S

$C++$ *Function Writing*

AMS595 Project 4

SUBMITTED BY

TAIWO ALARE

115360301

Table of Contents

Question 1 2
- Key Steps: 3
- Result 3

Question 2 3
- Key Steps: 5

Question 3 5
- Compilation 6
- Result 6

Question 4 (Bonus) 6

Question 1

Writing pi_appromiation using the given equations.

```
#include <iostream>
#include <cmath>

#define _USE_MATH_DEFINES  // I define this to make sure M_PI is available
#include <cmath>

// I define a struct to store results
struct PiResults {
    double approx;
    double error;
};

// I write a function to approximate pi and calculate the error
PiResults pi_approx(int N) {
    double sum = 0.0;
    double delta_x = 1.0 / N;

    for (int k = 1; k <= N; ++k) {
        double x_k = (double)k / N;
        double f_x_k = 1.0 / (1.0 + x_k * x_k);
        double f_x_k_minus_1 = 1.0 / (1.0 + (x_k - delta_x) * (x_k - delta_x));

        sum += (f_x_k + f_x_k_minus_1) / 2.0;
    }

    double result = sum * delta_x;

    // I calculate the absolute error
    double pi_value = M_PI;
    double absolute_error = std::abs(result - pi_value);

    // I create and return a PiResults struct
    return {result, absolute_error};
}

int main() {
    int N = 1000;

    // I call the pi_approx function
    PiResults results = pi_approx(N);

    // I print the results
```

```cpp
    std::cout << "Approximated value of pi: " << results.approx << std::endl;
    std::cout << "Absolute error: " << results.error << std::endl;

    return 0;
}
```

Key Steps:

I. I defined a struct to store results.
II. I wrote a function to approximate π using the given equations in the question.
III. I calculated the truncation error due to the approximation.
IV. I calculated the absolute error.
V. I printed the result for N= 1000.

Result

```
Approximated value of pi: 3.14159
Absolute error: 2.22045e-16
```

Question 2

```cpp
#include <iostream>
#include <cmath>
#include <vector>

// I define a struct to store results
struct PiResults {
    double approx;  // Approximated value of pi
    double error;   // Absolute error compared to built-in value of pi
};

// I  write function to approximate pi and calculate the error
PiResults pi_approx(int N) {
    double sum = 0.0;
    double delta_x = 1.0 / N;

    // I use the trapezoidal rule to approximate the integral
    for (int k = 1; k <= N; ++k) {
        double x_k = (double)k / N;
        double f_x_k = 1.0 / (1.0 + x_k * x_k);
        double f_x_k_minus_1 = 1.0 / (1.0 + (x_k - delta_x) * (x_k - delta_x));
```

```cpp
        sum += (f_x_k + f_x_k_minus_1) / 2.0;
    }

    double result = sum * delta_x;

    // I calculate the absolute error
    double pi_value = M_PI;
    double absolute_error = std::abs(result - pi_value);

    // I create and return a PiResults struct
    return {result, absolute_error};
}

// I write function to compute approximations for each input value
PiResults* approximations(const std::vector<int>& intervals) {
    // Allocate memory for results array
    PiResults* results = new PiResults[intervals.size()];

    // I compute approximations for each interval
    for (size_t i = 0; i < intervals.size(); ++i) {
        results[i] = pi_approx(intervals[i]);
    }

    return results;
}

int main() {
    // Example usage
    std::vector<int> interval_values = {1000, 2000, 5000};
    PiResults* approx_results = approximations(interval_values);

    // Access and use the results as needed
    for (size_t i = 0; i < interval_values.size(); ++i) {
        std::cout << "Approximation for " << interval_values[i] << " intervals: "
<< approx_results[i].approx << std::endl;
    }

    // I free the allocated memory
    delete[] approx_results;

    return 0;
}
```

Key Steps:

I. I defined a struct to store results.
II. I wrote a function to approximate π using the given equations in the question.
III. I calculated the truncation error due to the approximation.
IV. I calculated the absolute error.

Question 3

Creating HW2Main.cpp file

```
#include <iostream>
#include <vector>
#include "approximations.h"
#include "pi_approx.h"

int main() {
    // Q1: Print the approximation and error for N = 10000 using Q1
    int N_q1 = 10000;
    PiResults result_q1 = pi_approx(N_q1);
    std::cout << "Q1 - Approximation for N = " << N_q1 << ": " <<
result_q1.approx << std::endl;
    std::cout << "Q1 - Absolute error: " << result_q1.error << std::endl;

    // Q2: Create a vector with elements {101, 102, 107} as an input
    std::vector<int> intervals_q2 = {101, 102, 107};
    // Print out the elements of the array from Q2 using this vector
    PiResults* results_q2 = approximations(intervals_q2);
    for (size_t i = 0; i < intervals_q2.size(); ++i) {
        std::cout << "Q2 - Approximation for N = " << intervals_q2[i] << ": "
<< results_q2[i].approx << std::endl;
    }

    // Don't forget to free the allocated memory
    delete[] results_q2;

    return 0;
}
```

Compilation

Writing the MakeFile to compile the code.

```
CC = g++
CFLAGS = -std=c++11 -Wall
TARGET = HW2main
OBJS = HW2main.o approximations.o pi_approx.o

all: $(TARGET)

$(TARGET): $(OBJS)
	$(CC) $(CFLAGS) -o $(TARGET) $(OBJS)

HW2main.o: HW2main.cpp approximations.h pi_approx.h
	$(CC) $(CFLAGS) -c HW2main.cpp

approximations.o: approximations.cpp approximations.h pi_approx.h
	$(CC) $(CFLAGS) -c approximations.cpp

pi_approx.o: pi_approx.cpp pi_approx.h
	$(CC) $(CFLAGS) -c pi_approx.cpp

clean:
	rm -f $(TARGET) $(OBJS)
```

Result

```
Approximated value of pi: 3.14159
Absolute error: 2.22045e-16
```

Question 4 (Bonus)

Writing pi_appromation function that allows user to input the maximum permitted approximation error.

```
#define PI_APPROX2_H


#include <iostream>
#include <cmath>

#define _USE_MATH_DEFINES
#include <cmath>

struct PiResults {
    double approx;
```

```
    double error;
};

PiResults pi_approx(double max_error) {
    int N = 1;  // Start with a small number of intervals

    double result, absolute_error;
    do {
        double sum = 0.0;
        double delta_x = 1.0 / N;

        for (int k = 1; k <= N; ++k) {
            double x_k = (double)k / N;
            double f_x_k = 1.0 / (1.0 + x_k * x_k);
            double f_x_k_minus_1 = 1.0 / (1.0 + (x_k - delta_x) * (x_k -
delta_x));

            sum += (f_x_k + f_x_k_minus_1) / 2.0;
        }

        result = sum * delta_x;

        // Calculate the absolute error
        double pi_value = M_PI;
        absolute_error = std::abs(result - pi_value);

        // Double the number of intervals for the next iteration
        N *= 2;

    } while (absolute_error > max_error);

    return {result, absolute_error};
}

int main() {
    double max_error;

    // Get the maximum acceptable error from the user
    std::cout << "Enter the maximum acceptable error: ";
    std::cin >> max_error;

    // Call the pi_approx function
    PiResults results = pi_approx(max_error);

    // Print the results
```

```
    std::cout << "Approximated value of pi: " << results.approx << std::endl;
    std::cout << "Absolute error: " << results.error << std::endl;

    return 0;
}
```