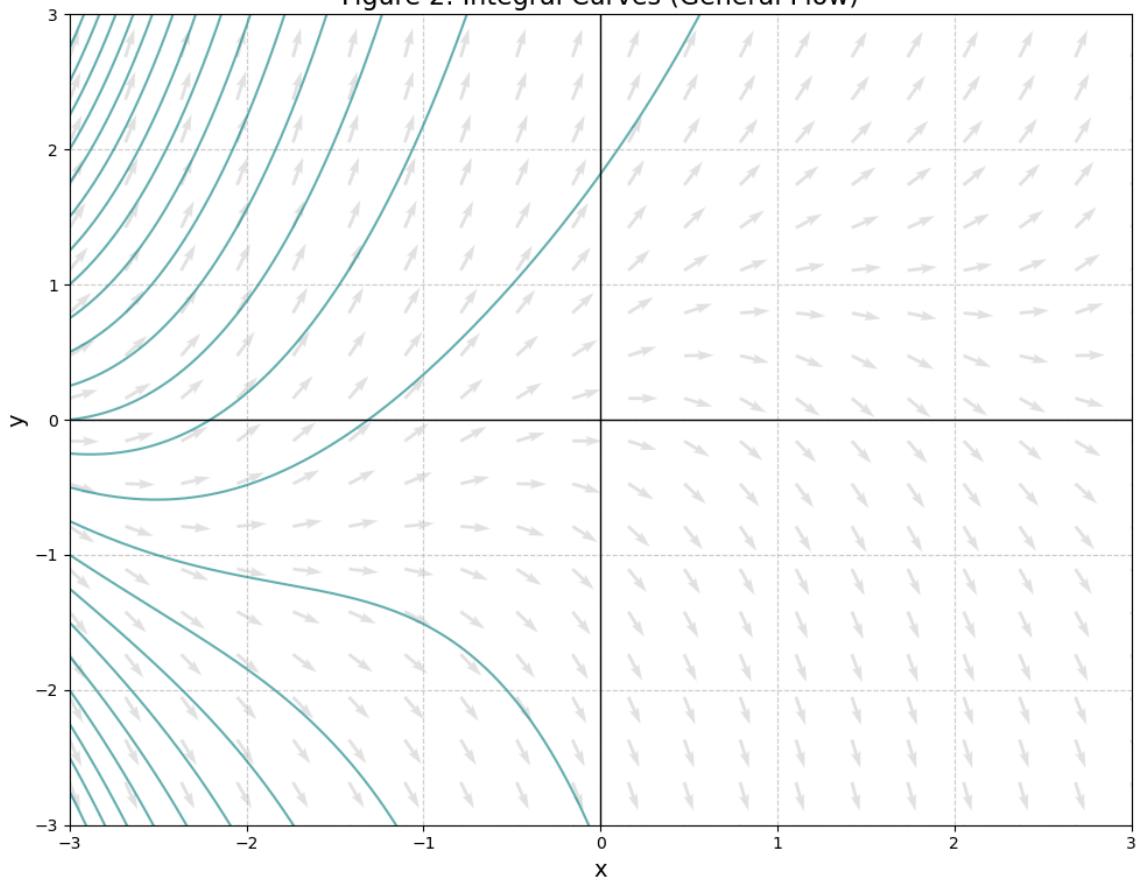
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



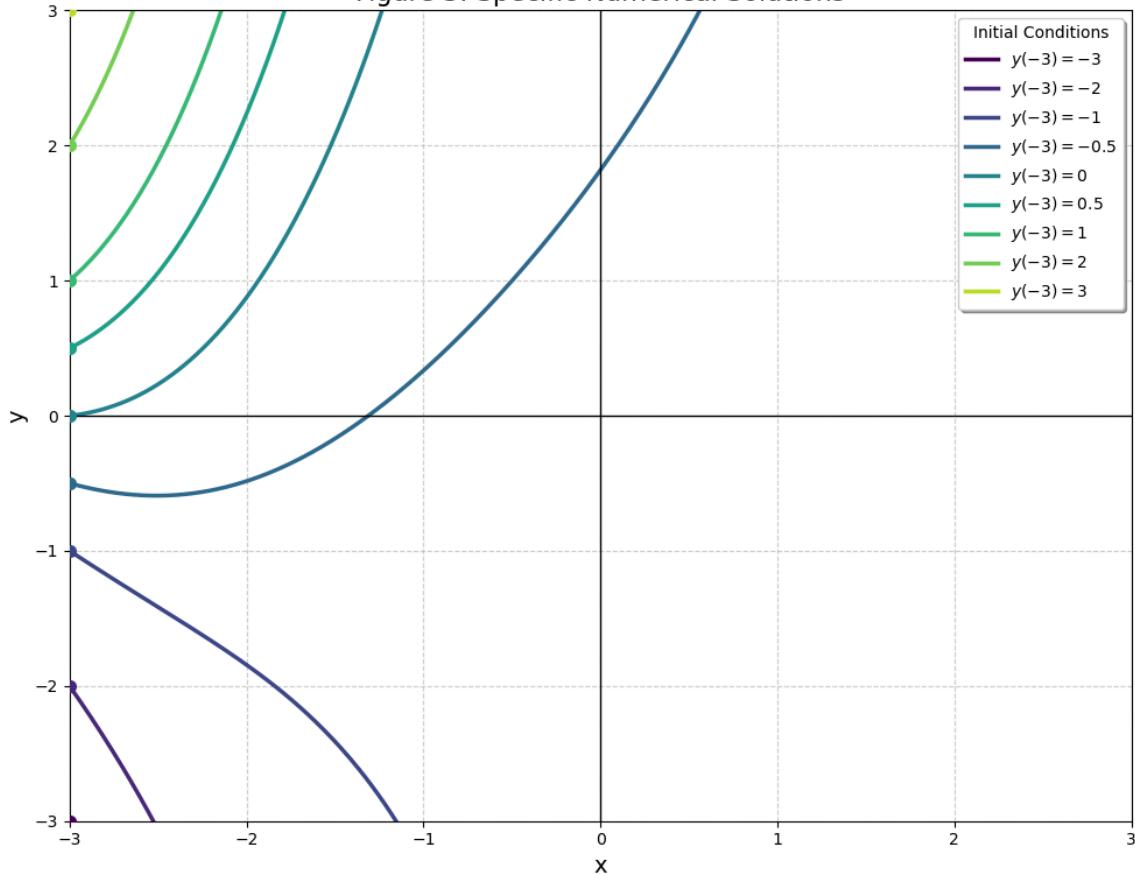
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

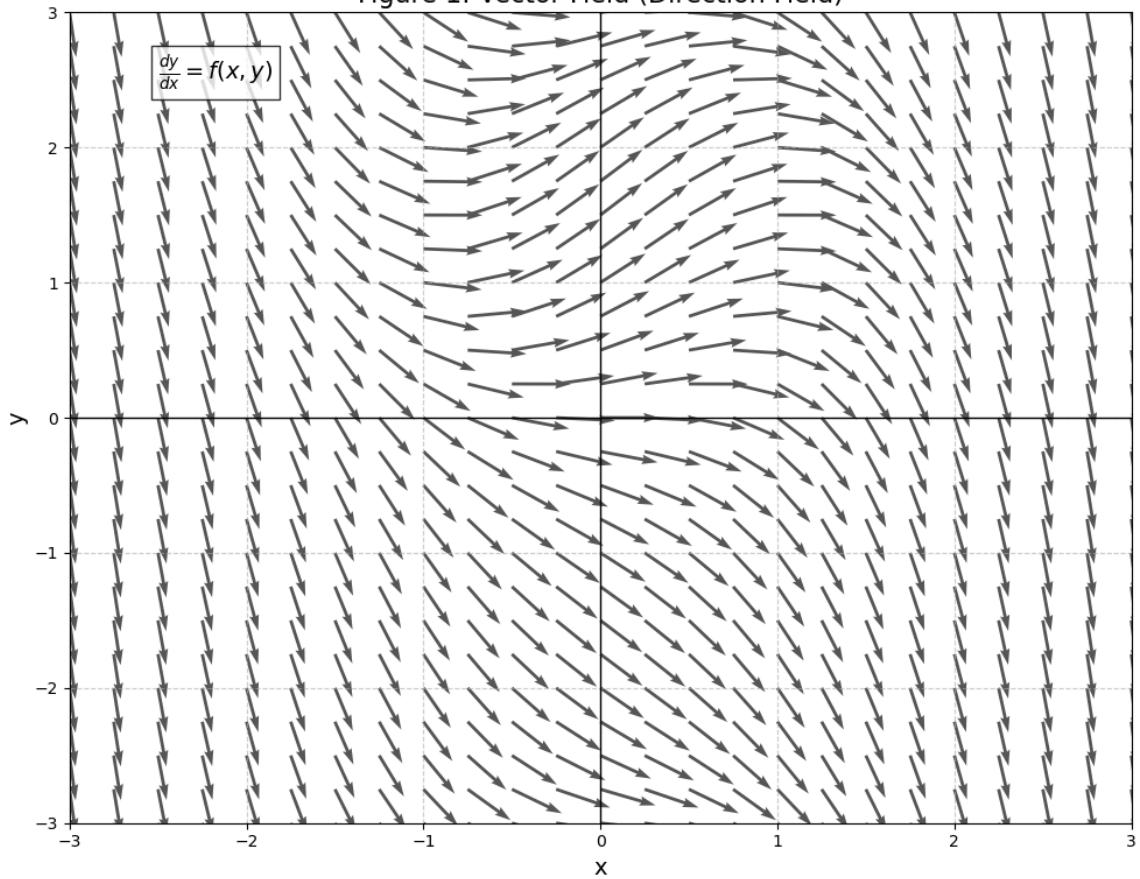
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f4)
```

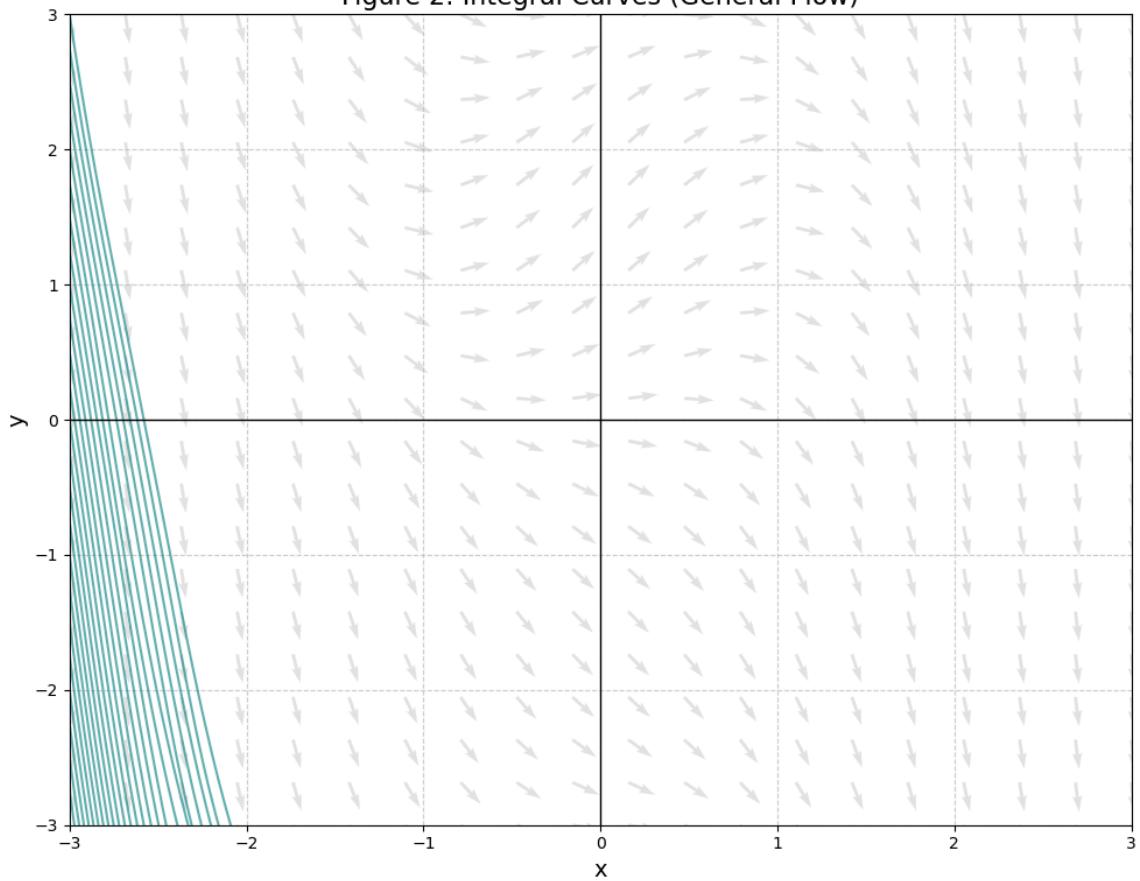
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



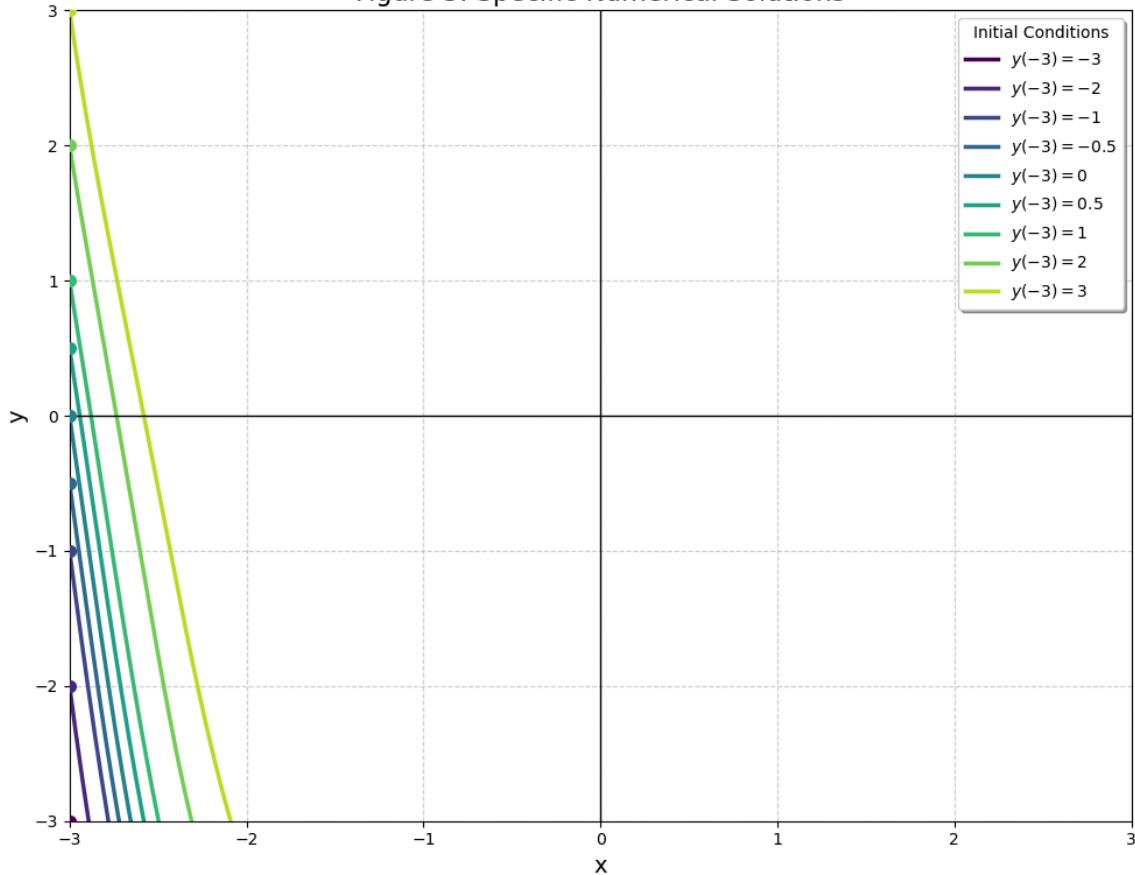
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

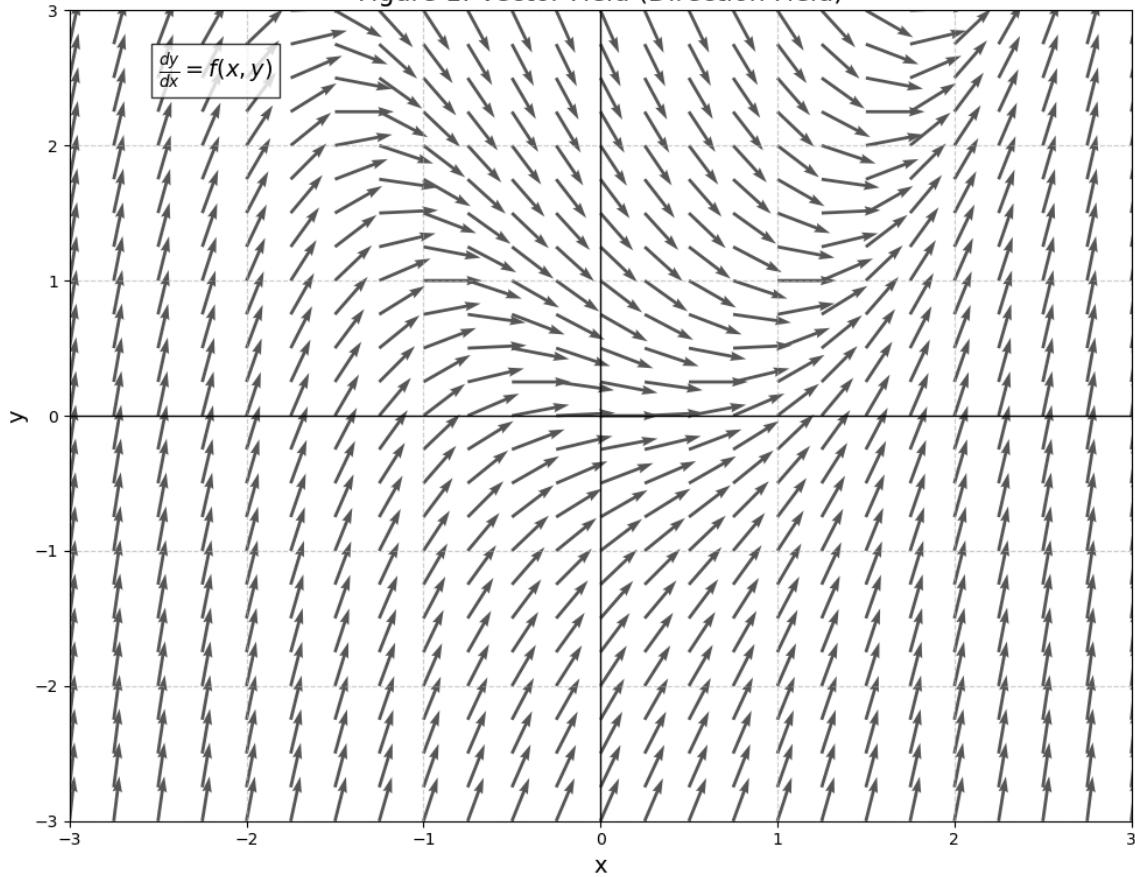
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f5)
```

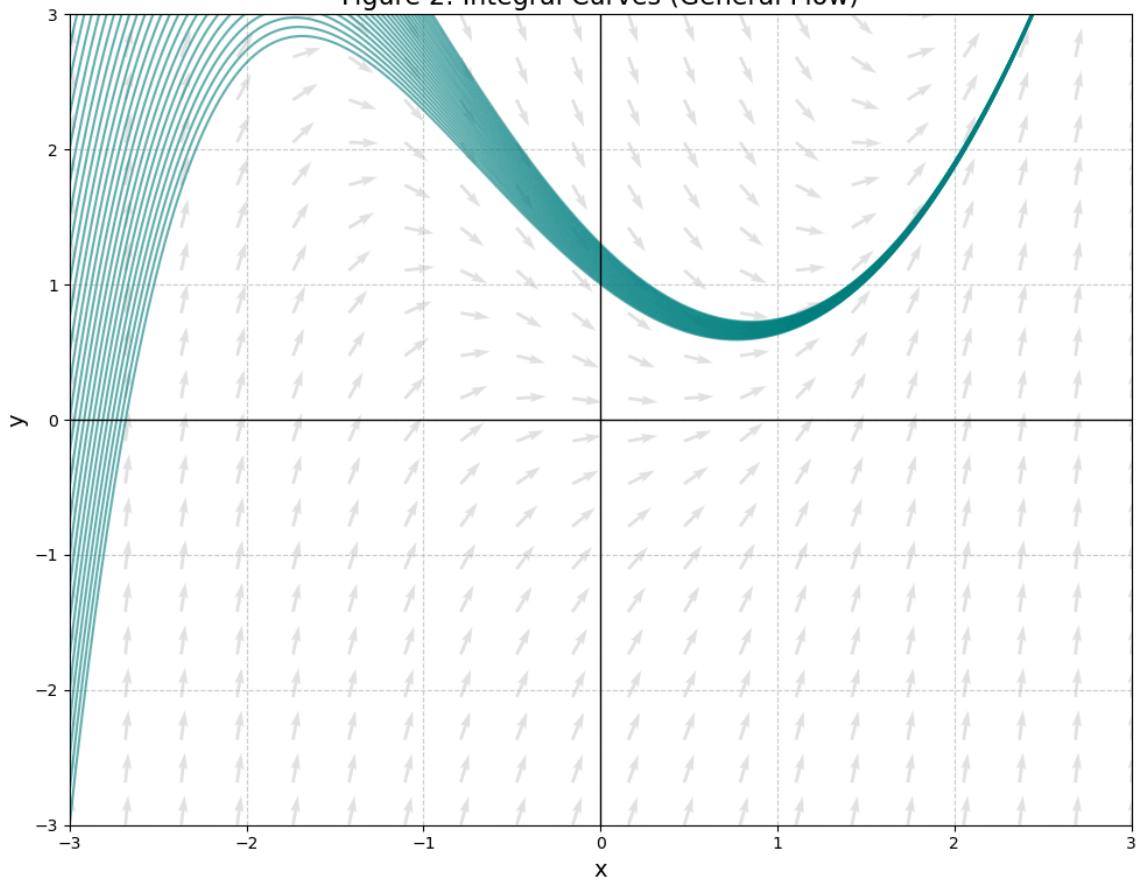
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



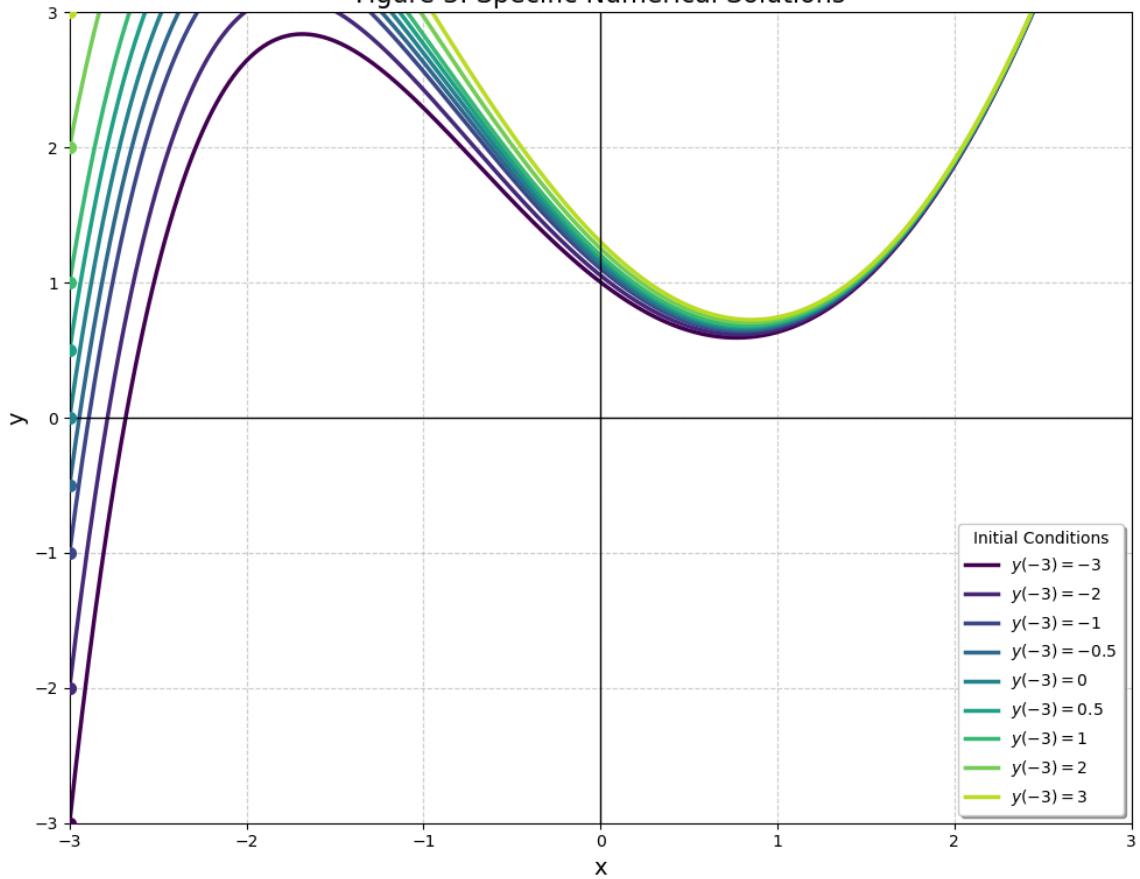
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

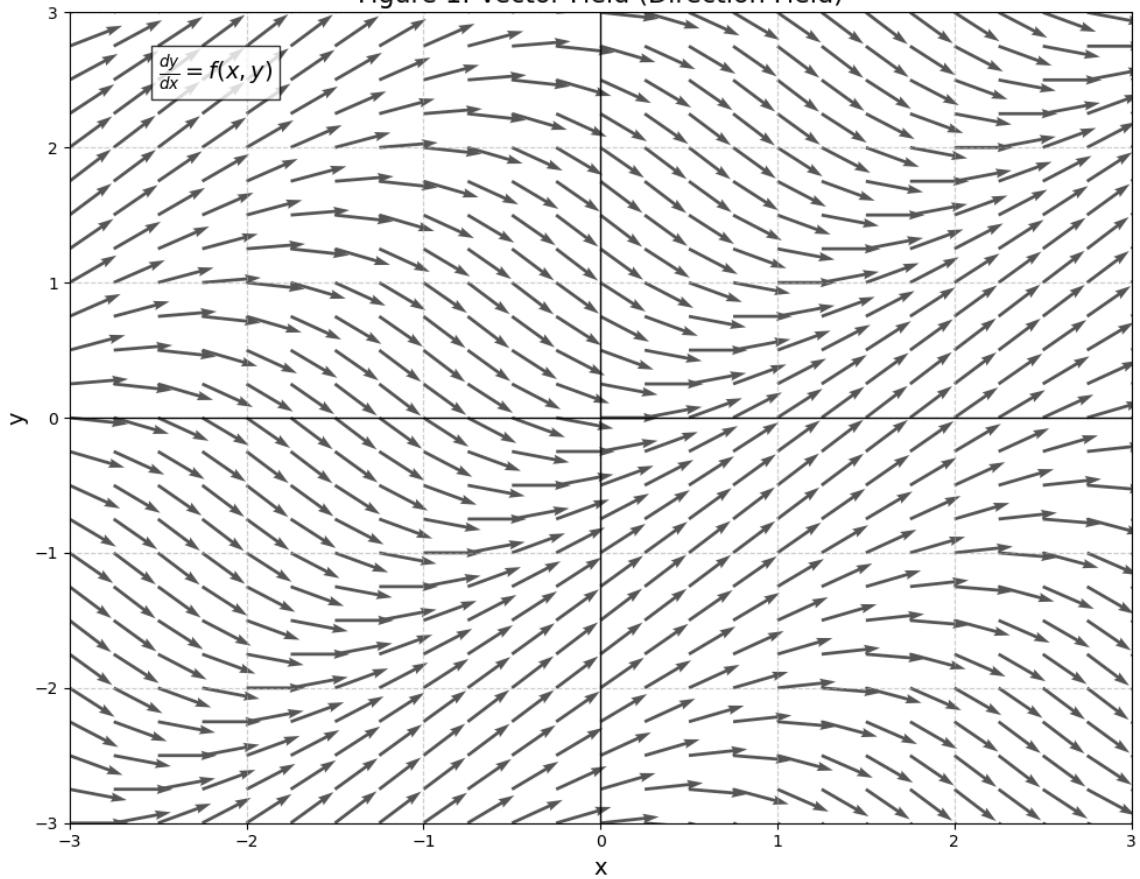
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f6)
```

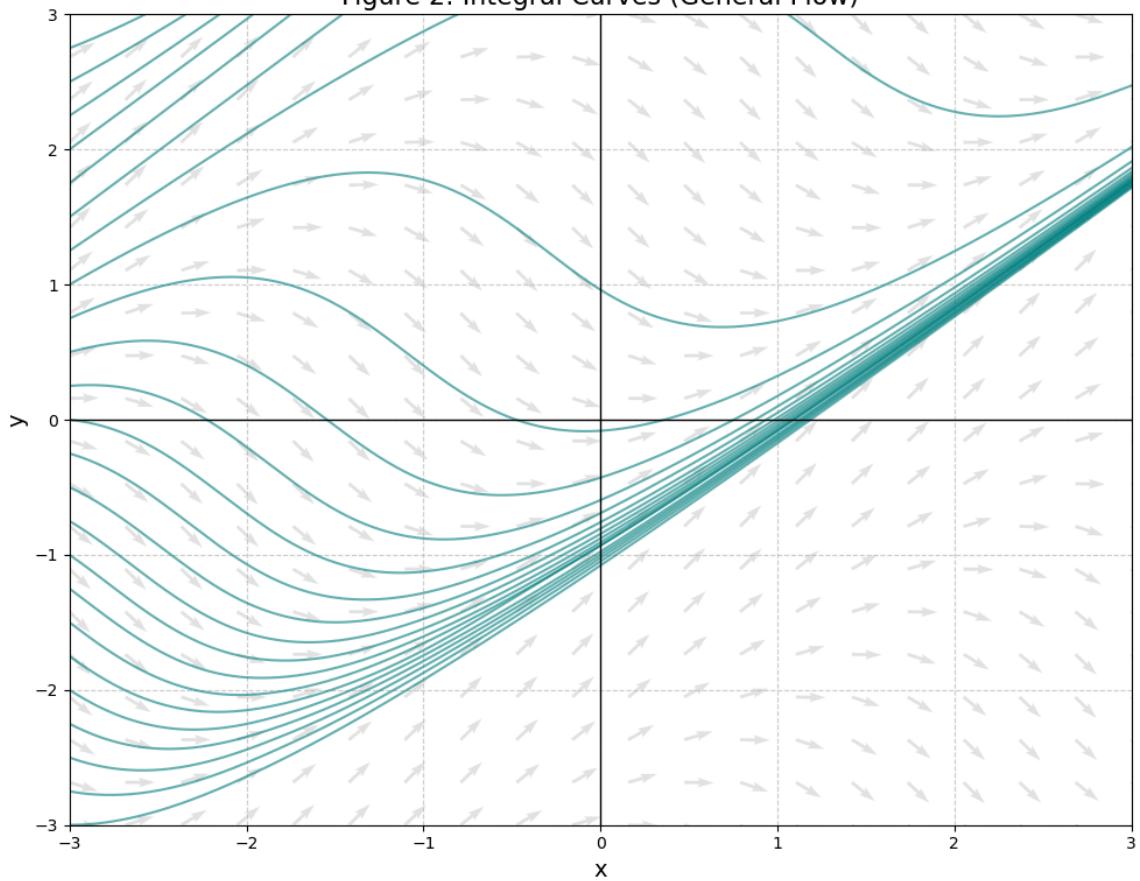
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



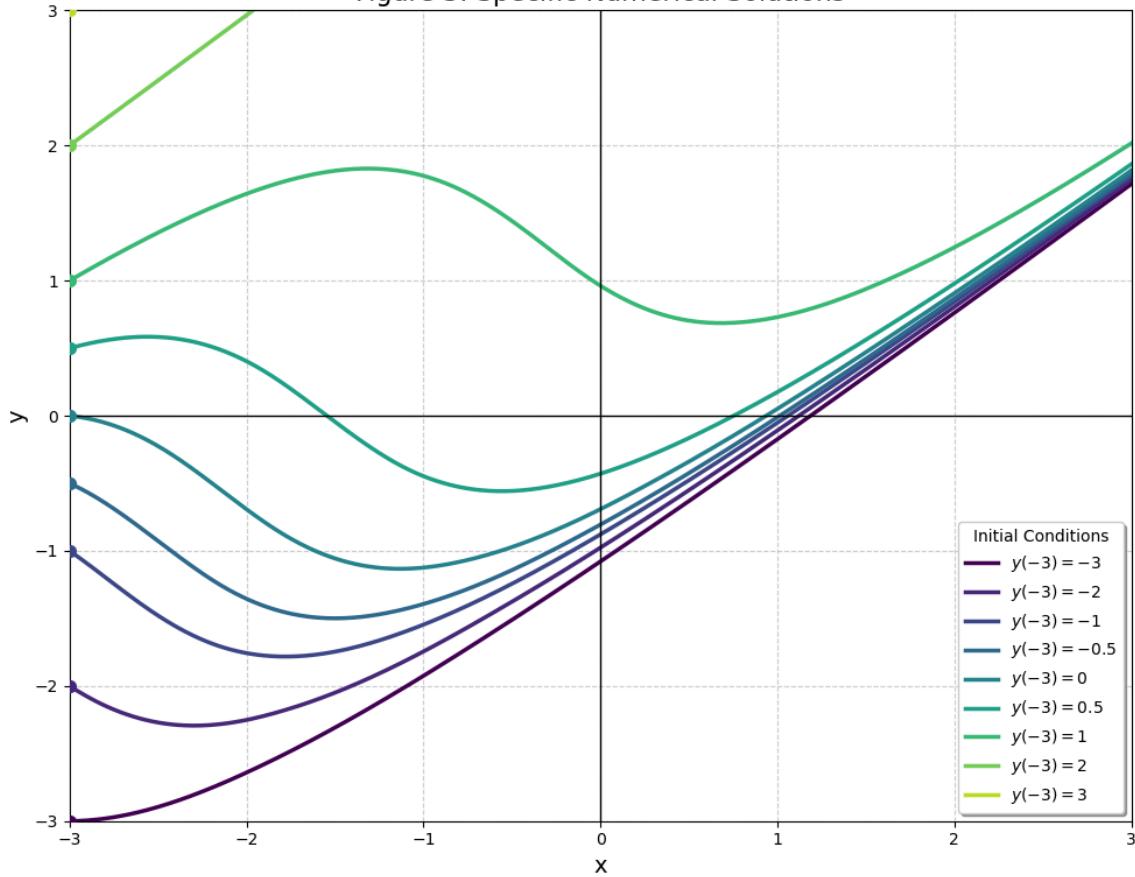
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

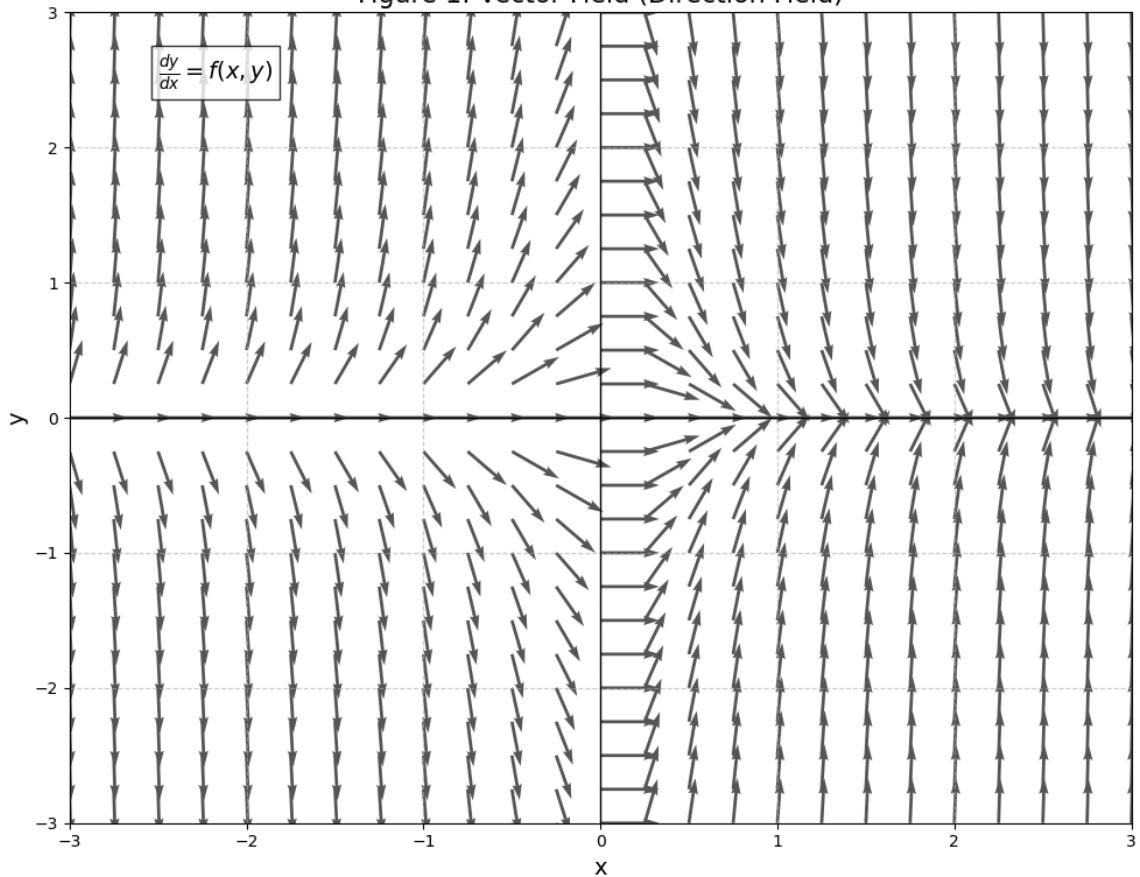
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f7)
```

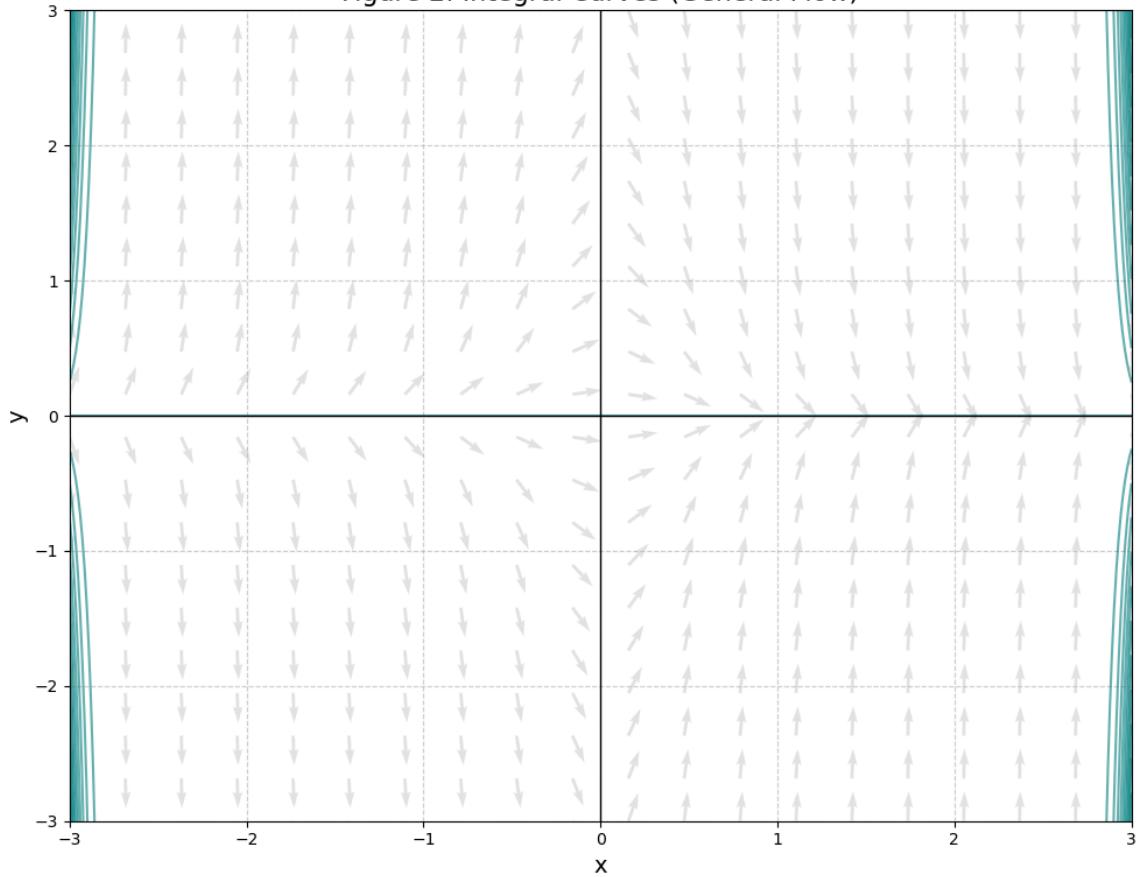
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



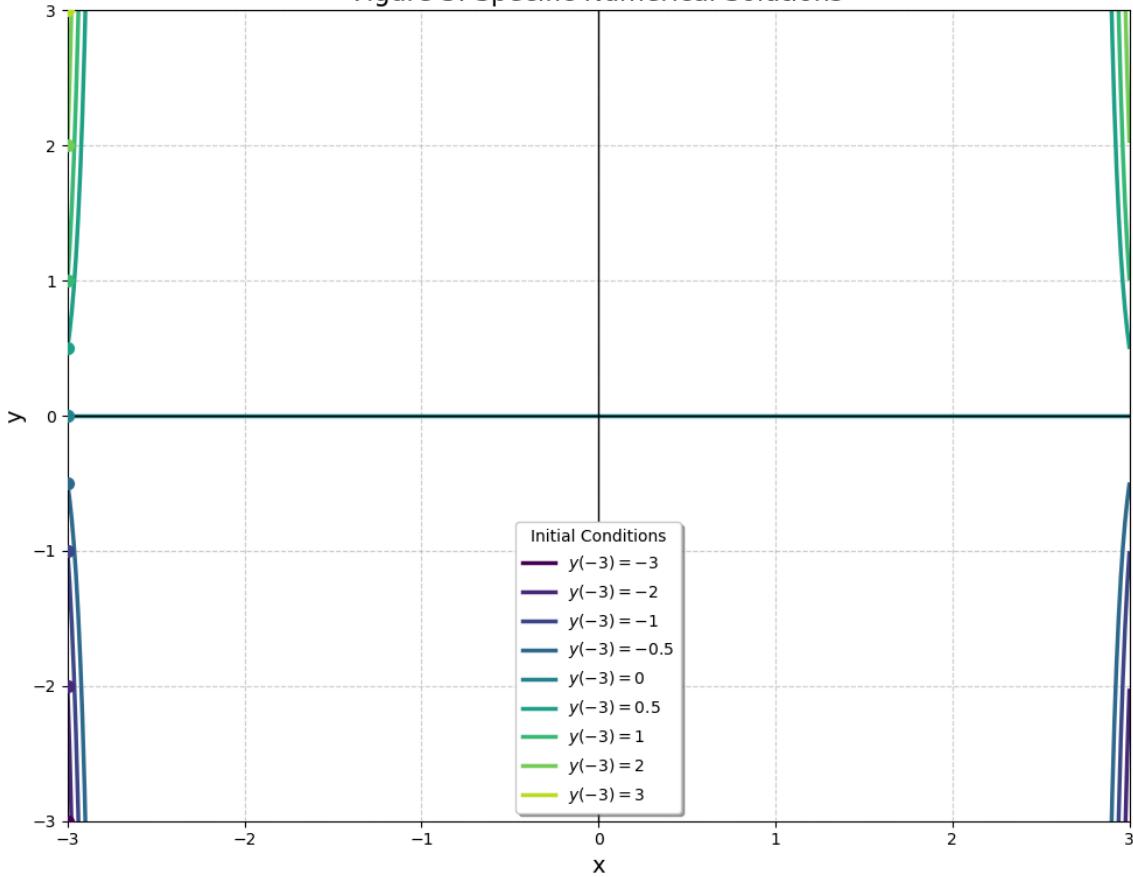
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

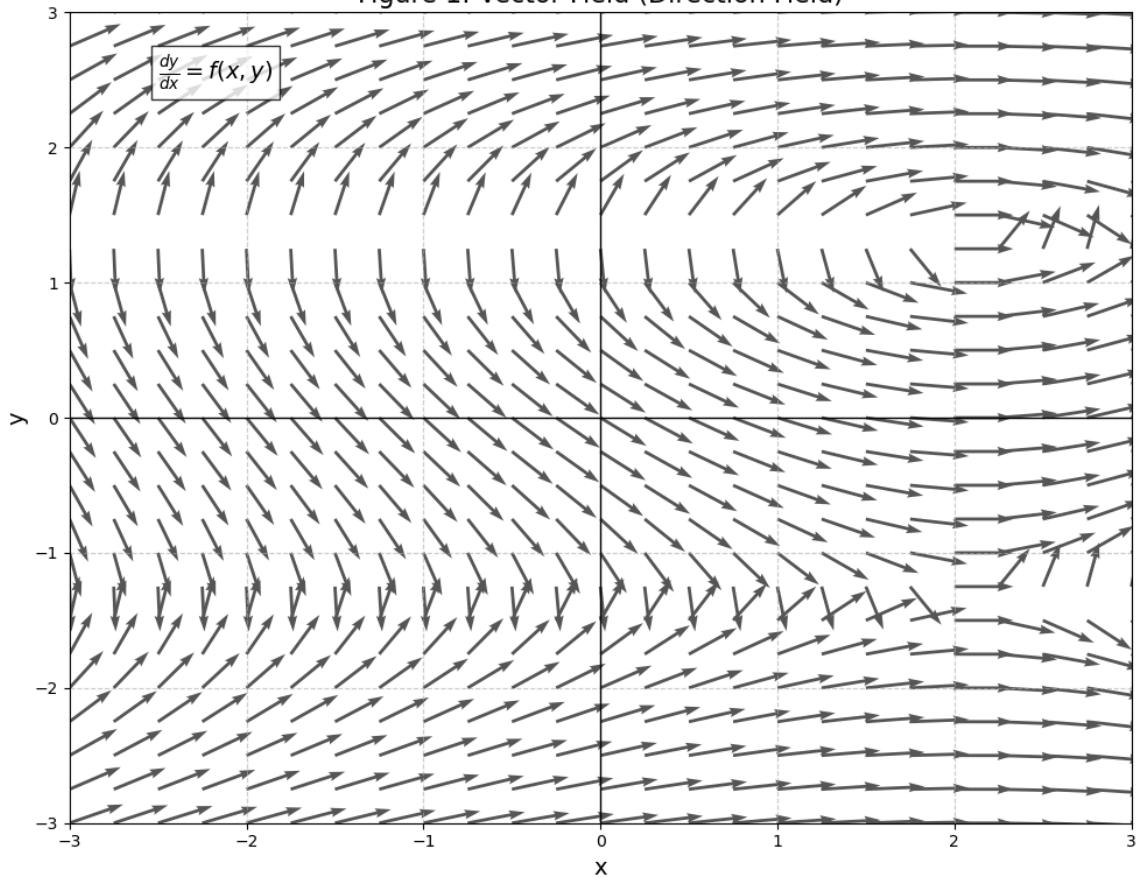
Figure 3: Specific Numerical Solutions



```
plots_ODE(ode_f8)
```

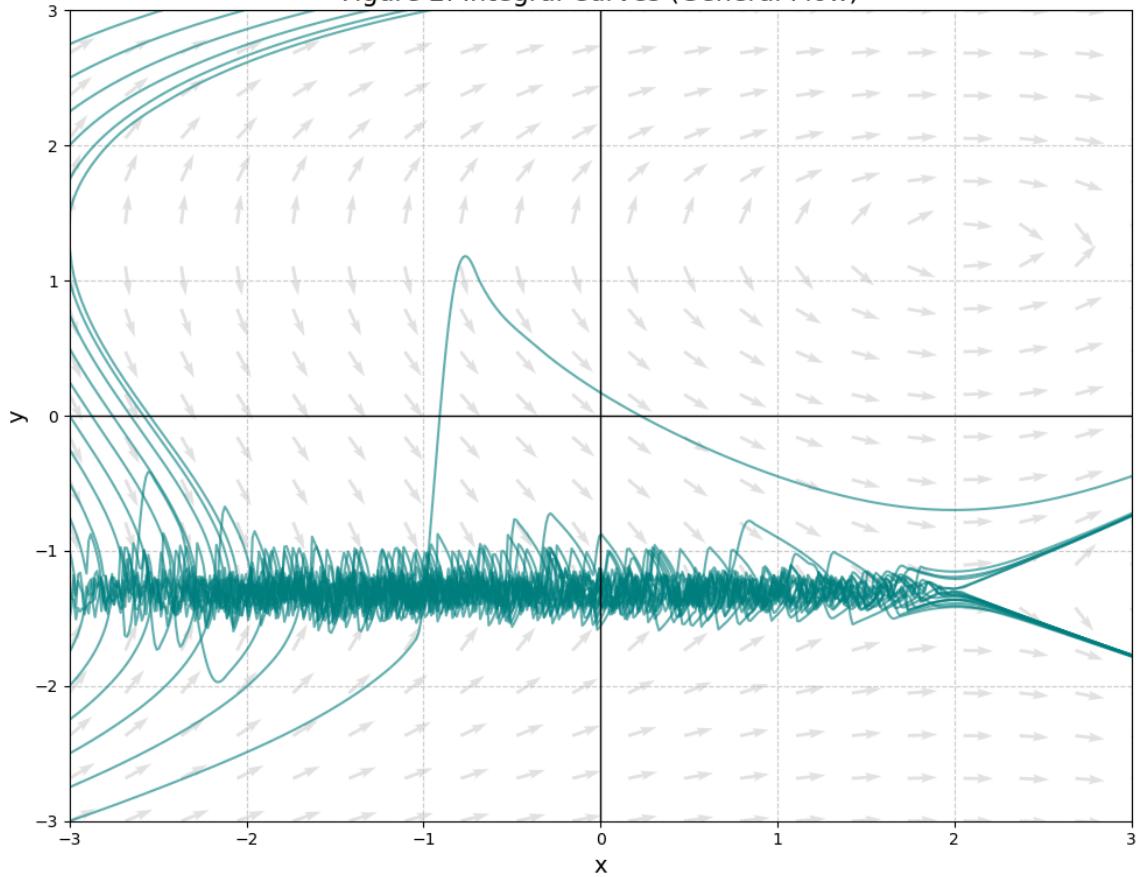
```
--- Differential Equation Visualizer ---
ODE: dy/dx = x - y
Domain: x in (-3, 3), y in (-3, 3)
```

Figure 1: Vector Field (Direction Field)



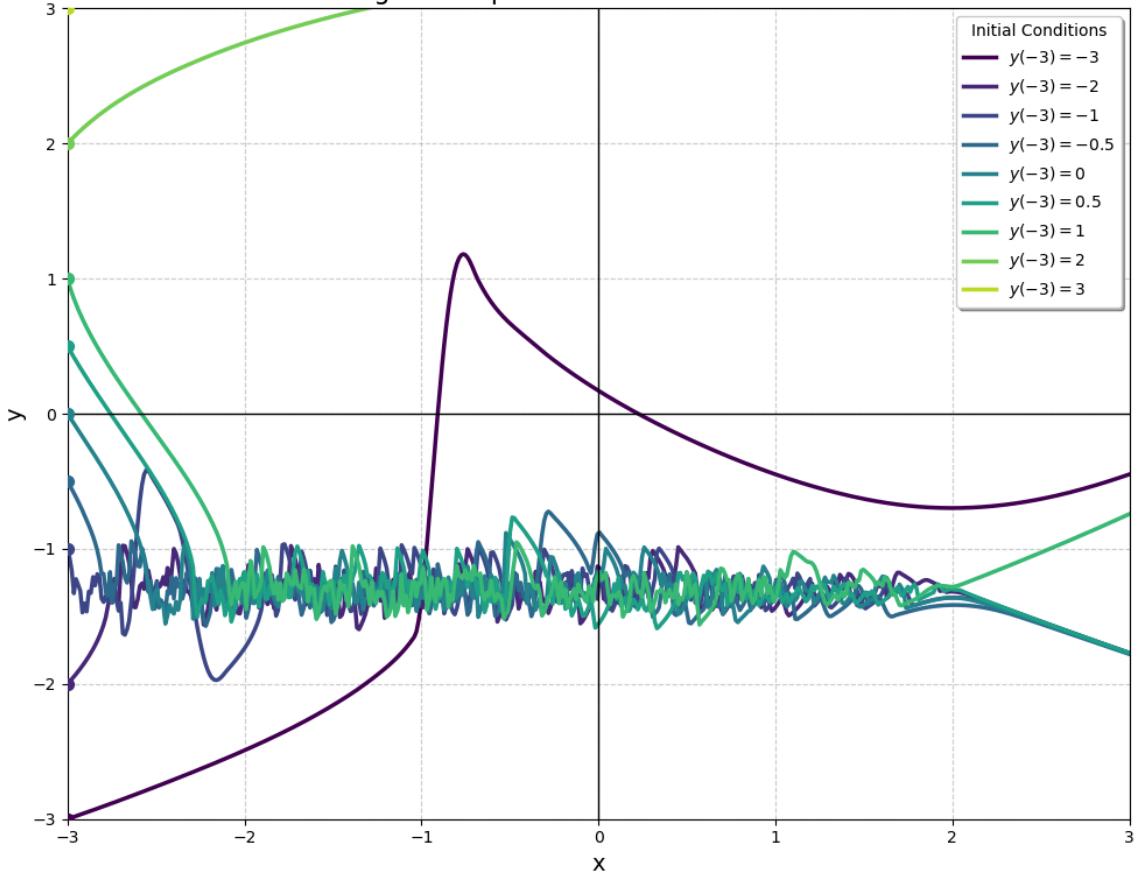
Generating 25 integral curves...

Figure 2: Integral Curves (General Flow)



Generating specific solutions...

Figure 3: Specific Numerical Solutions



```
import session_info
```

```
session_info.show(html=False)
```

```
-----
matplotlib      3.10.6
numpy          2.3.5
scipy          1.16.3
session_info    v1.0.1
-----
```

```
IPython         9.7.0
jupyter_client  8.6.3
jupyter_core   5.8.1
jupyterlab     4.4.7
notebook       7.4.5
-----
```

```
Python 3.13.9 | packaged by Anaconda, Inc. | (main, Oct 21 2025, 19:16:10) [GCC]
```

```
11.2.0]
Linux-6.14.0-37-generic-x86_64-with-glibc2.39
-----
Session information updated at 2026-02-09 02:27
```