**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**FACULTY OF APPLIED SCIENCE**

**🙞···☼···🙜**

**CALCULUS 2**

**PROJECT REPORT**

**Instructor:** Dr. Le Thai Thanh

**Class:** CC05

**Group:** 06

Ho Chi Minh City, May 25th, 2024

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**FACULTY OF APPLIED SCIENCE**

**🙞···☼···🙜**

**CALCULUS 2**

**PROJECT REPORT**

|  |  |  |
| --- | --- | --- |
| **No.** | **Student Name** | **Student ID** |
| 1 | Bùi Đăng Khoa | 2152668 |
| 2 | Hồ Minh Huy | 2352378 |
| 3 | Trần Đình Quang | 2352977 |

Ho Chi Minh City, May 25th, 2024

**Table of Contents**

[**QUESTION 01:** 3](#_Toc167021620)

[**Question 1a:** 3](#_Toc167021621)

[**Question 1b:** 6](#_Toc167021622)

[**Question 1c:** 6](#_Toc167021623)

[**QUESTION 02:** 8](#_Toc167021626)

[**1. Theory:** 8](#_Toc167021627)

[**2. Coding section:** 8](#_Toc167021631)

[**QUESTION 03:** 13](#_Toc167021632)

[**1.** **Introduction** 13](#_Toc167021633)

[**2.** **Types of surface integrals** 14](#_Toc167021634)

[**3.** **Applications of surface integrals** 14](#_Toc167021635)

[**4.** **Practical problems** 15](#_Toc167021636)

# **QUESTION 01:**

# **QUESTION 02:**

## **1. Theory**

**1.1 Parametric equations definition**

In mathematics, a parametric equation defines a group of quantities as functions of one or more independent variables called parameters. Parametric equations are commonly used to express the coordinates of the points that make up a geometric object such as a curve or surface, called a parametric curve and parametric surface, respectively. In such cases, the equations are collectively called a parametric representation, or parametric system, or parameterization (alternatively spelled as parametrisation) of the object.

For example, the equations

form a parametric representation of the unit circle, where t is the parameter: A point (x, y) is on the unit circle if and only if there is a value of t such that these two equations generate that point. Sometimes the parametric equations for the individual scalar output variables are combined into a single parametric equation in vectors:

**1.2 Find normal vector**

1. **Define the parametric equations**: Suppose the surface is given by the parametric equations
2. **Compute the partial derivatives**:

Find the partial derivative with respect to :

Find the partial derivative of r with respect to :

1. **Compute the cross product**: The normal vector is given by the cross product of :

The cross product of two vectors (a1,a2,a3) and (b1,b2,b3) is computed as:

**1.2 Find tangent plane equation**

1. **Define the parametric equations**: Suppose the surface is given by the parametric equations
2. **Compute the partial derivatives**:

Find the partial derivative with respect to :

Find the partial derivative of r with respect to :

1. **Compute the cross product**: The normal vector is given by the cross product of :
2. **Find the equation of the tangent plane:** The equation of the tangent plane at a point on the surface can be written as:

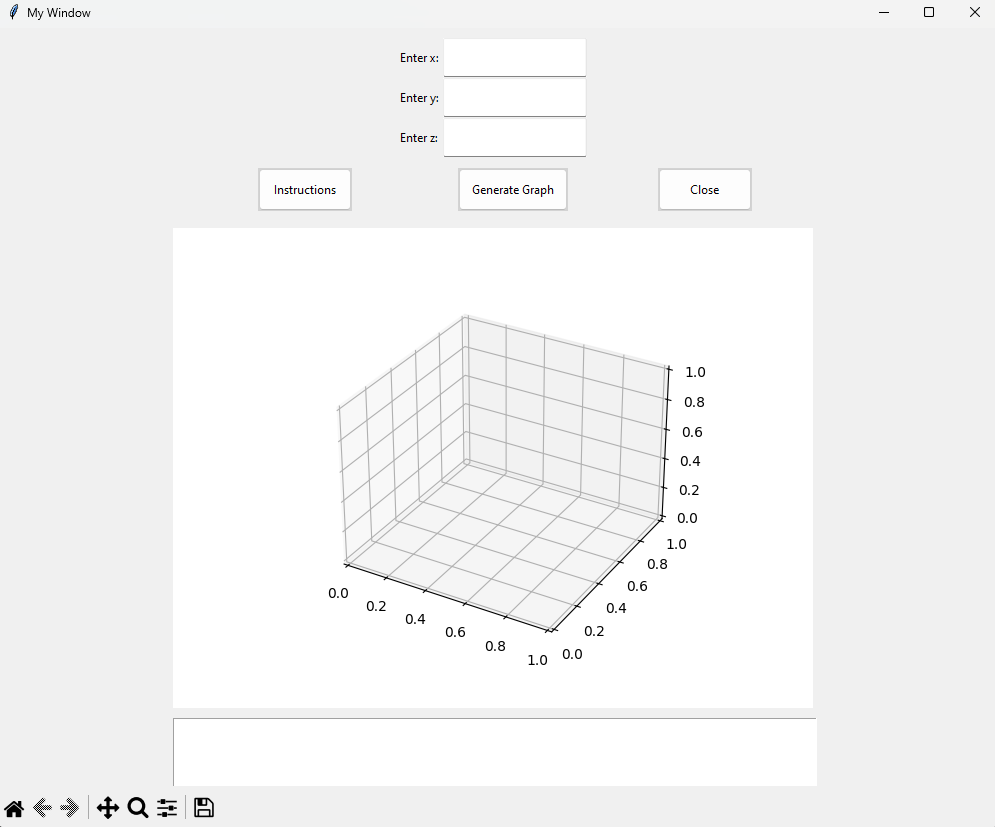
This can be expanded to:

Where

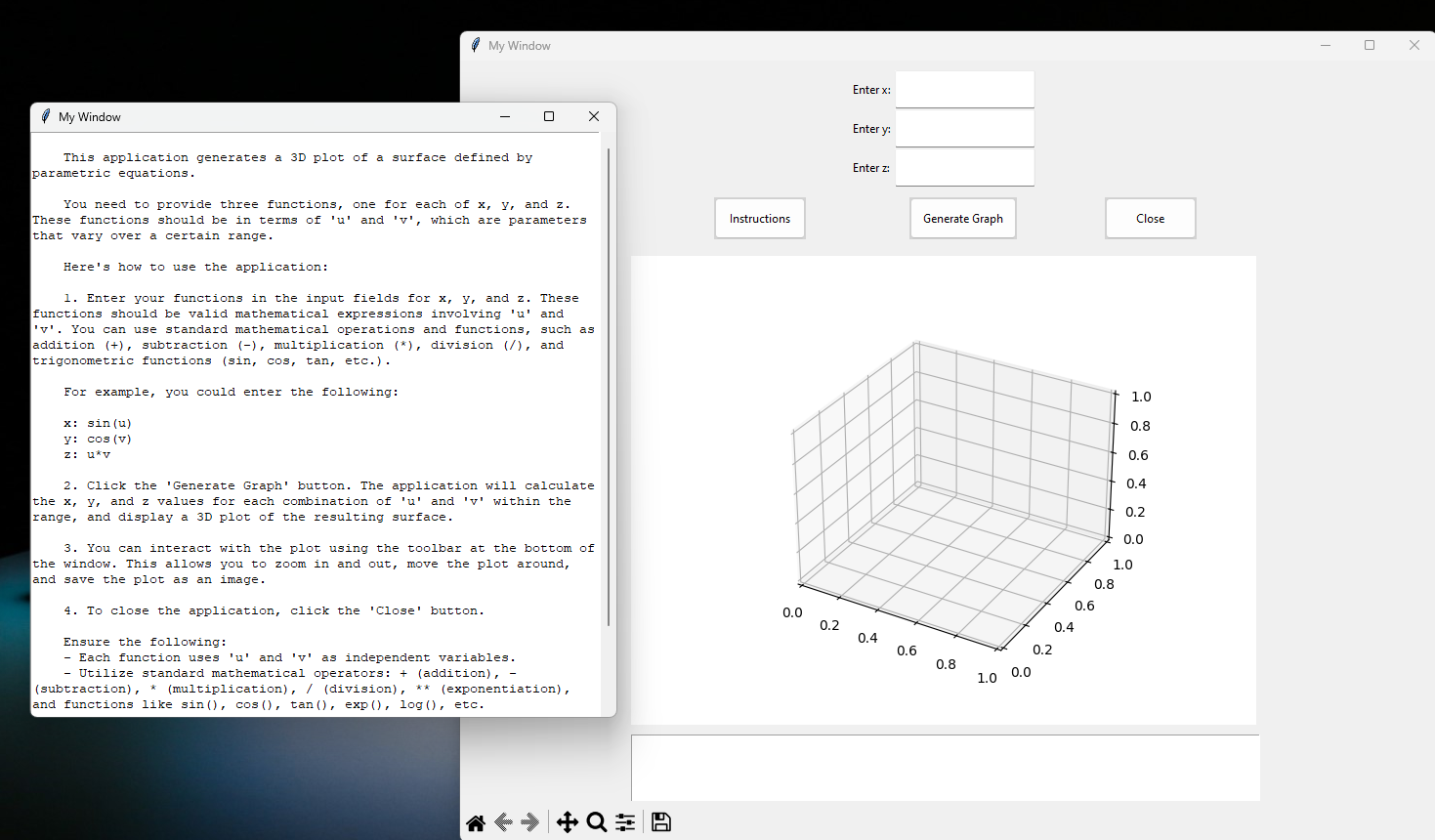
## **2. Coding section:**

Our application integrates a user-friendly interface. Users can enter a parametric equation then the system will plot and output the normal vector and tangent plane results of that equation. In addition, users can interact with the graph through the toolbar in the bottom corner.

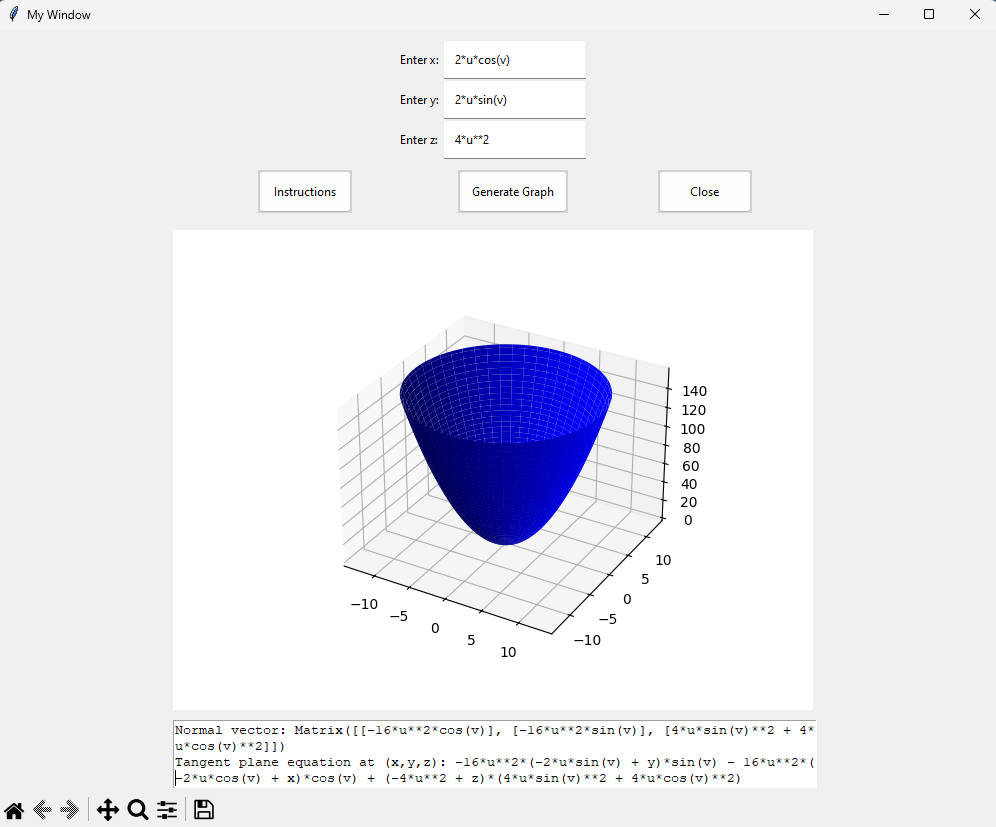
**2.1 When starting the program**



**2.1 When opening the instruction**



**2.1 When entering and running parametric equations**



**2.1 Source code**

import tkinter as tk

from tkinter import ttk  # Import ttk module

from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

from sympy import symbols, lambdify

from tkinter import messagebox

from sympy import diff, Matrix

from sympy import sympify

def create\_window():

    window = tk.Tk()  # Create a new window

    window.title("My Window")  # Set the title of the window

    window.geometry("1000x800")  # Set the size of the window

    # Create a style

    style = ttk.Style()

    # Configure the style for the input fields

    style.configure("TEntry", foreground="black", background="white", padding=10)

    # Configure the style for the buttons

    style.configure("TButton", foreground="black", background="lightgray", padding=10)

    # Create labels for x, y, and z

    x\_label = ttk.Label(window, text="Enter x:", style="TLabel")

    y\_label = ttk.Label(window, text="Enter y:", style="TLabel")

    z\_label = ttk.Label(window, text="Enter z:", style="TLabel")

    # Place the labels at custom positions

    x\_label.place(x=400, y=20)

    y\_label.place(x=400, y=60)

    z\_label.place(x=400, y=100)

    # Create input fields for x, y, and z

    x\_entry = ttk.Entry(window, style="TEntry")  # Use the style

    y\_entry = ttk.Entry(window, style="TEntry")

    z\_entry = ttk.Entry(window, style="TEntry")

    # Place the input fields at custom positions

    x\_entry.place(x=445, y=10)

    y\_entry.place(x=445, y=50)

    z\_entry.place(x=445, y=90)

    # Create a button that will display instructions when clicked

    instruction\_button = ttk.Button(window, text="Instructions", command=show\_instructions, style="TButton")  # Use the style

    instruction\_button.place(x=260, y=140)

    # Create a button that will generate the graph when clicked

    button = ttk.Button(window, text="Generate Graph", command=lambda: generate\_graph(x\_entry.get(), y\_entry.get(), z\_entry.get(), ax, canvas, result\_text), style="TButton")  # Use the style

    button.place(x=460, y=140)

    # Create a new figure for the plot

    fig = Figure()

    ax = fig.add\_subplot(111, projection='3d')

    # Create a canvas and add the plot to it

    canvas = FigureCanvasTkAgg(fig, master=window)

    canvas.get\_tk\_widget().place(x=100, y=200)

    # Create a text widget for displaying the normal vector and tangent plane

    result\_text = tk.Text(window, height=5, width=80)

    result\_text.place(x=175, y=690)

    # Add navigation toolbar (for zooming, moving the plot, etc.)

    toolbar = NavigationToolbar2Tk(canvas, window)

    toolbar.update()

    canvas.get\_tk\_widget().place(x=175, y=200)

    # Create a button to close the application

    close\_button = ttk.Button(window, text="Close", command=window.destroy, style="TButton")  # Use the style

    close\_button.place(x=660, y=140)

    return window

    #Create the instructions

def show\_instructions():

    instructions = """

    This application generates a 3D plot of a surface defined by parametric equations.

    You need to provide three functions, one for each of x, y, and z. These functions should be in terms of 'u' and 'v', which are parameters that vary over a certain range.

    Here's how to use the application:

    1. Enter your functions in the input fields for x, y, and z. These functions should be valid mathematical expressions involving 'u' and 'v'. You can use standard mathematical operations and functions, such as addition (+), subtraction (-), multiplication (\*), division (/), and trigonometric functions (sin, cos, tan, etc.).

    For example, you could enter the following:

    x: sin(u)

    y: cos(v)

    z: u\*v

    2. Click the 'Generate Graph' button. The application will calculate the x, y, and z values for each combination of 'u' and 'v' within the range, and display a 3D plot of the resulting surface.

    3. You can interact with the plot using the toolbar at the bottom of the window. This allows you to zoom in and out, move the plot around, and save the plot as an image.

    4. To close the application, click the 'Close' button.

    Ensure the following:

    - Each function uses 'u' and 'v' as independent variables.

    - Utilize standard mathematical operators: + (addition), - (subtraction), \* (multiplication), / (division), \*\* (exponentiation), and functions like sin(), cos(), tan(), exp(), log(), etc.

    - Avoid using constants directly; instead, express them as mathematical expressions involving 'u' and 'v'.

    - Verify that each function is valid and can be evaluated over the specified ranges.

    - Ensure your expressions are well-formed and free of syntax errors.

    """

    # Create a top-level window

    instruction\_window = tk.Toplevel()

    instruction\_window.geometry("600x600")  # Set the size of the window

    # Create a scrollbar

    scrollbar = tk.Scrollbar(instruction\_window)

    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

    # Create a text widget and add the instructions to it

    text\_widget = tk.Text(instruction\_window, wrap=tk.WORD, yscrollcommand=scrollbar.set)

    text\_widget.insert(tk.END, instructions)

    text\_widget.pack(side=tk.LEFT, fill=tk.BOTH)

    # Connect the scrollbar to the text widget

    scrollbar.config(command=text\_widget.yview)

def generate\_graph(x\_eq, y\_eq, z\_eq, ax, canvas, result\_text):

    try:

        # Clear the axes for the new plot

        ax.clear()

        # Create symbols

        u, v = symbols('u v')

        # Parse the user input into sympy expressions

        x\_expr = sympify(x\_eq)

        y\_expr = sympify(y\_eq)

        z\_expr = sympify(z\_eq)

        # Evaluate the expressions for the range of u and v

        u\_values = np.linspace(0, 2\*np.pi, 100)

        v\_values = np.linspace(0, 2\*np.pi, 100)

        x = np.array([[float(x\_expr.evalf(subs={u:u\_val, v:v\_val})) for u\_val in u\_values] for v\_val in v\_values])

        y = np.array([[float(y\_expr.evalf(subs={u:u\_val, v:v\_val})) for u\_val in u\_values] for v\_val in v\_values])

        z = np.array([[float(z\_expr.evalf(subs={u:u\_val, v:v\_val})) for u\_val in u\_values] for v\_val in v\_values])

        # Plot the surface

        ax.plot\_surface(x, y, z, color='b')

        # Calculate the partial derivatives

        x\_u = diff(x\_expr, u)

        x\_v = diff(x\_expr, v)

        y\_u = diff(y\_expr, u)

        y\_v = diff(y\_expr, v)

        z\_u = diff(z\_expr, u)

        z\_v = diff(z\_expr, v)

        # Calculate the normal vector

        normal\_vector = Matrix([x\_u, y\_u, z\_u]).cross(Matrix([x\_v, y\_v, z\_v]))

        # Calculate the equation of the tangent plane

        tangent\_plane\_eq = normal\_vector.dot(Matrix([symbols('x') - x\_expr, symbols('y') - y\_expr, symbols('z') - z\_expr]))

        # Update the text widget with the normal vector and tangent plane

        result\_text.delete('1.0', tk.END)

        result\_text.insert(tk.END, f"Normal vector: {normal\_vector}\n")

        result\_text.insert(tk.END, f"Tangent plane equation at (x,y,z): {tangent\_plane\_eq}")

        # Redraw the canvas

        canvas.draw()

    except Exception as e:

        messagebox.showerror("Error", str(e))

# Create a window and start the event loop

window = create\_window()

window.mainloop()

# **QUESTION 03:**