Obowiązkowe zadanie obliczeniowe

Równania Różniczkowe i Różnicowe

Przemysław Popowski

Wariant 4.1 : Równanie Transportu Ciepła

$$-k(x)\frac{d^2u(x)}{dx^2} = 0$$

$$u(2) = 0$$

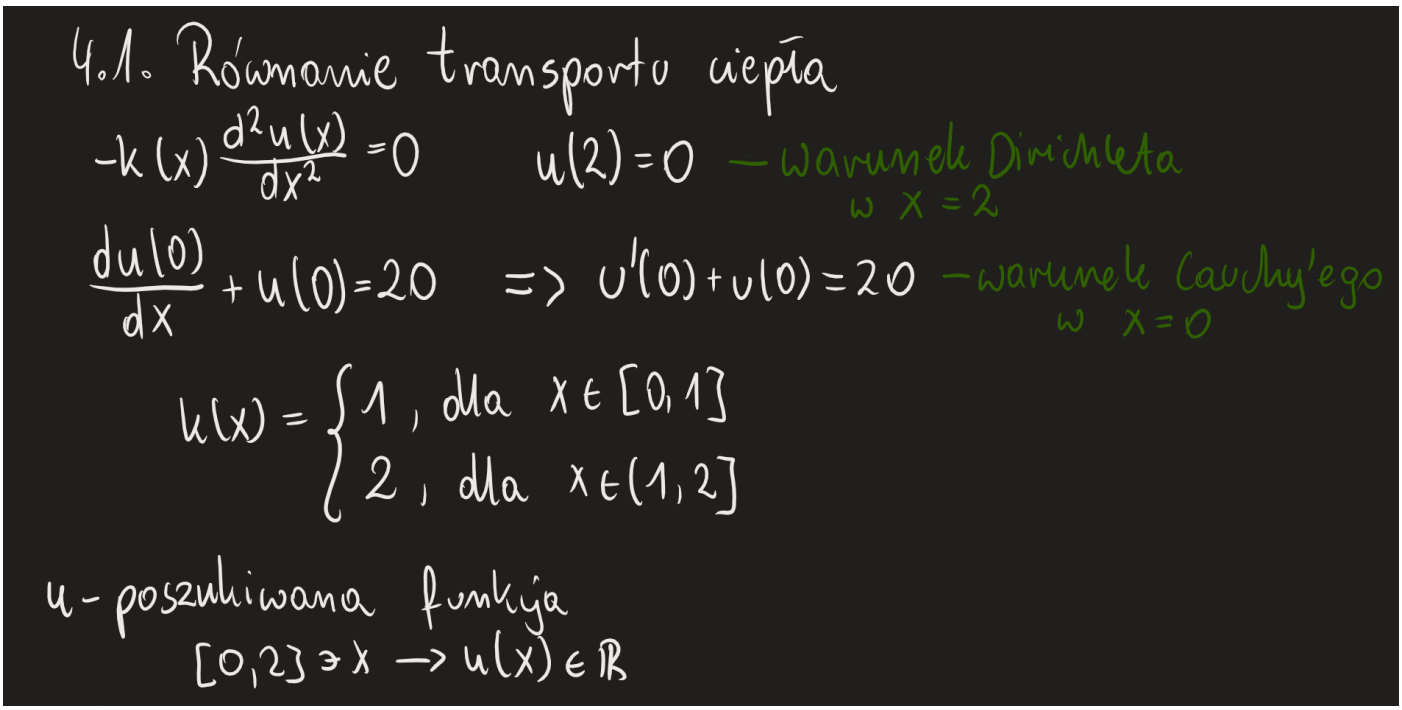
$$\frac{du(0)}{dx} + u(0) = 20$$

$$k(x) = \begin{cases} 1 & \textbf{dla } x \in [0,1] \\ 2 & \textbf{dla } x \in (1,2] \end{cases}$$

Gdzie *u* to poszukiwana funkcja

$[0, 2] \ni x \rightarrow u(x) \in \mathbb{R}$

Punkt 1. – Rozwiązanie równania



$-k(x) \cdot u'' = 0 \qquad x \in [0,2] \qquad \Omega = (0,2)$

$-k(x) \cdot u'' = 0 \quad / : -k(x)$

$u'' = 0$

mnożę obie strony przez funkcję testową $v \quad v \in V$

$v(2) = 0$

$u'' v = 0$

całkuję po dziedzinie

$$\int_0^2 u'' v \, dx = 0$$

$$[u'v]_0^2 - \int_0^2 u'v' \, dx = 0$$

$$u'(2)v(2) - (u'(0)v(0)) - \int_0^2 u'v' \, dx = 0$$

$v(2) = 0 \qquad u'(0) = 20 - u(0)$

$$-\int_0^2 u'v' \, dx - (20 - u(0))v(0) = 0$$

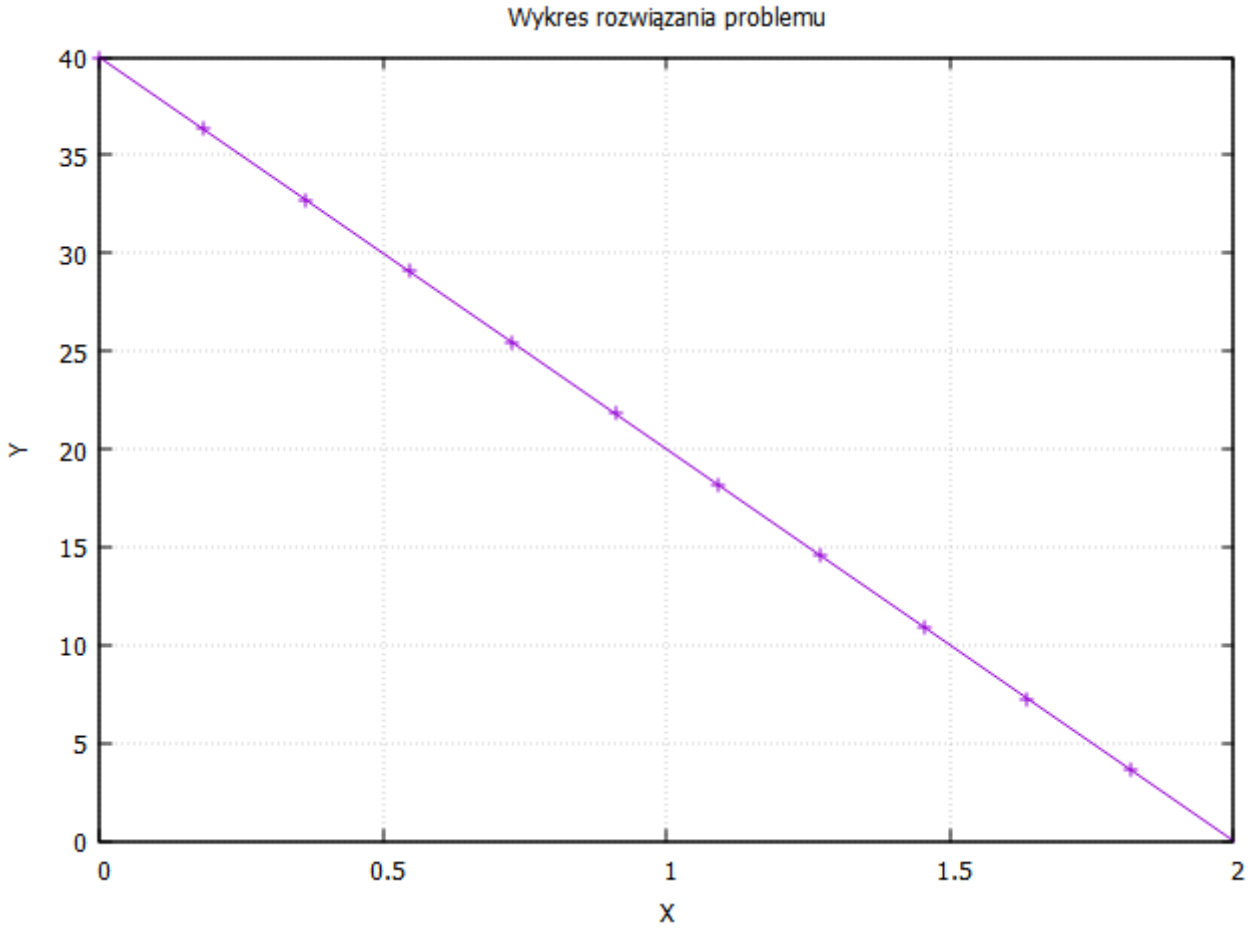
$$-20v(0) + u(0)v(0) - \int_0^2 u'v' \, dx = 0$$

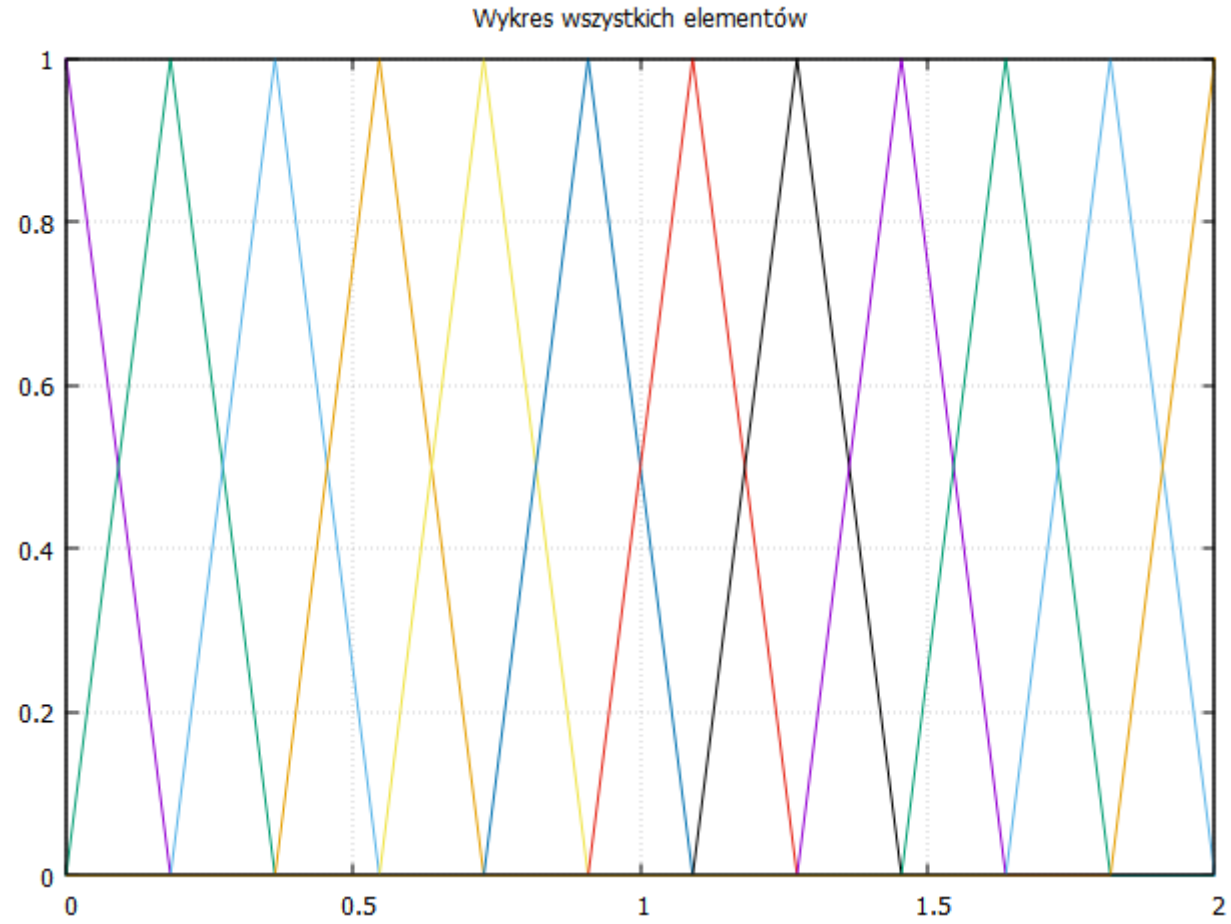
$$\underbrace{u(0)v(0) - \int_0^2 u'v' \, dx}_{B(u,v)} = \underbrace{20\,v(0)}_{L(v)}$$

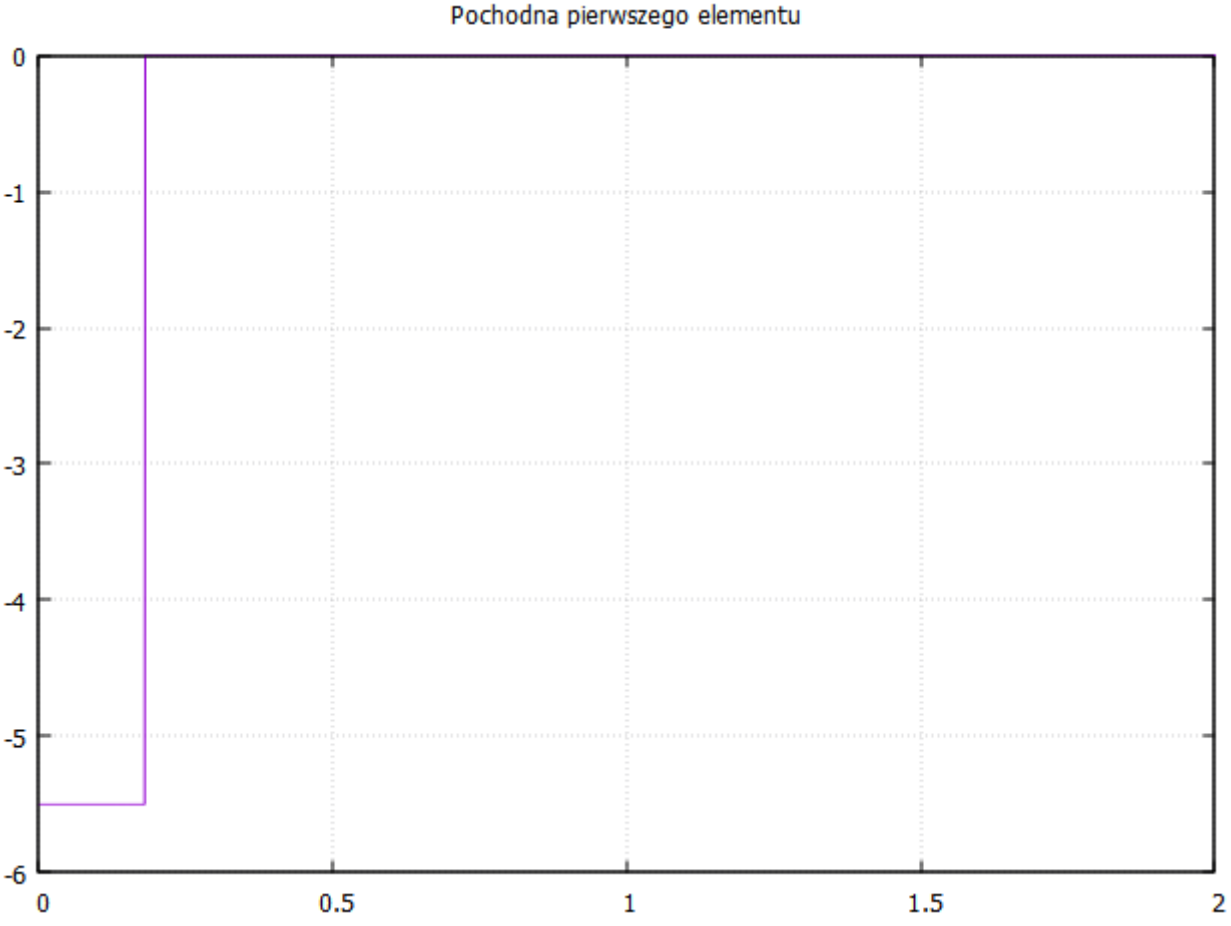
Punkt 2. – Prezentacja uzyskanych wyników dla przykładowego n = 11

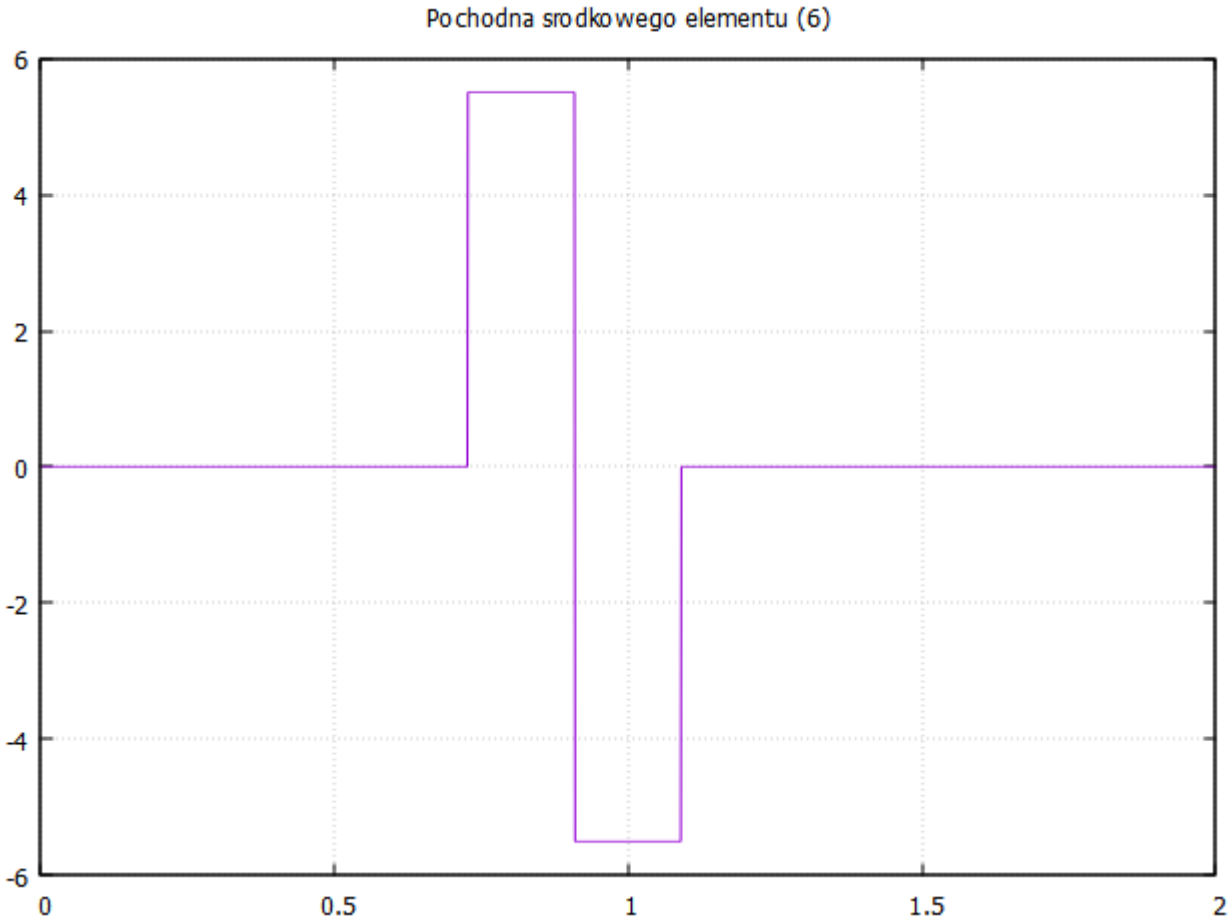
Punkty będące rozwiązaniem układu równań

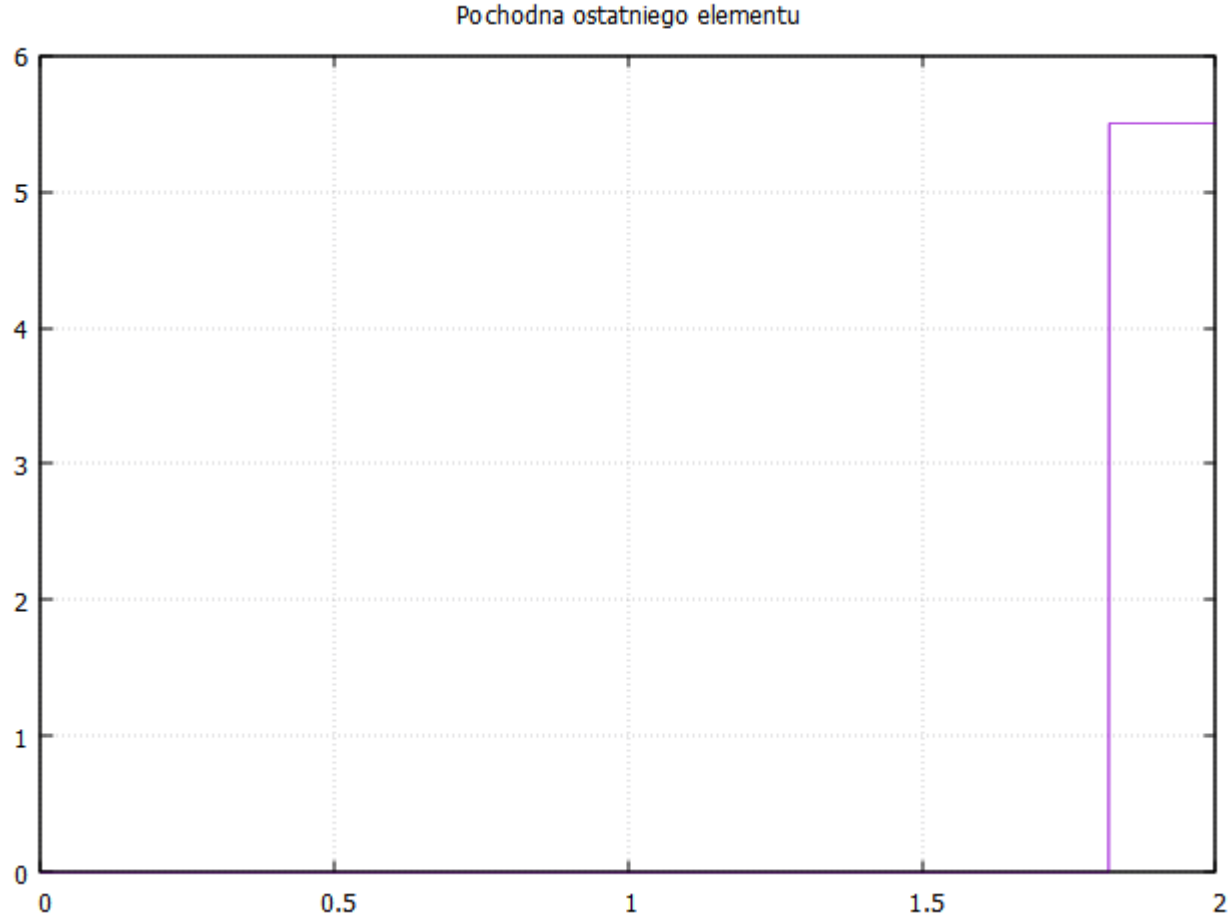
```
(x = 0 , y = 40)
(x = 0.181818 , y = 36.3636)
(x = 0.363636 , y = 32.7273)
(x = 0.545455 , y = 29.0909)
(x = 0.727273 , y = 25.4545)
(x = 0.909091 , y = 21.8182)
(x = 1.09091 , y = 18.1818)
(x = 1.27273 , y = 14.5455)
(x = 1.45455 , y = 10.9091)
(x = 1.63636 , y = 7.27273)
(x = 1.81818 , y = 3.63636)
(x = 2 , y = 0)
```




Wykres wszystkich elementów
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2


Pochodna pierwszego elementu
0
-1
-2
-3
-4
-5
-6
0
0.5
1
1.5
2


Pochodna srodkowego elementu (6)
6
4
2
0
-2
-4
-6
0
0.5
1
1.5
2


Pochodna ostatniego elementu
6
5
4
3
2
1
0
0
0.5
1
1.5
2

Punkt 3. – Kod programu (w języku C++)

Do rysowania wykresów zastosowałem bibliotekę „gnuplot”

```
// funkcja zwracająca punkt podziału
double xPodzial(double x, int n) {
    return (x*2.0)/n;
}
```

```
// funkcja zwracająca i-ty element
double e(double i, double x, int n) {
    double result = 0.0;
    if(x > xPodzial( x: i-1,n) && x <= xPodzial( x: i,n)) {
        result = ((n/2.0)*x) - i + 1;
    }
    else if(x > xPodzial( x: i,n) && x < xPodzial( x: i+1,n)) {
        result = ((-1)*((n/2.0)*x)) + i + 1;
    }
    return result;
}
```

```
// funkcja zwracająca pochodną i-tego elementu
double e_pochodna(double i, double x, int n) {
    double result = 0.0;
    if(x > xPodzial( x: i-1,n) && x <= xPodzial( x: i,n)) {
        result = (n/2.0);
    }
    else if(x > xPodzial( x: i,n) && x < xPodzial( x: i+1,n)) {
        result = (-1)*(n/2.0);
    }
    return result;
}
```

```
// funkcja zwracająca prawą stronę równania
double L(double i, int n) {
    return 20*e(i, x: 0.0, n);
}
```

```
// funkcja podcalkowa
double funkcja_podcalkowa(double x, double i, double j, int n) {
    double result = 0.0;
    result = e_pochodna(i, x, n) * e_pochodna( i: j, x, n);
    return result;
}
```

```
//// funkcja całkowania numerycznego przez kwadraturę Gauss-Legendre dla dwóch punktów
double calkuj(double i, double j, double dolnaGranica, double gornaGranica, int n) {
    double result = 0.0;
    double x1 = 1/sqrt( x: 3);
    double x2 = -1/sqrt( x: 3);
    double w1 = 1;
    double w2 = 1;
    double roznica = (gornaGranica - dolnaGranica)/2;
    double suma = (gornaGranica + dolnaGranica)/2;
    result = roznica * (w1*funkcja_podcalkowa( x: roznica*x1 + suma,i,j,n) + w2*funkcja_podcalkowa( x: roznica*x2 + suma,i,j,n));
    return result;
}
```

Zastosowanie kwadratury (dwa punkty) Gauss-Legendre do całkowania numerycznego:

Number of points, n	Points, x_i		Weights, w_i
2	$\pm\frac{1}{\sqrt{3}}$	±0.57735...	1

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}\xi_i + \frac{a+b}{2}\right)$$

Gdzie ξ_i -> to punkt x_i z tabelki powyżej

n = 2

```
// funkcja obliczająca lewą stronę równania
double B(double i, double j, int n) {
    double dolnaGranica;
    // wyznaczanie dolnej granicy całkowania
    if(0 > xPodzial( x: i-1,n) && 0 > xPodzial( x: j-1,n)) {
        dolnaGranica = 0.0;
    }
    else if(xPodzial( x: i-1,n) > 0 && xPodzial( x: i-1,n) > xPodzial( x: j-1,n)) {
        dolnaGranica = xPodzial( x: i-1,n);
    }
    else {
        dolnaGranica = xPodzial( x: j-1,n);
    }
    // wyznaczenie górnej granicy całkowania
    double gornaGranica;
    if(xPodzial( x: i+1,n) < xPodzial( x: j+1,n)) {
        gornaGranica = xPodzial( x: i+1,n);
    }
    else {
        gornaGranica = xPodzial( x: j+1,n);
    }
    double result;
    double calka = calkuj(i, j, dolnaGranica, gornaGranica, n);
    result = ( e(i, x: 0.0, n) * e( i: j, x: 0.0, n) ) - calka;
    return result;
}
```

```
// funkcja tworząca macierz n x n+1 wypełnioną zerami
// aby zadeklarować wstępnie miejsce w pamięci
double** inicjowanie_macierzy(int n) {
    double** Macierz = new double*[n];
    for(int i = 0; i < n; i++) {
        Macierz[i] = new double[n + 1];
        for(int j = 0; j < n + 1; j++) {
            Macierz[i][j] = 0;
        }
    }
    return Macierz;
}
```

```cpp
// tworzenie wektora o długości n wypełnionego zerami
// aby zadeklarować wstępnie miejsce w pamięci
double* inicjowanie_wektora(int n) {
    double* Wektor = new double[n];
    for (int i = 0; i < n; i++) {
        Wektor[i] = 0;
    }
    return Wektor;
}
```

```cpp
// zwalnianie pamięci macierzy
void wyczysc_macierz(double** Macierz, int n) {
    for(int i = 0; i < n; i++) {
        delete[] Macierz[i];
    }
    delete[] Macierz;
}
```

```cpp
// zwalnianie pamięci wektora
void wyczysc_wektor(double* Wektor) {
    delete[] Wektor;
}
```

```cpp
// macierz lewej strony równania
double** wypelnij_macierz(double** Macierz, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            Macierz[i][j] = B(i, j, n);
        }
    }
    return Macierz;
}
```

```cpp
// wektor prawej strony równania
double* wypelnij_wektor(double* Wektor, int n) {
    for(int i = 0; i < n; i++) {
        Wektor[i] = L(i, n);
    }
    return Wektor;
}
```

```cpp
// wypisywanie wektora (do testów)
void wypisz_wektor(double* Wektor, int n) {
    for (int i = 0; i < n; i++) {
        cout << "[" << Wektor[i] << "] ";
        cout << "\n";
    }
}

// wypisywanie macierzy (do testów)
void wypisz_macierz(double** Macierz, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << "[" << Macierz[i][j] << "] ";
        }
        cout << "\n";
    }
}
```

```cpp
// funkcja obliczająca układ równań metodą eliminacji Gaussa
double** eliminacjaGaussa(double** Macierz, double* Wektor, int n) {
    // dodanie wektora prawej strony do macierzy w kolumnie n
    for(int i = 0; i < n; i++) {
        Macierz[i][n] = Wektor[i];
    }
    // eliminacja Gaussa
    for(int i = 0; i < n; i++) {
        // szukam maksymalnej wartości w kolumnie i
        double maks = abs( x: Macierz[i][i]);
        int znalezionyWiersz = i;
        for(int j = i + 1; j < n; j++) {
            double wartosc = abs( x: Macierz[j][i]);
            if(wartosc > maks) {
                znalezionyWiersz = j;
                maks = wartosc;
            }
        }
        // zamiana wierszy
        swap( &: Macierz[i], &: Macierz[znalezionyWiersz]);
        // skalowanie głównego wiersza
        double lider = Macierz[i][i];
        for(int j = i; j <= n; j++) {
            Macierz[i][j] /= lider;
        }
        // eliminacja pozostałych wierszy
        for(int j = 0; j < n; j++) {
            if(j != i) {
                double wspolczynnik = Macierz[j][i];
                for(int k = i; k <= n; k++) {
                    Macierz[j][k] -= wspolczynnik * Macierz[i][k];
                }
            }
        }
    }
    return Macierz;
}
```

```
// funkcja rozwiązująca równanie
double* rozwiaz_rownanie(double** Macierz, double* Wektor, int n) {
    double* Wynik = inicjowanie_wektora( n: n + 1);
    double** rozwiazanaMacierz = eliminacjaGaussa(Macierz, Wektor, n);
    for(int i = 0; i < n; i++) {
        Wynik[i] = rozwiazanaMacierz[i][n];
    }
    return Wynik;
}
```

```
// funkcja wypisująca gotowe punkty
void wypisz_wyniki(double* Wektor, int n) {
    double x = 0.0;
    double skok = 2.0/n;
    for(int i = 0; i <= n; i++) {
        cout<<"(x = "<<x<<" , y = "<<Wektor[i]<<") \n";
        x+=skok;
    }
}
```

```cpp
// funkcja rysująca wykresy
void rysuj_wykresy(double** Macierz, double* Wynik, int n) {

    // inicjuję programy do tworzenia wykresów
    Gnuplot Rozwiazanie;
    Gnuplot WszystkieElementy;
    Gnuplot PochodnaPierwsza, PochodnaOstatnia, PochodnaSrodkowa;

    // stałe będące ograniczeniami osi wykresów
    double minX = 0.0, maxX = 2.0;
    double maxY = Wynik[0], minY = 0.0;


    // tworzę dane rozwiązania do wykresu
    vector<double> RozwiazanieX, RozwiazanieY;
    double x = 0.0;
    double skok = 2.0 / n;
    for (int i = 0; i <= n; i++) {
        RozwiazanieX.push_back(x);
        RozwiazanieY.push_back(Wynik[i]);
        x += skok;
    }

    // rysuję wykres rozwiązanego problemu
    Rozwiazanie << "set title 'Wykres rozwiązania problemu' \n";
    Rozwiazanie << "set grid \n";
    Rozwiazanie << "set xlabel 'X' \n";
    Rozwiazanie << "set ylabel 'Y' \n";
    Rozwiazanie << "set xrange [" << minX << ":" << maxX << "] \n";
    Rozwiazanie << "set yrange [" << minY << ":" << maxY << "] \n";
    Rozwiazanie << "plot '-' with linespoints title '' \n";
    Rozwiazanie.send( arg: make_tuple( &: RozwiazanieX, RozwiazanieY));
```

```cpp
// tworzę dane wszystkich elementów do wykresu
vector<double> pomocnicza;
vector<vector<double>> wszystkieElem;
vector<double> xWszystkich;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        pomocnicza.push_back(Macierz[i][j]);
    }
    wszystkieElem.push_back(pomocnicza);
    pomocnicza.clear();
}
```

```
// rysuję wykres wszystkich elementów
WszystkieElementy << "set title 'Wykres wszystkich elementów' \n";
WszystkieElementy << "set grid \n";
WszystkieElementy << "set xrange [" << minX << ":" << maxX << "] \n";
WszystkieElementy << "plot ";
for (int i = 0; i <= n; i++) {
    WszystkieElementy << " '-' with lines notitle";
    if (i < n) {
        WszystkieElementy << ", ";
    }
}
WszystkieElementy << "\n";
for (int i = 0; i < n; i++) {
    wszystkieElem[i].push_back(0);
    WszystkieElementy.send( arg: make_tuple( &: RozwiazanieX, wszystkieElem[i]));
}
vector<double> last;
for (int i = 0; i < n; i++) {
    last.push_back(0);
}
last.push_back(1);
WszystkieElementy.send( arg: make_tuple( &: RozwiazanieX, last));
```

```
    //tworzę dane do wykresów pochodnych
    vector<double> PochodneX;
    vector<double> pochodnaPocz, pochodnaSrod, pochodnaOst;
    int srodkowyElement = n / 2;
    for (double i = 0.001; i <= 2.000; i+=0.001) {
        PochodneX.push_back(i);
        pochodnaPocz.push_back(e_pochodna( i: 0.0,  x: i, n));
        pochodnaSrod.push_back(e_pochodna( i: srodkowyElement,  x: i, n));
        pochodnaOst.push_back(e_pochodna( i: n,  x: i, n));
    }

    // rysuję wykresy pochodnych
    // (pierwsza, srodkowa = n/2, ostatnia)
    PochodnaPierwsza << "set title 'Pochodna pierwszego elementu' \n";
    PochodnaPierwsza << "set grid \n";
    PochodnaPierwsza << "set xrange [" << minX << ":" << maxX << "] \n";
    PochodnaPierwsza << "plot '-' with lines title '' \n";
    PochodnaPierwsza.send( arg: make_tuple( &: PochodneX, pochodnaPocz));
    if (n % 2 == 1) srodkowyElement++;
    PochodnaSrodkowa << "set title 'Pochodna srodkowego elementu (" << srodkowyElement << ")' \n";
    PochodnaSrodkowa << "set grid \n";
    PochodnaSrodkowa << "set xrange [" << minX << ":" << maxX << "] \n";
    PochodnaSrodkowa << "plot '-' with lines title '' \n";
    PochodnaSrodkowa.send( arg: make_tuple( &: PochodneX, pochodnaSrod));

    PochodnaOstatnia << "set title 'Pochodna ostatniego elementu' \n";
    PochodnaOstatnia << "set grid \n";
    PochodnaOstatnia << "set xrange [" << minX << ":" << maxX << "] \n";
    PochodnaOstatnia << "plot '-' with lines title '' \n";
    PochodnaOstatnia.send( arg: make_tuple( &: PochodneX, pochodnaOst));
}
```

```
int main() {
    int n;
    cout << "Witaj w programie do obliczania rownania transportu ciepla metoda Galerkina \n";
    cout << "Ograniczenia programu: \n";
    cout << "(n musi byc liczba naturalna wieksza badz rowna 3) \n";
    cout << "Przyklady czasow rozwiazan: \n";
    cout << "(dla n = 1000 obliczenia trwaja okolo 3 sekundy) \n";
    cout << "(dla n = 1500 obliczenia trwaja okolo 7 sekund) \n";
    cout << "(dla n = 2000 obliczenia trwaja okolo 15 sekund) \n";
    cout << "(dla n = 2500 obliczenia trwaja okolo 28 sekund) \n";
    cout << "(dla n = 3000 obliczenia trwaja okolo 45 sekund) \n \n";
    cout << "Prosze wprowadzic parametr n:";
    cin >> n;
    double** Macierz = inicjowanie_macierzy(n);
    Macierz = wypelnij_macierz(Macierz, n);
    double* Wektor = inicjowanie_wektora(n);
    Wektor = wypelnij_wektor(Wektor, n);
    double* Wynik = rozwiaz_rownanie(Macierz, Wektor, n);

    wypisz_wyniki( Wektor: Wynik, n);

    rysuj_wykresy(Macierz, Wynik, n);

    // zwalnianie pamięci
    wyczysc_macierz(Macierz, n);
    wyczysc_wektor(Wektor);
    wyczysc_wektor( Wektor: Wynik);
}
```