MOwNiT Lab4 - sprawozdanie

Autor: Michał Flak

Zadanie 1

Tematem zadania będzie obliczanie różnymi sposobami całki funkcji x^2 oraz 1/sqrt(x) w przedziale (0,1). Proszę dla obydwu funkcji znaleźć dokładną wartość całki (całkując ręcznie)

Wyniki obliczone analityczne:

- Całka z x^2 na przedziale (0,1) = 1/3
- Całka z 1/sqrt(x) na przedziale (0,1) = 2

Zadanie 2

Napisać program obliczającą całkę metodą prostokątów.

```
double integrateRect(float a, float b, long long int n, double (*fun)
(double, void*))
{
  double y;
  double sum = 0;
  double increment = (b - a) / (double)n;
  for (double i = a + increment/2.0; i < b; i += increment)
  {
    y = fun(i, NULL);
    sum += y * (b-a) / (double)n;
  }
  return sum;
}
```

Zadanie 3

Zbadac, przy uzyciu programu z poprzedniego punktu, jak zmienia sie blad calkowania wraz ze wzrostem liczby podprzedzialow. Kiedy blad jest mniejszy niz 1e-3, 1e-4, 1e-5 i 1e-6?

Mierzymy błąd względny, porównując z oczekiwaną wartością policzoną analitycznie.

```
  double resultRect = integrateRect(a, b, n, fun);

  double errRel = fabs((desired - resultRect) / desired);
```

Uruchomiono program skryptem Bash w pętli - liczba podprzedziałów:

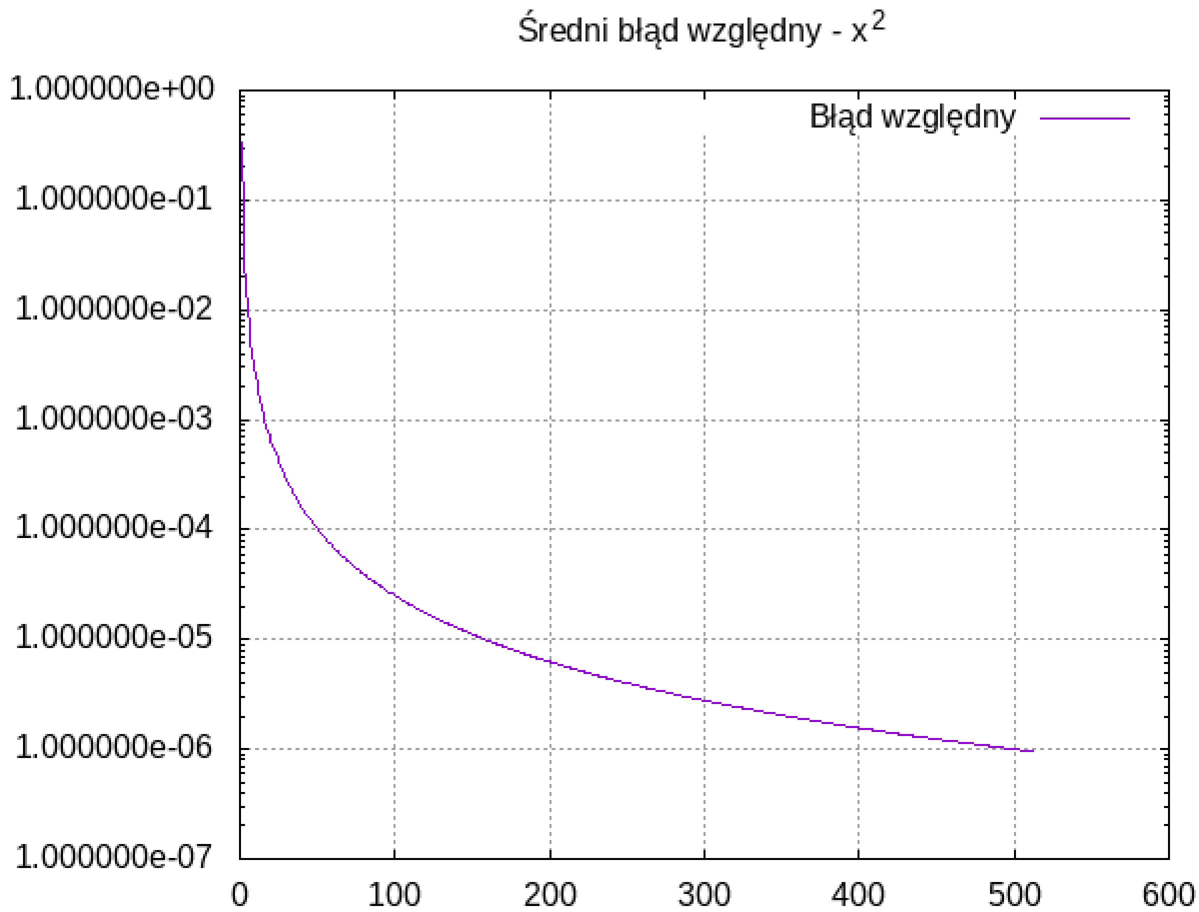
- dla funkcji 1 [1..512]

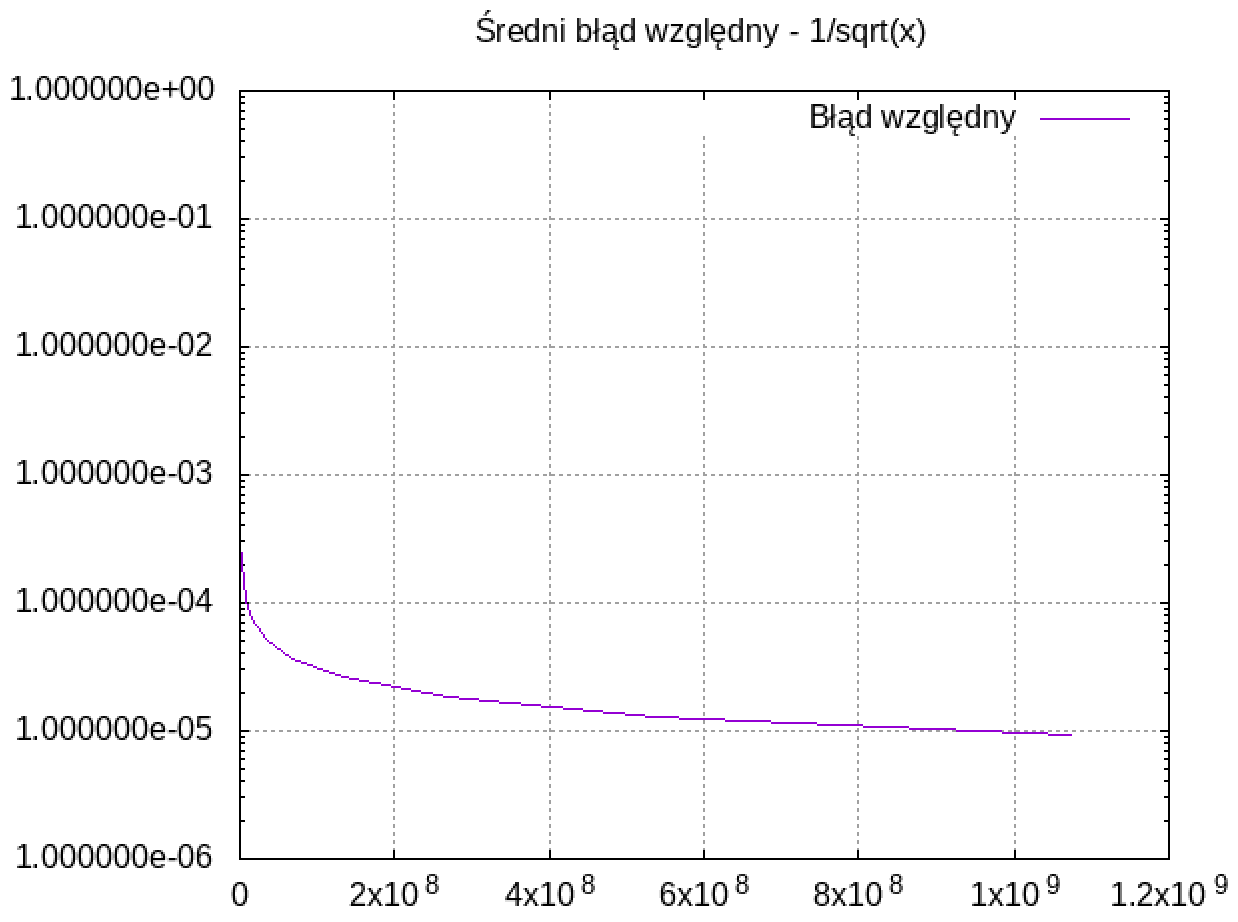
- dla funkcji 2 [1..4000000000], podwojenie liczby co iterację

Wyniki przekierowano do plików tekstowych [f1.txt, f2.txt]. Format:

```
n   ResultRect, ResultGsl, GSLN, ErrRel
310 0.333332    0.333333   1     2.60146e-06
311 0.333332    0.333333   1     2.58475e-06
312 0.333332    0.333333   1     2.56821e-06
```

Na tej podstawie narysowano wykresy:





Pierwsze uzyskane punkty, dla których błąd jest mniejszy niż:

Funkcja 1:

- 1e-3: n=16 err=0.000976562
- 1e-4: n=51 err=9.61169e-05
- 1e-5: n=159 err=9.88885e-06
- 1e-6: n=501 err=9.96012e-07

Funkcja 2:

- 1e-3: n=131072 err=0.000835406
- 1e-4: n=16777216 err=7.38402e-05
- 1e-5: n=1073741824 err=9.23002e-06
- 1e-6: Program działał zbyt dlugo

Zadanie 5

Obliczyć wartość całki korzystając z funkcji gsl_integration_qag metodą GSL_INTEG_GAUSS15 dla zadanych dokładności takich jak w p. 3. Sprawdzić, ile przedziałów (intervals) potrzebuje ta procedura aby osiągnąc zadaną dokładność (1e-3, 1e-4, 1e-5 i 1e-6). Porownac, ile przedzialow potrzebuje metoda prostokatow do osiagniecia podobnej dokladnosci. Patrz przykład w dokumentacji GSL.

Napisano funkcję obliczającą całkę o podanym maksymalnym błędzie przy użyciu GSL:

```
double integrateGSL(float a, float b, double acc, int nmax, double (*fun)
(double, void*), int* n)
{
  gsl_function F;
  F.function = fun;

  gsl_integration_workspace * ws = gsl_integration_workspace_alloc(nmax);
  double result;
  double abserr;

  int res = gsl_integration_qag(
    &F,
    a,
    b,
    0,
    acc,
    nmax,
    GSL_INTEG_GAUSS15,
    ws,
    &result,
    &abserr
    );

  *n = ws->size;
  gsl_integration_workspace_free(ws);
  return result;
}
```

Ilość przedziałów, jakiej potrzebuje procedura do osiągnięcia żądanej dokładności odczytano używając `ws->size`. Wyniki są następujące (oznaczone jako `GSLn`. Obok `n` z metody prostokątów dla porównania):

Funkcja 1:

- 1e-3: GSLn=1 n=16 err=0.000976562
- 1e-4: GSLn=1 n=51 err=9.61169e-05
- 1e-5: GSLn=1 n=159 err=9.88885e-06
- 1e-6: GSLn=1 n=501 err=9.96012e-07

Funkcja 2:

- 1e-3: GSLn=20 n=131072 err=0.000835406
- 1e-4: GSLn=27 n=16777216 err=7.38402e-05
- 1e-5: GSLn=33 n=1073741824 err=9.23002e-06
- 1e-6: Program działał zbyt dlugo

Jak widzimy, metoda prostokątów jest dużo mniej efektywna.

Kod programu do wglądu

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_integration.h>

double f1(double x, void* params)
{
    return x*x;
}

double f2(double x, void* params)
{
    return 1 / sqrt(x);
}

double integrateRect(float a, float b, long long int n, double (*fun)
(double, void*))
{
  double y;
  double sum = 0;
  double increment = (b - a) / (double)n;
  for (double i = a + increment/2.0; i < b; i += increment)
  {
    y = fun(i, NULL);
    sum += y * (b-a) / (double)n;
  }
  return sum;
}

double integrateGSL(float a, float b, double acc, int nmax, double (*fun)
(double, void*), int* n)
{
  gsl_function F;
  F.function = fun;

  gsl_integration_workspace * ws = gsl_integration_workspace_alloc(nmax);
  double result;
  double abserr;

  int res = gsl_integration_qag(
    &F,
    a,
    b,
    0,
    acc,
    nmax,
    GSL_INTEG_GAUSS15,
    ws,
    &result,
    &abserr
    );
```

```
  *n = ws->size;
  gsl_integration_workspace_free(ws);
  return result;
}

int main(int argc, char const *argv[])
{
  const double a = 0;
  const double b = 1;
  int n;
  double desired;
  double (*fun)(double, void*) = f1;
  int resultingN;

  //select function by parameter:
  //-f1: y = x*x
  //-f2: y = 1 / sqrt(x)
  //if(argc != 2) return -1;
  if(!strcmp(argv[1], "-f1"))
  {
    fun = f1;
    desired = 1.0/3.0;
  }
  else if(!strcmp(argv[1], "-f2"))
  {
    fun = f2;
    desired = 2.0;
  }

  //select desired iterations
  scanf("%d", &n);

  double resultRect = integrateRect(a, b, n, fun);

  double errRel = fabs((desired - resultRect) / desired);

  double resultGSL = integrateGSL(a, b, errRel, 1000, fun, &resultingN);

  //printf("ResultRect, ResultGsl, ResultingN, ErrRel\n");
  printf(
    "%d %lg %lg %d %lg, %d, %d, %d, %d\n",
    n,
    resultRect,
    resultGSL,
    resultingN,
    errRel,
    errRel <= 1e-3,
    errRel <= 1e-4,
    errRel <= 1e-5,
    errRel <= 1e-6
    );
  return 0;
```

```
}
```