Control Theory Simulation

Coauthors: ece21033, ChatGPT wherever mentioned

Table of Contents

1. Abstract

This report paper describes the development of a numerical method for simulating a second order dynamics problem with low-cost calculations that can be executed in real-time applications.

1. Problem Description

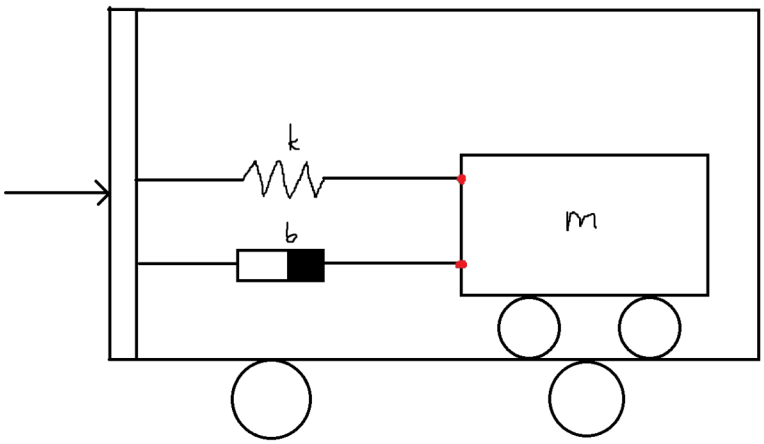

Figure 1

Consider the setup of a mass-less cart which contains a system of mass-spring-dampener as shown in Figure 1.

The input of the system x(t) represents the displacement of the outer cart from its initial rest position. The output of interest is the displacement of the inner cart from its initial rest position as a response to the input, which we shall label y(t).

The dampener has a viscous friction coefficient of b and its force is linearly proportional to the relative velocity of its ends.

The spring follows Hooke's Law and as such its force is linearly proportional to the relative displacement of its ends.

We require a mathematical representation of the system with zero initial conditions (rest state) and a method by which the defined problem can be solved and computed numerically.

2. Modeling

Writing the above in equation form and applying Newton's 2nd Law to the system we arrive at a 2nd order differential equation which fully describes the dynamics of the system:

$$m\frac{d^2y}{dt^2} = -b\left(\frac{dy}{dt} - \frac{dx}{dt}\right) - k\left(y - x\right) \Leftrightarrow$$
$$\Leftrightarrow m\frac{d^2y}{dt^2} + b\frac{dy}{dt} + ky = b\frac{dx}{dt} + kx$$

Where the term $\left(\frac{dy}{dt} - \frac{dx}{dt}\right)$ represents the relative velocity of the inner cart to the outer cart and thus the ends of the dampener, and the term $\left(y - x\right)$ is the elongation of the spring.

The choice of negative sign and coefficient for the forces is arbitrary, however, physically reasonable interpretations arise only for solutions with b>0, k>0 as per classical physics conventions.

The ensuing analysis is a solution to the differential equation under specific conditions, which will be summarized after the derivation. As a consequence, the behavior of the simulation beyond this framework of conditions is not well defined.

3. Solution

First, we apply the Laplace Transform to the differential equation:

$$\mathscr{L}\left\{m\frac{d^2y}{dt^2} + b\frac{dy}{dt} + ky\right\} = \mathscr{L}\left\{b\frac{dx}{dt} + kx\right\} \Leftrightarrow$$

$$\Leftrightarrow \left(ms^2 + bs + k\right)\boldsymbol{Y}\left(s\right) = \left(bs + k\right)\boldsymbol{X}\left(s\right)$$

Because of the assumed zero initial conditions, we get a reduced form of the equation.

Next, we find the transfer function of the system as per the definition:

$$\boldsymbol{G}(s)=\frac{\boldsymbol{Y}(s)}{\boldsymbol{X}(s)}=\frac{bs+k}{ms^2+bs+k}$$

Following classical solution methods we proceed to rearrange this expression to fit standard forms.

For the denominator we complete the square and define the quantity ω^2 with the end goal of matching it to a specific Laplace Transform pair with a shifted s term, as follows:

$$ms^2+bs+k=m\left(s^2+\frac{b}{m}s+\frac{k}{m}\right)=m\left(\alpha^2+2\alpha\beta+\beta^2+c\right)\Rightarrow\beta=\frac{b}{2m}$$

$$\Rightarrow ms^2+bs+k=m\left(\left(s+\frac{b}{2m}\right)^2+c\right)$$

$$c=\omega^2,\omega>0:c+\beta^2=\frac{k}{m}\Leftrightarrow\omega^2+\frac{b^2}{4m^2}=\frac{k}{m}\Rightarrow\omega=\sqrt{\frac{k}{m}-\left(\frac{b}{2m}\right)^2}\Rightarrow$$

$$\Rightarrow ms^2+bs+k=m\left(\left(s+\frac{b}{2m}\right)^2+\omega^2\right)$$

The standard form could be solved to include negative values of ω, as well as the edge case of ω=0 in the form of separate cases.

In order to reduce the analytical complexity for this simple proof of concept simulation we rely on the ω term here being real valued and thus we have the constraint that:

$$\frac{k}{m}\geq\left(\frac{b}{2m}\right)^2\Rightarrow b\leq 2\sqrt{k}$$

For the numerator we factor out a term $\left(s+\frac{b}{2m}\right)$:

$$bs+k=a\left(s+\frac{b}{2m}\right)+\beta=as+\frac{ab}{2m}+\beta\Rightarrow a=b\Rightarrow bs+k=bs+\frac{b^2}{2m}+\beta\Rightarrow$$
$$\Rightarrow\beta=k-\frac{b^2}{2m}\Rightarrow$$

$$\Rightarrow bs+k=b\left(s+\frac{b}{2m}\right)+k-\frac{b^2}{2m}$$

Using the above we decompose the transfer function G(s) into the sum:

$$\boldsymbol{G}(s)=\frac{b\left(s+\frac{b}{2m}\right)}{m\left(\left(s+\frac{b}{2m}\right)^2+\omega^2\right)}+\frac{k-\frac{b^2}{2m}}{m\left(\left(s+\frac{b}{2m}\right)^2+\omega^2\right)}$$

We match these terms to the following Laplace Transform pairs and apply fundamental properties to derive the time domain weighting function:

$$\mathscr{L}^{-1}\{\frac{s+a}{(s+a)+\omega^2}\}=e^{-at}cos\omega t$$
$$\mathscr{L}^{-1}\{\frac{1}{(s+a)+\omega^2}\}=\frac{e^{-at}sin\omega t}{\omega}$$

$$g(t)=\mathscr{L}^{-1}\{\boldsymbol{G}(s)\}=\frac{b}{m}e^{-\frac{b}{2m}t}cos\omega t+\frac{k-\frac{b^2}{2m}}{m\omega}e^{-\frac{b}{2m}t}sin\omega t$$

From the differential equation it can be shown that this is an LTI system and as such its output for any time t is given by the convolution of the input with the weighting function:

$$y(t)=\int_{-\infty}^{+\infty}x(\tau)g(t-\tau)d\tau=\int_{0}^{t}x(\tau)g(t-\tau)d\tau$$

which takes the above form because of causality constraints and zero initial conditions.

In summary, these solutions are only defined for:

- Zero Initial Conditions
- $b\leq 2\sqrt{k}$ i. e. under-damped or critically damped systems only
- One dimensional motion
- Causal real-time input on LTI systems

Therefore, the role of the simulation module is to compute this integral for any given time moment in an efficient manner.

4. Method of computation

In order to compute this convolution integral numerically we discretize the formula by the substitution $t=n\Delta t$, $\tau=k\Delta t$, where n is the current time sample for which we calculate the output, and k is the integration index:

$$y(n\Delta t)\approx\sum_{k=0}^{n}g((n-k)\Delta t)\,x(k\Delta t)\,\Delta t$$

We have arrived at a discrete version of the convolution. There exist two main computational problems with applying this formula as is:

- Firstly, for a reasonable real-time simulation with a small-enough Δt and a theoretically unknown run-time, the space required to store all the previous samples is theoretically unlimited and practically unachievable, even with an approximate time window.
- Secondly, the time cost of calculating this integral over all the samples is barring for real-time applications.

We attempt to bypass these problems by leveraging recursive relationships to further simplify the formula and reduce the computational complexity for a real time application.

To this end, we define the following notation and rewrite the formula before continuing:

$$x[n]=x(n\Delta t) \quad y[n]=y(n\Delta t) \quad g[n-k]=g((n-k)\Delta t)$$

$$y[n]=\Delta t\sum_{k=0}^{n} g[n-k]\,x[k]$$

5. Required Mathematical Proof

There exists a very specific relationship for calculating the cosine of angle multiples recursively, known as the Chebyshev method. The relationship can be proven in many ways that are outside the scope of this project. These can be found in [1][2].

The following proof outline makes the connection of this relationship to the required integral more clear and requires only a fundamental understanding of linear algebra. More information on the topic can be found in [3] and any general linear algebra textbook.

We begin by asserting without explicit notation the basic relationships for a change of basis transform. From these known relationships and the geometric definition of the sine and cosine we can very simply derive the matrix which performs a rotation by an arbitrary angle θ as shown in figure 2.

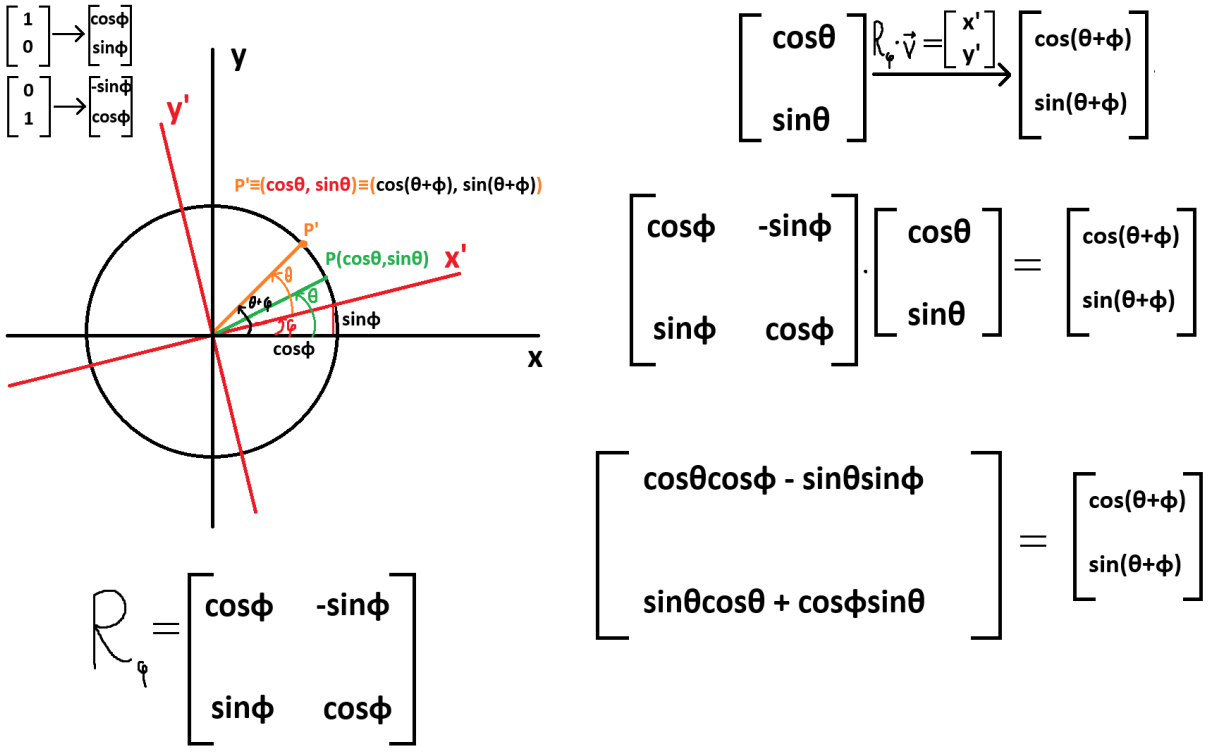

Figure 2

As such a recursive formula for computing the cosine of the n-th angle multiple naturally arises.

We have the rotation matrix:

$$R_\theta = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

By simple substitution of nθ in the above formulas we get the relationship:

$$R_\theta \begin{bmatrix} \cos(n\theta) \\ \sin(n\theta) \end{bmatrix} = \begin{bmatrix} \cos((n+1)\theta) \\ -\sin((n+1)\theta) \end{bmatrix}$$

which holds in general for arbitrary angle θ.

Expand:

$$R_\theta \begin{bmatrix} \cos(n\theta) \\ \sin(n\theta) \end{bmatrix} = \begin{bmatrix} cos\theta cos(n\theta) - sin\theta sin(n\theta) \\ \sin(\theta)\sin(n\theta) + cos\theta sin(n\theta) \end{bmatrix} = \begin{bmatrix} \cos((n+1)\theta) \\ -\sin((n+1)\theta) \end{bmatrix}$$

We now write the general expression for the k-th cosine and utilize the above:

$$cosk\theta = cos\theta cos((k-1)\theta) - sin\theta sin((k-1)\theta) \Leftrightarrow \ \textit{Expand} \ \sin((k-1)\theta)$$

$$\Leftrightarrow cosk\theta = cos\theta cos((k-1)\theta) - sin\theta[sin\theta cos((k-2)\theta) + cos\theta sin((k-2)\theta)] \Leftrightarrow \ \textit{Distribute}$$

$$\Leftrightarrow cosk\theta = cos\theta cos((k-1)\theta) - \sin^2\theta cos((k-2)\theta) - sin\theta cos\theta sin((k-2)\theta)$$

At this point we notice the term: $sin\theta sin((k-2)\theta) = cos\theta cos((k-2)\theta) - \cos((k-1)\theta)$ from the above

$$\Leftrightarrow cosk\theta = cos\theta cos((k-1)\theta) - \sin^2\theta cos((k-2)\theta) - cos\theta[cos\theta cos((k-2)\theta) - \cos((k-1)\theta)] \Leftrightarrow \textit{Distribute}$$

$$\Leftrightarrow cosk\theta = cos\theta cos((k-1)\theta) - \sin^2\theta cos((k-2)\theta) - \cos^2\theta \cos((k-2)\theta) + cos\theta cos((k-1)\theta) \Leftrightarrow \ \textit{Gather , Factor}$$

$$\Leftrightarrow cosk\theta = 2cos\theta cos((k-1)\theta) - \cos((k-2)\theta)[\sin^2\theta + \cos^2\theta] \Leftrightarrow$$

$$cosk\theta = 2\,cos\theta cos((k-1)\theta) - \cos((k-2)\theta)$$

$$Q.E.D$$

A similar relationship exists for sinkθ the proof of which is not shown.

6. Proof of recursive relationship for transfer function

By considering the above formula we now understand that because g(t) contains cosine terms it can be calculated as a combination of only a few of its previous values, which correspond to previous iterations of the n-th cosine. With this intuition we attempt a derivation of a general recursive relationship for g[n] as follows:

As a reminder, the relationship for g[n] is:

$$g[n]=g(n\Delta t)=\frac{b}{m}e^{-\frac{b}{2m}n\Delta t}cos\omega\Delta t+\frac{k-\frac{b^2}{2m}}{m\omega}e^{-\frac{b}{2m}n\Delta t}sin\omega\Delta t$$

We rewrite:

$$g[n]=e^{-\frac{b}{2m}n\Delta t}[\frac{b}{m}cos\omega\Delta t+\frac{k-\frac{b^2}{2m}}{m\omega}sin\omega\Delta t]$$

and for convenience let us name $A=\frac{b}{m},B=\frac{k-\frac{b^2}{2m}}{m\omega}$ to get:

$$g[n]=e^{-\frac{b}{2m}n\Delta t}[Acos(\omega\Delta t)+Bsin(\omega\Delta t)]=$$

$$=e^{-\frac{b}{2m}n\Delta t}(A[2\cos(\omega\Delta t)\cos((n-1)\omega\Delta t)-\cos((n-2)\omega\Delta t)]+B[2\cos(\omega\Delta t)\sin((n-1)\omega\Delta t)-\sin((n-2)\omega\Delta t)])=$$

$$=e^{-\frac{b}{2m}n\Delta t}[2Acos(\omega\Delta t)\cos((n-1)\omega\Delta t)-Acos((n-2)\omega\Delta t)+2Bcos(\omega\Delta t)\sin((n-1)\omega\Delta t)-Bsin((n-2)\omega\Delta t)]=$$

$$=e^{-\frac{b}{2m}n\Delta t}(2\cos(\omega\Delta t)[Acos((n-1)\omega\Delta t)+Bsin((n-1)\omega\Delta t)]-[Acos((n-2)\omega\Delta t)+Bsin((n-2)\omega\Delta t)])$$

We perform the following factorization:

$$e^{-\frac{b}{2m}n\Delta t}=e^{-\frac{b}{2m}(n-1+1)\Delta t}=e^{-\frac{b}{2m}(n-1)\Delta t-\frac{b}{2m}\Delta t}=e^{-\frac{b}{2m}(n-1)\Delta t}e^{-\frac{b}{2m}\Delta t}$$

$$e^{-\frac{b}{2m}n\Delta t}=e^{-\frac{b}{2m}(n-2+2)\Delta t}=e^{-\frac{b}{2m}(n-2)\Delta t-\frac{2b}{2m}\Delta t}=e^{-\frac{b}{2m}(n-2)\Delta t}e^{-\frac{b}{m}\Delta t}$$

To achieve:

$$g[n]=2e^{-\frac{b}{2m}}\cos(\omega\Delta t)e^{-\frac{b}{2m}(n-1)\Delta t}[Acos((n-1)\omega\Delta t)+Bsin((n-1)\omega\Delta t)]-e^{-\frac{b}{m}\Delta t}e^{-\frac{b}{2m}(n-2)\Delta t}[Acos((n-2)\omega\Delta t)+Bsin((n-2)\omega\Delta t)]$$

By observation we notice that the terms we have constructed are the factors g[n-1] and g[n-2], thus:

$$g[n]=2e^{-\frac{b}{2m}\Delta t}\cos(\omega\Delta t)g[n-1]-e^{-\frac{b}{m}\Delta t}g[n-2]$$

What we have shown here is that there exists a recursive relationship for the weighting function. This then leads to a recursive relationship for the output function.

7. Proof of recursive relationship for output

The following are two different methods by which the output can be expressed as a combination of previous outputs and inputs using the recursive g[k].

$$g[n-k]=2e^{-\frac{b}{2m}}\cos(\omega\Delta t)g[(n-k)-1]-e^{-\frac{b}{m}\Delta t}g[(n-k)-2]\Rightarrow$$

$$\Rightarrow g[n-k]x[k]=2e^{-\frac{b}{2m}}\cos(\omega\Delta t)g[(n-k)-1]x[k]-e^{-\frac{b}{m}\Delta t}g[(n-k)-2]x[k]$$

For simplicity in the following proofs we name $a=\frac{b}{2m}$ and we write:

$$\sum_{k=0}^{n}g[n-k]x[k]=2e^{-a\Delta t}\cos(\omega\Delta t)\sum_{k=0}^{n}g[(n-k)-1]x[k]-e^{-2a\Delta t}\sum_{k=0}^{n}g[(n-k)-2]x[k]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)\left[\sum_{k=0}^{n-1}g[(n-k)-1]x[k]+g[-1]x[n]\right]-e^{-2a\Delta t}\left[\sum_{k=0}^{n-2}g[(n-k)-2]x[k]+g[-1]x[n-1]+g[-2]x[n]\right]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)g[-1]x[n]-e^{-2a\Delta t}g[-2]x[n]-e^{-2a\Delta t}g[-1]x[n-1]+2e^{-a\Delta t}\cos(\omega\Delta t)\left[\sum_{k=0}^{n-1}g[(n-k)-1]x[k]\right]-e^{-2a\Delta t}\left[\sum_{k=0}^{n-2}g[(n-k)-2]x[k]\right]=$$

$$=x[n]g[0]-e^{-2a\Delta t}g[-1]x[n-1]+2e^{-a\Delta t}\cos(\omega\Delta t)\left[\sum_{k=0}^{n-1}g[(n-k)-1]x[k]\right]-e^{-2a\Delta t}\left[\sum_{k=0}^{n-2}g[(n-k)-2]x[k]\right]=$$

$$=x[n]g[0]+2e^{-a\Delta t}\cos(\omega\Delta t)\frac{y[n-1]}{\Delta t}-e^{-2a\Delta t}\frac{y[n-2]}{\Delta t}-e^{-2a\Delta t}g[-1]x[n-1]$$

The negative index creates a slight problem in interpretation and practical implementation that can be bypassed as follows:

$$\sum_{k=0}^{n} g[n-k]x[k] = x[n]g[0] + 2e^{-a\Delta t}\cos(\omega\Delta t)\frac{y[n-1]}{\Delta t} - e^{-2a\Delta t}\frac{y[n-2]}{\Delta t} - e^{-2a\Delta t}g[-1]x[n-1] + 2e^{-a\Delta t}\cos(\omega\Delta t)g[0]x[n-1] - 2e^{-a\Delta t}\cos(\omega\Delta t)g[0]x[n-1] =$$

$$= x[n]g[0] + 2e^{-a\Delta t}\cos(\omega\Delta t)\frac{y[n-1]}{\Delta t} - e^{-2a\Delta t}\frac{y[n-2]}{\Delta t} + g[1]x[n-1] - 2e^{-a\Delta t}\cos(\omega\Delta t)g[0]x[n-1] =$$

$$= x[n]g[0] + 2e^{-a\Delta t}\cos(\omega\Delta t)\frac{y[n-1]}{\Delta t} - e^{-2a\Delta t}\frac{y[n-2]}{\Delta t} + (g[1] - 2e^{-a\Delta t}\cos(\omega\Delta t)g[0])x[n-1]$$

Therefore, the output relationship we are looking for is:

$$y[n] = \Delta t\sum_{k=0}^{n} g[n-k]x[k] = 2e^{-a\Delta t}\cos(\omega\Delta t)y[n-1] - e^{-2a\Delta t}y[n-2] + x[n]g[0]\Delta t + [(g[1] - 2e^{-a\Delta t}\cos(\omega\Delta t)g[0])\Delta t]x[n-1]$$

This second method avoids the undefined behavior of negative indexes for g[k] altogether:

We begin this proof constructively by writing expanded expressions for the values of the output and separating out very specific sums which will be used in later steps.

$$y[n-1] = \Delta t\sum_{k=0}^{n-1} g[k]x[n-k-1] = \Delta t\left[g[0]x[n-1] + g[1]x[n-2] + g[2]x[n-3] + \ldots + g[n-1]x[0]\right]$$

$$\sum_{k=2}^{n-1} g[k]x[n-k-1] = \frac{y[n-1]}{\Delta t} - g[0]x[n-1] - g[1]x[n-2]$$

$$y[n-2] = \Delta t\sum_{k=0}^{n-2} g[k]x[n-k-2] = \Delta t\left[g[0]x[n-2] + g[1]x[n-3] + g[2]x[n-4] + \ldots + g[n-2]x[0]\right]$$

$$\sum_{k=1}^{n-2} g[k]x[n-k-2] = \frac{y[n-2]}{\Delta t} - g[0]x[n-2]$$

$$y[n] = \Delta t\sum_{k=0}^{n} g[k]x[n-k] = \Delta t\left[g[0]x[n] + g[1]x[n-1] + g[2]x[n-2] + \ldots + g[n]x[0]\right] =$$

$$= \Delta t\left[g[0]x[n] + g[1]x[n-1] + g[2]x[n-2] + \sum_{k=3}^{n} g[k]x[n-k]\right]$$

$$\sum_{k=3}^{n} g[k]x[n-k]=\sum_{k=3}^{n}[2e^{-a\Delta t}\cos(\omega\Delta t)g[k-1]-e^{-2a\Delta t}g[k-2]]x[n-k]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)\sum_{k=3}^{n} g[k-1]x[n-k]-e^{-2a\Delta t}\sum_{k=3}^{n} g[k-2]x[n-k]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)\sum_{k=2}^{n-1} g[k]x[n-k-1]-e^{-2a\Delta t}\sum_{k=1}^{n-2} g[k]x[n-k-2]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)\left[\frac{y[n-1]}{\Delta t}-g[0]x[n-1]-g[1]x[n-2]\right]-e^{-2a\Delta t}\left[\frac{y[n-2]}{\Delta t}-g[0]x[n-2]\right]=$$

$$=2e^{-a\Delta t}\cos(\omega\Delta t)\frac{y[n-1]}{\Delta t}-2e^{-a\Delta t}\cos(\omega\Delta t)g[0]x[n-1]-2e^{-a\Delta t}\cos(\omega\Delta t)g[1]x[n-2]-e^{-2a\Delta t}\frac{y[n-2]}{\Delta t}+e^{-2a\Delta t}g[0]x[n-2]=$$

We substitute this relationship back into y[n] and name $A=2e^{-a\Delta t}\cos(\omega\Delta t)$, $B=-e^{-2a\Delta t}$ to get:

$$y[n]=Ay[n-1]+By[n-2]+\Delta t(g[0]x[n]+g[1]x[n-1]+g[2]x[n-2]-Ag[0]x[n-1]-Ag[1]x[n-2]-Bg[0]x[n-2])$$

And we note that the last two terms can be combined to reduce the formula:

$$y[n]=Ay[n-1]+By[n-2]+\Delta t\left(g[0]x[n]+(g[1]-Ag[0])x[n-1]+g[2]x[n-2]-g[2]x[n-2]\right)$$

$$y[n]=2e^{-a\Delta t}\cos(\omega\Delta t)y[n-1]-e^{-2a\Delta t}y[n-2]+\Delta t g[0]x[n]+[(g[1]-2e^{-a\Delta t}\cos(\omega\Delta t)g[0])\Delta t]x[n-1]$$

The output is the same formula.

What we have shown here is a method by which we can calculate the convolution integral from pre-computed factors and a few previous samples for the input and output, which significantly reduces computational cost to almost constant time.

8. Proof by the graphical method

In order to validate the final formula for use in real-time applications we must ensure that its results match as closely as possible to the calculation of the convolution integral for any time t of the fundamental definition process. We use the definition integral to calculate the graph of a unit input response as a reference in non real-time, and then calculate the same output using the simplified calculation method, again in non real-time, using fixed time steps to simulate frames of a real-time application. We observe that the resulting graphs are identical.

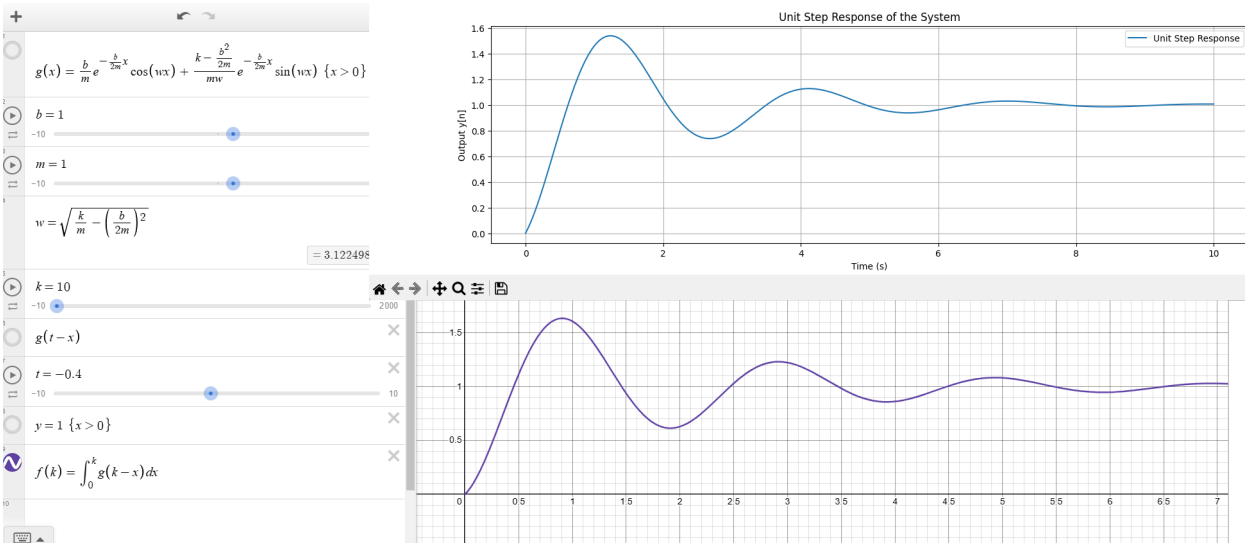

Figure 3: Output graph comparison

9. Code implementation

Below is a tool assisted (ChatGPT) implementation of the above equations in python. The RecursiveConvolution class is a standalone module that was modified and used for graphing purposes and then integrated as a part of a larger framework.

```
import math
import matplotlib.pyplot as plt

class RecursiveConvolution:
    def __init__(self):
        """
        Initialize the class with default parameters and state variables.
        """
        self.g0 = None
        self.g1 = None
        self.g2 = None
        self.alpha = None
        self.omega = None
        self.prev_y1 = 0  # Store the previous output value y[n-1]
        self.prev_y2 = 0  # Store the previous output value y[n-2]
        self.prev_x = 0   # Store the previous input value x[n-1]
        self.dt = None    # The time step size

    def update_parameters(self, b, m, k, dt):
        """
        Update the system parameters and compute g[0] and g[1].

        Parameters:
        b (float): Damping coefficient.
        m (float): Mass.
        k (float): Spring constant.
        dt (float): Time step size.
```

```
        """
        # Compute alpha and omega
        self.alpha = b / (2 * m)
        self.omega = math.sqrt(max(0, k / m - self.alpha ** 2))  # Natural
frequency

        self.g0 = b / m

        # Precompute g[1] for the given parameters
        self.g1 = (
            (b / m) * math.exp(-self.alpha * dt) * math.cos(self.omega * dt) +
            ((k - (b ** 2) / (2 * m)) / (m * self.omega)) * math.exp(-self.alpha *
dt) * math.sin(self.omega * dt)
        )

        self.dt = dt  # Store the time step size

    def calculate_output(self, x):
        """
        Compute the output y[n] for the given input x[n].

        Parameters:
        x (float): Current input value x[n].

        Returns:
        float: Output value y[n].
        """
        if self.g1 is None or self.dt is None:
            raise ValueError("System parameters not initialized. Call
'update_parameters' first.")

        # Compute the coefficients for the recursive calculation
        exp_neg_alpha_dt = math.exp(-self.alpha * self.dt)
        cos_term = math.cos(self.omega * self.dt)
        exp_neg_2alpha_dt = math.exp(-2 * self.alpha * self.dt)

        # Recursive calculation
        y = (
            2 * exp_neg_alpha_dt * cos_term * self.prev_y1 -
            exp_neg_2alpha_dt * self.prev_y2 +
            self.dt * self.g0 * x +
            self.dt * (self.g1 - 2 * exp_neg_alpha_dt * self.g0) * self.prev_x
        )

        # Update state variables
        self.prev_y2 = self.prev_y1
        self.prev_y1 = y
        self.prev_x = x

        return y
```

Shown: the RecursiveConvolution class used to achieve the final result

10. Result

The real-time application can be found at https://github.com/ECE3633/Control-Theory-Simulation.git

Below is the test-bench code that was used for generating the graph.

```
if __name__ == "__main__":
    # Initialize the convolution system
    conv = RecursiveConvolution()

    # Update parameters
    b, m, k, dt = 1, 1, 10, 0.001
    conv.update_parameters(b=b, m=m, k=k, dt=dt)

    # Generate unit step response
    num_points = 10000
    time_points = [n * dt for n in range(num_points)]
    unit_step_input = [1 for n in range(num_points)]

    # Calculate the output for the unit step input
    unit_step_output = []
    for x in unit_step_input:
        y = conv.calculate_output(x)
        unit_step_output.append(y)

    # Plot the output
    plt.figure(figsize=(10, 6))
    plt.plot(time_points, unit_step_output, label="Unit Step Response")
    plt.title("Unit Step Response of the System")
    plt.xlabel("Time (s)")
    plt.ylabel("Output y[n]")
    plt.grid(True)
    plt.legend()
    plt.show()
```

11. Limitations, parameters, test conditions

During development the following issues and observations were noted:

- The sampling rate of both the input and output were considered equal for simplicity of derivation, however a reasonable improvement would be to calculate the output integral multiple times per frame with a smaller Δt for more accurate approximation. Since the algorithm is recursive the change in the implementation is relatively simple by reusing the latest input of the current frame multiple times per frame as if the consecutive inputs were equal in value without changing the formula. The reasoning behind this is that the framerate can be limited to the typical 30 or 60 fps since a realistic input does not change with a frequency higher than 1/60 Hz and thus can be accurately captured at that rate (see: Nyquist frequency in [5]). As such, performing the simulation with a smaller time step within a single frame would give better approximation results without being dependent on or hindering performance.

- Although not mentioned, the formulas for the simplified calculation of the convolution assume evenly spaced sampling time Δt at a fixed rate. The choice for simplicity means that the current implementation does not account for this and that affects the performance of the system under unexpected conditions such as frame drops. This relates to the next point.

- If the factors are assumed constant (with a constant Δt) and pre-computed then any fluctuations in the frametime would result in physically inaccurate results in relation to actual time. The simulation still remains consistent in terms of frames since each frame is regarded as one Δt step of the simulation in this implementation and no additional inputs or outputs are taken within a single frame regardless of its actual time difference with the previous frame.

The aforementioned problems could be solved by implementing a system that splits the frametime estimate into steps and performs the simulation in a way that is somewhat detached from the frame loop. A few unsuccessful attempts were made, with a common cause of failure:

- If the formula is computed as is with Δt being a variable frametime (difference of time between frames), stability issues are introduced at slow framerates because of the large Δt step and the simulation becomes entirely inaccurate. To perform calculations with a variable framerate would require changing the analytical formula in some way that is beyond the scope of this project.

12. Future works and applications:

There are many potentially reasonable remedies to the practical problems, each with its own additional complexities best saved for future work. For instance, a hybrid approach wherein the frametime is split into multiples of some fixed time step Δt and the simulation runs multiple times per frame for as many such multiples as are available is one such future work idea, inspired by [6]

The scope of this problem was fairly limited by the solution conditions. A worthwhile investigation would be to ask if the analytical solutions could theoretically be expanded to account for more complex problems, whether or not the relationships can be vectorized in 3D and whether or not the formulas could possibly be used for over-damped systems.

The tools applied here were optimizations specific to the problem at hand and increased computational efficiency at the cost of application range. Potential applications include any two-body system connected by such a theoretical spring-dampener system.

13. Bibliography

1. https://en.wikipedia.org/wiki/List_of_trigonometric_identities#:~:text=%5Bedit%5D-,The%20Chebyshev%20method,-is%20a%20recursive
2. https://trans4mind.com/personal_development/mathematics/trigonometry/multipleAnglesRecursiveFormula.htm
3. http://www.boris-belousov.net/2016/05/31/change-of-basis/

4. Modern Control Engineering (K. Ogata). ISBN: 978-0136156734
5. Modern Digital and Analog Communication Systems (B.P. Lathi). ISBN: 978-0190686864
6. https://www.youtube.com/watch?v=4r_EvmPKOvY