**Derivative Calculator**

**SUBMITTED BY**

Abdul Rafay Zahid (65540)

Emaz Ali Khan (65566)

Sumaika Asif (65651)

**SUPERVISED BY**

ARAIB AQEEL



REPORT SUBMITTED TO THE FACULTY OF COMPUTING, KARACHI INSTITUTE OF ECONOMICS AND TECHNOLOGY, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE

SPRING 2024

# INTRODUCTION

**ABOUT PROGRAM**

This project presents a versatile Derivative Calculator designed to assist users in both symbolic and numerical differentiation of functions. The calculator offers an intuitive menu-driven interface, allowing users to seamlessly choose between different modes of differentiation and handle a variety of mathematical expressions. This project is made on **python** programming language.

**OBJECTIVE**

The Derivative Calculator was developed to offer a versatile and accurate tool for both symbolic and numerical differentiation. Key objectives included providing precise symbolic differentiation for a wide range of functions, including common trigonometric functions with hardcoded derivatives. Numerical differentiation using the central difference method was also a priority, ensuring accurate approximations of derivatives at specific points.

The calculator supports the application of the quotient rule, allowing users to input numerator and denominator functions and compute their derivatives both symbolically and numerically. An intuitive, menu-driven interface was designed to enhance user experience, with clear error messages to assist in troubleshooting.

Educational value was emphasized, providing step-by-step differentiation processes to aid in understanding. Robust error handling and broad compatibility with various mathematical functions and expressions were also critical objectives, making the calculator a comprehensive and reliable tool for students, educators, and professionals.

## FEATURES

**SYMBOLIC DIFFERENTIATION**

Symbolic differentiation involves computing the exact derivative of a function in terms of symbolic expressions. This feature is particularly useful for obtaining precise mathematical derivatives, which can be further analyzed or simplified. The calculator supports: Differentiation of general functions input by the user. Utilization of hardcoded derivatives for common trigonometric functions such as sin(x), cos(x), tan(x), and others. Application of the quotient rule, allowing users to specify numerator (u(x)) and denominator (v(x)) functions and automatically compute their derivative.

**NUMERICAL DIFFERENTIATION**

Numerical differentiation provides an approximate derivative value at a specific point, based on the chosen step size. This method is essential for situations where a symbolic derivative is either difficult to obtain or unnecessary. The calculator's numerical differentiation feature includes: Evaluation of user-defined functions using standard numerical libraries like NumPy. Implementation of the central difference method for precise numerical differentiation. Handling of the quotient rule for numerical differentiation, enabling users to input numerator and denominator functions and compute the derivative at a specified point.

* **US****AGE**

The Derivative Calculator is designed to be user-friendly and interactive, prompting users to input the required information at each step. It allows for flexibility in choosing between symbolic and numerical methods, catering to different user needs, whether for exact analytical solutions or practical numerical approximations.

* **ERROR HANDLING**

The calculator includes robust error handling mechanisms to guide users through potential issues such as syntax errors or parsing difficulties. Clear and concise error messages ensure that users can quickly correct their inputs and achieve the desired results. Overall, this Derivative Calculator serves as a comprehensive tool for students, educators, and professionals in mathematics, engineering, and related fields, providing a reliable and efficient way to perform differentiation tasks.

* **CONCLUSION**

The Derivative Calculator is a versatile and user-friendly tool designed for both symbolic and numerical differentiation. It caters to diverse needs by supporting general functions and the quotient rule. With hardcoded derivatives for common trigonometric functions and robust error handling, it ensures accurate and efficient computation. Ideal for students, educators, and professionals, this calculator simplifies differentiation tasks, enhancing understanding and application in mathematics and related fields.

* **Code**

import numpy as np

import sympy as sp

def symbolic\_differentiation():

# Ask the user to choose between a general function and specifying u(x) and v(x) for the quotient rule

choice = input("Do you want to differentiate a general function or use the quotient rule? (general/quotient): ").strip().lower()

if choice == 'general':

# Ask the user to enter the function they want to differentiate

function\_str = input("Enter the function you want to differentiate (e.g., 'tan(x)', 'sin(x)', 'cos(x)', 'x\*\*2', 'x \* sin(x)', 'x / cos(x)', etc.): ")

# Define the symbolic variable x

x = sp.symbols('x')

# Hardcoded derivatives for trigonometric functions

hardcoded\_derivatives = {

'tan(x)': sp.sec(x)\*\*2,

'cot(x)': -sp.csc(x)\*\*2,

'csc(x)': -sp.csc(x) \* sp.cot(x),

'sec(x)': sp.sec(x) \* sp.tan(x),

'sin(x)': sp.cos(x),

'cos(x)': -sp.sin(x)

}

try:

# Check if the input function is a hardcoded trigonometric function

if function\_str in hardcoded\_derivatives:

# Use the hardcoded derivative for the function

derivative = hardcoded\_derivatives[function\_str]

print(f"The symbolic derivative of {function\_str} is: {derivative}")

else:

# Parse the input function using `sympy.sympify`

function = sp.sympify(function\_str)

# Compute the symbolic derivative

derivative = sp.diff(function, x)

# Simplify the derivative

simplified\_derivative = sp.simplify(derivative)

# Print the result

print(f"The simplified symbolic derivative of {function\_str} is: {simplified\_derivative}")

except sp.SympifyError:

# Handle the case where the function can't be parsed

print(f"Error: Could not parse the function '{function\_str}'. Please check the syntax and try again.")

except Exception as e:

# Handle other potential exceptions and print an error message

print(f"An error occurred while computing the derivative: {e}")

elif choice == 'quotient':

# Ask the user to enter u(x) and v(x) for the quotient rule

u\_str = input("Enter the function u(x) for the quotient rule: ")

v\_str = input("Enter the function v(x) for the quotient rule: ")

try:

# Parse the input functions using `sympy.sympify`

u = sp.sympify(u\_str)

v = sp.sympify(v\_str)

# Define the symbolic variable x

x = sp.symbols('x')

# Compute the derivatives of u(x) and v(x)

u\_prime = sp.diff(u, x)

v\_prime = sp.diff(v, x)

# Apply the quotient rule: (u'v - uv') / v^2

derivative = (u\_prime \* v - u \* v\_prime) / v\*\*2

# Simplify the derivative

simplified\_derivative = sp.simplify(derivative)

# Print the result

print(f"The simplified symbolic derivative of ({u\_str}) / ({v\_str}) is: {simplified\_derivative}")

except sp.SympifyError:

# Handle the case where the function can't be parsed

print(f"Error: Could not parse the functions '{u\_str}' or '{v\_str}'. Please check the syntax and try again.")

except Exception as e:

# Handle other potential exceptions and print an error message

print(f"An error occurred while computing the derivative: {e}")

else:

print("Invalid choice. Please try again.")

def numerical\_differentiation():

# Ask the user to choose between a general function and specifying u(x) and v(x) for the quotient rule

choice = input("Do you want to differentiate a general function or use the quotient rule? (general/quotient): ").strip().lower()

if choice == 'general':

# Ask the user to enter the function

function\_str = input("Enter the function you want to differentiate (e.g., 'np.sin(x)', 'np.cos(x)', 'np.tan(x)', 'x\*\*2', etc.): ")

# Define the symbolic variable x

x = sp.symbols('x')

try:

# Define the function using eval and lambda to support np functions

function = eval(f"lambda x: {function\_str}", {"np": np, "sqrt": np.sqrt, "sin": np.sin, "cos": np.cos, "tan": np.tan, "exp": np.exp, "log": np.log, "pi": np.pi})

# Compute the numerical derivative using the central difference method

x\_value = float(input("Enter the value of x at which you want to compute the derivative: "))

h = float(input("Enter the step size h (small value like 0.001): "))

# Compute the numerical derivative using the central difference method

numerical\_derivative = (function(x\_value + h) - function(x\_value - h)) / (2 \* h)

# Print the numerical derivative

print(f"The numerical derivative of {function\_str} at x = {x\_value} is: {numerical\_derivative}")

except Exception as e:

print(f"An error occurred while computing the derivative: {e}")

elif choice == 'quotient':

# Ask the user to enter u(x) and v(x) for the quotient rule

u\_str = input("Enter the function u(x) for the quotient rule: ")

v\_str = input("Enter the function v(x) for the quotient rule: ")

try:

# Parse the input functions using `sympy.sympify`

u = sp.sympify(u\_str)

v = sp.sympify(v\_str)

# Define the symbolic variable x

x = sp.symbols('x')

# Compute the numerical derivatives of u(x) and v(x)

u\_prime = sp.diff(u, x)

v\_prime = sp.diff(v, x)

# Define lambda functions for u(x) and v(x)

u\_func = eval(f"lambda x: {u\_str}", {"np": np, "sqrt": np.sqrt, "sin": np.sin, "cos": np.cos, "tan": np.tan, "exp": np.exp, "log": np.log, "pi": np.pi})

v\_func = eval(f"lambda x: {v\_str}", {"np": np, "sqrt": np.sqrt, "sin": np.sin, "cos": np.cos, "tan": np.tan, "exp": np.exp, "log": np.log, "pi": np.pi})

# Compute the numerical derivative of u(x) and v(x) at a specific x value

x\_value = float(input("Enter the value of x at which you want to compute the derivative: "))

u\_value = u\_func(x\_value)

v\_value = v\_func(x\_value)

# Apply the quotient rule: (u'v - uv') / v^2

numerical\_derivative = (u\_prime.evalf(subs={x: x\_value}) \* v\_value - u\_value \* v\_prime.evalf(subs={x: x\_value})) / v\_value\*\*2

# Print the numerical derivative

print(f"The numerical derivative of ({u\_str}) / ({v\_str}) at x = {x\_value} is: {numerical\_derivative}")

except Exception as e:

print(f"An error occurred while computing the derivative: {e}")

else:

print("Invalid choice. Please try again.")

def main():

while True:

print("\n-----------------------------DERIVATIVE CALCULATOR----------------------------------")

print("\nDifferentiation Menu:")

print("1. Symbolic Differentiation")

print("2. Numerical Differentiation")

print("3. Exit")

choice = input("Enter your choice (1, 2, or 3): ")

if choice == '1':

symbolic\_differentiation()

elif choice == '2':

numerical\_differentiation()

elif choice == '3':

print("Exiting the program.")

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()





