HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY – VNU HCMC
OFFICE FOR INTERNATIONAL STUDY PROGRAM
FACULTY OF APPLIED SCIENCE

———————— * ————————


# CALCULUS 1
# CALCULUS APPLICATION REPORT

Lecturer : **M.S. Phan Thị Khánh Vân**
Subject : **Calculus 1 (Exercise)**
Class : **CC03**
Group : **06**
Members : **Huỳnh Dương Gia Bảo** – 2252063
**Lương Triển Đạt** – 2252144
**Phùng Nhật Huy** – 2252261
**Tạ Anh Huy** – 2211272
**Phạm Hoàng Quân** – 2252687
**Võ Minh Thăng** – 2252762

Ho Chi Minh City, May 24$^{th}$, 2023

# Contents

# I Derivative

## 1. Theoretical Basis

The derivative is a fundamental concept in calculus that measures the rate at which a function changes. It provides valuable information about the slope or rate of change of a function at any given point. The derivative allows us to analyze how a function behaves locally, providing insights into its increasing or decreasing nature, the presence of maximum or minimum points, and the concavity of the function.

The derivative of a function $f(x)$ is defined using the concept of limits. Specifically, the derivative at a point $x = x_0$ is obtained by taking the limit of the difference quotient as the interval around a shrinks to zero. The difference quotient represents the average rate of change of the function $f(x)$ over a small interval, and by taking the limit, we can determine the instantaneous rate of change or the slope of the function at that specific point. Mathematically, the derivative is expressed as:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

This definition captures the concept of an infinitesimal change in $x$ and the corresponding change in the function value, allowing us to precisely calculate the derivative at a specific point.

For a better understanding of the aforementioned expression, we have created this Matlab script that simulates the graph by generating the secant line based on the limit and gradually reducing the variable $h$ from 2 to approximate 0. As $h$ approaches 0, the resulting line represents the tangent line at $x_0$ of the function $f(x)$. Below is the code:

```
clc; clearvars;

% User input for the function f(x)
f_str = input("Enter the function f(x): ", 's');
f = str2func(['@(x)', f_str]);

% User input for the initial point x0
x0 = input("Enter the point x0: ");

% Define the step size
h = 2;

% Create a range of x-values
x = linspace(x0 - h, x0 + 2 * h); % Offset the view

% Compute the corresponding y-values
y = f(x);

% Plot the function
plot(x, y, 'linewidth', 2)
hold on
grid on

```

```
% Compute the slope of the secant line
secant_line_slope = (f(x0 + h) - f(x0)) / (x0 + h - x0);

% Initialize the secant line
secant_line = secant_line_slope * (x - x0) + f(x0);

% Plot the secant line
secant_plot = plot(x, secant_line, '--', 'linewidth', 2);

% Plot the endpoint
end_point = plot(x0 + h, f(x0 + h), 'g.', 'LineWidth', 2, 'MarkerSize', 25);

% Plot the initial point
plot(x0, f(x0), 'r.', 'LineWidth', 2, 'MarkerSize', 25)

% Set axis labels and title
xlabel('x');
ylabel('y');
title(sprintf('x0 = %.2f, h = %.2f', x0, h));

% Set the initial view
xlim([x0 - h, x0 + 2 * h]);

% Compute the y-axis limits
y_real = real(f(x));
min_f = min(y_real);
max_f = max(y_real);
avg = (abs(max_f) + abs(min_f)) / 2;
ylim([min_f - avg * 1/2, max_f + avg * 1/2]);

pause(0.5)

% Animation loop
for h_ = h:-0.01:0.000000001
    secant_line_slope = (f(x0 + h_) - f(x0)) / (x0 + h_ - x0);

    % Update the secant line
    secant_line = secant_line_slope * (x - x0) + f(x0);

    % Update the plot
    set(secant_plot, 'YData', secant_line);
    set(end_point, 'XData', x0 + h_, 'YData', f(x0 + h_));

    title(sprintf('x0 = %.2f, h = %.2f', x0, h_));

    % Pause for a short duration to create an animation effect
    pause(0.01);
end

% Set the final secant line type to a straight line
set(secant_plot, 'LineStyle', '-');
```

Input and Result:

```
Enter the position of point A [x, y]: [0,0]
Enter the vector A [x, y]: [1,1]
Enter the position of point B [x, y]: [0,3]
Enter the duration of the simulation: 5
```

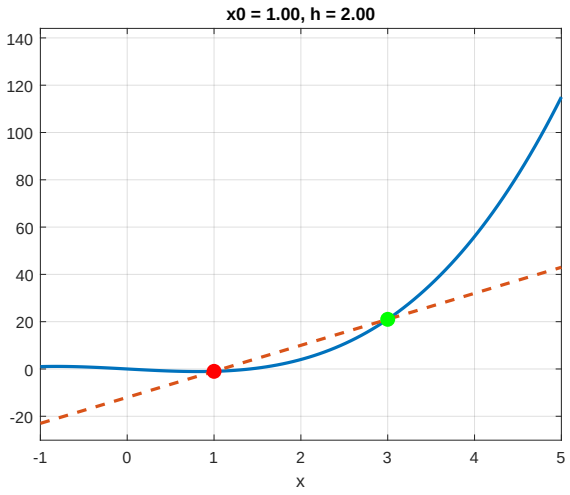



Figure 1: Initial state

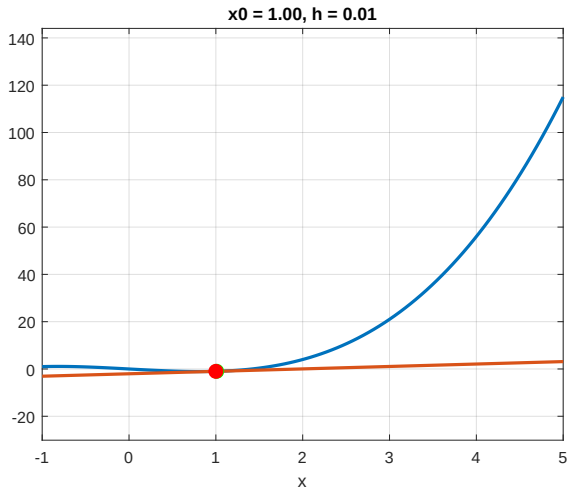



Figure 2: Final state

The derivative provides crucial information about the behavior of functions in various fields, such as physics, economics, engineering, and more. It enables us to study and model dynamic systems, optimize functions, analyze rates of change, and understand fundamental properties of functions. The power of the derivative lies in its ability to capture instantaneous behavior and provide valuable insights into the intricate details of mathematical functions.

# 2. Applications

## a. Electronic Speedometer

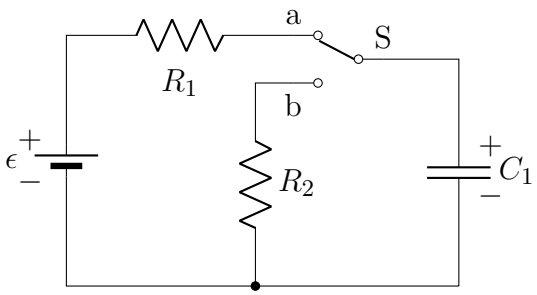
An electronic speedometer is a key component in modern vehicles, providing drivers with real-time speed information. It relies on wheel speed sensors that generate electrical signals corresponding to the rotation of the wheels. By counting the pulses from these sensors within a specific time interval, the electronic control unit (ECU) calculates the derivative of the pulse count, which represents the rate of change of wheel speed. This derivative value is then converted into a speed measurement using calibration parameters and scaling factors. The final speed value is displayed on the speedometer's digital or analog display, ensuring that drivers have accurate and instantaneous speed readings.


To ensure accurate readings, electronic speedometers often employ smoothing and filtering techniques. These techniques involve processing the derivative values to reduce noise and fluctuations caused by variations in wheel rotation. Filtering algorithms such as moving average or low-pass filters are applied to obtain a smoother representation of the speed. This helps drivers perceive a more stable speed reading, particularly in situations where wheel rotation may be inconsistent.

Overall, the application of derivatives in electronic speedometers enables the continuous monitoring and calculation of vehicle speed. By leveraging the principles of calculus, electronic speedometers provide drivers with reliable and up-to-date speed information, enhancing safety and enabling better control of the vehicle. Through the integration of wheel speed sensors, signal processing units, and display mechanisms, electronic speedometers ensure that drivers stay informed about their current speed while driving.

## b. Volume vs. Current in Capacitor

The derivative of a capacitor represents how its voltage changes with respect to time. A capacitor is an electronic component that stores electrical energy in the form of an electric field. It consists of two conductive plates separated by an insulating material, known as a dielectric.




When a voltage is applied across the capacitor, the plates accumulate opposite charges, creating an electric field between them. The voltage across the capacitor is directly proportional to the amount of charge stored on the plates. Mathematically, we can express this relationship as:

$$Q = CV$$

where $Q$ is the charge stored on the capacitor, $C$ is the capacitance (a measure of the capacitor's ability to store charge), and $V$ is the voltage across the capacitor.

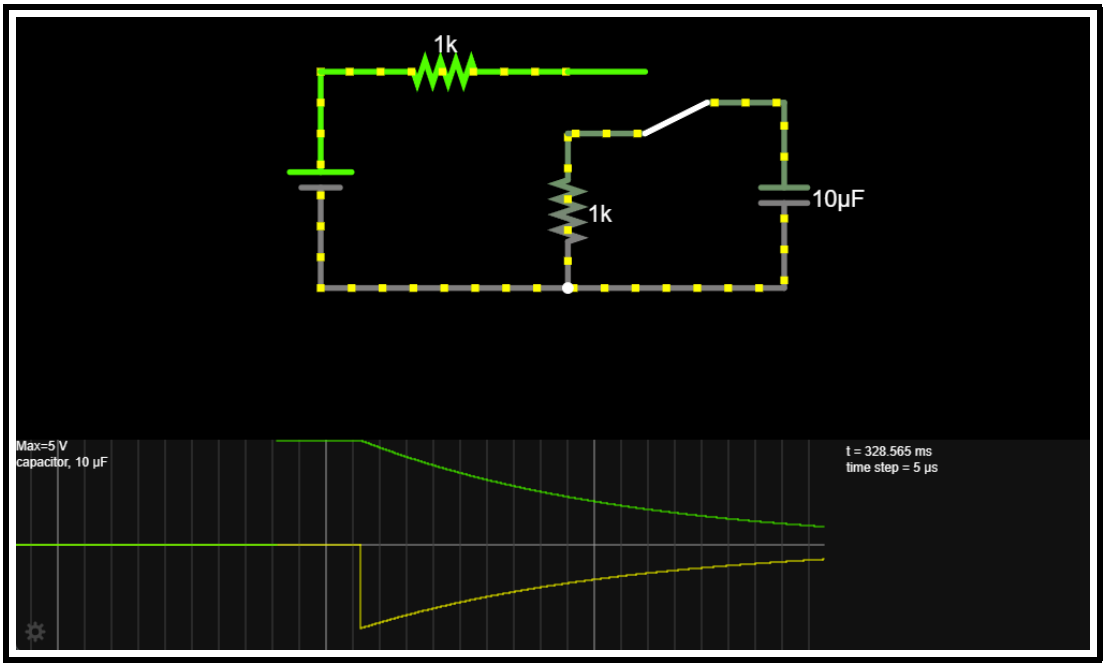
Now, let's differentiate both sides of the equation with respect to time $t$:

$$\frac{dQ}{dt} = C\frac{dV}{dt}$$

The left side of the equation $\frac{dQ}{dt}$ represents the rate of change of charge with respect to time, which is the current flowing into or out of the capacitor. The right side of the equation $C\frac{dV}{dt}$ represents the capacitance multiplied by the rate of change of voltage with respect to time.

In other words, the derivative of a capacitor's voltage $\frac{dV}{dt}$ multiplied by its capacitance $C$ gives you the current flowing through the capacitor. This relationship is known as the capacitor current-voltage relationship.

In practice, when you apply a changing voltage across a capacitor (for example, in an AC circuit), the derivative of the voltage waveform determines the current flowing through the capacitor. The derivative $\frac{dV}{dt}$ represents the slope of the voltage waveform, indicating how quickly the voltage is changing at any given time. The capacitance $C$ determines how much current is required to change the voltage at a particular rate.




Taking this example, after the capacitor fully charged, let's define the time $t_0 = 0$ as the time when we switch the switch in the other direction, and $V(t_0)$ currently reaches its maximum. As the capacitor discharges, the function $V(t)$ gradually decreases to 0, following the plot of the exponential function $V(t) = e^{-ax}$. With the same duration, the current is given by $I(t) = C\dfrac{dV}{dt} = -Cae^{-ax}$.

It's important to note that the derivative of the voltage across a capacitor is discontinuous when the voltage suddenly changes, such as during the charging or discharging of the capacitor. At those instances, the derivative approaches infinity momentarily, indicating a high current flow.

# II Integral

## 1. Theoretical Basis

Integrals are one of the fundamental concepts in mathematics and are used extensively in many fields of science and engineering. An integral is a mathematical representation of the area under a curve, and it provides a powerful tool for solving a wide range of problems.

The concept of integrals was first introduced in the 17$^{\text{th}}$ century by mathematicians such as Isaac Newton[1] and Gottfried Wilhelm Leibniz[2]. The integral is represented by the following formula:

$$\int_a^b f(x)dx = F(x)\Big|_a^b = F(b) - F(a)$$

Since its inception, the concept of integrals become a cornerstone of modern mathematics. Integrals have been used in many fields, including engineering, physics, economics, biology, computer science, and music theory.

To gain a clear understanding of integrals, the Riemann sum[3] is a technique used to approximate their values by dividing the area under a curve into smaller rectangles. This method provides an estimation by summing the areas of these rectangles, with each rectangle's width and height determined by the partitioning of the interval and the function's evaluation at specific points. As the number of rectangles increases, the approximation becomes more accurate, approaching the exact value of the integral. The Riemann sum holds a foundational role in integral calculus and plays a crucial part in comprehending advanced integration methods.

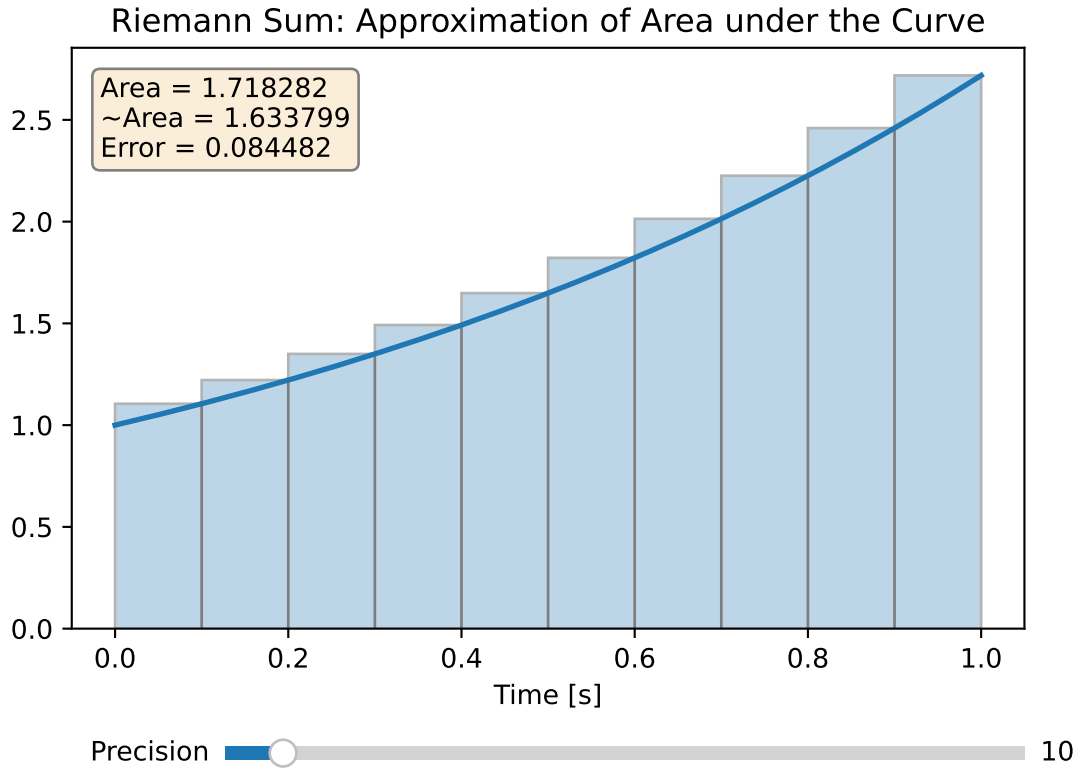
Now, let's examine the Python code we have developed to demonstrate the Riemann sum approximation for the area under a curve. The code includes a slider that controls the number of rectangles used to approximate the area. This allows us to observe the convergence of the Riemann sum approximation towards the exact area:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib.widgets import Slider
4
5  # User input for f(x) function
6  try:
7      fx = input("Enter the function f(x): ")
8      f = lambda x: eval(fx)
9  except Exception as e:
10     print("Error:", e)
11     print("Please enter a valid function expression.")
12
13 def riemann_sum(f, a, b, n):
14     """Compute the Riemann sum for the function f over the interval [a, b]"""
```

[1] Issac Newton: https://en.wikipedia.org/wiki/Isaac_Newton
[2] Gottfried Wilhelm Leibniz: https://en.wikipedia.org/wiki/Gottfried_Wilhelm_Leibniz
[3] Riemann sum: https://en.wikipedia.org/wiki/Riemann_sum

```
    dx = (b - a) / n  # Width of each subinterval
    x = np.linspace(a, b, n+1)  # Partition interval [a, b] into n subintervals
    heights = f(x[:-1])  # Heights of the rectangles using left endpoints
    total_area = np.sum(heights) * dx  # Sum the areas of all rectangles
    return total_area

# Define interval [a, b] and number of subintervals (n)
while True:
    try:
        a, b = map(float, input("Enter the domain values for a and b: ").split())
        break
    except ValueError:
        print("Invalid input. Please enter valid numeric values.")

n = 10

# Plotting the function
x = np.linspace(a, b, 10000)
y = f(x)

# Create the figure and the line
fig, ax = plt.subplots()
line, = ax.plot(x, y, lw=2)

plt.title('Riemann Sum: Approximation of Area under the Curve')

# Creating rectangles
x_bar = np.linspace(a, b, n+1)[:-1]
y_bar = f(x_bar + (b - a) / n)
bar = ax.bar(x_bar, y_bar, width=(b - a) / n, align='edge', alpha=0.3, edgecolor='k')
ax.set_xlabel('Time [s]')

total_area = np.trapz(y, x)
approx_area = riemann_sum(f, a, b, n)
error = total_area - approx_area
textString = f"Area = {total_area:.6f}\n~Area = {approx_area:.6f}\nError = {error:.6f}"
props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
text = ax.text(0.03, 0.95, textString, transform=ax.transAxes, fontsize=10,
        verticalalignment='top', bbox=props)

# Adjust main plot to make room for the slider
fig.subplots_adjust(bottom=0.25)

# Make a horizontal slider to control the precision
axfreq = fig.add_axes([0.25, 0.1, 0.65, 0.03])
freq_slider = Slider(
    ax=axfreq,
    label='Precision',
    valmin=3,
    valmax=100,
    valinit=n,
    valfmt='%0.0f'
)

# The function to be called anytime the slider's value changes
def update(val):
    global bar, approx_area, error, text
    n = int(val)
    x_bar = np.linspace(a, b, n+1)[:-1]
    y_bar = f(x_bar + (b - a) / n)
    bar.remove()
```

```
    bar = ax.bar(x_bar, y_bar, width=(b - a) / n, align='edge', alpha=0.3,
    ↪  edgecolor='k')

    approx_area = riemann_sum(f, a, b, n)
    error = total_area - approx_area
    textString = f"Area = {total_area:.6f}\n~Area = {approx_area:.6f}\nError =
    ↪  {error:.6f}"
    text.set_text(textString)

    fig.canvas.draw_idle()

freq_slider.on_changed(update)

plt.show()
```




In this report, we will explore the various applications of integrals in different fields. We will discuss how integrals are used to solve problems and provide examples of real-world applications. We will also examine the potential for future developments in the field of integral applications.

By the end of this report, readers will have a better understanding of the importance of integrals in various fields and will appreciate the versatility of this fundamental mathematical concept.

# 2. Applications

## a. Physics

Integrals can be used to calculate quantities such as area, volume, and work, and are an essential tool for modeling physical systems and analyzing their behavior. In the field of physics, integrals are particularly important for understanding the relationship between force and potential energy in conservative systems. By integrating the force function, we can obtain the corresponding potential energy function, and use it to gain insights into the behavior of the system.

For example, suppose we have a system where $F(x) = -kx$, where $k$ is a constant. We can find the corresponding potential energy function by integrating the force with respect to position:

$$U(x) = -\int F(x)dx = -\int (-kx)dx = \frac{1}{2}kx^2 + C$$

where $C$ is a constant of integration

We can plot both the force and potential energy functions to compare them. Here is an example plot for the case where $k = 1$ and $C = 0$:

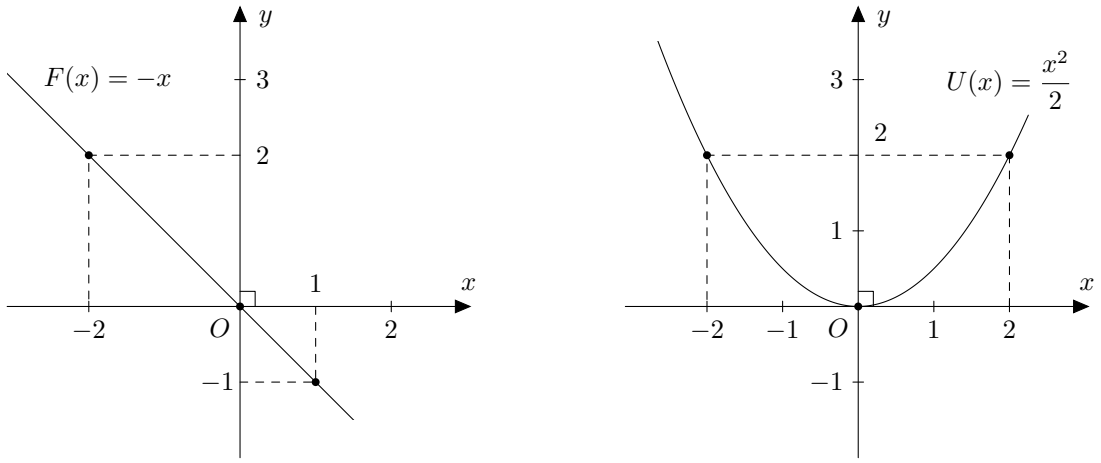


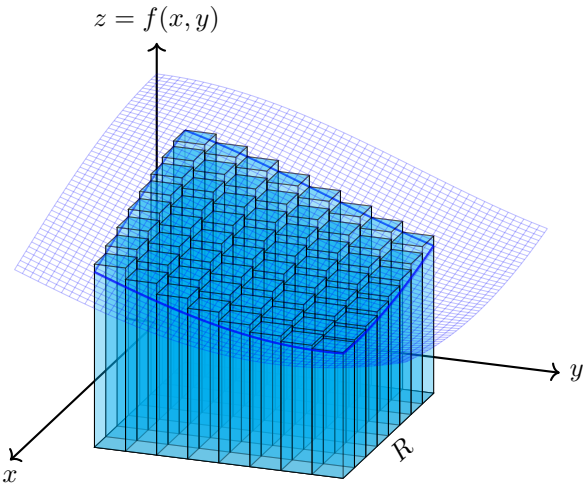

Figure 3: Force function and potential energy function for a one-dimensional system.

As we can see from the plot, the force function is linear and the potential energy function is quadratic. The potential energy function is minimum at $x = 0$, which is the equilibrium position where the force is zero.

This example demonstrates how integrals can be used to analyze the relationship between force and potential energy in a physical system. By knowing the force, we can integrate to find the corresponding potential energy function and gain insights into the behavior of the system.

## b. Computer Graphics

Double integrals are a powerful tool in mathematics and have numerous applications in various fields, including computer graphics. One of these applications is in the calculation of volumes under curved surfaces. In computer graphics, double integrals can be used to calculate the volume under a 3D curve that is defined by a function of two variables, such as $z = f(x, y)$. This calculation is essential in creating 3D models, simulations, and animations. By using double integrals, computer graphic designers and animators can accurately calculate the volume under the surface of a 3D model, which is critical in creating realistic and accurate visual representations.




Volume under $f(x, y)$ bounded by a given region $R$ defined by $x \in [a, b]$ and $y \in [c, d]$ is

$$\iint_R f(x, y)dA = \int_a^b \int_c^d f(x, y)\, dy\, dx$$

where $dA = dx\, dy$

Figure 4: Double integral virtualization

To enhance the exploration and understanding of various functions, we have developed a Python code that allows users to input functions in terms of $x$ and $y$, as well as define the corresponding domains. Running the program enables users to visualize the function's graph and wireframe volume, providing valuable insights into its properties. This interactive simulation offers a dynamic and visually engaging platform to analyze and gain a deeper comprehension of diverse functions. Below is the code:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # User input for f(x) function
5  try:
6      fxy = input("Enter the function f(x, y): ")
7      f = lambda x, y: eval(fxy)
8  except Exception as e:
9      print("Error:", e)
10     print("Please enter a valid function expression.")
11
12 def simulate_double_integral():
13     """Simulates a double integral and plots the result"""
14     while True:
15         try:
16             x_min, x_max = map(float, input("Enter x-min and x-max
                ↪  (space-separated): ").split())
17             y_min, y_max = map(float, input("Enter y-min and y-max
                ↪  (space-separated): ").split())
```

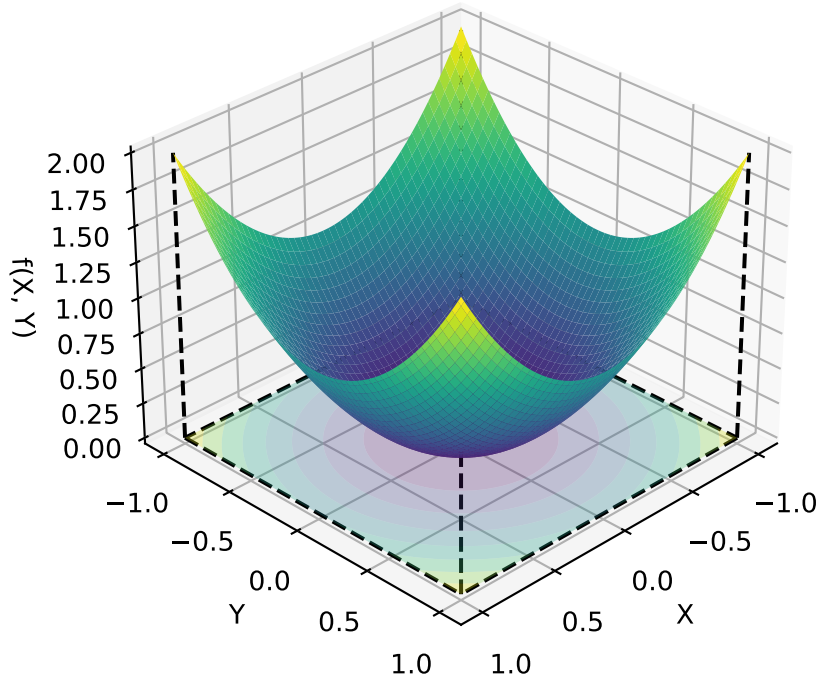
```
            break
        except ValueError:
            print("Invalid input. Please enter valid numeric values.")

    num_points = 100  # Number of points in each dimension

    x = np.linspace(x_min, x_max, num_points)
    y = np.linspace(y_min, y_max, num_points)
    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)

    # Calculate the volume under the surface using the trapezoidal rule
    while True:
        try:
            volume = np.trapz(np.trapz(Z, dx=(x_max - x_min) / (num_points -
            ↪  1), axis=0), dx=(y_max - y_min) / (num_points - 1))
            break
        except Exception as e:
            print("Error:", e)
            print("Please enter a valid function expression.")
            Z = f(X, Y)

    # Plotting the result
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # 3D surface plot
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=1.0)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('f(X, Y)')

    # Set the z-axis limits based on function values
    z_min, z_max = np.min(Z), np.max(Z)
    if z_min >= 0:
        ax.set_zlim(0, z_max)
    elif z_max <= 0:
        ax.set_zlim(z_min, 0)
    else:
        ax.set_zlim(z_min, z_max)

    # Rotate the camera to a different view
    ax.view_init(elev=30, azim=45)

    # Projection on XY-plane
    ax.contourf(X, Y, Z, zdir='z', offset=0, cmap='viridis', alpha=0.3)

    # Draw dashed lines from surface to base
    ax.plot([x_min, x_min], [y_min, y_min], [0, Z[0, 0]], 'k--')
    ax.plot([x_max, x_max], [y_min, y_min], [0, Z[0, -1]], 'k--')
    ax.plot([x_min, x_min], [y_max, y_max], [0, Z[-1, 0]], 'k--')
    ax.plot([x_max, x_max], [y_max, y_max], [0, Z[-1, -1]], 'k--')

    # Draw boundary dashed lines
    ax.plot(x, y_min * np.ones_like(x), zs=0, linestyle='dashed',
    ↪  color='black')
    ax.plot(x, y_max * np.ones_like(x), zs=0, linestyle='dashed',
    ↪  color='black')
    ax.plot(x_min * np.ones_like(y), y, zs=0, linestyle='dashed',
    ↪  color='black')
    ax.plot(x_max * np.ones_like(y), y, zs=0, linestyle='dashed',
    ↪  color='black')

    plt.title(f"Volume: {volume:.6f}")
    plt.show()
```

```
78
79  # Run the simulation
80  simulate_double_integral()
```

Input and Result:

```
Enter the function f(x, y): x**2+y**2
Enter x-min and x-max (space-separated): -1 1
Enter y-min and y-max (space-separated): -1 1
```

Volume: 2.667211




Moreover, the more advanced tool, triple integrals, is even more powerful than double integrals, as they can calculate volumes of 3D objects that are more complex. Triple integrals can integrate over a region in 3D space, whereas double integrals can integrate only on a 2D plane. In summary, both double and triple integrals are powerful mathematical tools with numerous real-world applications, especially in fields such as computer graphics.

## c. Signal Processing

Integrals are a fundamental mathematical tool used in many engineering applications, particularly in the analysis of signals and systems. One important application of integrals is in the field of audio engineering, where they are used to analyze and manipulate sound signals. The Fourier Transform is a mathematical technique that uses integrals to extract frequency components of a signal, and it is widely used in audio engineering.

The Fourier Transform is a widely used algorithm for calculating the Fourier Transform of discrete-time signals. It is used to plot the amplitude of sound waves as a function of frequency. Engineers use Fourier Transform to sample the sound signal at regular intervals,

then apply the algorithm to the signal, which calculates the amplitude of the signal at different frequencies. The resulting plot shows the amplitude of the signal at each frequency, allowing engineers to identify the different frequency components of the sound wave.

Continuous-time Fourier Transform have this basic formula:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt$$

where $f$ is the frequency, $x(t)$ is signal with respect to time

To simplify the process, we utilize an external Python package in our Python code to simulate the sound wave using the Fast Fourier Transform:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.fftpack import fft
4  from scipy.io import wavfile
5
6  sr, data = wavfile.read('sample.wav')
7  print(data.shape, sr)
8  signal = data[:sr,0]
9  Signal = fft(signal)
10 fig, (axt, axf) = plt.subplots(2, 1, constrained_layout=1, figsize=(11.8,3))
11 axt.plot(signal, lw=0.15) ; axt.grid(1)
12 axf.plot(np.abs(Signal[:sr//2]), lw=0.15) ; axf.grid(1)
13 plt.show()
```

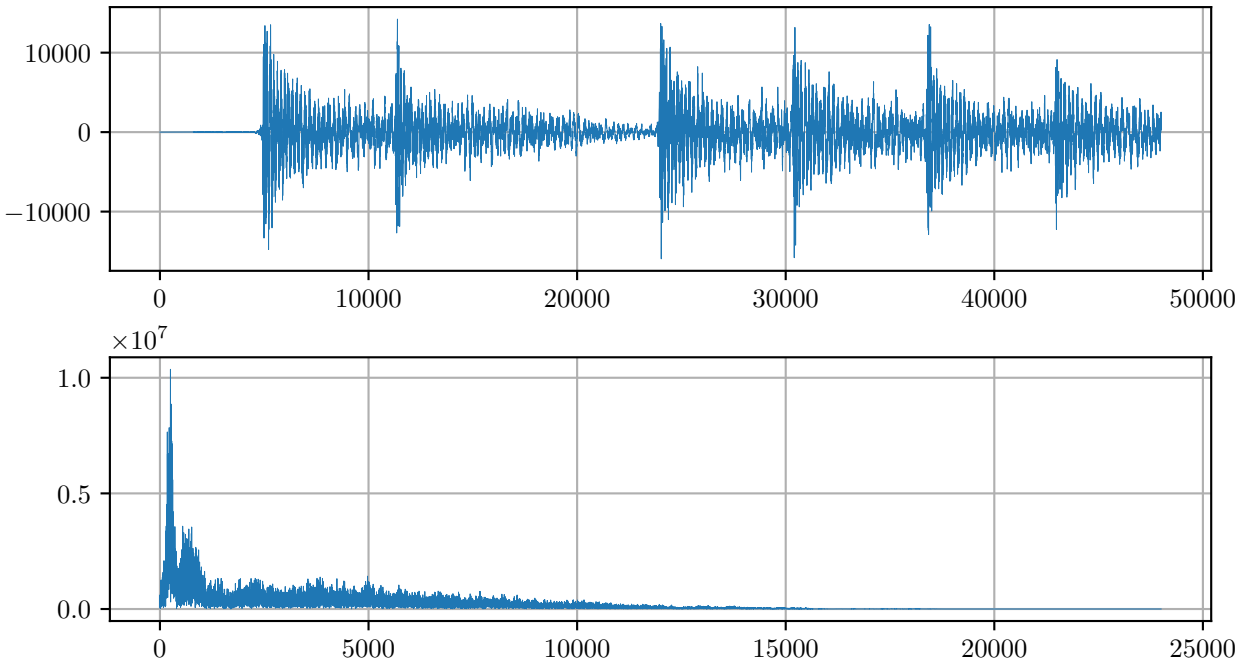



Figure 5: A PGF histogram from `matplotlib`

Fourier Transform has many practical applications in engineering, such as in audio processing, communication systems, and control systems. For example, engineers may use

Fourier Transform to analyze the frequency content of a speech signal and filter out noise or other unwanted components. They may also use Fourier Transform to design audio filters or equalizers that selectively amplify or attenuate specific frequency ranges of a sound signal. In summary, the use of integrals, particularly in the analysis of signals and systems using Fourier Transform, is a powerful technique that has many practical applications in audio and signal processing.

## d. Sculpting

Integrals are commonly used in engineering to design structures with complex shapes, where calculating the amount of material needed to construct a structure with a variable cross-sectional area is critical. This is particularly important in industries such as aerospace and automotive engineering, where weight and strength are key factors.

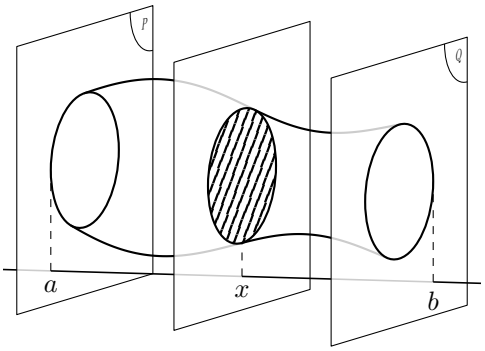



Figure 6: Screen projection of a surface of revolution.

Similarly, integrals are used in pottery sculpting to create pots with specific volumes. By integrating the cross-sectional area function over the height of the pot, potters can determine the total volume of clay needed to create the pot, which allows them to create pots with specific volumes for cooking and serving purposes.

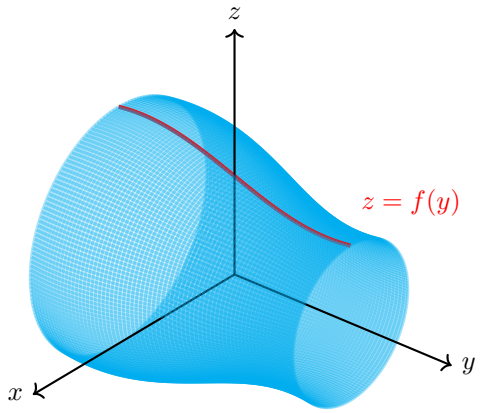



Figure 7: Surface of revolution

Surface area around the y-axis is

$$\int_a^b \left(2\pi f(y)\sqrt{1+(f'(y))^2}\right)\, dy$$

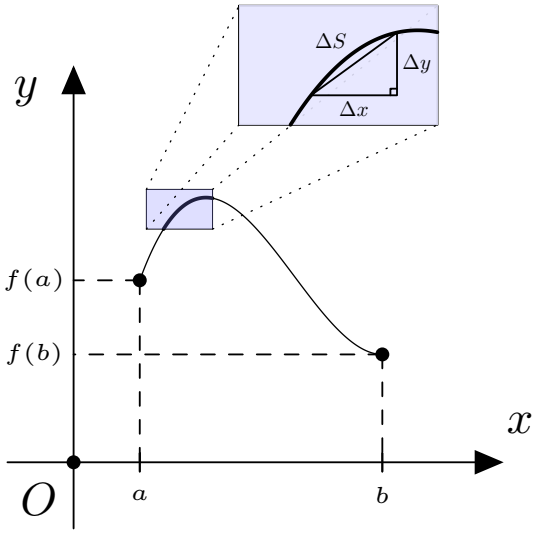
with the domain $y \in [a, b]$

Overall, integrals play a critical role in these fields by enabling the calculation of volumes and amounts of material needed to construct a structure or create a pot with a variable

cross-sectional area. The ability to calculate volumes accurately is essential, as it ensures that structures and pots are both functional and efficient.

## e. Civil Engineering

The application of integral in calculating the length of a curve is particularly relevant in civil engineering when designing roads and bridges. In designing these structures, curves are often used to navigate around obstacles or to follow natural contours of the landscape. By calculating the length of these curves, engineers can accurately determine the amount of materials needed and ensure the safety and efficiency of the structure.




From the graph, we can approximate arc length of the function:

$$\sum \Delta S = \sum \lim_{\Delta x \to 0} \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2}$$

Arc length of a function $y = f(x)$ with $x \in [a, b]$ is

$$\int_a^b \sqrt{1 + (f'(x))^2}\, dx$$

Figure 8: Arc length of a function

By accurately calculating the length of curves in road and bridge design, engineers can also minimize the impact on the environment. By following natural contours of the landscape, curves can be designed to avoid environmentally sensitive areas such as wetlands or forests.

Overall, the application of calculating the length of curves is crucial in civil engineering, particularly in designing roads and bridges. Accurately calculating the length of these curves can ensure the safety and efficiency of the structure, minimize environmental impact, and reduce unnecessary costs.

# III Differential Equation

## 1. Theoretical Basis

A differential equation is a mathematical equation that relates an unknown function to its derivatives. It involves the concept of rates of change and describes the relationship between a function and its derivatives. Differential equations play a fundamental role in various fields of science, engineering, and mathematics, providing a powerful tool for modeling and understanding dynamic systems.

In general, a differential equation can be written in the form:

$$g(x) = f_n(x)\frac{d^n y}{dx^n} + \cdots + f_1(x)\frac{dy}{dx} + f_0(x)y$$

such that:

$$y(x_0) = y_0,\ y'(x_0) = y_0',\ y''(x_0) = y_0'', \ldots$$

For any nonzero $f_n(x)$, if $\{f_0, f_1, \ldots\}$ and $g$ are continuous on some interval containing $x_0, y$ is unique and exists.

Solving a differential equation means finding the function $y$ that satisfies the equation. This typically involves finding an expression for $y$ in terms of $x$ or determining its behavior. Solutions to differential equations can be obtained through analytical methods, such as separation of variables, integration, or using specific techniques for different types of equations. Numerical methods and computer simulations are also widely used to approximate solutions for complex differential equations.

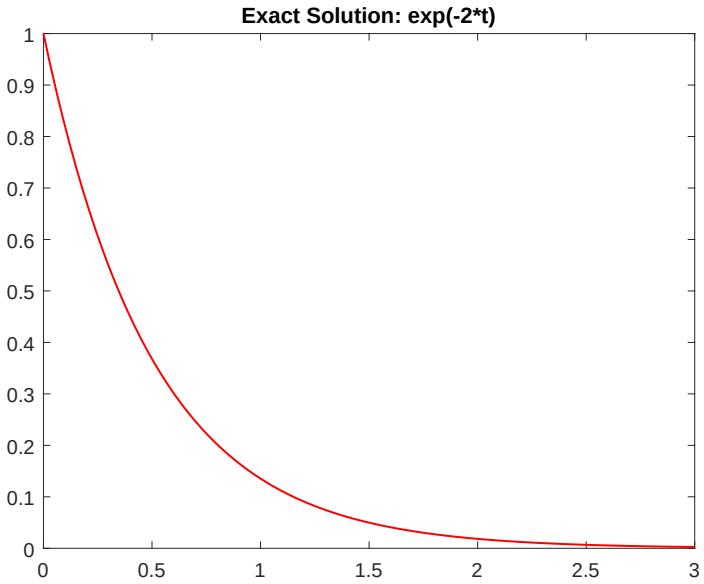
This is the Matlab script that we have developed to calculate the exact solution while inputting the differential equation:

```
clc; clearvars;

% User input for the derivative function dy/dt
dydt_func = input("Enter the derivative function dy/dt: ", 's');

% User input for the initial condition y0
y0 = input("Enter the initial condition y0: ");

% Define the time range for simulation
t_min = 0;
t_max = input("Enter the duration: ");

% Generate the time values
t_exact = linspace(t_min, t_max);

% Convert the derivative function to a function handle
dydt = str2func(['@(t, y)', dydt_func]);

% Solve the ODE using ode45
[t, y] = ode45(@(t, y) dydt(t, y), t_exact, y0);

```

```
% Plot the numerical solution
plot(t, y, 'r-', 'LineWidth', 1, 'MarkerSize', 20)

% Define the symbolic variable
syms y(t)

% Solve the ODE symbolically using dsolve
exact_sol = dsolve(diff(y,t) == dydt(t, y), y(0) == y0);

% Set the plot title as the exact solution
title(['Exact Solution: ', char(exact_sol)]);

```

Input and Result:

```
Enter the derivative function dy/dt: -2*y
Enter the initial condition y0: 1
Enter the duration: 3
```




Differential equations have broad applications in diverse areas, including physics, engineering, biology, economics, and more. They provide a powerful framework for modeling dynamic systems, predicting behavior, and understanding the underlying principles governing various phenomena. By studying and analyzing differential equations, we can gain insights into the behavior and dynamics of real-world systems, making them an essential tool in scientific research and problem-solving.

# 2. Applications

## a. Growth and Decay

Growth and decay are fundamental concepts that occur in various natural and man-made systems. Whether it's the growth of a population, the decay of a radioactive substance, or the depreciation of an asset, understanding and modeling these processes are essential in many fields. Ordinary differential equations provide a powerful framework for studying growth and decay phenomena by expressing the rate of change of a quantity as a function of its current value.

By solving these equations, we can analyze the dynamics, predict future behavior, and make informed decisions. The application of ordinary differential equations to growth and decay processes enables us to gain insights into the underlying mechanisms and optimize outcomes in fields such as biology, economics, physics, and more.

Let's consider an example of population growth or decay using an ordinary differential equation. Suppose we have a population of bacteria, and we want to model the growth or decay of this population over time.

Let $N(t)$ represent the population size at time $t$. If we assume that the rate of change of the population is proportional to the population itself, we can express this relationship using the following ordinary differential equation:

$$\frac{dN}{dt} = kN(t)$$

where $k$ is constant that determines the growth or decay rate

To solve this ordinary differential equation, we can separate the variables, integrate both sides, and then simplify by exponentiating both sides:

$$\begin{aligned}\frac{1}{N(t)}dN = kdt \Rightarrow \int \frac{1}{N}dN &= \int kdt \\ \Leftrightarrow \ln|N(t)| &= kt + C^* \\ \Leftrightarrow N(t) &= \pm e^{C^*}e^{kt} = Ce^{kt}\end{aligned}$$

The equation $N(t) = Ce^{kt}$ represents the general solution to the ordinary differential equation governing population growth or decay. The constant $C$ represents the initial population size at $t = 0$.

By specifying the initial condition $N(0) = N_0$, we can determine the particular solution for a specific scenario:

$$N(t) = N_0e^{kt}$$

This equation allows us to predict the population size $N(t)$ at any given time t based on the initial population size $N_0$ and the growth or decay rate represented by the constant k.

In summary, ordinary differential equations provide a mathematical framework to model growth and decay processes in various systems, including population dynamics. By solving these equations, we can understand and predict the behavior of the system over time, making them valuable tools in the field of mathematical modeling and analysis.

## b. Pursuit Curve

Pursuit curves find practical applications in various real-life scenarios, demonstrating the relevance of ordinary differential equations. One such application is in the field of aerospace engineering, particularly in the tracking and interception of flying objects. Consider a scenario where a rocket or a missile aims to intercept a target moving through space. By modeling the motion of the target and the rocket using differential equations, engineers can determine the trajectory that the rocket should follow to intercept the target successfully. The pursuit curve, derived from the solution of the differential equation, provides the desired path for the rocket to pursue and eventually reach the target.

Pursuit curves find applications in robotics and autonomous navigation, such as tracking moving targets for autonomous drones and self-driving cars. By using differential equations, engineers can develop algorithms to calculate optimal trajectories and maintain desired positions relative to the target, enabling accurate and efficient tracking.

In the field of animal behavior and predator-prey dynamics, pursuit curves offer insights into predator-prey interactions. Differential equations are used to model the movements of predators and prey, allowing ecologists and biologists to understand the strategies employed by predators and the evasive maneuvers adopted by prey. The pursuit curve derived from these equations sheds light on the dynamics of natural predator-prey relationships.

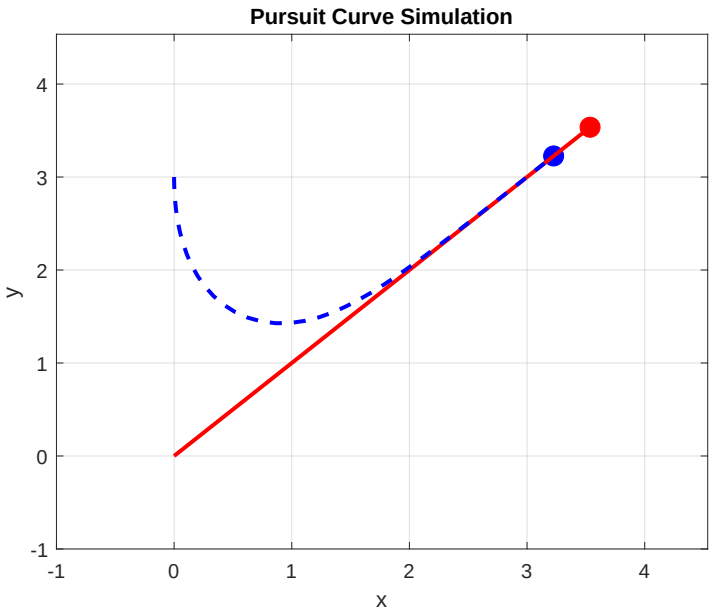
To gain a clearer understanding of this topic, we have developed a Matlab program that simulates the trajectories of two objects. By specifying a unit velocity vector for object 1 and the duration of the simulation, the program allows the second object to pursue and chase the first object, forming a pursuit curve. Below is the code:

```
clc; clearvars;

% User input for the positions of points A and B
posA = input("Enter the position of point A [x, y]: ");
x1 = posA(1);
y1 = posA(2);
% User input for the vector A
A = input("Enter the vector A [x, y]: ");
A = A / norm(A);  % Convert A to a unit vector

posB = input("Enter the position of point B [x, y]: ");
x2 = posB(1);
y2 = posB(2);

% User input for the duration of the simulation
duration = input("Enter the duration of the simulation: ");

% Define the velocity of point A
v = 1;

```

```
% Set the time span
tspan = [0 duration];

% Define the initial conditions
initial_condition = [x1, y1, x2, y2];

% Solve the system of ordinary differential equations
[t, positions] = ode45(@(t, y) pursuitCurve(t, y, v, A), tspan,
↪  initial_condition);

% Extract the positions of points A and B
x1 = positions(:, 1);
y1 = positions(:, 2);
x2 = positions(:, 3);
y2 = positions(:, 4);

% Animate the pursuit curve
figure;
for i = 1:length(t)
    plot(x1(i), y1(i), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');  %
↪  Point A
    hold on;
    plot(x2(i), y2(i), 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'b');  %
↪  Point B
    plot(x1(1:i), y1(1:i), 'r-', 'LineWidth', 2);  % Trajectory of point A
    plot(x2(1:i), y2(1:i), 'b--', 'LineWidth', 2);  % Trajectory of point B
    xlim([min([x1; x2])-1, max([x1; x2])+1]);
    ylim([min([y1; y2])-1, max([y1; y2])+1]);
    xlabel('x');
    ylabel('y');
    title('Pursuit Curve Simulation');
    grid on;
    hold off;
    pause(0.1);
end

% Function defining the system of ordinary differential equations
function dydt = pursuitCurve(~, y, v, A)
    x1 = y(1);
    y1 = y(2);
    x2 = y(3);
    y2 = y(4);

    dx1dt = v * A(1);
    dy1dt = v * A(2);
    dx2dt = (x1 - x2) / norm([x1 - x2, y1 - y2]);
    dy2dt = (y1 - y2) / norm([x1 - x2, y1 - y2]);

    dydt = [dx1dt; dy1dt; dx2dt; dy2dt];
end
```

Input and Result:

```
Enter the position of point A [x, y]: [0,0]
Enter the vector A [x, y]: [1,1]
Enter the position of point B [x, y]: [0,3]
Enter the duration of the simulation: 5
```




In summary, pursuit curves derived from ordinary differential equations have practical applications in diverse fields such as aerospace engineering, robotics, and ecology. By formulating pursuit problems as differential equations, engineers and scientists can devise optimal strategies for tracking and intercepting targets, design autonomous systems capable of following moving objects, and gain insights into predator-prey dynamics. These real-life applications demonstrate the significance of differential equations and the pursuit curve concept in solving complex problems and understanding dynamic systems.

## c. Disease Prediction

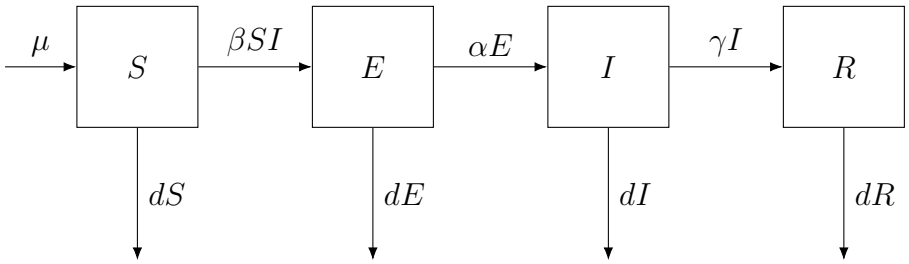
The $SEIR$ model is an extension of the $SIR$ model[4] that incorporates an additional compartment to represent individuals who are Exposed ($E$) to the disease but not yet infectious. This model is particularly useful for studying diseases with an incubation period, such as COVID-19. The $SEIR$ model is based on a system of ordinary differential equations that describe the transitions between the compartments.

Let's consider an example of using the $SEIR$ model to make predictions about COVID-19. The following set of differential equations represents the $SEIR$ model:

$$\frac{dS}{dt} = \mu - dS - \beta SI$$
$$\frac{dE}{dt} = \beta SI - (d + \alpha)E$$
$$\frac{dI}{dt} = \alpha E - (d + \gamma)I$$
$$\frac{dR}{dt} = \gamma I - dR$$

[4]Compartmental models in epidemiology: `https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology`

Here, $S(t)$, $E(t)$, $I(t)$, and $R(t)$ represent the number of susceptible, exposed, infected, and recovered individuals at time $t$, respectively. $\beta$ represents the effective contact rate, $\alpha$ represents the rate at which exposed individuals become infectious, and $\gamma$ represents the recovery rate. Here is the flow chart for $SEIR$ model:




The $SEIR$ model is utilized to forecast the spread of diseases such as COVID-19. Estimating the model's parameters, including $\beta$, $\alpha$, and $\gamma$, is essential for achieving accurate predictions. Numerical integration techniques and solver functions in programming languages like Python are employed to solve the aforementioned model's differential equations.

Simulating the $SEIR$ model with estimated parameter values allows us to generate predictions for the future course of the outbreak. These predictions offer insights into factors such as the number of infections, the timing and duration of the peak, and the impact of interventions like vaccination and quarantine measures.

It is important to consider the limitations of the $SEIR$ model. Prediction accuracy depends on factors like data quality, model assumptions, and parameter uncertainties. Regular updates and refinement of the model with new data are necessary to improve prediction accuracy.

In summary, the $SEIR$ model is a valuable tool for predicting the spread of diseases like COVID-19. By estimating model parameters and numerically solving the differential equations, we can generate predictions that aid in understanding disease dynamics, evaluating control strategies, and informing public health interventions.

# IV References


[1] Falstad. *CircuitJS*. URL: https://www.falstad.com/circuit/circuitjs.html.

[2] Wikipedia. *Differential equation*. URL: https://en.wikipedia.org/wiki/Differential_equation.

[3] Wikipedia. *Riemann sum*. URL: https://en.wikipedia.org/wiki/Riemann_sum.

[4] Wikipedia. *Derivative*. URL: https://en.wikipedia.org/wiki/Derivative.

[5] Wikipedia. *Compartmental models in epidemiology*. URL: https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology.