

Wydział WIMiIP	Imię i nazwisko Mateusz Witkowski	Rok II	Grupa 4
Temat: Kwadratury Gaussa 2D			Prowadzący dr hab. inż. Hojny Marcin, prof. AGH
Data ćwiczenia 16.04.2020	Data oddania 23.04.2020	Data zaliczenia	OCENA

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się oraz implementacja kwadratury Gaussa 2D na podstawie załączonej instrukcji oraz przykładowego programu.

2. Wprowadzenie teoretyczne

Kwadratura Gaussa 2D jest popularną kwadraturą złożoną do obliczania pola powierzchni figury określonej w dwóch kierunkach. Metody złożone charakteryzują się zdecydowanie lepszą dokładnością w porównaniu do podstawowych metod przybliżonego obliczania całek. Polegają na podzieleniu przedziału $[a, b]$ na pewną liczbę podprzedziałów, zastosowaniu metody w podprzedziałach a następnie zsumowaniu wyników.

Kwadratura Gaussa 2D polega na przekształceniu układu współrzędnych w taki sposób, by element kwadratowy został odwzorowany przez kwadrat o wymiarach 2×2 . Transformacja układu współrzędnych określona jest równaniem:

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{bmatrix} \{N\}^T & 0 \\ 0 & \{N\}^T \end{bmatrix} \begin{Bmatrix} x_n \\ y_n \end{Bmatrix}$$

- $\{x_n\} = \{x_1, x_2, x_3, x_4\}^T$
- $\{y_n\} = \{y_1, y_2, y_3, y_4\}^T$
- $\{N\} = \frac{1}{4} \{(1 - \xi)(1 - \eta), (1 + \xi)(1 - \eta), (1 + \xi)(1 + \eta), (1 - \xi)(1 + \eta)\}^T$

W kwadraturze Gaussa 2D określony jest przedział $[a, b]$, punkty całkowania oraz ich wagi:

- $[a, b] = [-1, 1]$
- $w_0 = w_1 = 1$
- $\xi_0 = \eta_0 = 0,5773502692$
- $\xi_1 = \eta_1 = -0,5773502692$

Ważnym elementem jest wyliczenie pochodnych cząstkowych ξ, η :

$$\begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{Bmatrix} = [J] \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{Bmatrix}$$

Gdzie: $[J]$ – jest to macierz Jacobiego, z której wyznacznik $[J_0]$ jest Jakobianem transformacji układu współrzędnych, obliczanym za pomocą wzoru:

$$\det |J| = \frac{\partial x}{\partial \xi} \frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi} \frac{\partial x}{\partial \eta}$$

Pochodne to suma pochodnych cząstkowych czterech wierzchołków przemnożona przez wartość współrzędnej x lub y, dana wzorem:

$$\frac{\partial x}{\partial \xi} = \sum_{i=1}^4 \frac{\partial N_i}{\partial \xi} x_i \qquad \frac{\partial y}{\partial \xi} = \sum_{i=1}^4 \frac{\partial N_i}{\partial \xi} y_i$$

$$\frac{\partial x}{\partial \eta} = \sum_{i=1}^4 \frac{\partial N_i}{\partial \eta} x_i \qquad \frac{\partial y}{\partial \eta} = \sum_{i=1}^4 \frac{\partial N_i}{\partial \eta} y_i$$

Dla każdego z wierzchołków od 1 do 4 obliczamy N:

- $N_1(\xi, \eta) = 0.25(1 - \xi)(1 - \eta)$
- $N_2(\xi, \eta) = 0.25(1 + \xi)(1 - \eta)$
- $N_3(\xi, \eta) = 0.25(1 + \xi)(1 + \eta)$
- $N_4(\xi, \eta) = 0.25(1 - \xi)(1 + \eta)$

Uwzględniając wszystkie powyższe kroki, całkowanie funkcji w układzie ξ, η , za pomocą metody Gaussa można zapisać wzorem:

$$\int \int f(x, y) dx dy = \int_{-1}^1 \int_{-1}^1 f(\xi, \eta) J_0 d\eta d\xi = \sum_{i=0}^n \sum_{j=0}^n w_i w_j f(\xi_i, \eta_j) J_0$$

Gdzie:

- $w_0 = w_1 = 1$
- $\xi_0 = \eta_0 = 0,5773502692$
- $\xi_1 = \eta_1 = -0,5773502692$

3. Kod programu

Zdefiniowano globalnie tablice statyczne przechowujące punkty całkowania oraz ich wagi, dodatkowo utworzono wektor oraz zmienna przechowującą nazwę pliku, potrzebne do pobrania danych dotyczących wierzchołków w późniejszej części programu.

```
vector < double > dane;  
string nazwaPliku = "punkty.txt";  
double waga[2] = { 1.0, 1.0 };  
double punkt[2] = { -0.5773502692, 0.5773502692 };
```

Rysunek 1. Deklaracja w przestrzeni globalnej

Zdefiniowano funkcję pobierającą dane z podanego pliku txt oraz uzupełniającą przekazany wektor.

```
//Funkcja pobierająca dane z pliku tekstowego i zapisujące je do wektora.  
int pobranie_z_pliku(vector <double>* data, string fileName) {  
    string linia;  
    fstream plik;  
  
    plik.open(fileName, ios::in);           //otwarcie pliku  
    if (plik.good() == true)               //sprawdzenie poprawności pliku  
    {  
        while (!plik.eof())  
        {  
            getline(plik, linia, ',');      //zapisuje słowa oddzielane przecinkami  
            double d = atof(linia.c_str()); //konwersja stringa na double  
            data->push_back(d);              //zapis słowa parsowanego na typ double do wektora  
        }  
        plik.close();                       //zamknięcie pliku  
    }  
    if (data->size() % 2 == 1) {             //zabezpieczenie w razie le wyrażonego pliku tekstowego  
        cout << "bledne dane w pliku" << endl;  
        return -1;  
    }  
    return data->size();                    //funkcja zwraca wielkość wektora potrzebna do stworzenia tablic X, Y  
}
```

Rysunek 2. Funkcja pobierająca dane z pliku

W funkcji main następuje sprawdzenie poprawności pliku oraz rozdzielanie pobranych danych o punktach na odpowiednie tablice – X i Y.

```
//////////SPRAWDZANIE POPRAWNOSCI PLIKU//////////
int wielkosc_wektora = pobranie_z_pliku(&dane, nazwaPliku);
if (wielkosc_wektora == -1) //program zostanie przerwany jesli plik tekstowy byl bledny
{
    return -1;
}

//////////WYPEENIANIE TABLIC//////////
int liczba_punktow = wielkosc_wektora / 2; //wielkosc wektora podzielona przez 2 powinna nam dać ilo punktów zapisanych w pliku
double* X = new double[liczba_punktow]; //tablica współczynników x
double* Y = new double[liczba_punktow]; //tablica współczynników y
int j = 0;
for (size_t i = 0; i < dane.size(); i++) //wypełnienie tablic
{
    if (i % 2 == 0) {
        X[j] = dane[i];
    }
    if (i % 2 == 1) {
        Y[j] = dane[i];
        j++;
    }
}
```

Rysunek 3. Sprawdzenie poprawności, podział danych

Utworzono plik txt i wypełniono go danymi.

punkty.txt	X	Źródło.cpp*
1	0,	0,
2	5,	0,
3	5,	5,
4	0,	4

Rysunek 4. Plik z wierzchołkami czworokąta

Definicja funkcji liczącej powierzchnię przyjmuje dwa argumenty – tablice wierzchołków, wewnątrz zadeklarowano niezbędne wielowymiarowe tablice potrzebne do dalszych obliczeń, przechowujące pochodne względem ξ i η oraz wyznacznik z macierzy Jacobiego.

```
double Gauss2D(double * X, double * Y) {
    double Poch_KSI[2][4]; //Pochodne względem ksi
    double Poch_NI[2][4]; //pochodne względem ni
    double Fun_DETJ[2][2]; //Wyznacznik macierzy Jacobiego
```

Rysunek 5. Deklaracja tablic

W podwójnej pętli obliczono wartości pochodnych względem ξ i η

```
for (size_t j = 0; j < 2; j++)
{
    for (size_t i = 0; i < 2; i++)
    {
        Poch_KSI[j][0] = -0.25 * (1.0 - punkt[j]);
        Poch_KSI[j][1] = 0.25 * (1.0 - punkt[j]);
        Poch_KSI[j][2] = 0.25 * (1.0 + punkt[j]);
        Poch_KSI[j][3] = -0.25 * (1.0 + punkt[j]);

        Poch_NI[i][0] = -0.25 * (1.0 - punkt[i]);
        Poch_NI[i][1] = -0.25 * (1.0 + punkt[i]);
        Poch_NI[i][2] = 0.25 * (1.0 + punkt[i]);
        Poch_NI[i][3] = 0.25 * (1.0 - punkt[i]);
    }
}
```

Rysunek 6. Wyliczenie pochodnych

Wyliczamy wartości iloczynów sum pochodnych cząstkowych i wartości opowiadających współrzędnych x bądź y. Następnie wyznaczamy wartość macierzy Jacobiego.

```
for (size_t j = 0; j < 2; j++)
{
    for (size_t i = 0; i < 2; i++)
    {
        double dx_dKSI = Poch_KSI[j][0] * X[0] + Poch_KSI[j][1] * X[1] + Poch_KSI[j][2] * X[2] + Poch_KSI[j][3] * X[3];
        double dy_dKSI = Poch_KSI[j][0] * Y[0] + Poch_KSI[j][1] * Y[1] + Poch_KSI[j][2] * Y[2] + Poch_KSI[j][3] * Y[3];

        double dx_dNI = Poch_NI[i][0] * X[0] + Poch_NI[i][1] * X[1] + Poch_NI[i][2] * X[2] + Poch_NI[i][3] * X[3];
        double dy_dNI = Poch_NI[i][0] * Y[0] + Poch_NI[i][1] * Y[1] + Poch_NI[i][2] * Y[2] + Poch_NI[i][3] * Y[3];

        Fun_DETJ[i][j] = dx_dKSI * dy_dNI - dx_dNI * dy_dKSI;
    }
}
```

Rysunek 7. Wyliczenie wartości macierzy Jacobiego

Finalnie wyliczono wartość powierzchni zadanego czworokąta, poprzez przemnożenie wartości bezwzględnej z obliczonego wyznacznika Jacobiego i odpowiednich wag. Zwrócono wynik i wypisano go w main'nie na konsolę.

```
double Powierzchnia = 0.0;
for (size_t j = 0; j < 2; j++)
{
    for (size_t i = 0; i < 2; i++)
    {
        Powierzchnia = Powierzchnia + fabs(Fun_DETJ[i][j]) * waga[i] * waga[j];
    }
}
return Powierzchnia;
```

Rysunek 8. Obliczenie powierzchni

Cały kod programu:

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;
vector < double > dane;
string nazwaPliku = "punkty.txt";
double waga[2] = { 1.0, 1.0 };
double punkt[2] = { -0.5773502692, 0.5773502692 };

//Funkcja pobierająca dane z pliku tekstowego i zapisujące je do wektora.
int pobranie_z_pliku(vector <double>* data, string fileName) {
    string linia;
    fstream plik;

    plik.open(fileName, ios::in);          //otwarcie pliku
    if (plik.good() == true)              //sprawdzenie poprawności pliku
    {
        while (!plik.eof())
        {
            getline(plik, linia, ',');      //zapisuje słowa oddzielane przecinkami
            double d = atof(linia.c_str()); //konwersja stringa na double
            data->push_back(d);              //zapis słowa parsowanego na typ double do wektora
        }
        plik.close();                      //zamknięcie pliku
    }
    if (data->size() % 2 == 1) {             //zabezpieczenie w razie lewyplonego pliku tekstowego
        cout << "błędne dane w pliku" << endl;
        return -1;
    }
    return data->size();                    //funkcja zwraca wielkość wektora potrzebna do stworzenia tablic X, Y
}

double Gauss2D(double * X, double * Y) {
    double Poch_KSI[2][4]; //Pochodne względem ksi
    double Poch_NI[2][4];  //pochodne względem ni
    double Fun_DETJ[2][2]; //Wyznacznik macierzy Jacobiego

    for (size_t j = 0; j < 2; j++)
    {
        for (size_t i = 0; i < 2; i++)
        {
            Poch_KSI[j][0] = -0.25 * (1.0 - punkt[j]);
            Poch_KSI[j][1] = 0.25 * (1.0 - punkt[j]);
            Poch_KSI[j][2] = 0.25 * (1.0 + punkt[j]);
            Poch_KSI[j][3] = -0.25 * (1.0 + punkt[j]);

            Poch_NI[i][0] = -0.25 * (1.0 - punkt[i]);
            Poch_NI[i][1] = -0.25 * (1.0 + punkt[i]);
            Poch_NI[i][2] = 0.25 * (1.0 + punkt[i]);
            Poch_NI[i][3] = 0.25 * (1.0 - punkt[i]);
        }
    }

    for (size_t j = 0; j < 2; j++)
    {
        for (size_t i = 0; i < 2; i++)
        {
            double dx_dKSI = Poch_KSI[j][0] * X[0] + Poch_KSI[j][1] * X[1] + Poch_KSI[j][2] * X[2] + Poch_KSI[j][3] * X[3];
            double dy_dKSI = Poch_KSI[j][0] * Y[0] + Poch_KSI[j][1] * Y[1] + Poch_KSI[j][2] * Y[2] + Poch_KSI[j][3] * Y[3];

            double dx_dNI = Poch_NI[i][0] * X[0] + Poch_NI[i][1] * X[1] + Poch_NI[i][2] * X[2] + Poch_NI[i][3] * X[3];
            double dy_dNI = Poch_NI[i][0] * Y[0] + Poch_NI[i][1] * Y[1] + Poch_NI[i][2] * Y[2] + Poch_NI[i][3] * Y[3];

            Fun_DETJ[i][j] = dx_dKSI * dy_dNI - dx_dNI * dy_dKSI;
        }
    }
}
```

```

double Powierzchnia = 0.0;
for (size_t j = 0; j < 2; j++)
{
    for (size_t i = 0; i < 2; i++)
    {
        Powierzchnia = Powierzchnia + fabs(Fun_DETJ[i][j]) * waga[i] * waga[j];
    }
}
return Powierzchnia;
}

int main()
{
    //////////SPRAWDZANIE POPRAWNOSCI PLIKU//////////
    int wielkosc_wektora = pobranie_z_pliku(&dane, nazwaPliku);
    if (wielkosc_wektora == -1) //program zostanie przerwany jesli plik tekstowy byl bledny
    {
        return -1;
    }
    ////////////WYPEENIANIE TABLIC//////////
    int liczba_punktow = wielkosc_wektora / 2; //wielkosc wektora podzielona przez 2 powinna nam dać ilo punktów zapisanych w pliku
    double* X = new double[liczba_punktow]; //tablica współczynników x
    double* Y = new double[liczba_punktow]; //tablica współczynników y
    int j = 0;
    for (size_t i = 0; i < dane.size(); i++) //wypełnienie tablic
    {
        if (i % 2 == 0) {
            X[j] = dane[i];
        }
        if (i % 2 == 1) {
            Y[j] = dane[i];
            j++;
        }
    }

    cout << "Powierzchnia wynosi = " << Gauss2D(X,Y) << endl;
    system("Pause");

    return 0;
}

```

4. Testy

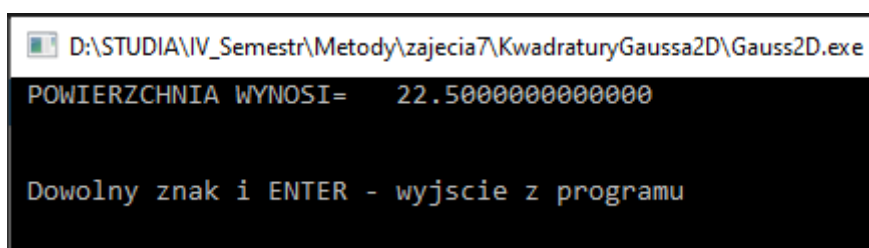
W celu zweryfikowania wyników z programu porównano je z tymi, zwróconymi przez program załączony w instrukcji do ćwiczenia.

1. Przykład

Tabela 1. Dane z przykładu .

X	0	5	5	0
y	0	0	5	4

Załączony program (Fortran):



```

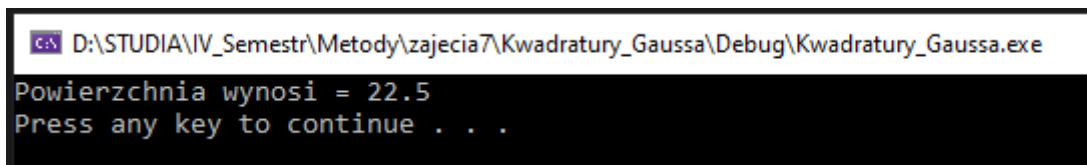
D:\STUDIA\IV_Semestr\Metody\zajecia7\KwadraturyGaussa2D\Gauss2D.exe
POWIERZCHNIA WYNOSI= 22.5000000000000

Dowolny znak i ENTER - wyjscie z programu

```

Rysunek 9. Wynik działania programu

Implementacja C++



```
C:\> D:\STUDIA\IV_Semestr\Metody\zajecia7\Kwadratury_Gaussa\Debug\Kwadratury_Gaussa.exe
Powierzchnia wynosi = 22.5
Press any key to continue . . .
```

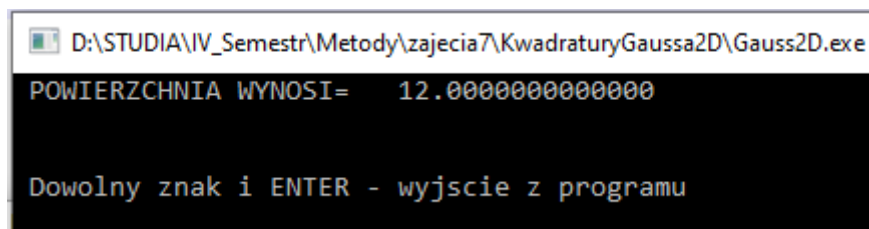
Rysunek 10. Wynik działania programu

2. Przykład

Tabela 2. Dane z przykładu .

X	0	-2	0	2
y	0	3	6	-3

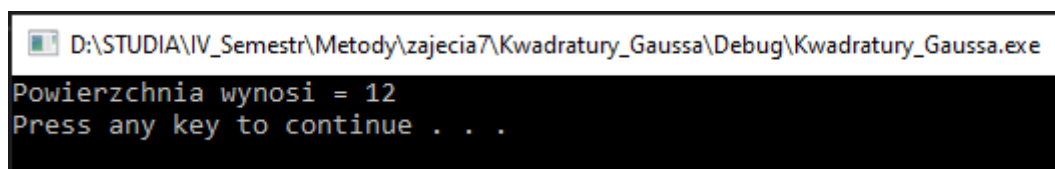
Załączony program (Fortran):



```
D:\STUDIA\IV_Semestr\Metody\zajecia7\KwadraturyGaussa2D\Gauss2D.exe
POWIERZCHNIA WYNOSI= 12.0000000000000
Dowolny znak i ENTER - wyjscie z programu
```

Rysunek 11. Wynik działania programu

Implementacja C++



```
D:\STUDIA\IV_Semestr\Metody\zajecia7\Kwadratury_Gaussa\Debug\Kwadratury_Gaussa.exe
Powierzchnia wynosi = 12
Press any key to continue . . .
```

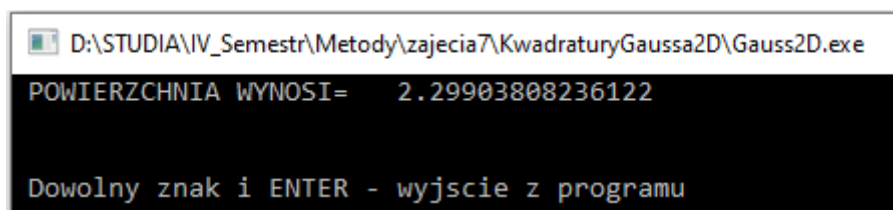
Rysunek 12. Wynik działania programu

3. Przykład

Tabela 3. Dane z przykładu .

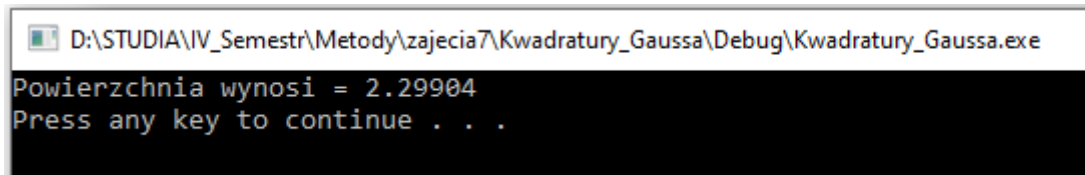
X	0.5	-0.5	0	1
y	0	1	4	-3

Załączony program (Fortran):



```
D:\STUDIA\IV_Semestr\Metody\zajecia7\KwadraturyGaussa2D\Gauss2D.exe
POWIERZCHNIA WYNOSI= 2.29903808236122
Dowolny znak i ENTER - wyjscie z programu
```

Rysunek 13. Wynik działania programu



```
D:\STUDIA\IV_Semestr\Metody\zajecia7\Kwadratury_Gaussa\Debug\Kwadratury_Gaussa.exe
Powierzchnia wynosi = 2.29904
Press any key to continue . . .
```

Rysunek 14. Wynik działania programu

5. Wnioski

Kwadratura Gaussa jest niezwykle dokładna ze względu na specjalnie dobrane węzły i wagi, dzięki czemu metoda tak osiąga najwyższy możliwy stopień dokładności. Opiera się ona o aproksymację funkcji całkowanej wielomianem interpolacyjnym. Wykonane powyżej testy dowodzą, że wyniki programu są poprawne oraz bardzo precyzyjne.