Numerical Solution to the Differential Equation $mg - \beta\dot{x}^2 - k(x - x_0) = m\ddot{x}$

Numerical Solution with Initial Conditions $x(0) = x_0$, $\dot{x}(0) = v_0$

To solve the nonlinear differential equation $mg - \beta\dot{x}^2 - k(x - x_0) = m\ddot{x}$ with initial conditions $x(0) = x_0$ and $\dot{x}(0) = v_0$, where m, g, β, k, and x_0 are constants, $\dot{x} = \frac{dx}{dt}$, and $\ddot{x} = \frac{d^2x}{dt^2}$, we use a numerical approach due to the nonlinear $\beta\dot{x}^2$ term. We transform the second-order equation into a system of first-order ODEs and apply the fourth-order Runge-Kutta (RK4) method.

Step 1: Transform to a First-Order System

Rewrite the equation:

$$m\ddot{x} + \beta\dot{x}^2 + k(x - x_0) = mg \tag{1}$$

Substitute $y = x - x_0$, so $x = y + x_0$, $\dot{x} = \dot{y}$, $\ddot{x} = \ddot{y}$:

$$m\ddot{y} + \beta\dot{y}^2 + ky = mg \tag{2}$$

Initial conditions:

$$x(0) = x_0 \implies y(0) = 0, \quad \dot{x}(0) = v_0 \implies \dot{y}(0) = v_0$$

Define:

$$y_1 = y, \quad y_2 = \dot{y} = \frac{dy}{dt}$$

Then:

$$\dot{y}_1 = y_2$$

$$\dot{y}_2 = \ddot{y} = \frac{mg - \beta\dot{y}^2 - ky}{m} = \frac{mg - \beta y_2^2 - ky_1}{m}$$

The system is:

$$\frac{dy_1}{dt} = y_2 \tag{3}$$

$$\frac{dy_2}{dt} = \frac{mg - \beta y_2^2 - ky_1}{m} \tag{4}$$

Initial conditions: $y_1(0) = 0$, $y_2(0) = v_0$. In vector form:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} y_2 \\ \frac{mg - \beta y_2^2 - k y_1}{m} \end{bmatrix}$$

This is an autonomous system since $\mathbf{f}$ does not explicitly depend on t.

Step 2: Choose a Numerical Method

The RK4 method is used for its fourth-order accuracy. For $\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y})$, the update from t_n to $t_{n+1} = t_n + h$ is:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{5}$$

where:

$$\mathbf{k}_1 = \mathbf{f}(t_n, \mathbf{y}_n) \tag{6}$$

$$\mathbf{k}_2 = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right) \tag{7}$$

$$\mathbf{k}_3 = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right) \tag{8}$$

$$\mathbf{k}_4 = \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_3) \tag{9}$$

Step 3: Apply RK4 to the System

For:

$$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} y_2 \\ \frac{mg - \beta y_2^2 - k y_1}{m} \end{bmatrix}$$

Compute:

- $\mathbf{k}_1 = \begin{bmatrix} y_{2,n} \\ \frac{mg - \beta y_{2,n}^2 - k y_{1,n}}{m} \end{bmatrix}$
- $\mathbf{k}_2 = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right)$
- $\mathbf{k}_3 = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right)$
- $\mathbf{k}_4 = \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_3)$

Update:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

Step 4: Python Implementation

Below is a Python script using NumPy to implement RK4, with example parameters.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters
m = 1.0    # mass (kg)
g = 9.81   # gravity (m/s^2)
beta = 0.5 # damping coefficient (kg/m)
k = 10.0   # spring constant (N/m)
x0 = 0.0   # reference position (m)
v0 = 1.0   # initial velocity (m/s)

# Time settings
t_max = 10.0  # total time (s)
h = 0.01      # step size (s)
n_steps = int(t_max / h)
t = np.linspace(0, t_max, n_steps + 1)

# Initialize arrays
y = np.zeros((n_steps + 1, 2))  # [y1, y2]
y[0] = [0.0, v0]               # initial conditions: y1(0) = 0, y2(0) = v0

# Define the ODE system
def f(t, y):
    y1, y2 = y
    return np.array([y2, (m*g - beta*y2**2 - k*y1) / m])

# RK4 solver
for n in range(n_steps):
    y_n = y[n]
    k1 = f(t[n], y_n)
    k2 = f(t[n] + h/2, y_n + (h/2) * k1)
    k3 = f(t[n] + h/2, y_n + (h/2) * k2)
    k4 = f(t[n] + h, y_n + h * k3)
    y[n + 1] = y_n + (h/6) * (k1 + 2*k2 + 2*k3 + k4)

# Convert back to x
x = y[:, 0] + x0
v = y[:, 1]

# Plot results
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(t, x, label='Position $x(t)$')
plt.xlabel('Time (s)')
plt.ylabel('Position (m)')
plt.grid(True)
plt.legend()

plt.subplot(2, 1, 2)
```

```
plt.plot(t, v, label='Velocity $\dot{x}(t)$', color='orange')
plt.xlabel('Time (s)')
plt.ylabel('Velocity (m/s)')
plt.grid(True)
plt.legend()

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

Step 5: Interpret the Numerical Solution

- **Parameters**: Example values: $m = 1$, $g = 9.81$, $\beta = 0.5$, $k = 10$, $x_0 = 0$, $v_0 = 1$. Adjust based on the physical system.
- **Step Size**: $h = 0.01$ ensures accuracy; smaller h improves precision.
- **Output**: The script computes $y_1(t) = y(t) = x(t) - x_0$ and $y_2(t) = \dot{x}(t)$, converting to $x(t) = y_1(t) + x_0$. Plots show position and velocity.
- **Behavior**: The quadratic damping $\beta\dot{y}^2$ causes the system to approach equilibrium $y = \frac{mg}{k}$ (or $x = x_0 + \frac{mg}{k}$) with damped oscillations, depending on v_0 and β.

Step 6: Considerations

- **Stability**: RK4 is stable for small h. For stiff systems (large k/m), consider adaptive methods (e.g., `scipy.integrate.solve_ivp`).
- **Accuracy**: Verify by reducing h and checking convergence.
- **Physical Interpretation**: The system models a mass on a spring with quadratic damping and gravity, capturing damped oscillations or monotonic approaches to equilibrium.

Final Answer

To numerically solve $mg - \beta\dot{x}^2 - k(x - x_0) = m\ddot{x}$ with $x(0) = x_0$, $\dot{x}(0) = v_0$:

1. Transform to:

$$\frac{dy_1}{dt} = y_2 \tag{10}$$

$$\frac{dy_2}{dt} = \frac{mg - \beta y_2^2 - ky_1}{m} \tag{11}$$

   with $y_1(0) = 0$, $y_2(0) = v_0$.

2. Apply RK4 with a small step size (e.g., $h = 0.01$).
3. Implement in Python (as shown) to compute $x(t) = y_1(t) + x_0$ and $\dot{x}(t) = y_2(t)$.
4. Analyze results to understand the system's behavior, influenced by m, β, k, g, and v_0.