

Metody różnic skończonych  
Sprawozdanie nr 4  
Zagadnienie Poissona i metody iteracyjne

Kacper Dąbrowski

Czerwiec 2024

**Spis treści**

<b>1 Równanie cząstkowe II rzędu</b>	<b>3</b>
<b>2 Równanie eliptyczne</b>	<b>3</b>
<b>3 Metoda MRS dla równania Poissona</b>	<b>3</b>
<b>4 Metody iteracyjne dla równań eliptycznych</b>	<b>5</b>
4.1 Metoda Jakobiego . . . . .	6
4.2 Metoda Gaussa-Seidela . . . . .	6
4.3 Metoda nadrelaksacji Younga . . . . .	7
4.4 Metoda Peacemana-Rahforda . . . . .	7
<b>5 Siatki adaptacyjne</b>	<b>9</b>
<b>6 Programy do metod</b>	<b>10</b>
<b>7 Zadanie 1</b>	<b>18</b>
7.1 Metoda MRS . . . . .	18
7.2 Metoda Jakobiego . . . . .	19
7.3 Metoda Gaussa-Seidela . . . . .	19
7.4 Metoda nadrelaksacji Younga . . . . .	20
7.5 Metoda Peacemana-Rachforda . . . . .	21
7.6 Siatki adaptacyjne . . . . .	22
7.7 Zestawienie wyników . . . . .	23

<b>8 Zadanie 2</b>	<b>26</b>
8.1 Metoda MRS . . . . .	26
8.2 Metoda Jakobiego . . . . .	27
8.3 Metoda Gaussa-Seidela . . . . .	28
8.4 Metoda nadrelaksacji Younga . . . . .	29
8.5 Metoda Peacemana-Rachforda . . . . .	30
8.6 Siatki adaptacyjne . . . . .	31
8.7 Zestawienie wyników . . . . .	32
<b>9 Zadanie 3</b>	<b>35</b>
9.1 Metoda MRS . . . . .	35
9.2 Metoda Jakobiego . . . . .	36
9.3 Metoda Gaussa-Seidela . . . . .	37
9.4 Metoda nadrelaksacji Younga . . . . .	38
9.5 Metoda Peacemana-Rachforda . . . . .	39
9.6 Siatki adaptacyjne . . . . .	40
9.7 Zestawienie wyników . . . . .	41
<b>10 Wnioski</b>	<b>44</b>

## 1 Równanie cząstkowe II rzędu

**Uwaga.** Tak jak w poprzednich sprawozdaniach stosowany jest skrócony zapis. Dla funkcji dwóch zmiennych następująco:  $\psi(x_i, y_j) = \psi_{ij}$ ,  $\psi(x_i + h, y_j) = \psi_{i+1,j}$

Równaniem cząstkowym II rzędu nazywamy równanie postaci

$$\varphi(x, y, \frac{\partial \psi}{\partial x}, \frac{\partial \psi}{\partial y}, \frac{\partial^2 \psi}{\partial x^2}, \frac{\partial^2 \psi}{\partial y^2}, \frac{\partial^2 \psi}{\partial x \partial y}, \psi) = f(x, y)$$

Z twierdzenia Schwarza pochodne mieszane są równe, więc tylko jedna została zapisana.

## 2 Równanie eliptyczne

Niech dane będzie równanie cząstkowe postaci

$$a(x, y) \frac{\partial^2 \psi}{\partial x^2} + 2b(x, y) \frac{\partial^2 \psi}{\partial x \partial y} + c(x, y) \frac{\partial^2 \psi}{\partial y^2} + d(x, y) \frac{\partial \psi}{\partial x} + e(x, y) \frac{\partial \psi}{\partial y} + f(x, y) \psi = g(x, y) \quad (1)$$

Dla równania (1) dany jest wyróżnik wzorem

$$\Delta(x, y) = \Delta = [b(x, y)]^2 - a(x, y)c(x, y) \quad (2)$$

Równaniem eliptycznym nazywamy równanie cząstkowe II rzędu, dla którego wyróżnik (2) jest mniejszy od 0. Równania te są równaniami stacjonarnymi, to oznacza, że są niezależne od czasu. Dla  $\Delta = -1$  otrzymujemy równanie Poissona postaci

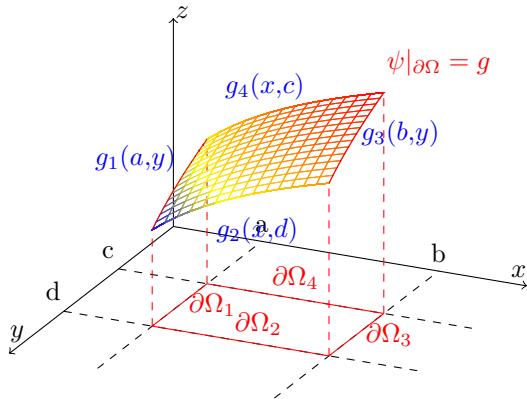
$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = g(x, y)$$

Dla  $g(x, y) = 0$  równanie nazywamy równaniem Laplace'a.

## 3 Metoda MRS dla równania Poissona

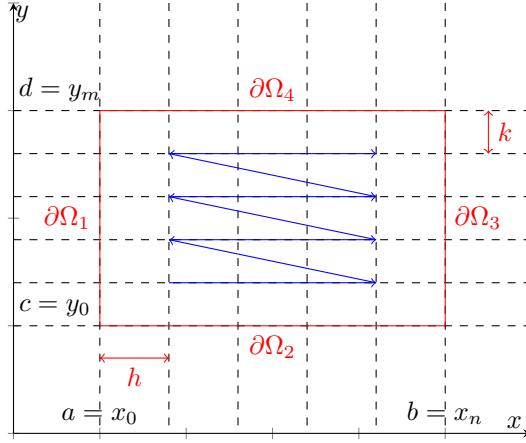
Mamy dany problem postaci

$$\begin{cases} \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = f(x, y) \\ \psi|_{\partial\Omega} = g \end{cases}$$



$$\Omega = \{(x,y) \in \mathbb{R} : a \leq x \leq b \text{ & } c \leq y \leq d\}$$

Budujemy siatkę w następujący sposób



Wartości w węzłach będziemy aproksymować schematami centralnymi drugiego rzędu dla równań cząstkowych. Wyglądarki one następująco

$$\frac{\partial^2 \psi}{\partial x^2} \approx D_{c,x}^2 \psi = \frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2}, \quad \frac{\partial^2 \psi}{\partial y^2} \approx D_{c,y}^2 \psi = \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{k^2}$$

Zatem dla równania Poissona możemy zapisać, że

$$\text{dla } x_i, y_j : \frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{k^2} = f_{ij}$$

a po przekształceniach

$$\frac{1}{h^2}(\psi_{i+1,j} + \psi_{i-1,j}) + \frac{1}{k^2}(\psi_{i,j+1} + \psi_{i,j-1}) - 2(\frac{1}{h^2} + \frac{1}{k^2})\psi_{i,j} = f_{ij}$$

Te schematy przeliczają punkty w węzłach bazujące na wartościach dookoła. Wiemy, że brzeg jest znany więc jak przeliczamy punkty w rogach i przy brzegach to możemy znane wartości przenieść na drugą stronę równania. W ten sposób utworzy się nam macierz zblokowana postaci

$$\mathbf{L} = \begin{bmatrix} -2(k^2 + h^2) & k^2 & & & \\ k^2 & -2(k^2 + h^2) & k^2 & & \\ & k^2 & -2(k^2 + h^2) & k^2 & \\ & & \ddots & \ddots & \ddots \\ & & & k^2 & -2(k^2 + h^2) \end{bmatrix}_{n-1 \times n-1}$$

$$\mathbf{H} = \begin{bmatrix} h^2 & & & \\ & h^2 & & \\ & & \ddots & \\ & & & h^2 \end{bmatrix}_{n-1 \times n-1} \quad \mathbf{A} = \begin{bmatrix} \mathbf{L} & \mathbf{H} & & \\ \mathbf{H} & \mathbf{L} & \mathbf{H} & \\ & \ddots & \ddots & \ddots \\ & & \mathbf{H} & \mathbf{L} \end{bmatrix}_{m-1 \times m-1}$$

$$\mathbf{f} = \begin{bmatrix} h^2 k^2 f_{1,1} - k^2 g_{0,1} - h^2 g_{1,0} \\ h^2 k^2 f_{2,1} - h^2 g_{2,0} \\ \vdots \\ h^2 k^2 f_{n-1,1} - h^2 g_{n,1} - k^2 g_{n-1,0} \\ h^2 k^2 f_{1,2} - k^2 g_{0,2} \\ h^2 k^2 f_{2,2} \\ \vdots \\ h^2 k^2 f_{n-1,2} - k^2 g_{n,2} \\ \vdots \\ h^2 k^2 f_{1,n-1} - k^2 g_{0,n-1} - h^2 g_{1,n} \\ h^2 k^2 f_{2,n-1} - h^2 g_{2,n} \\ \vdots \\ h^2 k^2 f_{n-1,n-1} - k^2 g_{n,n-1} - h^2 g_{n-1,n} \end{bmatrix}_{(m-1)(n-1) \times (m-1)(n-1)}$$

## 4 Metody iteracyjne dla równań eliptycznych

Polegają na zbliżaniu się do rozwiązania wraz z kolejną iteracją. Przy metodach iteracyjnych patrzymy, w którym momencie nasze rozwiązanie osiągnie, względem poprzedniego, wartości odpowiednio małe. Przy tych metodach musimy uważać na dobór tolerancji i ograniczyć liczbę iteracji, ponieważ niektóre algorytmy mogą nawet nie osiągnąć zadanej toleracji. Będziemy rozpatrywać równanie Poissona z zagadnieniem Dirichleta

$$\begin{cases} \nabla^2 \psi = f \text{ na } \Omega \\ \psi|_{\partial\Omega} = g \end{cases}$$

## 4.1 Metoda Jakobiego

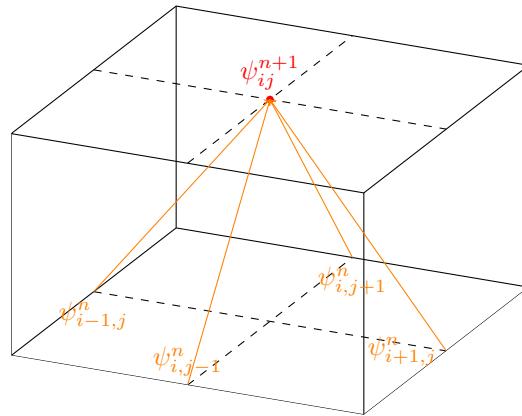
Jest to podstawowa metoda iteracyjna, korzystamy tutaj z równania

$$k^2\psi_{i+1,j} + k^2\psi_{i-1,j} + h^2\psi_{i,j+1} + h^2\psi_{i,j-1} - 2h^2\psi_{ij} - 2k^2\psi_{ij} = h^2k^2f_{ij}$$

Po odpowiednich przekształceniach otrzymujemy postać iteracyjną

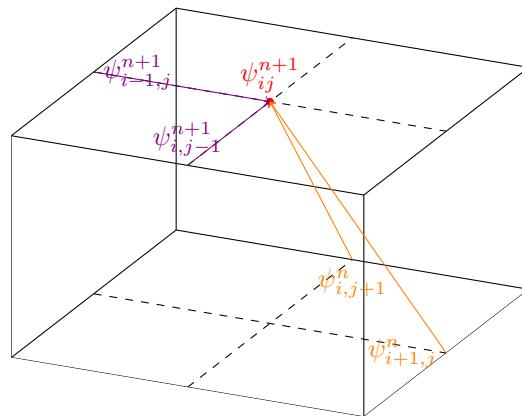
$$\psi_{ij}^{n+1} = \frac{1}{2(h^2+k^2)}(k^2\psi_{i+1,j}^n + k^2\psi_{i-1,j}^n + h^2\psi_{i,j+1}^n + h^2\psi_{i,j-1}^n - h^2k^2f_{ij})$$

Dla pierwszej iteracji należy dobrać warunki początkowe. Oczywiście na brzegach mamy określone dokładne rozwiązanie. Metoda korzysta z punktów następująco



## 4.2 Metoda Gaussa-Seidela

Ta metoda korzysta z tego samego podejścia co metoda Jakobiego, ale wykorzystuje punkty wyliczane na bierząco. To znaczy



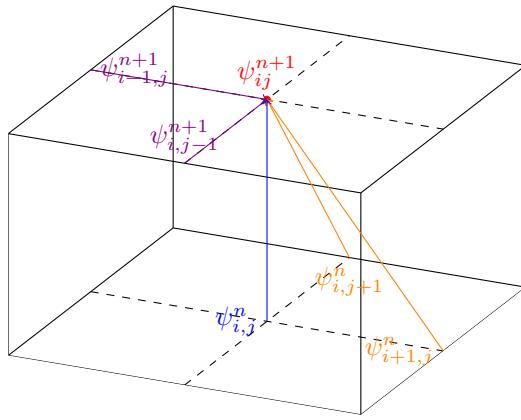
Zarówno metoda Gaussa-Seidela oraz Jakobiego są metodami jawnymi. Oznacza to, że nie musimy rozwiązywać żadnych układów równań, ponieważ w każdych kolejnych iteracjach korzystamy ze znanych węzłów.

### 4.3 Metoda nadrelaksacji Younga

Ta metoda jest modyfikacją metody Gaussa-Seidela. Polega ona na wprowadzeniu parametru  $\omega$ , który wpływa pozytywnie na zbieżność. Wzór rekurencyjny wygląda następująco

$$\psi_{ij}^{n+1} = \frac{1}{2(h^2+k^2)}\omega(k^2\psi_{i+1,j}^n + k^2\psi_{i-1,j}^n + h^2\psi_{i,j+1}^n + h^2\psi_{i,j-1}^n) + (1-\omega)\psi_{ij}^n - \frac{h^2k^2f_{ij}}{2(h^2+k^2)}$$

Na rysunku różni się to w taki sposób



Aby wyliczyć optymalny parametr Younga należy skorzystać ze wzorów

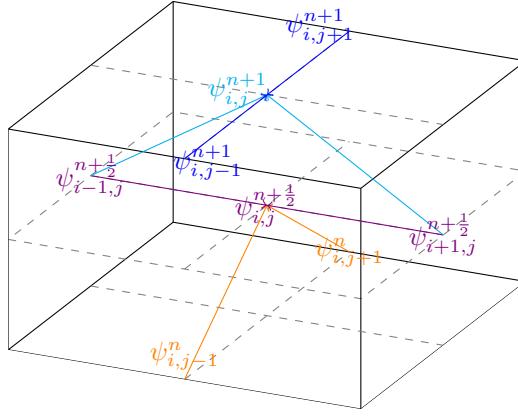
$$\lambda = \frac{1}{4}(\cos \frac{\pi}{M} + \cos \frac{\pi}{N})^2$$

$$\omega_{opt} = 1 + \frac{\lambda}{(1+\sqrt{1-\lambda})^2}$$

Metoda Younga należy do metod jawnych.

### 4.4 Metoda Peacemana-Rahforda

Metoda Peacemana-Rahforda implementuje poziomy połówkowe. Działanie tego algorytmu wygląda następująco



Oznacza to, że aby rozwiązać równanie musimy najpierw rozwiązać układ równań utworzony z poziomu najniższego. Wtedy otrzymamy wszystkie wartości na poziomie połówkowym, a potem możemy rozwiązać końcowy układ równań. Rozwiązanie to prezentuje się następująco

$$\psi_{i,j}^{n+\frac{1}{2}} = r \left( \psi_{i-1,j}^{n+\frac{1}{2}} + \psi_{i+1,j}^{n+\frac{1}{2}} - 2\psi_{i,j}^{n+\frac{1}{2}} \right) + r \left( \psi_{i,j-1}^n + \psi_{i,j+1}^n - 2\psi_{i,j}^n \right) + \psi_{i,j}^n \quad (3)$$

$$\psi_{i,j}^{n+1} = r \left( \psi_{i-1,j}^{n+1} + \psi_{i+1,j}^{n+1} - 2\psi_{i,j}^{n+1} \right) + r \left( \psi_{i,j-1}^{n+\frac{1}{2}} + \psi_{i,j+1}^{n+\frac{1}{2}} - 2\psi_{i,j}^{n+\frac{1}{2}} \right) + \psi_{i,j}^{n+\frac{1}{2}} \quad (4)$$

Parametr  $r$  należy dobrąć w specyficzny sposób i ulega on zmianie z każdą iteracją. Dla  $n-1$  węzłów wewnętrznych obszaru obliczamy parametr  $\alpha = \sin^2(\frac{\pi}{2n})$ . Gdy mamy już wyliczony parametr  $\alpha$  potrzebujemy wiedzieć jak duży będzie utworzony wektor  $r$ . Jego długość wyliczamy z własności

$$k < \frac{\log \alpha}{2 \log(\sqrt{2}) - 1}$$

a każdy element  $r_i$  jest wyliczany następująco

$$r_i = \alpha^{\frac{1-2i}{2k}}$$

Elementy wektora  $r$  są używane cyklicznie co oznacza

$$r_1, r_2, \dots, r_k, r_1, \dots$$

Rozważając równanie (3) jesteśmy w stanie wyznaczyć pierwszy układ równań. Po przeniesieniu niewiadomych na jedną stronę otrzymujemy następującą postać macierzową

$$\mathbf{A} = \begin{bmatrix} 1+2r & -r & & & \\ -r & 1+2r & -r & & \\ & -r & 1+2r & -r & \\ & & \ddots & \ddots & \ddots \\ & & & -r & 1+2r \end{bmatrix} \psi^{n+\frac{1}{2}} = \begin{bmatrix} \psi_{1,j}^{n+\frac{1}{2}} \\ \psi_{2,j}^{n+\frac{1}{2}} \\ \vdots \\ \psi_{n-1,j}^{n+\frac{1}{2}} \end{bmatrix}$$

$$\hat{\psi}^n = \begin{bmatrix} r(\psi_{1,j-1}^n + \psi_{1,j+1}^n - 2\psi_{1,j}^n) + \psi_{1,j}^n + r\psi_{0,j}^{n+\frac{1}{2}} \\ r(\psi_{2,j-1}^n + \psi_{2,j+1}^n - 2\psi_{2,j}^n) + \psi_{2,j}^n \\ \vdots \\ r(\psi_{n-1,j-1}^n + \psi_{n-1,j+1}^n - 2\psi_{n-1,j}^n) + \psi_{n-1,j}^n + r\psi_{n,j}^{n+\frac{1}{2}} \end{bmatrix}$$

Teraz wystarczy rozwiązać układ równań liniowych dla  $j = 1, 2, \dots, n-1$

$$\mathbf{A}\psi^{n+\frac{1}{2}} = \hat{\psi}^n$$

Po rozwiązaniu tego układu równań możemy rozważyć równość (4) i utworzyć kolejny układ równań

$$\mathbf{A} = \begin{bmatrix} 1+2r & -r & & & \\ -r & 1+2r & -r & & \\ & -r & 1+2r & -r & \\ & & \ddots & \ddots & \ddots \\ & & & -r & 1+2r \end{bmatrix} \psi^{n+1} = \begin{bmatrix} \psi_{i,1}^{n+1} \\ \psi_{i,2}^{n+1} \\ \vdots \\ \psi_{i,m-1}^{n+1} \end{bmatrix}$$

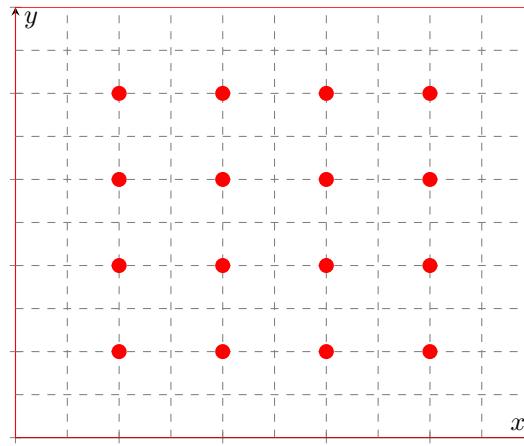
$$\hat{\psi}^{n+\frac{1}{2}} = \begin{bmatrix} r(\psi_{i-1,1}^{n+\frac{1}{2}} + \psi_{i+1,1}^{n+\frac{1}{2}} - 2\psi_{i,1}^{n+\frac{1}{2}}) + \psi_{i,1}^{n+\frac{1}{2}} + r\psi_{i,0}^{n+1} \\ r(\psi_{i-1,2}^{n+\frac{1}{2}} + \psi_{i+1,2}^{n+\frac{1}{2}} - 2\psi_{i,2}^{n+\frac{1}{2}}) + \psi_{i,2}^{n+\frac{1}{2}} \\ \vdots \\ r(\psi_{i-1,m-1}^{n+\frac{1}{2}} + \psi_{i+1,m-1}^{n+\frac{1}{2}} - 2\psi_{i,m-1}^{n+\frac{1}{2}}) + \psi_{i,m-1}^{n+\frac{1}{2}} + r\psi_{i,m}^{n+1} \end{bmatrix}$$

Teraz końcowo rozwiązuje kolejny układ równań dla  $i = 1, 2, \dots, m-1$

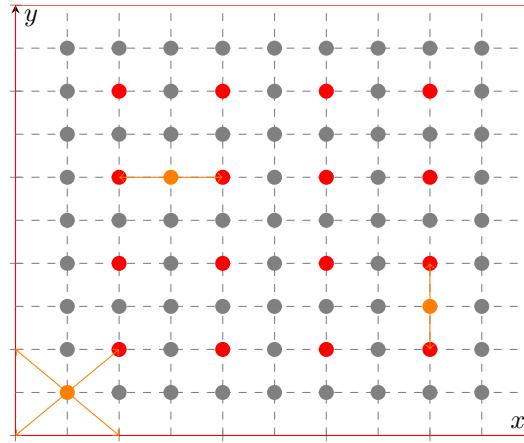
$$\mathbf{A}\psi^{n+1} = \hat{\psi}^{n+\frac{1}{2}}$$

## 5 Siatki adaptacyjne

Siatki adaptacyjne wykorzystują znane wcześniej metody iteracyjne. Na samym początku musimy skorzystać z jakiejś metody iteracyjnej dla małej liczby iteracji. Teraz możemy przejść do zagęszczania siatki. W tym celu podwajamy liczbę punktów  $n$  i  $m$  i dodajemy 1. Wtedy rozważamy taką siatkę



Po wyliczeniu kilku iteracji wyliczamy punkty, które zostały utworzone po zagęszczeniu siatki. Robimy to porzez średnią z sąsiednich punktów w taki sposób



Teraz możemy użyć tej siatki jako punkty początkowe do innej metody iteracyjnej. W naszym przypadku użyta została metoda Jakobiego, a potem metoda Peacemana-Rahforda.

## 6 Programy do metod

Wszystkie programy zostały napisane w języku Python. Konstruktory klas nie różnią się wcale od siebie (poza MRS dla równań eliptycznych). Tak samo z sporządzaniem wykresów.

Biblioteki potrzebne do działania programu:

- MatPlotLib - obsługa wykresów z Matlaba
- Numpy - obsługa numeryki w Pythonie

Na rysunku 1 został przedstawiony konstruktor klasy *Poisson*. Za argumenty przyjmuje funkcję źródłową  $f$  oraz funkcje określone na krawędziach prostokąta na którym rozważamy problem. Rysunek 2 przedstawia dodatkowe zmienne dodane do metod iteracyjnych.



```

class Poisson:
    ...
    Solves Poisson partial differential equation defined on rectangle.
    ...
    def __init__(self, f, lowerU, upperU, leftU, rightU) -> None:
        ...
        Initiation of Poisson partial differential equation on rectangle.

    Args:
        f (lambda function): function f
        lowerU (lambda function): lower boundary of rectangle
        upperU (lambda function): upper boundary of rectangle
        leftU (lambda function): left boundary of rectangle
        rightU (lambda function): right boundary of rectangle...
        self.f = f
        self.lowerU = lowerU
        self.upperU = upperU
        self.leftU = leftU
        self.rightU = rightU

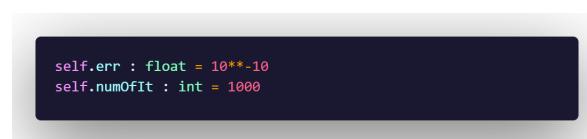
        self.solution = None
        self.label : str = ""

        self.sol_exist : bool = False
        self.set_n : bool = False
        self.set_interval : bool = False

        self.n : int = 1
        self.m : int = 1
        self.x0 : float = 0
        self.xn : float = 1
        self.y0 : float = 0
        self.ym : float = 1
        self.k : float = 0
        self.h : float = 0

        self.x : np.ndarray
        self.y : np.ndarray
        self.unknown_x : np.ndarray
        self.unknown_y : np.ndarray
    
```

Rysunek 1: Kod konstruktora klasy



Rysunek 2: Dodatkowe zmienne dla metod iteracyjnych

Dalsze modyfikacje dotyczą także funkcji odpowiadającej za ustawienia zmiennych. W tym wypadku mamy do ustawienia parę dodatkowych parametrów ( $m, k$ , punkty  $y_m$ )

Rysunek 3 przedstawia zedytowaną funkcję do wyświetlania wykresów 3D.

```
def show(self) -> str:
    ...
    Plots the solution.

    Returns:
        str: Error norm if can.''
    self.plt_config()
    U = self.solve()
    X, Y = np.meshgrid(self.x, self.y)

    lowBoundX : list[float] = list(map(self.lowerU, self.x))
    upBoundX : list[float] = list(map(self.upperU, self.x))
    lowBoundY : list[float] = list(map(self.leftU, self.y))
    upBoundY : list[float] = list(map(self.rightU, self.y))

    lowBoundX = np.array(lowBoundX)
    upBoundX = np.array(upBoundX)
    lowBoundY = np.array(lowBoundY)
    upBoundY = np.array(upBoundY)

    U = np.vstack((U, lowBoundX[1:len(self.x)-1]))
    U = np.vstack((upBoundX[1:len(self.x)-1], U))

    U = np.hstack((np.flipud(lowBoundY.reshape([self.m+2, 1])), U))
    U = np.hstack((U, np.flipud(upBoundY.reshape([self.m+2, 1]))))

    fig = plt.figure()

    if self.sol_exist:
        gs = GridSpec(2, 2, figure=fig)
        ax = fig.add_subplot(gs[0, 0], projection='3d')
        ax1 = fig.add_subplot(gs[1, 0], projection='3d')
        ax1.plot_wireframe(X, Y, self.solution(X, Y), label="Rozwiązańe dokładne", color="#00853f")
        ax1.set_title("Rozwiązańe dokładne")
        ax2 = fig.add_subplot(gs[:, 1], projection='3d')
        ax2.set_title("Błąd")
        surf = ax2.plot_surface(X, Y, np.abs(self.solution(X, Y) - np.flipud(U)), cmap="Reds",
                               linewidth=0, antialiased=False, label="Błąd")
        fig.colorbar(surf, shrink=0.5, aspect=10)
    else:
        ax = fig.add_subplot(projection='3d')
        ax.plot_wireframe(X, Y, np.flipud(U), label="Rozwiązańe numeryczne", color="#f68c14")
        ax.set_title("Rozwiązańe numeryczne")
        fig.suptitle(self.label)
        plt.grid(True)
        plt.show()
    if self.sol_exist:
        return LA.norm(self.solution(X, Y) - np.flipud(U), np.inf)
```

Rysunek 3: Funkcja obsługująca wykresy 3D

Rysunki 4-9 przedstawiają zaimplementowane algorytmy metod

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.''
    if not all([self.set_interval, self.set_n]):
        raise ValueError("Cannot solve equation without setting parameters")
    T : np.ndarray = np.diag(-2*(1/self.h**2+1/self.k**2)*np.ones(self.n)) + np.diag
    (1/self.h**2*np.ones(self.n-1),-1) + np.diag(1/self.h**2*np.ones(self.n-1),1)
    B : np.ndarray = np.diag(1/self.k**2*np.ones(self.n))
    A : np.ndarray = np.kron(np.diag(np.ones(self.m)),T) + np.kron(np.diag(np.ones(
    self.m-1),-1),B) + np.kron(np.diag(np.ones(self.m-1),1),B)
    F : list = []
    for j in range(self.m):
        for i in range(self.n):
            values : float = self.f(self.unknown_x[i],self.unknown_y[j])
            if j == 0:
                values -= self.lowerU(self.unknown_x[i])/self.k**2
            elif j == self.m-1:
                values -= self.upperU(self.unknown_x[i])/self.k**2
            if i == 0:
                values -= self.leftU(self.unknown_y[j])/self.h**2
            elif i == self.n-1:
                values -= self.rightU(self.unknown_y[j])/self.h**2
            F.append(values)
    F : np.ndarray = np.array(F)
    U : np.ndarray = LA.solve(A,F)
    U = U.reshape([self.m,self.n])
    U = np.flipud(U)
    return U

```

Rysunek 4: Metoda MRS dla równania Poissona

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.''''
    if not all([self.set_interval, self.set_n]):
        raise ValueError("Cannot solve equation without setting parameters")
    U : np.ndarray = np.zeros([self.n+2, self.m+2])
    U[:, 0] = np.array(list(map(self.lowerU, self.x)))
    U[:, -1] = np.array(list(map(self.upperU, self.x)))
    U[0, :] = np.array(list(map(self.leftU, self.y)))
    U[-1, :] = np.array(list(map(self.rightU, self.y)))
    for _ in range(self.numOfIt):
        U1 = copy.deepcopy(U)
        for i in range(1, self.n+1):
            for j in range(1, self.m+1):
                U1[i, j] = ((U[i+1, j]+U[i-1, j])/self.h**2 + (U[i, j+1]+U[i, j-1])/self.
k**2 - self.f(self.x[i], self.y[j]))/(2*(1/self.h**2+1/self.k**2))
        e = LA.norm(U-U1,np.inf)
        U = copy.deepcopy(U1)
        if e <= self.err:
            print(_)
            print(e)
            return U
    print(e)
    return U

```

Rysunek 5: Metoda Jakobiego

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.'''''
    if not all([self.set_interval, self.set_n]):
        raise ValueError("Cannot solve equation without setting parameters")
    U : np.ndarray = np.zeros([self.n+2, self.m+2])
    U[:, 0] = np.array(list(map(self.lowerU, self.x)))
    U[:, -1] = np.array(list(map(self.upperU, self.x)))
    U[0, :] = np.array(list(map(self.leftU, self.y)))
    U[-1, :] = np.array(list(map(self.rightU, self.y)))
    for _ in range(self.numOfIt):
        U1 = copy.deepcopy(U)
        for i in range(1, self.n+1):
            for j in range(1, self.m+1):
                U1[i, j] = ((U[i+1, j]+U[i-1, j])/self.h**2 + (U[i, j+1]+U[i, j-1])/
self.k**2 - self.f(self.x[i], self.y[j]))/(2*(1/self.h**2+1/self.k**2))
        e = LA.norm(U-U1,np.inf)
        U = copy.deepcopy(U1)
        if e <= self.err:
            print(_)
            print(e)
            return U
    print(e)
    return U

```

Rysunek 6: Metoda Gaussa-Seidela

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.''
    if not all([self.set_interval, self.set_n]):
        raise ValueError("Cannot solve equation without setting parameters")
    U : np.ndarray = np.zeros([self.n+2, self.n+2])
    U[:, 0] = np.array(list(map(self.lowerU, self.x)))
    U[:, -1] = np.array(list(map(self.upperU, self.x)))
    U[0, :] = np.array(list(map(self.leftU, self.y)))
    U[-1, :] = np.array(list(map(self.rightU, self.y)))
    lamb : float = 1/4*(np.cos(np.pi/self.n)+np.cos(np.pi/self.m))**2
    omega : float = 1 + lamb/(1+np.sqrt(1-lamb))**2
    for k in range(self.numOfIt):
        U1 = copy.deepcopy(U)
        for i in range(1, self.n+1):
            for j in range(1, self.m+1):
                U1[i, j] = omega*((U[i+1, j]+U1[i-1, j])/self.h**2 + (U[i, j+1]+U1[i, j-1])/self.k**2)/(2*(1/self.h**2+1/self.k**2)) + (1-omega)*U[i, j] - self.f(self.x[i], self.y[j])/(2*(1/self.h**2+1/self.k**2))
        e = LA.norm(U-U1, np.inf)
        e = LA.norm(U-U1, np.inf)
        U = copy.deepcopy(U1)
        if e <= self.err:
            print(f"Policzono w {k} kroków")
            print(e)
            return U
    print(e)
    return U

```

Rysunek 7: Metoda nadrelaksacji Younga

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable. ***
        if not all([self.set_interval, self.set_n]):
            raise ValueError("Cannot solve equation without setting parameters")
    temp : int = 0; e : float = 10
    alpha : float = (np.sin(np.pi/(2*np.max([self.m,self.n]))))**2
    pk : int = 1
    while (np.sqrt(2)-1)**(2*pk) >= alpha:
        pk += 1
    pr : list[float] = []
    for i in range(pk):
        pr.append(alpha*((1-(i+1))/(2*pk)))
    r : np.ndarray = np.kron(np.ones(self.numOfIt*self.m*self.n),np.array(pr))

    U = np.transpose(np.zeros((self.n+2,self.m+2)))
    U[0,:] = np.array(list(map(self.lowerU, self.x)))
    U[-1,:] = np.array(list(map(self.upperU, self.x)))
    U[:,0] = np.array(list(map(self.leftU, self.y)))
    U[:, -1] = np.array(list(map(self.rightU, self.y)))

    U = np.transpose(U)
    while temp <= self.numOfIt and e >= self.err:
        U1 : list = []
        U2 : list = []
        v1 = (1+2*r[temp])*np.ones(self.m)
        v2 = -r[temp]*np.ones(self.m-1)
        A = np.diag(v1) + np.diag(v2,-1) + np.diag(v2,1)

        for j in range(1,self.n+1):
            F = []
            for i in range(1,self.m+1):
                F.append(U[j,i] + r[temp]*(U[j+1,i]+U[j-1,i]-2*U[j,i]))
            F[0] += r[temp]*U[j,0]
            F[-1] += r[temp]*U[j,-1]
            F = np.array(F)
            N1 = LA.solve(A,F)
            U1.append(N1)
        U1 = np.vstack((U1, U[-1, 1:-1]))
        U1 = np.vstack(([0,1:-1],U1))
        U1 = np.hstack((U1, np.atleast_2d(U[:, -1]).T))
        U1 = np.hstack((np.atleast_2d(U[:,0]).T, U1))

        v1 = (1+2*r[temp])*np.ones(self.n)
        v2 = -r[temp]*np.ones(self.m-1)
        A = np.diag(v1) + np.diag(v2,-1) + np.diag(v2,1)

        for j in range(1,self.m+1):
            F = []
            for i in range(1,self.n+1):
                F.append(U[i,j] + r[temp]*(U[i,j+1]+U[i,j-1]-2*U[i,j]))
            F[0] += r[temp]*U[0,j]
            F[-1] += r[temp]*U[-1,j]
            F = np.array(F)
            N2 = LA.solve(A,F)
            U2.append(N2)
        U2 = np.transpose(U2)
        U2 = np.vstack((U2, U[-1, 1:-1]))
        U2 = np.vstack(([0,1:-1],U2))
        U2 = np.hstack((U2, np.atleast_2d(U[:, -1]).T))
        U2 = np.hstack((np.atleast_2d(U[:,0]).T, U2))

        e = LA.norm(U2-U, np.inf)
        U = copy.deepcopy(U2)
        temp += 1
    print("Policzono w {} kroków".format(temp))
    return U

```

Rysunek 8: Metoda Peacemana-Rahforda

```

def solve(self) -> np.ndarray:
    ...
    Solves the problem.

    Returns:
        float: solution depending on solve type variable. ***
        if not all([self.set_interval, self.set_n]):
            raise ValueError("Cannot solve equation without setting parameters")
    temp : int = 0; e : float = 10
    alpha : float = (np.sin(np.pi/(2*np.max([self.m,self.n]))))**2
    pk : int = 1
    while (np.sqrt(2)-1)**(2*pk) >= alpha:
        pk += 1
    pr : list[float] = []
    for i in range(pk):
        pr.append(alpha*((1-(i+1))/(2*pk)))
    r : np.ndarray = np.kron(np.ones(self.numOfIt*self.m*self.n),np.array(pr))

    U = np.transpose(np.zeros((self.n+2,self.m+2)))
    U[0,:] = np.array(list(map(self.lowerU, self.x)))
    U[-1,:] = np.array(list(map(self.upperU, self.x)))
    U[:,0] = np.array(list(map(self.leftU, self.y)))
    U[:, -1] = np.array(list(map(self.rightU, self.y)))

    U = np.transpose(U)
    while temp <= self.numOfIt and e >= self.err:
        U1 : list = []
        U2 : list = []
        v1 = (1+2*r[temp])*np.ones(self.m)
        v2 = -r[temp]*np.ones(self.m-1)
        A = np.diag(v1) + np.diag(v2,-1) + np.diag(v2,1)

        for j in range(1,self.n+1):
            F = []
            for i in range(1,self.m+1):
                F.append(U[j,i] + r[temp]*(U[j+1,i]+U[j-1,i]-2*U[j,i]))
            F[0] += r[temp]*U[j,0]
            F[-1] += r[temp]*U[j,-1]
            F = np.array(F)
            N1 = LA.solve(A,F)
            U1.append(N1)
        U1 = np.vstack((U1, U[-1, 1:-1]))
        U1 = np.vstack(([0,1:-1],U1))
        U1 = np.hstack((U1, np.atleast_2d(U[:, -1]).T))
        U1 = np.hstack((np.atleast_2d(U[:,0]),U1))

        v1 = (1+2*r[temp])*np.ones(self.n)
        v2 = -r[temp]*np.ones(self.m-1)
        A = np.diag(v1) + np.diag(v2,-1) + np.diag(v2,1)

        for j in range(1,self.m+1):
            F = []
            for i in range(1,self.n+1):
                F.append(U[i,j] + r[temp]*(U[i,j+1]+U[i,j-1]-2*U[i,j]))
            F[0] += r[temp]*U[0,j]
            F[-1] += r[temp]*U[-1,j]
            F = np.array(F)
            N2 = LA.solve(A,F)
            U2.append(N2)
        U2 = np.transpose(U2)
        U2 = np.vstack((U2, U[-1, 1:-1]))
        U2 = np.vstack(([0,1:-1],U2))
        U2 = np.hstack((U2, np.atleast_2d(U[:, -1]).T))
        U2 = np.hstack((np.atleast_2d(U[:,0]),U2))

        e = LA.norm(U2-U, np.inf)
        U = copy.deepcopy(U2)
        temp += 1
    print("Policzono w {} kroków".format(temp))
    return U

```

Rysunek 9: Siatki adaptacyjne

## 7 Zadanie 1

Rozważamy równanie Laplace'a postaci

$$\begin{cases} \nabla^2 \psi = 0 \\ \psi|_{x,y=0} = 0 \\ \psi|_{x,y=1} = x \\ \psi|_{x=0,y} = 0 \\ \psi|_{x=1,y} = y \end{cases} \quad \Omega = \{(x,y) : 0 \leq x \leq 1, 0 \leq y \leq 1\}$$

Rozwiążanie analityczne

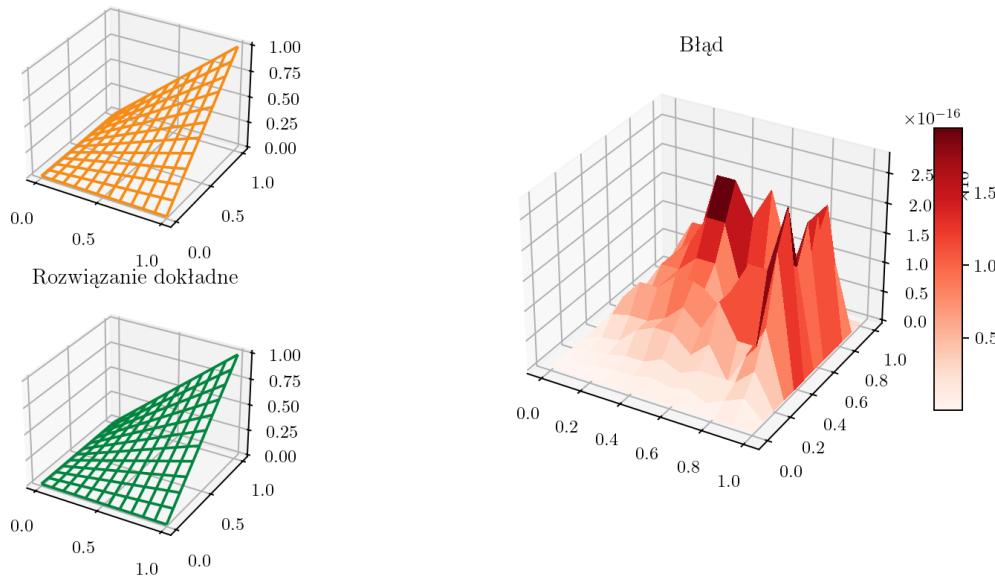
$$\psi_{dok} = xy$$

### 7.1 Metoda MRS

Rysunek 10 przedstawia rozwiązanie zadania metodą MRS dla  $n = 10$  oraz  $m = 10$

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0$$

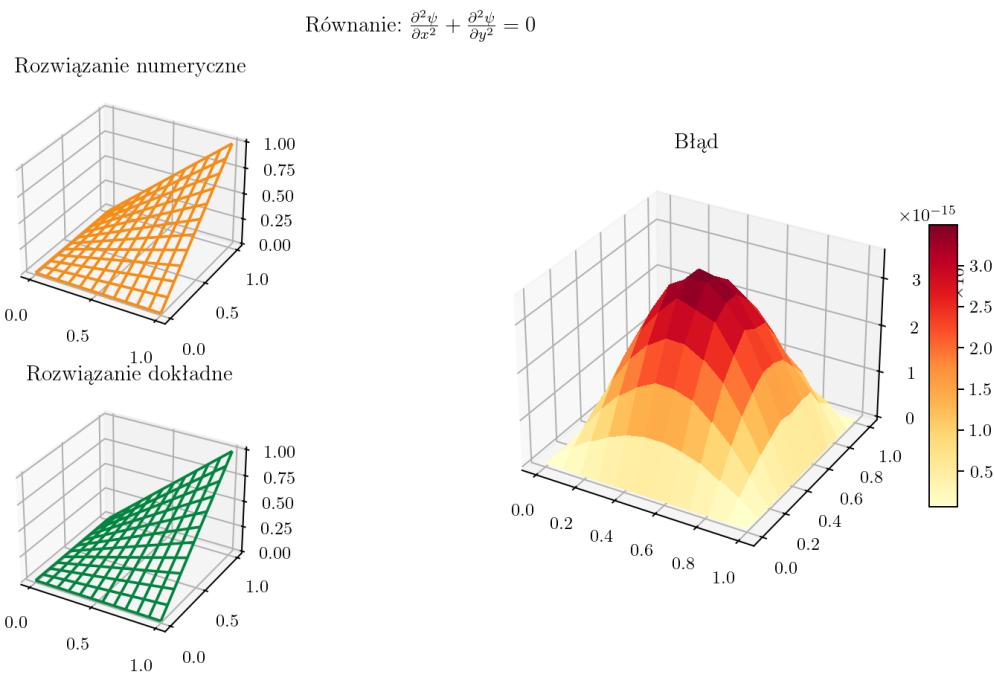
Rozwiążanie numeryczne



Rysunek 10: Wykres z rozaniem dla  $n = 10$  oraz  $m = 10$

## 7.2 Metoda Jakobiego

Rysunek 11 przedstawia rozwiązanie zadania metodą Jakobiego dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.



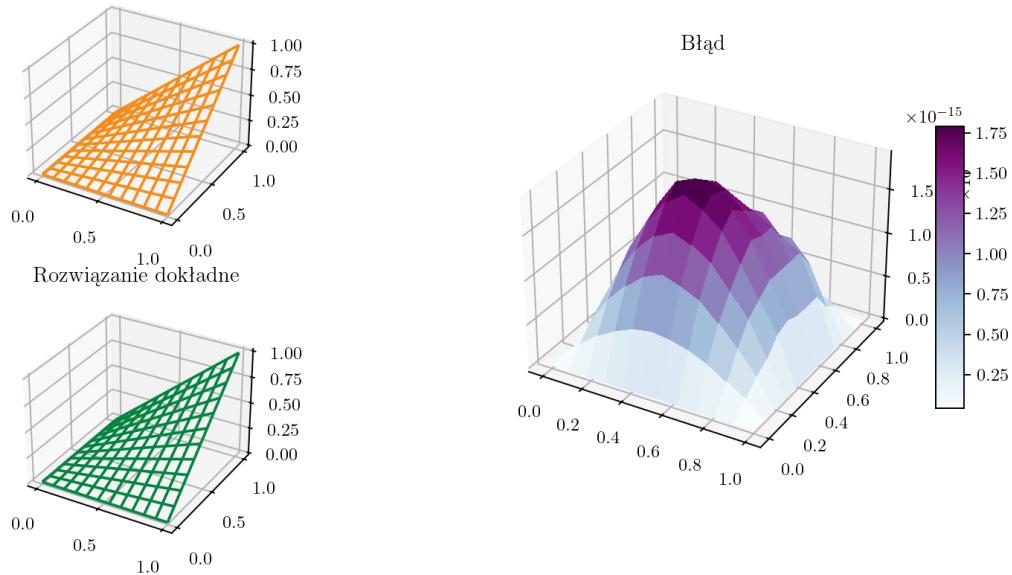
Rysunek 11: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

## 7.3 Metoda Gaussa-Seidela

Rysunek 12 przedstawia rozwiązanie zadania metodą Gaussa-Seidela dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0$$

Rozwiązańe numeryczne



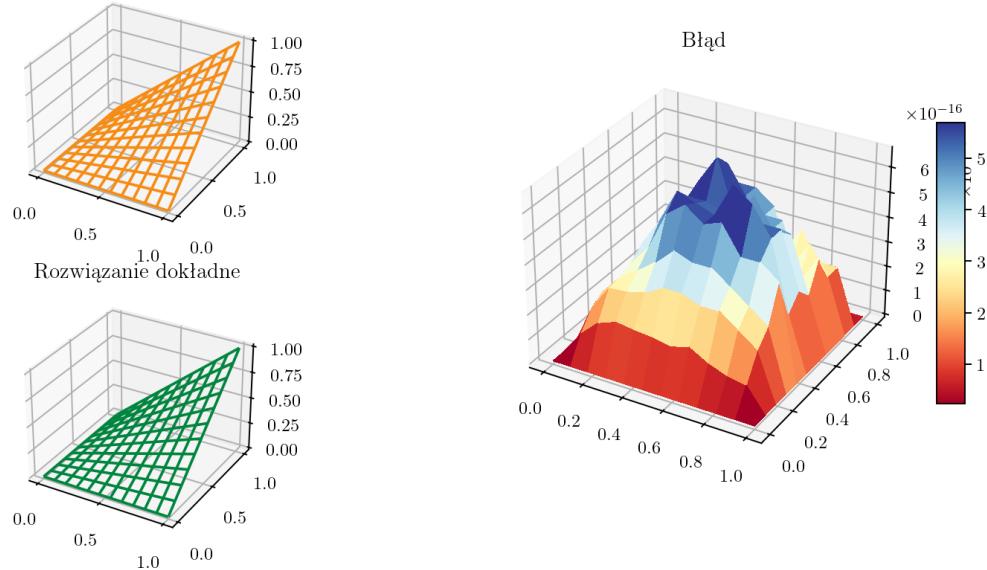
Rysunek 12: Wykres z rozwiązańem dla  $n = 10$  oraz  $m = 10$

#### 7.4 Metoda nadrelaksacji Younga

Rysunek 13 przedstawia rozwiązanie zadania metodą nadrelaksacji Younga dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0$$

Rozwiązanie numeryczne



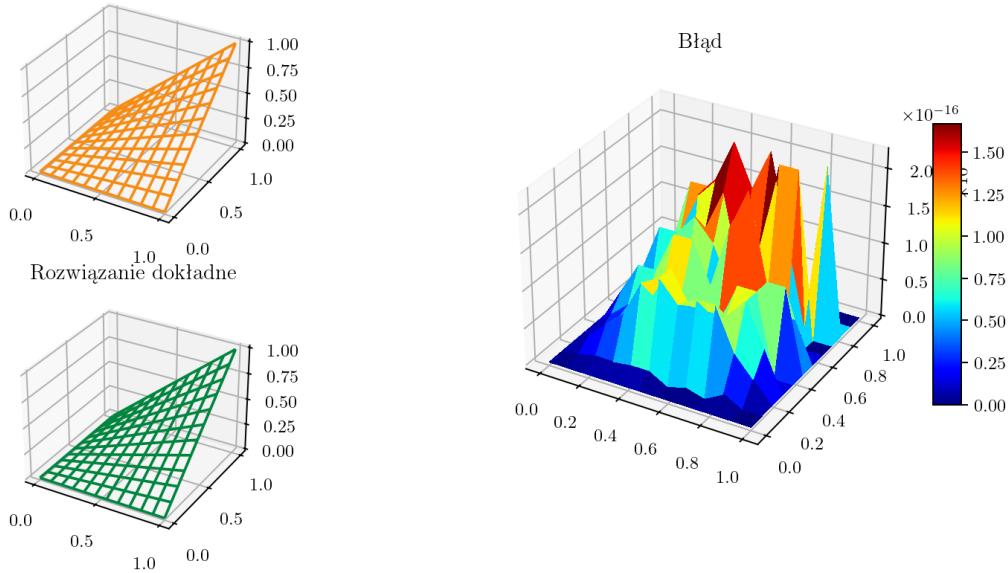
Rysunek 13: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

## 7.5 Metoda Peacemana-Rachforda

Rysunek 14 przedstawia rozwiązanie zadania metodą Peacemana-Rachforda dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0$$

Rozwiązańe numeryczne



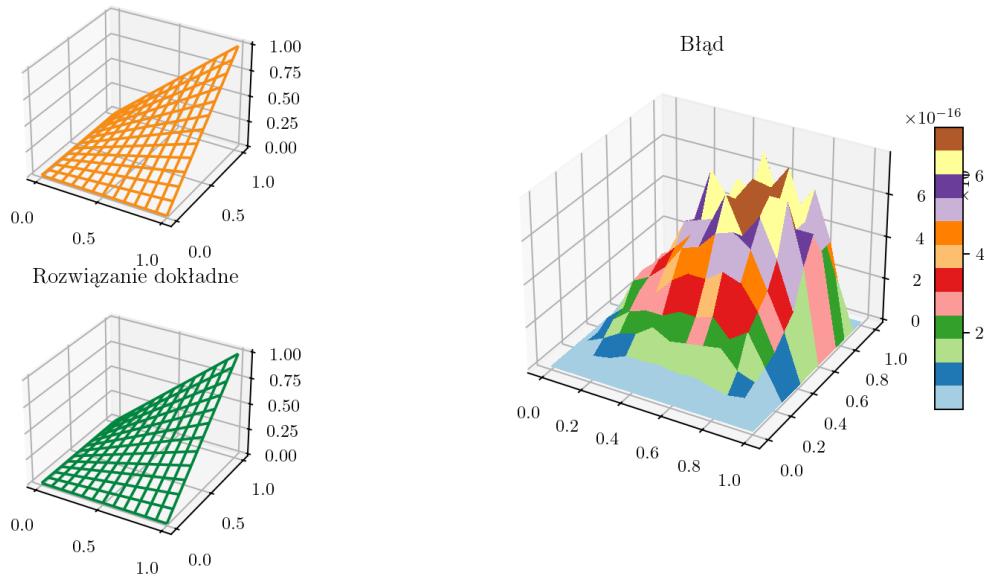
Rysunek 14: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

## 7.6 Siatki adaptacyjne

Rysunek 15 przedstawia rozwiązanie zadania siatkami adaptacyjnymi dla  $n = 5$ ,  $m = 5$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0$$

Rozwiązańe numeryczne



**Rysunek 15:** Wykres z rozwiązaniem dla  $n = 5$  oraz  $m = 5$

## 7.7 Zestawienie wyników

Tabele od 1 do 3 zawierają zestawienie wyników dla różnych  $m$  oraz  $n$  (tolerancja  $10^{-15}$  i 10000 iteracji)

Tabela 1: Zestawienie czasów działań algorytmów

Czas [s]							
$n$	$m$	Metoda MRS	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	0,0000	0,0185	0,0110	0,0100	0,0339	0,0040
10	10	0,0040	0,2686	0,1402	0,0713	0,0952	0,0578
20	20	0,0063	3,0607	1,7923	17,4799	49,8631	15,2215
30	30	0,0329	14,3844	7,7617	37,5619	127,1649	32,5078
40	40	0,1247	48,7358	23,8962	63,5070	192,5389	52,6825
50	50	0,4009	70,8053	55,2160	91,1348	454,1799	82,2498

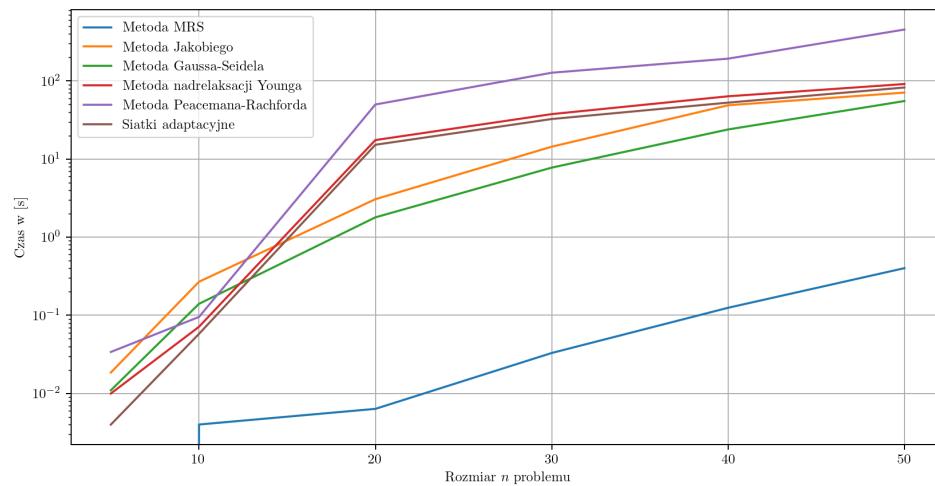
Tabela 2: Zestawienie dokładności względem rozwiązania analitycznego

Dokładność względem rozwiązania							
$n$	$m$	Metoda MRS	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	3,61E-16	5,76E-15	1,98E-15	8,33E-16	6,80E-16	8,05E-16
10	10	1,50E-15	2,49E-14	1,34E-14	4,61E-15	1,39E-15	5,66E-15
20	20	6,22E-15	6,76E-14	2,82E-14	1,97E-14	3,87E-15	1,69E-14
30	30	1,79E-14	1,93E-13	1,12E-13	4,67E-15	1,48E-14	5,78E-15
40	40	1,76E-14	1,99E-12	3,97E-13	1,85E-13	3,28E-14	1,48E-13
50	50	4,50E-14	7,48E-08	5,70E-13	2,36E-13	2,34E-14	3,48E-13

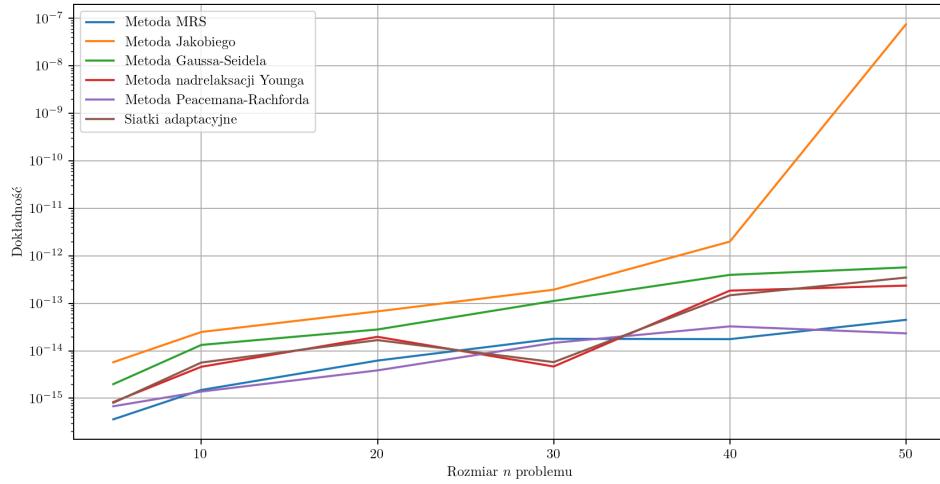
Tabela 3: Zestawienie liczby iteracji metod

Liczba iteracji						
$n$	$m$	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	229	119	59	35	38
10	10	783	403	98	53	135
20	20	2830	1451	10000	10000	10000
30	30	6104	3124	10000	10000	10000
40	40	10000	5402	10000	10000	10000
50	50	10000	8279	10000	10000	10000

Rysunki 16 i 17 przedstawiają zestawienie rozwiązań analitycznego oraz zestawienie czasów w postaci wykresów.



Rysunek 16: Zestawienie czasów



Rysunek 17: Zestawienie dokładności

## 8 Zadanie 2

Rozważamy równanie Poissona postaci

$$\begin{cases} \nabla^2 \psi = (x^2 + y^2) + e^{xy} \\ \psi|_{x,y=0} = 1 \\ \psi|_{x,y=1} = e^x \\ \psi|_{x=0,y} = 1 \\ \psi|_{x=2,y} = e^{2y} \end{cases} \quad \Omega = \{(x,y) : 0 \leq x \leq 2, 0 \leq y \leq 1\}$$

Rozwiązańanie analityczne

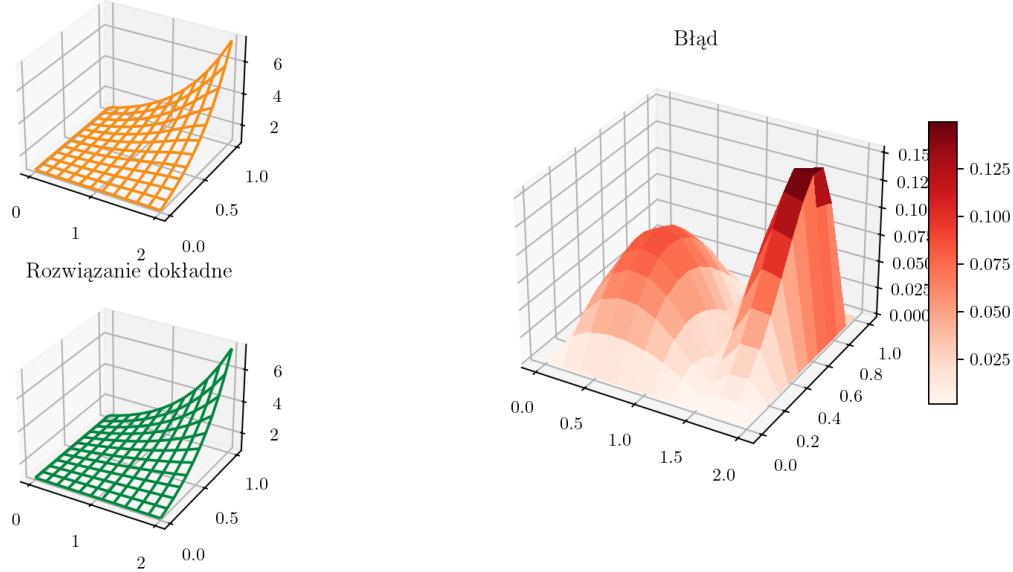
$$\psi_{dok} = e^{xy}$$

### 8.1 Metoda MRS

Rysunek 18 przedstawia rozwiązanie zadania metodą MRS dla  $n = 10$  oraz  $m = 10$

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązańe numeryczne



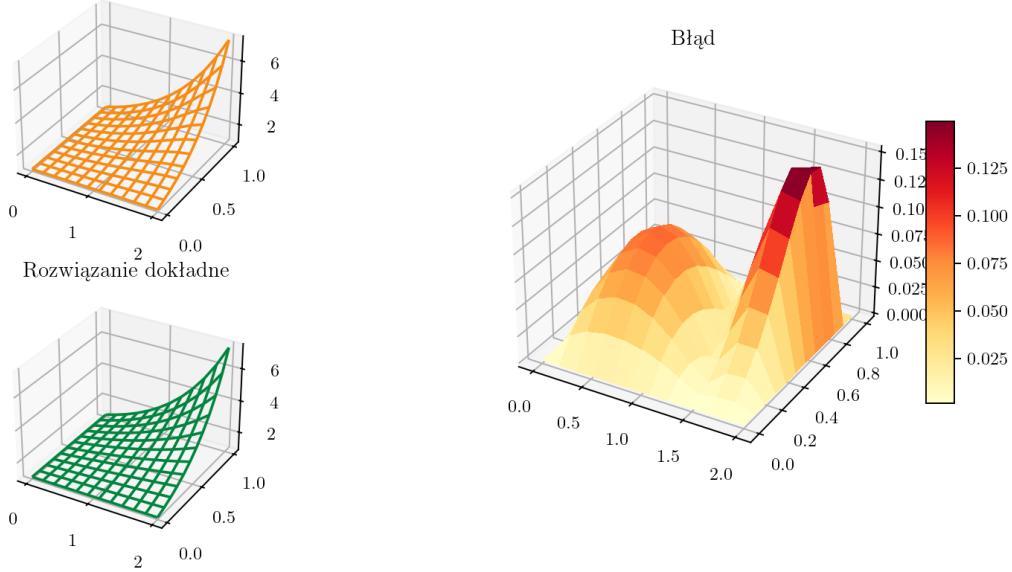
**Rysunek 18:** Wykres z rozwiązańem dla  $n = 10$  oraz  $m = 10$

## 8.2 Metoda Jakobiego

Rysunek 19 przedstawia rozwiązanie zadania metodą Jakobiego dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązańe numeryczne



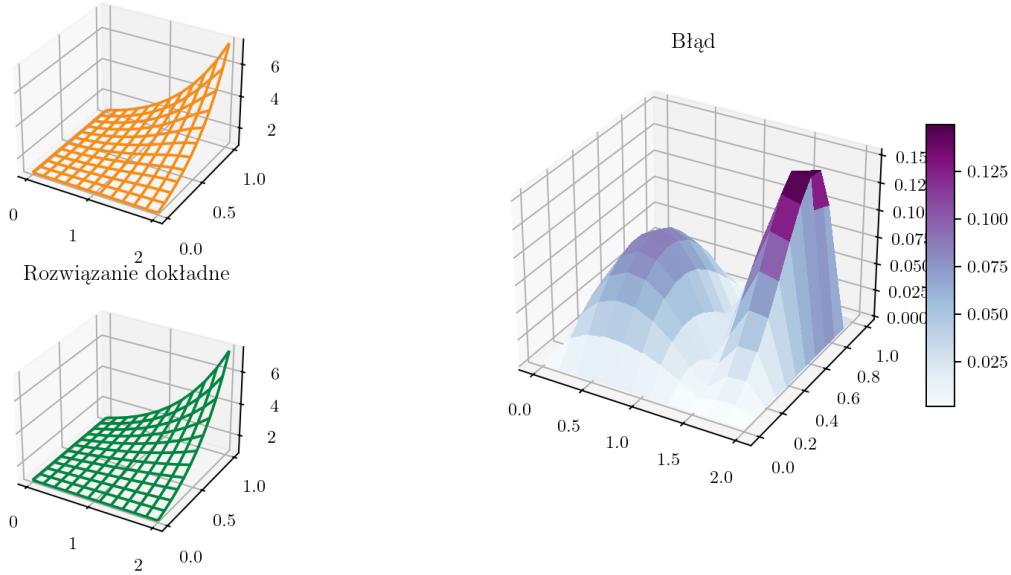
Rysunek 19: Wykres z rozwiązańem dla  $n = 10$  oraz  $m = 10$

### 8.3 Metoda Gaussa-Seidela

Rysunek 20 przedstawia rozwiązańe zadania metodą Gaussa-Seidela dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązańe numeryczne



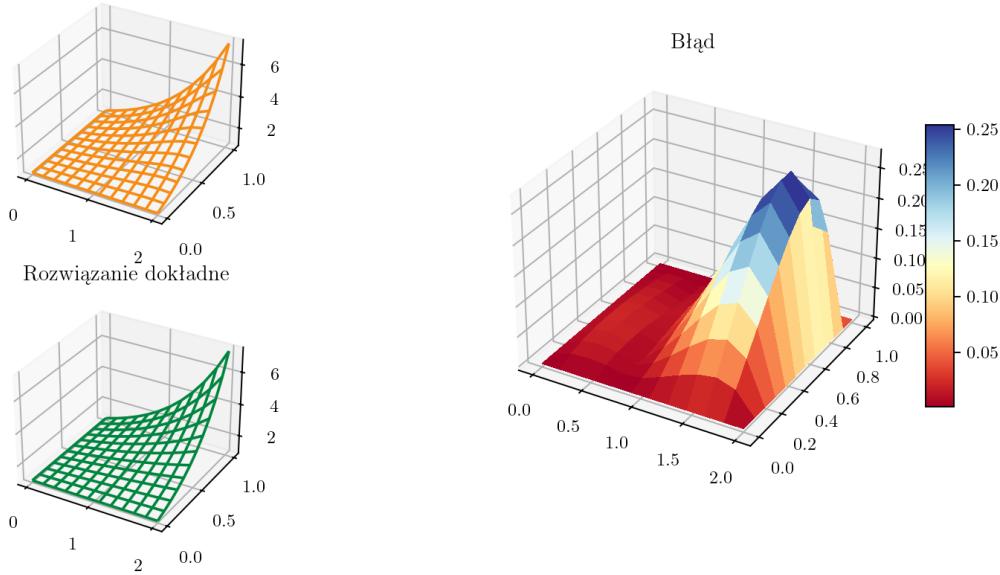
Rysunek 20: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

#### 8.4 Metoda nadrelaksacji Younga

Rysunek 21 przedstawia rozwiązanie zadania metodą nadrelaksacji Younga dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązańe numeryczne



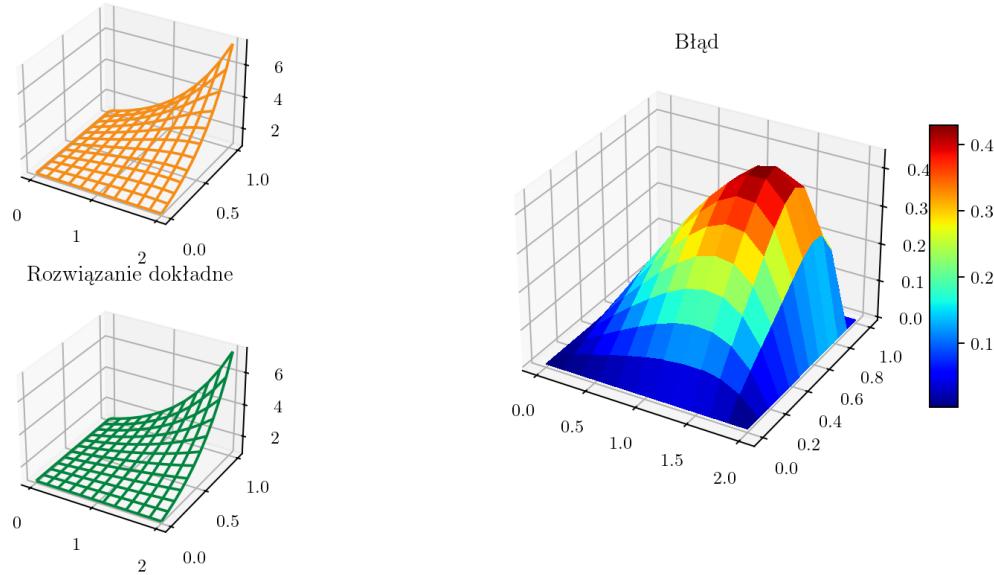
Rysunek 21: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

## 8.5 Metoda Peacemana-Rachforda

Rysunek 22 przedstawia rozwiązanie zadania metodą Peacemana-Rachforda dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązań numeryczne



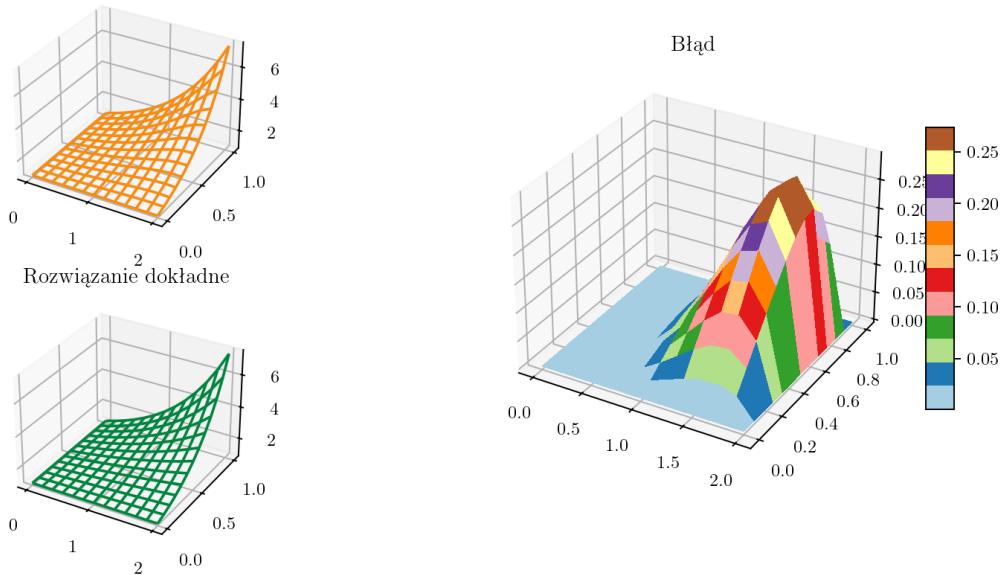
Rysunek 22: Wykres z rozwiązań dla  $n = 10$  oraz  $m = 10$

## 8.6 Siatki adaptacyjne

Rysunek 23 przedstawia rozwiązanie zadania siatkami adaptacyjnymi dla  $n = 5$ ,  $m = 5$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = (x^2 + y^2) + e^{xy}$$

Rozwiązańe numeryczne



Rysunek 23: Wykres z rozwiązańem dla  $n = 5$  oraz  $m = 5$

## 8.7 Zestawienie wyników

Tabele od 4 do 6 zawierają zestawienie wyników dla różnych  $m$  oraz  $n$  (tolerancja  $10^{-10}$  i 10000 iteracji)

Tabela 4: Zestawienie czasów działań algorytmów

Czas [s]							
$n$	$m$	Metoda MRS	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	0,0000	0,0309	0,0232	0,0169	0,0544	0,0112
10	10	0,1316	0,5845	0,3248	0,0840	0,1220	0,1068
20	20	0,1384	4,9550	2,1627	0,3198	0,6561	0,5255
30	30	0,5246	17,8244	10,8062	1,3249	1,9741	1,0919
40	40	0,3563	55,0162	32,0814	2,5088	4,7123	2,5416
50	50	0,5537	119,8372	86,7084	4,7489	10,3188	4,5858

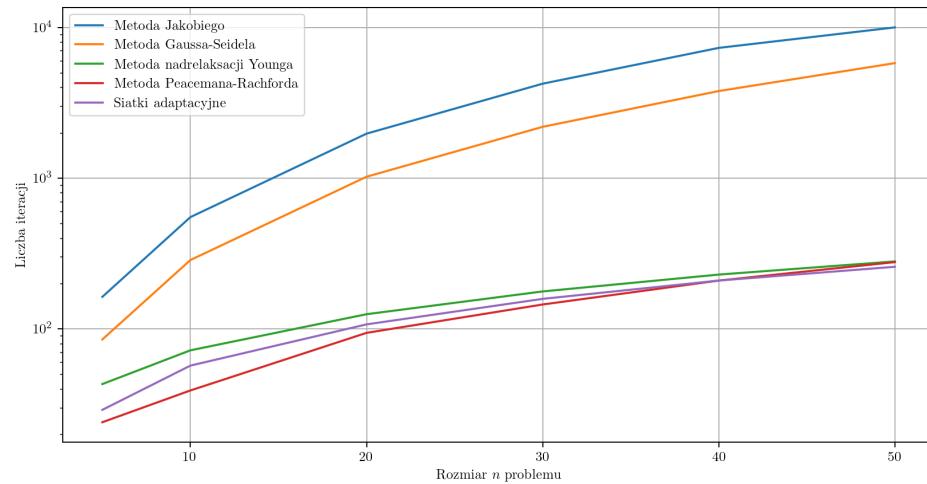
Tabela 5: Zestawienie dokładności względem rozwiązania analitycznego

Dokładność względem rozwiązania							
$n$	$m$	Metoda MRS	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	0,3630	87,6133	87,6133	87,6133	87,6133	87,6133
10	10	0,7479	137,3835	137,3835	137,3835	137,3835	147,4621
20	20	1,4938	238,7093	238,7093	238,7093	238,7093	248,8780
30	30	2,2319	340,5017	340,5017	340,5017	340,5017	350,6897
40	40	2,9589	442,4197	442,4197	442,4197	442,4197	452,6149
50	50	3,6889	544,3894	544,3894	544,3894	544,3894	554,5881

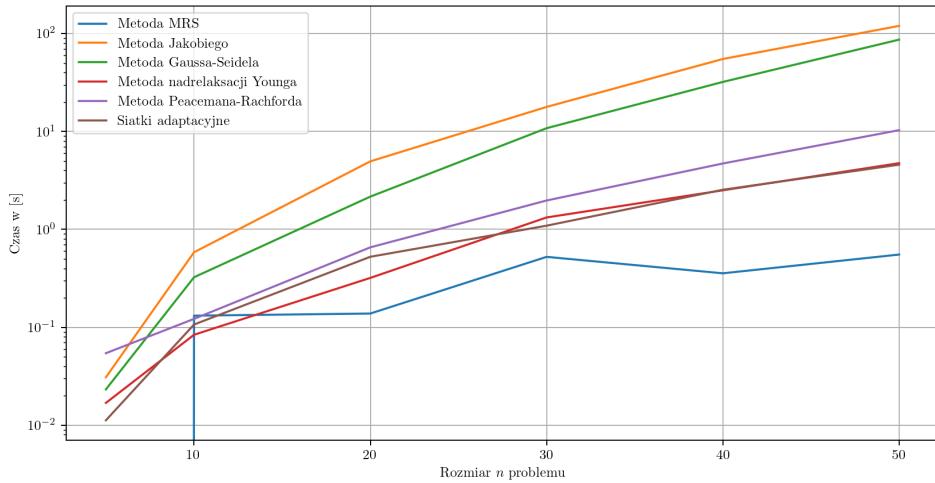
Tabela 6: Zestawienie liczby iteracji metod

Liczba iteracji						
$n$	$m$	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	163	85	43	24	29
10	10	551	286	72	39	57
20	20	1972	1021	125	94	107
30	30	4231	2189	177	145	158
40	40	7312	3782	229	209	209
50	50	10000	5794	280	277	258

Rysunki 24 i 25 przedstawiają zestawienie liczby iteracji oraz zestawienie czasów w postaci wykresów.



Rysunek 24: Zestawienie czasów



Rysunek 25: Zestawienie liczby iteracji

## 9 Zadanie 3

Rozważamy równanie Poissona postaci

$$\begin{cases} \nabla^2 \psi = \frac{x}{y} + \frac{y}{x} \\ \psi|_{x,y=1} = x \ln x \\ \psi|_{x,y=2} = x \ln 4x^2 \quad \Omega = \{(x,y) : 1 \leq x \leq 2, 1 \leq y \leq 2\} \\ \psi|_{x=1,y} = y \ln y \\ \psi|_{x=2,y} = 2y \ln 2y \end{cases}$$

Rozwiązanie analityczne

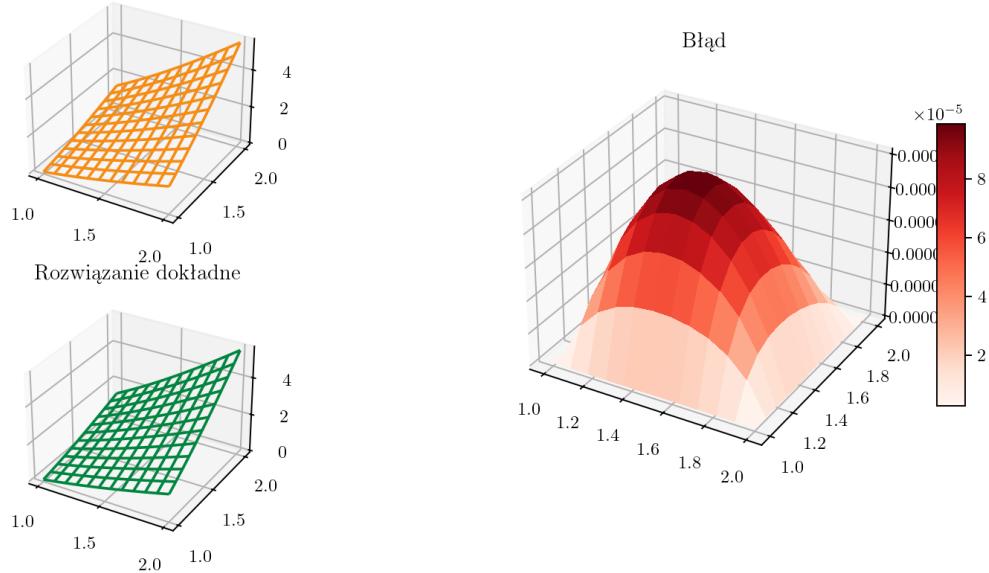
$$\psi_{dok} = xy \ln xy$$

### 9.1 Metoda MRS

Rysunek 26 przedstawia rozwiązanie zadania metodą MRS dla  $n = 10$  oraz  $m = 10$

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



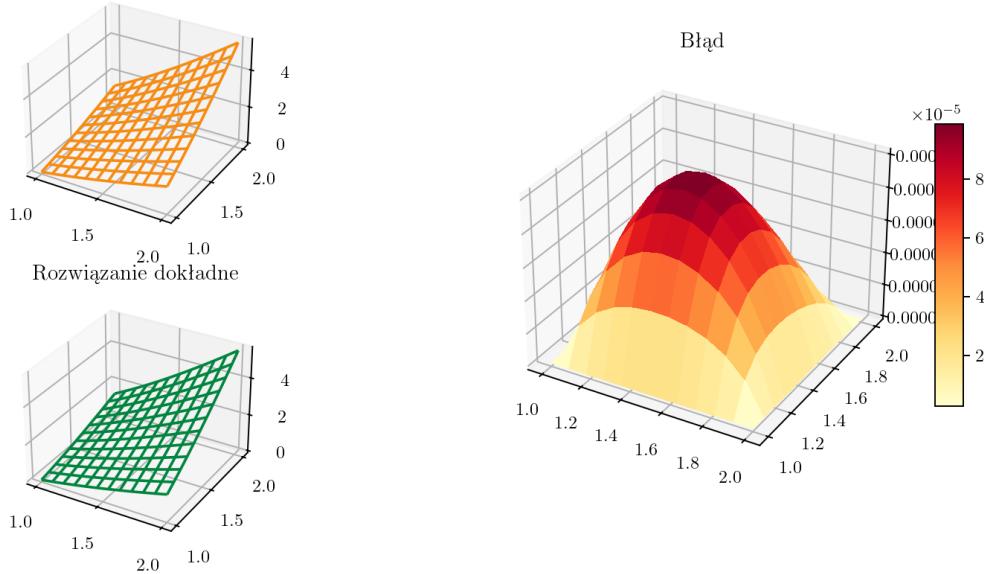
**Rysunek 26:** Wykres z rozwiązańem dla  $n = 10$  oraz  $m = 10$

## 9.2 Metoda Jakobiego

Rysunek 27 przedstawia rozwiązanie zadania metodą Jakobiego dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



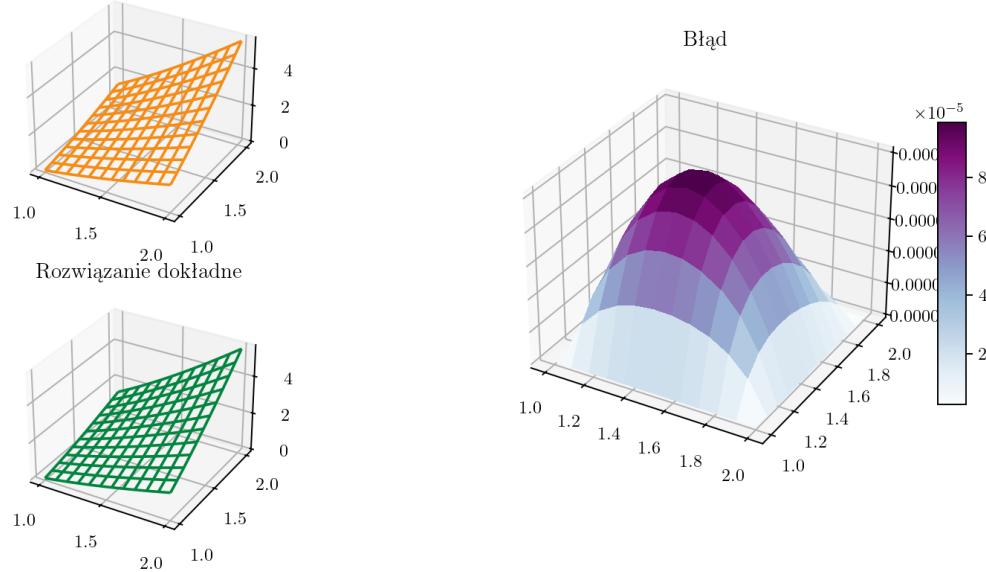
Rysunek 27: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

### 9.3 Metoda Gaussa-Seidela

Rysunek 28 przedstawia rozwiązanie zadania metodą Gaussa-Seidela dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



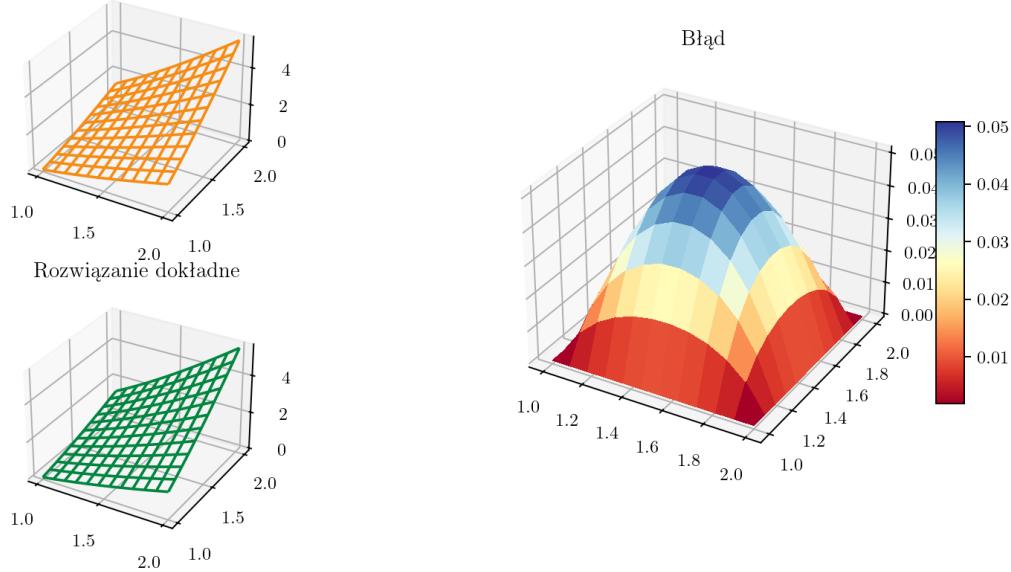
**Rysunek 28:** Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

#### 9.4 Metoda nadrelaksacji Younga

Rysunek 29 przedstawia rozwiązanie zadania metodą nadrelaksacji Younga dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



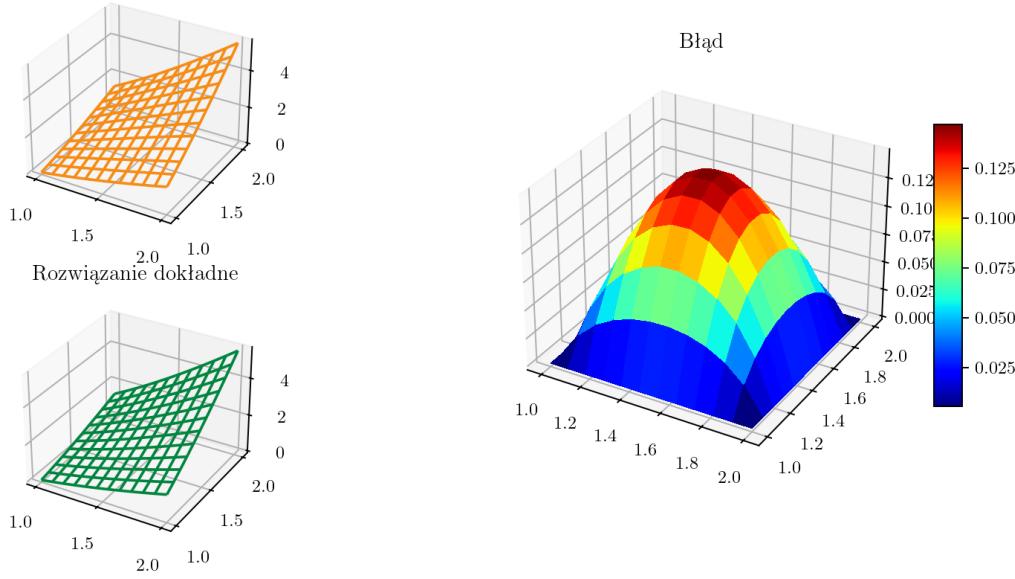
Rysunek 29: Wykres z rozwiązańem dla  $n = 10$  oraz  $m = 10$

## 9.5 Metoda Peacemana-Rachforda

Rysunek 30 przedstawia rozwiązańe zadania metodą Peacemana-Rachforda dla  $n = 10$ ,  $m = 10$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



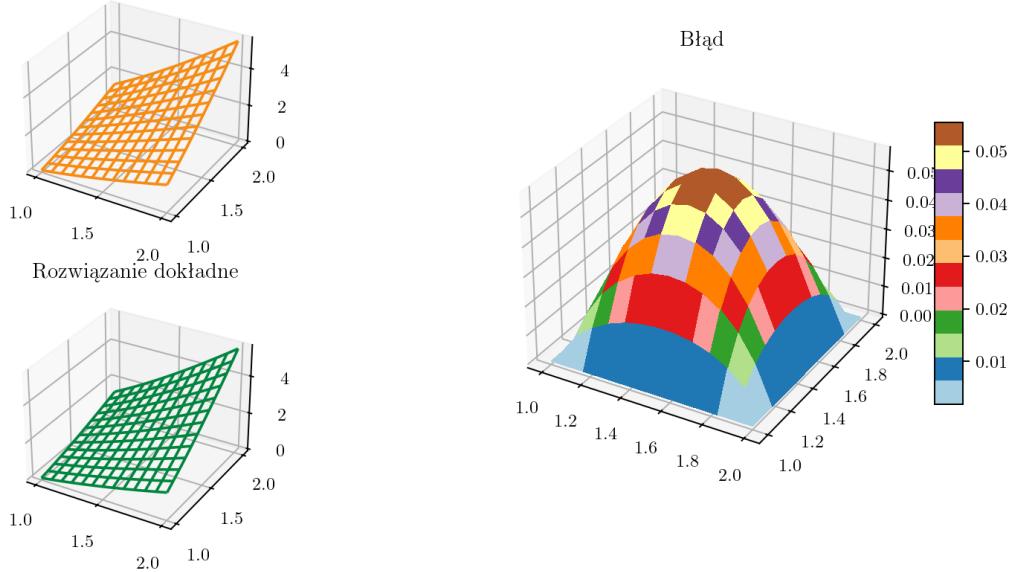
Rysunek 30: Wykres z rozwiązaniem dla  $n = 10$  oraz  $m = 10$

## 9.6 Siatki adaptacyjne

Rysunek 31 przedstawia rozwiązań zadania siatkami adaptacyjnymi dla  $n = 5$ ,  $m = 5$ , minimalnym błędem między iteracjami równym  $10^{-15}$  i 10000 maksymalnymi iteracjami.

$$\text{Równanie: } \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{x}{y} + \frac{y}{x}$$

Rozwiązańe numeryczne



Rysunek 31: Wykres z rozwiązaniem dla  $n = 5$  oraz  $m = 5$

## 9.7 Zestawienie wyników

Tabele od 7 do 9 zawierają zestawienie wyników dla różnych  $m$  oraz  $n$  (tolerancja  $10^{-10}$  i 10000 iteracji)

Tabela 7: Zestawienie czasów działań algorytmów

Czas [s]							
$n$	$m$	Metoda MRS	Metoda Jakobiego-go	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peaceman-Rachforda	Siatki adaptacyjne
5	5	0,0013	0,0379	0,0110	0,0143	0,0301	0,0050
10	10	0,0008	0,3827	0,1403	0,1224	0,1224	0,0444
20	20	0,0004	2,8977	1,3665	0,4259	0,6770	0,2736
30	30	0,0003	14,8123	6,5969	0,9962	2,0480	0,7092
40	40	0,0002	43,4072	21,5865	2,2635	4,3792	1,6423
50	50	0,0002	94,0034	53,3270	4,5925	11,2313	3,4703

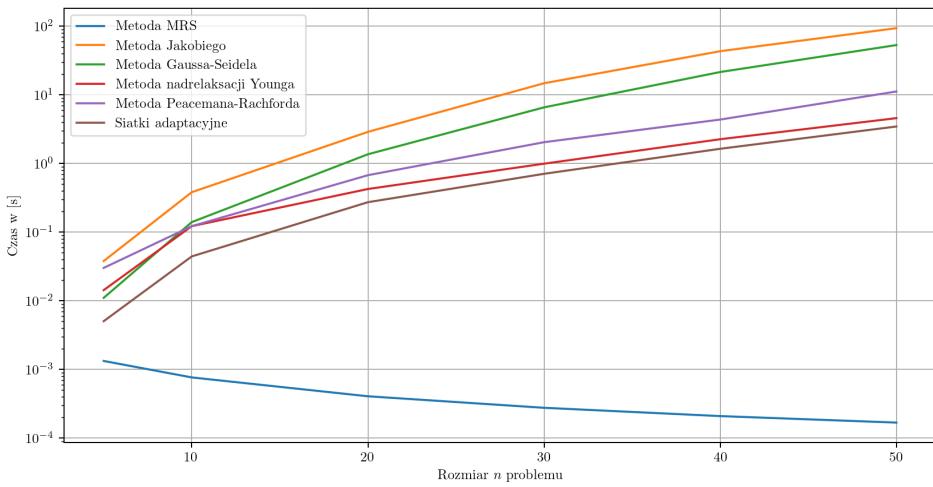
Tabela 8: Zestawienie dokładności względem rozwiązania analitycznego

Dokładność względem rozwiązania							
$n$	$m$	Metoda MRS	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	0,0229	0,0013	0,0013	0,1222	0,5875	0,1673
10	10	0,1657	0,0008	0,0008	0,3843	1,1108	0,4655
20	20	0,1538	0,0004	0,0004	0,9092	2,1551	0,9774
30	30	0,3850	0,0003	0,0003	1,4292	3,1918	1,4923
40	40	0,2299	0,0002	0,0002	1,9475	4,2264	2,0079
50	50	0,4945	0,0002	0,0002	2,4651	5,2602	2,5237

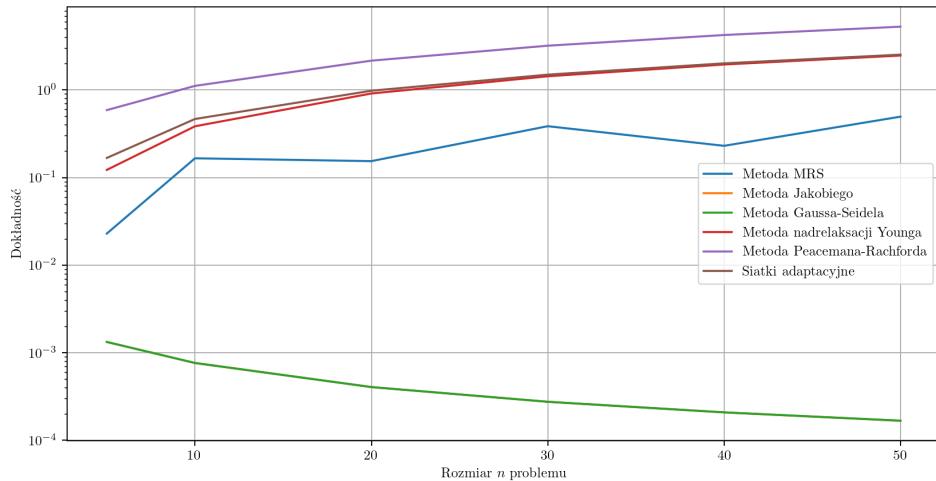
Tabela 9: Zestawienie liczby iteracji metod

Liczba iteracji						
$n$	$m$	Metoda Jakobiego	Metoda Gaussa-Seidela	Metoda nadrelaksacji Younga	Metoda Peacemana-Rachforda	Siatki adaptacyjne
5	5	163	85	43	24	29
10	10	551	286	72	39	57
20	20	1972	1021	125	94	107
30	30	4231	2189	177	145	158
40	40	7312	3782	229	209	209
50	50	10000	5794	280	277	258

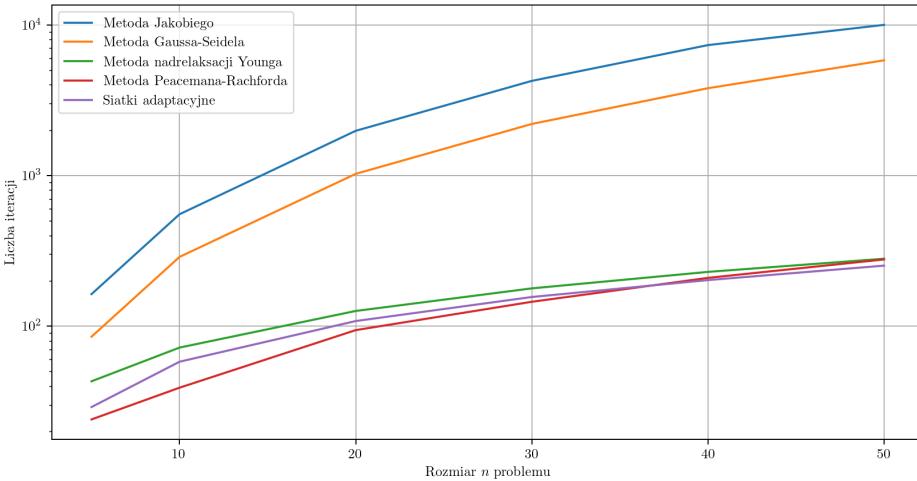
Rysunki 32, 33 i 34 przedstawiają zestawienie rozwiązania analitycznego, zestawienie czasów oraz zestawienie względem liczby iteracji w postaci wykresów.



Rysunek 32: Zestawienie czasów



Rysunek 33: Zestawienie dokładności



Rysunek 34: Zestawienie liczby iteracji

## 10 Wnioski

Metoda MRS osiągała znaczco lepsze czasy od reszty metod. Jest to spowodowane faktem, że metody iteracyjne wykonywały wiele iteracji i przez to są bardziej złożone obliczeniowo. Na czasy wpływa istotnie język programowania. Algorytmy wykorzystujące wielokrotne pętle są najbardziej narażone na słabsze czasy. Najlepiej to widać w algorytmie na siatki adaptacyjne. Tam w najgorszym przypadku wyliczamy 10000 razy zagnieszczone pętle  $m \times n$ . Mamy tutaj do czynienia z złożeniem obliczeniowym postaci  $O(mn \cdot 10000)$ . Nieefektywne również w tym algorytmie jest generowanie wektora  $r$ . Wystarczy wygenerować jeden wektor  $r$  i na iterator nakładać operator modulo  $k$ , gdzie  $k$  jest długością  $r$ .

Dla toleracji  $10^{-15}$  algorytmy iteracyjne nie były w stanie czasami dojść do takiej tolerancji przy 10000 krokach. W zadaniu 1 widać, że liczba kroków osiągała szybko wartości maksymalne. Mialo to ogromny wpływ na czasy obliczeń. Po ograniczeniu tolerancji do  $10^{-10}$  algorytmy były w stanie szybko obliczyć rozwiązań. Metoda Jakobiego wypadła najgorzej w porównaniu do reszty metod. Skorzystanie z siatek adaptacyjnych było w stanie poprawić liczbę iteracji dla dużych  $n$  oraz  $m$ .

Dokładność dawała sporo do życzenia. Metody iteracyjne z jakiegoś powodu dawały kiepski rezultat w zadaniu 2. W zadaniu 3 było trochę lepiej, metody Gaussa-Seidela oraz Jakobiego dawały lepsze rezultaty niż metoda MRS. Nadrelaksacja, Peaceman-Rachford i siatki wypadły najgorzej.

Dla dobrze dobranej tolerancji mimo większej złożoności obliczeniowej siatki adaptacyjne dawały najlepsze rezultaty czasowe, względem innych metod iteracyjnych.