Monte Carlo Method & Simpson's Rule

Jakub Straupisz, Zuzanna Chmielecka

January 2024

1 Introduction

First of all to calculate the approximate value of a polynomial, using Monte Carlo Method, on a given interval, we have to calculate the approximate value of the integral, of the given polynomial.

2 Labels

Interval:
$[a, b]$

Polynomial:
$f(x)$

Number of trials:
n

Formula:
$$\int_a^b f(x)dx = (\frac{b-a}{n}) * \sum_{i=0}^{n} f(x_i)$$

3 Instructions

The program starts off, writing out the names of methods which are going to be used to calculate the integrals - Monte Carlo and Simpson method (for clarification how those work see further points - shortly explaining its purpose. Next, the program ask for the following input:

- degree of the polynomial
- lower limit of the integral which is being calculated (a)
- upper limit of the integral which is being calculated (b)
- number of samples on the the interval [a,b] on which the calculations will be done
- *arrayCoefficients - which is our pointer to array with all coeficcients,
- a, b, n - which are our lower, upper limits, and number of samples.

With the given data the program returns the value of the interval using the Monte Carlo and Simpson's method and the estimated error for Monte Carlo. Additionally, it writes all this data to a file named "data.txt" and creates to plots, one using the result from the Monte Carlo method and the other using the results from the Simpson's method.

4 Main

In main() to compile our code, we include all headers and all typical libraries we need. First we ask user to input

- degree - which is our polynomial degree,
- *arrayCoefficients - which is our pointer to array with all coeficcients,
- a, b, n - which are our lower, upper limits, and number of samples.

Then we create four arrays (TabX, TabY), (tabX, tabY) - which are our arrays for Monte Carlo, and Simpsons, with random x, and f(x) for both in range of [a, b].

Thanks to having these arrays we can calculate approximate values of integral using the Monte Carlo Method and Simpson's Rule. Accordingly monte_carlo(a, b, TabY, n), simpsons(a, b, tabY, n, tablica, st).

Once we have calculated our values, we extract the entire specifications into a separate file.
And in yet another files we write out our x and f(x) for both methods. writeDataToFile_monte(TabX, TabY, n) and writeDataToFile_simpson(tabX, TabY, n)

Once we have performed all these steps we draw the graphs of both methods using drawPlot_monte() and drawPlot_simpson().

```c
// MONTE CARLO PROJECT

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "plot.h"
#include "input.h"
#include "monte_carlo.h"
#include "simpson.h"
#include "func.h"
#include "data.h"
#include "calculation_error.h"


int main()
{
    printf("\nMonte Carlo and Simpson Integration, based on Riemann Sum and Thomas Simpson\n\n");
    printf("Ten program wylicza wartosc calek numerycznych wielomianow za pomoca metody Monte Carlo oraz metody Simpsona.\nProsze wpisac nastepujace dane.\n\n");


    double* TabX;
    double* TabY;
    double* tabX;
    double* tabY;

    int degree = degreeOfPolynomial();
    int *arrayCoefficients = arrayWithCoefficients(degree);

    int a = lowerLimit();
    int b = upperLimit();
    int n = numOfSamples();


    createTabXandTabYMonte(&TabX, &TabY, a, b, n, arrayCoefficients, degree);
    createTabXandTabYSimpson(&tabX, &tabY, a, b, n, arrayCoefficients, degree);


    printf("Przyblizona wartosc calki numerycznej metoda Monte Carlo z wielomianu to: %f\n\n", monteCarlo(a, b, TabY, n));
    printf("Przyblizona wartosc calki numerycznej metoda Simpsona z wielomianu to: %f\n\n", simpson(a, b, n, arrayCoefficients, degree));

    double integrationMonte = monteCarlo(a, b, TabY, n);
    double integrationSimpson = simpson(a, b, n, arrayCoefficients, degree);
    double errorMonte = error(TabY, n, integrationMonte);

    writePolynomialToFile(degree, arrayCoefficients, a, b, n, integrationMonte, integrationSimpson, errorMonte);

    writeDataToFileMonte(TabX, TabY, n);
    drawPlotMonte();

    writeDataToFileSimpson(tabX, tabY, n);
    drawPlotSimpson();

    printf("Bl d obliczeniowy wynosi:%f\n\n", error(TabY, n, integrationMonte));

    free(TabX);
    free(TabY);
    free(tabX);
    free(tabY);

    return 0;
}
```

Listing 1: C main

5 Functions

Here we made a function which helps us calculate f(x) using Monte Carlo Method and Simpson's Rule. We are using x, tab - our coeficcients, stopien - degree of polynomial. To calculate f(x) we use Horner's diagram.

```c
#include <stdio.h>
#include <stdlib.h>
#include "func.h"

double f(double x, int tab[], int stopien) {
    double wynik = tab[0];

    for (int i = 1; i <= stopien; i++)
        wynik = wynik * x + tab[i];

    return wynik;
}
```

Listing 2: C functions

6 Input

Here are some usefull functions we made to interact with the user.

- degreeOfPolynomial() - takes the input of the degree of a polynomial from the user.
- *arrayWithCoefficients() - takes the input of the rates of change of a polynomial from the user
- lowerLimit(), upperlimit() - takes the input of the down and top limit of the interval on which the integration is being calculated from the user
- numOfSamples() - takes the input of the number of values for which the integration is being calculated from the user

```c
#include <stdio.h>
#include <stdlib.h>
#include "input.h"
#include <stdbool.h>
//wciecia
int degreeOfPolynomial(){
    int degree;
    while (true) {
        printf("Podaj stopien (liczbe do ktorej podnoszona jest najwieksza potega) wielomianu:
 ");
        if (scanf("%d", &degree) == 1) {
            printf("\n");
            break;
        }
        else {
            printf("Nieprawidlowe dane. Podaj poprawny stopien wielomianu (liczba).\n");
            while (getchar() != '\n');
        }
    }
    return degree;
}

int* arrayWithCoefficients(int degree){
    int* array_coefficients = (int*)malloc(degree*sizeof(int));
    printf("Podaj prosze wspolczynniki liczonego wielomianu zaczynajac od tego przy najwyzszej
 potedze.\n\n");
    while(1){
        for (int i = 0; i < degree+1; i++){
            printf("Podaj %d wspolczynnik:", i);
            if(scanf("%d", &array_coefficients[i])==1){
                continue;
        }
            else{
                printf("Nieprawidlowe dane. Podaj poprawny stopien wielomianu (liczba).\n");
                while (getchar() != '\n');
                i--;
            }
        }
        printf("\n");
        break;
    }
    int n = degree;
    printf("Liczony wielomian to:");
    for(int i=0; i<n; i++){
        printf(" %dx^%d +",array_coefficients[i],degree);
        degree--;
        }
    printf(" %d\n",array_coefficients[n]);

    printf("\n");
    return array_coefficients;
}

int lowerLimit(){
    int a;
    printf("Calka liczona jest dla danego przedzialu <a,b>\n");
    while(1){
        printf("Podaj dolna granice (a) zasiegu liczenia calki:");
        if(scanf("%d",&a)==1){
            break;
        }
            else{
                printf("Nieprawidlowe dane. Podaj poprawny stopien wielomianu (liczba).\n");
                while (getchar() != '\n');
            }
    }
    return a;
}


int upperLimit() {
    int b;
    while(1){
        printf("Podaj gorna granice (b) zasiegu liczenia calki:");
        if(scanf("%d",&b)==1){
            printf("\n");
            break;
        }
            else {
                printf("Nieprawidlowe dane. Podaj poprawna granice dolna(liczba).\n");
                while (getchar() != '\n');
            }
    }
    return b;
}

int numOfSamples(){
    int n;
    printf("Obie metody liczenia opieraja sie na wyliczeniu wartosci dla wielomianu dla danej ilosci probek.\nIm wyzsza liczba probek tym wyzsza dokladnosc calki.\n\n");
    while(1){
        printf("Podaj liczbe probek dla ktorej chcesz liczyc calke (dowlona dodatnia liczba):");
        if(scanf("%d", &n)==1){
            printf("\n");
             break;
        }
            else{
                printf("Nieprawidlowe dane. Podaj poprawna dolna granice (liczba).\n");
                while (getchar() != '\n');
            }
    }
    return n;
}
```
Listing 3: C Input

7 Monte Carlo Method

In createTabXandTabY_monte as the name says we create two arrays which we will use to calculate our integral in function below named monte_carlo we use the label from the beggining. To specify:

- szerekosc_przedziału - width where we calculate integral
- (*xs), (*ys) - our pointers to arrays where we record our x's and f(x's)
- xi - is a random generated float between a and b
- (double)rand()/RAND_MAX is random value between (0, 1], after we add an a, we get number between [a , b]

In monte_carlo() we simply use our labels to calculate value as we said before.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "monte_carlo.h"
#include "func.h"

void createTabXandTabYMonte(double** xs,double** ys, int a, int b, int n, int tab[], int
    stopien) {

    srand(time(NULL));
    double szerokosc_przedzialu =(double) b - a;

    *xs = (double*) malloc(n*sizeof(double));
    *ys = (double*) malloc(n*sizeof(double));

    if (*xs == NULL || *ys == NULL) {
    fprintf(stderr, "Memory allocation failed\n");
    exit(EXIT_FAILURE);}

    for (int i = 0; i < n; i++) {
        double xi = a + ((double)rand() / RAND_MAX) * szerokosc_przedzialu; // Losowa wartosc
    z zakresu [a, b]
        (*xs)[i] = xi;
        (*ys)[i] = f(xi, tab, stopien);
    }
}

double monteCarlo(int a,int b, double ys[], int n){
    double suma = 0.0;
    double szerokosc_przedzialu = (double) b - a;

    for (int i = 0; i < n; i++) suma += ys[i];

    double calka = (suma * szerokosc_przedzialu) / n;
    return calka;
}
```

Listing 4: C monte carlo method

8 Simpson's rule

This function is very similar to the previous one. We change how we calculate the value in simpsons(). Instead of random numbers we take equal compartments. For even multiplies the results by two, for odd by four and then multiplies all the numbers added to themselves by width and divids by three.

```c
#include <stdio.h>
#include <stdlib.h>
#include "simpson.h"
#include "func.h"

void createTabXandTabYSimpson(double** xs,double** ys,int a, int b, int n, int tab[],int
    stopien){
    double przedzial = (double)(b - a)/n;
    (*xs) = (double*) malloc(n*sizeof(double*));
    (*ys) = (double*) malloc(n*sizeof(double*));

    if (*xs == NULL || *ys == NULL) {
    fprintf(stderr, "Memory allocation failed\n");
    exit(EXIT_FAILURE);}

    for(int i=0; i<=n; i++){
        double k = a + przedzial * i;
        (*xs)[i] = k;
        (*ys)[i] = f(k, tab, stopien);
    }
}

double simpson(int a,int b, int n,int tab[],int stopien){
    double przedzial = (double) (b - a) / n;
    double suma = 0.0;

   for (int i = 0; i <= n; i++) {
     double k = a + przedzial * i;
        if (i == 0 || i == n) suma += f(k, tab, stopien);
        else if (i % 2 == 0)  suma += 2 * f(k, tab, stopien);
        else                  suma += 4 * f(k, tab, stopien);
    }

    double calka = (przedzial / 3) * suma;
    return calka;
}
```

Listing 5: C simpson's rule

9 Calculation Error

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "calculation_error.h"

double error(double ys[], int n, double monte){
double y_squared = 0.0;
double monte_squared = 0.0;
double error;

double* ys_squared = (double*)malloc(n*sizeof(double));
for(int i = 0; i < n ; i++){
    ys_squared[i] = ys[i]*ys[i];
}

for(int j = 0; j < n; j++){
    y_squared += ys_squared[j];
}

y_squared = y_squared/n;
monte_squared = monte*monte;
error = sqrt((monte_squared-y_squared)/(n));

return error;
}
```

Listing 6: C Calculation Error

10 Plot

In this section we draw a plots.

- drawPlot_monte() - using the data from the .txt data file for Monte Carlo
- drawPlot_simpson - same but for Simpson's calculations
- writeDataToFile_monte() - writing the data from the arrays with the points of a Monte Carlo integration, so then we can draw plots
- writeDataToFile_simpson() - same but for Simpson's

```c
#include <stdio.h>
#include <stdlib.h>
#include "plot.h"

void drawPlotMonte(){
    FILE *gnuplotPipeMonte = popen("gnuplot -persistent", "w");
    if (gnuplotPipeMonte != NULL) {
        fprintf(gnuplotPipeMonte, "set terminal pngcairo enhanced color\n");
        fprintf(gnuplotPipeMonte, "set output 'calka_wykres_monte.png'\n");
        fprintf(gnuplotPipeMonte, "set xlabel 'X'\n");
        fprintf(gnuplotPipeMonte, "set ylabel 'Y'\n");
        fprintf(gnuplotPipeMonte, "set title 'Calka Monte Carlo'\n");
        fprintf(gnuplotPipeMonte, "plot 'plot_data_monte.txt' smooth unique title 'Data'\n");

    } else {
        printf("Error: Gnuplot not found or unable to open a pipe.\n");
        exit(EXIT_FAILURE);
    }
}

void drawPlotSimpson(){
    FILE *gnuplotPipeSimpson = popen("gnuplot -persistent", "w");
    if (gnuplotPipeSimpson != NULL) {
        fprintf(gnuplotPipeSimpson, "set terminal pngcairo enhanced color\n");
        fprintf(gnuplotPipeSimpson, "set output 'calka_wykres_simpson.png'\n");
        fprintf(gnuplotPipeSimpson, "set xlabel 'X'\n");
        fprintf(gnuplotPipeSimpson, "set ylabel 'Y'\n");
        fprintf(gnuplotPipeSimpson, "set title 'Calka Simpson'\n");
        fprintf(gnuplotPipeSimpson, "plot 'plot_data_simpson.txt' smooth unique title 'Data'\n
    ");
    } else {
        printf("Error: Gnuplot not found or unable to open a pipe.\n");
        exit(EXIT_FAILURE);
    }
}

void writeDataToFile(char* name, double x[], double y[], int n) {
    char filename [100];
    snprintf(filename, sizeof(filename),"plot_data_%s.txt",name);
    FILE *dataFile = fopen(filename, "w");
    if (dataFile != NULL) {
        for (int i = 0; i < n; i++) {
            fprintf(dataFile, "%f %f\n", x[i], y[i]);
        }
        fclose(dataFile);
    } else {
        printf("Error: Unable to open data file.\n");
        exit(EXIT_FAILURE);
    }
}

void writeDataToFileMonte(double x[],double y[],int n) {
    writeDataToFile("monte", x, y, n);
}

void writeDataToFileSimpson(double x[],double y[],int n){
    writeDataToFile("simpson", x, y, n);
}
```

Listing 7: C Plot

11 Data

In this section we use all data we got from the the code and simply write out it to file.

```c
#include <stdio.h>
#include <stdlib.h>
#include "data.h"

void writePolynomialToFile(int st, int tab[],int a, int b, int n, double cal1, double cal2,
    double error){
    FILE *dataFile = fopen("data.txt", "a");
    if (dataFile != NULL) {
        fprintf(dataFile, "Dla wielomianu:\n ");
        int m = st;
        for(int i=0; i<m; i++){
            fprintf(dataFile, "%dx^%d + ",tab[i],st);
            st--;
        }
        fprintf(dataFile, "%d ",tab[m]);
        fprintf(dataFile, " w przedziale(%d,%d) ", a, b);
        fprintf(dataFile, "dla %d ilosci probek:\n\n", n);
        fprintf(dataFile, "Monte Carlo:%f\n", cal1);
        fprintf(dataFile, "Simpson:%f\n\n", cal2);
        fprintf(dataFile, "Blad obliczeniowy:%f\n\n", error);
        fprintf(dataFile, "
    -------------------------------------------------------------------\n");

        fclose(dataFile);
    } else {
        printf("Error: Unable to open data file.\n");
        exit(EXIT_FAILURE);
    }
}
```

Listing 8: Data

12 Makefile

```makefile
CC = gcc
CFLAGS = -Wall -Wextra

# lista plikow .o
OBJECTS = func.o input.o main.o monte_carlo.o plot.o simpson.o data.o calculation_error.o
PNG = calka_wykres_monte.png calka_wykres_simpson.png
TXT = plot_data_monte.txt plot_data_simpson.txt
all: Monte_carlo

# linkowanie
Monte_carlo: $(OBJECTS)
  $(CC) $(CFLAGS) -o $@ $^

# kompilowanie
%.o: %.c
  $(CC) $(CFLAGS) -c -o $@ $<

# czyszczenie
clean:
  rm -f $(OBJECTS) Monte_carlo
  rm -f $(PNG) Monte_carlo
  rm -f $(TXT) Monte_carlo

run: Monte_carlo
   ./Monte_carlo
```

Listing 9: Makefile