PRESENTATION

Dwarak A - EE24BTECH11019

January 15, 2025

1. Problem Statement
2. Theoretical Solution
   - Function and its Derivatives
   - Critical Points
   - Minima, Maxima Condition
   - Minimum, Maximum Values
3. Computational Solution
   - Gradient Descent and Ascent
   - Parameters and Results
4. C Code
5. Python Code
6. Plot

Problem Statement

Find the maximum and minimum if any for the function

$$f(x) = \sin(2x) + 5 \tag{2.1}$$

Function and its Derivatives

$$f(x) = \sin(2x) + 5 \tag{3.1}$$

$$f'(x) = 2\cos(2x) \tag{3.2}$$

$$f''(x) = -4\sin(2x) \tag{3.3}$$

Critical Points

To find the critical points, we set the derivative equal to zero:

$$f'(x) = 0 \tag{3.4}$$

$$\implies 2\cos(2x) = 0 \tag{3.5}$$

$$\implies 2x = \frac{\pi}{2} + n\pi, \quad n \in \mathbb{Z} \tag{3.6}$$

$$\implies x = \frac{\pi}{4} + \frac{n}{2}\pi, \quad n \in \mathbb{Z} \tag{3.7}$$

Local Minima, Maxima Condition

Let A be the set of critical points,
For local minima:

$$f''(x) > 0 \tag{3.8}$$
$$\implies -4\sin(2x) > 0, \quad x \in A \tag{3.9}$$
$$\implies \sin(2x) < 0 \tag{3.10}$$
$$\implies x_{min} = \frac{\pi}{4} + \frac{2m-1}{2}\pi, \quad m \in \mathbb{Z} \tag{3.11}$$

For local maxima:

$$f''(x) < 0 \tag{3.12}$$
$$\implies -4\sin(2x) < 0, \quad x \in A \tag{3.13}$$
$$\implies \sin(2x) > 0 \tag{3.14}$$
$$\implies x_{max} = \frac{\pi}{4} + m\pi, \quad m \in \mathbb{Z} \tag{3.15}$$

Minimum, Maximum Values

Minimum value of $f(x)$ using (3.11):

$$f(x_{min}) = \sin(2x_{min}) + 5 \tag{3.16}$$
$$= \sin\left(\frac{\pi}{2} + (2m-1)\pi\right), \quad m \in \mathbb{Z} \tag{3.17}$$
$$= -1 + 5 \tag{3.18}$$
$$= 4 \tag{3.19}$$

Maximum value of $f(x)$ using (3.15):

$$f(x_{max}) = \sin(2x_{max}) + 5 \tag{3.20}$$
$$= \sin\left(\frac{\pi}{2} + (2m)\pi\right), \quad m \in \mathbb{Z} \tag{3.21}$$
$$= 1 + 5 \tag{3.22}$$
$$= 6 \tag{3.23}$$

Gradient Descent and Ascent

For the derivative:

$$f'(x_n) = 2\cos(2x_n) \tag{4.1}$$

Gradient Descent to find the local minimum:

$$x_{n+1} = x_n - \eta f'(x_n) \tag{4.2}$$

$$x_{n+1} = x_n - 2\eta\cos(2x_n) \tag{4.3}$$

Gradient Ascent to find the local maximum:

$$x_{n+1} = x_n + \eta f'(x_n) \tag{4.4}$$

$$x_{n+1} = x_n + 2\eta\cos(2x_n) \tag{4.5}$$

Here, η represents the learning rate. The learning rate in gradient descent is a parameter that determines the step size taken in the direction of the negative gradient (for minimization) or the positive gradient (for maximization) during each iteration.

$$0 < \eta < 1$$

Parameters and Results

Assuming the following parameters:

$$\eta = 0.1 \quad \text{(learning rate)} \tag{4.6}$$

$$\text{tolerance} = 1e - 6 \tag{4.7}$$

$$x_0 = 0.0 \quad \text{(initial guess)} \tag{4.8}$$

The computed results are:

$$x_{\text{min}} = -0.7853968861361207, \quad y_{\text{min}} = 4.000000000003263 \tag{4.9}$$

$$x_{\text{max}} = 0.7853968861361207, \quad y_{\text{max}} = 5.999999999996737 \tag{4.10}$$

C Code I

```c
#include <math.h>
#include <stdlib.h>

// f(x) = sin(2x) + 5
double func(double x) {
    return sin(2 * x) + 5;
}

// f'(x) = 2cos(2x)
double func_deriv(double x) {
    return 2 * cos(2 * x);
}

// Generate points for given function
void points_gen(double x_start, double x_end, double* x_vals, double*
     y_vals, double h, double (*func)(double)) {
    int i = 0;
    for (double x_i = x_start; x_i < x_end; x_i += h) {
      x_vals[i] = x_i;
```

C Code II

```c
        y_vals[i] = func(x_i);
        i++;
    }
}

void gradient_method(double x0, double learning_rate, double tol, double
     extremum[2], double (*func)(double), double (*func_deriv)(double))
    {
  double x1 = x0 + learning_rate * func_deriv(x0);

  while (fabs(x1 - x0) >= tol) {
    x0 = x1;
    x1 = x0 + learning_rate * func_deriv(x0);
  }

  extremum[0] = x1;
  extremum[1] = func(x1);
}
```

Python Code I

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

# Load the shared object file
gradient_lib = ctypes.CDLL('./gradient.so')

# Define argument and return types for func and func_deriv
gradient_lib.func.argtypes = [ctypes.c_double]  # double
    func(double x)
gradient_lib.func.restype = ctypes.c_double

gradient_lib.func_deriv.argtypes = [ctypes.c_double]  # double
    func_deriv(double x)
gradient_lib.func_deriv.restype = ctypes.c_double

gradient_lib.points_gen.argtypes = [
    ctypes.c_double,  # double x_start
```

Python Code II

```
    ctypes.c_double,  # double x_end
    ctypes.POINTER(ctypes.c_double),  # double* x_vals
    ctypes.POINTER(ctypes.c_double),  # double* y_vals
    ctypes.c_double,  # double h
    ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double)  # double (*func)(double)
]
gradient_lib.points_gen.restype = None

gradient_lib.gradient_method.argtypes = [
    ctypes.c_double,  # double x0
    ctypes.c_double,  # double learning_rate
    ctypes.c_double,  # double tol
    ctypes.POINTER(ctypes.c_double),  # double extremum[2]
    ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double),  # function pointer for func
    ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double)   # function pointer for func_deriv
```

Python Code III

```python
]
gradient_lib.gradient_method.restype = None

# Create function pointers using ctypes from the shared library
    functions directly
FuncType = ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double)

# Get function pointers from the shared library
func_ptr = FuncType(gradient_lib.func)
func_deriv_ptr = FuncType(gradient_lib.func_deriv)

# Parameters
x_start = -2.0       # Start of range
x_end = 2.0          # End of range
h = 0.01             # Step size for trapezoidal integration
num_points = int((x_end - x_start) / h) # Points for trapezoidal
    integration

```

Python Code IV

```python
x0 = 0.0
learning_rate = 0.1
tol = 1e-6

# Allocate memory for the output arrays
x_vals = (ctypes.c_double * num_points)()
y_vals = (ctypes.c_double * num_points)()

# Allocate memory for the extremum
min_coords = (ctypes.c_double * 2)()
max_coords = (ctypes.c_double * 2)()

# Generate points using func pointer
gradient_lib.points_gen(x_start, x_end, x_vals, y_vals, h,
    func_ptr)

# Gradient Descent
```

Python Code V

```
gradient_lib.gradient_method(x0, -learning_rate, tol,
    min_coords, func_ptr, func_deriv_ptr)

# Gradient Ascent
gradient_lib.gradient_method(x0, learning_rate, tol, max_coords,
    func_ptr, func_deriv_ptr)

# Convert the results to NumPy arrays
x_vals = np.array(x_vals)
y_vals = np.array(y_vals)

# Print the minimum found by gradient descent
print(f"Minimum found at x = {min_coords[0]}, y =
    {min_coords[1]}")
print(f"Maximum found at x = {max_coords[0]}, y =
    {max_coords[1]}")

# Plot the function and the gradient descent result
```

Python Code VI

```
plt.plot(x_vals, y_vals, label='function')
plt.scatter(min_coords[0], min_coords[1], color='red',
    label='minimum')
plt.scatter(max_coords[0], max_coords[1], color='green',
    label='maximum')

plt.legend()
plt.axis('equal')
plt.grid()

plt.savefig("../figs/plot.png")
plt.show()
```

Plot

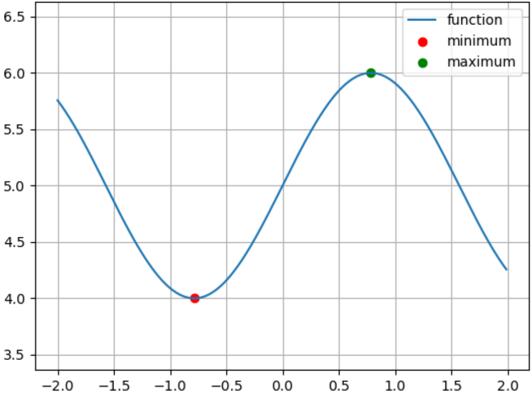

Figure: Plot of local maximum and minimum