Metody różnic skończonych
Sprawozdanie nr 2
Zagadnienie Neumanna i Robina

Kacper Dąbrowski

Maj 2024

Spis treści

1 Zagadnienie brzegowe Neumanna dla podstawowego równania różniczkowego II rzędu

Uwaga. Funkcje $\alpha(x), \beta(x), \gamma(x), f(x), y(x)$ będą zapisywane bez oznaczania ich zależności od zmiennej x tz. np. $\alpha(x) = \alpha$. Podobnie skrócony jest zapis funkcji w punkcie tz. funkcja $\alpha(x_1) = \alpha_1$.

Zagadnieniem Neumanna nazywamy warunki brzegowe równania różniczkowego dane w następujący sposób

$$\begin{cases} y'' = f \\ y'|_{x=a} = \tilde{y}_a \quad x \in [a,b] \\ y'|_{x=b} = \tilde{y}_b \end{cases}$$

Oznacza to, że mamy podane wartości pochodnej na brzegach funkcji. Aproksymację zadania wykonujemy przy pomocy schematów różnicowych. Macierz, którą będziemy rozwiązywać będzie musiała zostać poszerzona o wartości w punktach y_0 oraz y_{n+1}, ponieważ w tych punktach znamy tylko wartość pochodnej. Po podzieleniu funkcji na przedziały x_i otrzymujemy, że

$$\text{dla } x_i : \frac{y_{i-1}-2y_i+y_{i+1}}{h^2} = f_i \text{ dla } i \in \{1,2,...,n\}$$

Dla $i = 0$ użyjemy schematu w przód

$$\frac{-y_0+y_1}{h} = \tilde{y}_a$$

Dla $i = n + 1$ użyjemy schematu w tył

$$\frac{-y_n+y_{n+1}}{h} = \tilde{y}_b$$

Po przekształceniach otrzymujemy taką postać macierzową

$$A = \begin{bmatrix} -1 & 1 & 0 & \dots & & & \\ 1 & -2 & 1 & 0 & \dots & & \\ 0 & 1 & -2 & 1 & 0 & \dots & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & & \dots & 0 & -1 & 1 \end{bmatrix}_{n+2\times n+2} \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} 0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ 0 \end{bmatrix} \mathbf{g} = \begin{bmatrix} -\tilde{y}_a \cdot h \\ 0 \\ \vdots \\ -\tilde{y}_b \cdot h \end{bmatrix}$$

Teraz możemy wyliczyć wartości w punktach węzłowych poprzez rozwiązanie układu równań

$$A\mathbf{y} = \mathbf{f} - \mathbf{g}$$

1.1 Podejście drugie

Podejście drugie polega na dołączeniu węzła pozornego o indeksie -1. Następnie korzystamy z

$$y''(x_0) \approx D_c^2 y_0 = \frac{y_{-1}-2y_0+y_1}{h^2}$$

oraz

$$y'(a) \approx D_c y_0 = \frac{-y_{-1}+y_1}{2h}$$

Z tego mamy następujący układ równań

$$\begin{cases} \frac{y_{-1}-2y_0+y_1}{h^2} = f_0 \\ \frac{-y_{-1}+y_1}{2h} = \tilde{y}_a \end{cases}$$

Rozwiązując dla y_{-1} mamy

$$y_{-1} = y_1 - 2h\tilde{y}_a$$

Po wstawieniu otrzymujemy

$$-y_0 + y_1 = \tfrac{1}{2}h^2 f_0 + h\tilde{y}_a$$

Dla prawego brzegu obliczenia są analogicznie i postać wygląda nastąpująco

$$y_{n+2} = y_n + 2h\tilde{y}_b$$

$$y_n - y_{n+1} = \tfrac{1}{2}h^2 f_{n+1} - h\tilde{y}_b$$

Postać macierzowa prezentuje się następująco

$$A = \begin{bmatrix} -1 & 1 & 0 & \dots & & & \\ 1 & -2 & 1 & 0 & \dots & & \\ 0 & 1 & -2 & 1 & 0 & \dots & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & & \dots & 1 & -2 & 1 \\ 0 & \dots & & \dots & 0 & 1 & -1 \end{bmatrix} \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} \frac{1}{2}h^2 f_0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ \frac{1}{2}h^2 f_{n+1} \end{bmatrix} \mathbf{g} = \begin{bmatrix} -\tilde{y}_a \cdot h \\ 0 \\ \vdots \\ \tilde{y}_b \cdot h \end{bmatrix}$$

Teraz możemy wyliczyć wartości w punktach węzłowych poprzez rozwiązanie układu równań:

$$A\mathbf{y} = \mathbf{f} - \mathbf{g}$$

1.2 Podejście trzecie

Podejście trzecie polega na aproksymacji pochodnej schematami różnicowymi wyższych rzędów.

$$\text{dla } x_0 : \frac{-3/2 y_0 + 2y_1 - 1/2 y_2}{h} = \tilde{y}_a$$

$$x_0 : -3y_0 + 4y_1 - y_2 = \tilde{y}_a 2h$$

Postać macierzowa

$$A = \begin{bmatrix} -3 & 4 & -1 & \dots & & & \\ 1 & -2 & 1 & 0 & \dots & & \\ 0 & 1 & -2 & 1 & 0 & \dots & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & & \dots & 3 & -4 & -1 \end{bmatrix}_{n+2 \times n+2} \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} 0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ 0 \end{bmatrix} \mathbf{g} = \begin{bmatrix} -\tilde{y}_a \cdot 2h \\ 0 \\ \vdots \\ -\tilde{y}_b \cdot 2h \end{bmatrix}$$

Teraz możemy wyliczyć wartości w punktach węzłowych poprzez rozwiązanie układu równań

$$A\mathbf{y} = \mathbf{f} - \mathbf{g}$$

2 Zagadnienie brzegowe Neumanna dla ogólnego równania różniczkowego II rzędu

Zagadnienie różni się od podstawowego tym, że rozważamy teraz pełne ogólne równanie różniczkowe II rzędu. To znaczy

$$\begin{cases} \alpha y'' + \beta y' + \gamma y = f \\ y'|_{x=a} = \tilde{y}_a \\ y'|_{x=b} = \tilde{y}_b \end{cases} \qquad x \in [a,b]$$

Różnice względem poprzedniego zagadnienia tyczą się głównie postaci macierzowej, tamta macierz A jest zastąpiona następującą

$$A = \begin{bmatrix} -1 & 1 & 0 & \dots & & & \\ \alpha_1 - \frac{1}{2}\beta_1 h & -2\alpha_1 + h^2\gamma_1 & \alpha_1 + \frac{1}{2}\beta_1 h & 0 & \dots & & \\ 0 & \alpha_2 - \frac{1}{2}\beta_2 h & -2\alpha_2 + h^2\gamma_2 & \alpha_2 + \frac{1}{2}\beta_2 h & 0 & \dots & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & \alpha_n - \frac{1}{2}\beta_n h & -2\alpha_n + h^2\gamma_n & \alpha_n + \frac{1}{2}\beta_n h \\ 0 & \dots & & \dots & 0 & -1 & 1 \end{bmatrix}_{n+2 \times n+2}$$

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix} \mathbf{f} = \begin{bmatrix} 0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ 0 \end{bmatrix} \mathbf{g} = \begin{bmatrix} \tilde{y}_a \cdot h \\ 0 \\ \\ \vdots \\ \tilde{y}_b \cdot h \end{bmatrix}$$

Teraz możemy wyliczyć wartości w punktach węzłowych poprzez rozwiązanie układu równań

$$A\mathbf{y} = \mathbf{f} - \mathbf{g}$$

3 Program do rozwiązywania równania z zagadnieniem Neumanna

Program do obsługi zagadnienia Neumanna został napisany w języku Python. Jest on przystosowany do zagadnienia podstawowego i mieszanego, czyli do zagadnienia z punktu 1 z jednym brzegiem danym zagadnieniem Dirichleta, a drugim Neumanna. Klasa zawiera szereg metod opisanych poniżej.

Biblioteki potrzebne do działania programu:

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

3.1 Kontruktor klasy

Kontruktor klasy *Neumann* przedstawiony jest na rysunku 1. Argumentami wejściowymi klasy jest funkcja źródłowa f podana w poprzed funkcję *lambda* oraz parametr *way* wskazujący, którym sposobem dany problem ma być rozwiązany. Przy inicjacji klasy tworzone są inne zmienne globalne, które będą przydatne w dalszej części.

```python
class Neumann:
    '''
    Solves second order differencial equations with Neumann boundary conditions, given by equation y'' = f(x).
    '''
    def __init__(self, f, way : int = 2) -> None:
        '''
        Initiation of problem given by equation: y'' = f(x).

        Args:
            f (lambda function): function f
            way (int): pick way with which you want to solve the problem (from 0 to 2, defaults to 2)'''
        self.f = f
        self.way : int = way

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

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

        self.derivative : list[bool] = [False, False]

        self.n = 1
        self.a = 0
        self.b = 2
        self.val_a = 0
        self.val_b = 0
        self.h = 0

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

Rysunek 1: Kod konstruktora klasy

3.2 Metody konfigurujące klasę

Tak samo jak w przypadku zagadnienia Dirichleta mamy praktycznie te same metody konfigurujące klasę. Jedyne co jest nowego w tej klasie to metody wskazujące, po której stronie mamy pochodną. Są to odpowiednio funkcje *setLeft* i *setRight*. Przedstawione one są na rysunku 2.

Funkcje te zmieniają wartość zmiennej globalnej *derivative*, która początkowo jest listą dwuelementową wypełnioną wartościami *False*. Jeśli pochodna jest przy lewym brzegu to lewa wartość jest zapisywana jako *True*, jak prawa to na odwrót. Później ta zmienna pomoże nam w rozwiązywaniu układu równań.

```python
def setLeft(self) -> None:
    '''Consider left side as derivative.'''
    self.derivative : list[bool]= [True, False]
    self.setH()

def setRight(self) -> None:
    '''Consider right side as derivative.'''
    self.derivative : list[bool]= [False, True]
    self.setH()
```

Rysunek 2: Kod metody ustalającej brzeg z pochodną

3.3 Skrypt rozwiązujący problem

Za rozwiązanie podstawowego równania różniczkowego II rzędu z zagadnieniem Neumanna odpowiada funkcja *solve*, przedstawiona na rysunku 3. Na samym początku sprawdzane jest, czy wszystkie parametry, konieczne do rozwiązania zadania, zostały spełnione. Brany pod uwagę tym razem jest też warunek na to czy został wybrany brzeg, na którym jest pochodna. Jeśli nie zostały wcześniej wywołane tamte funkcje, program zwróci błąd mówiący, że nie można rozwiązać zadania. Następnie generowane są wszystkie wektory potrzebne do rozwiązania problemu. Jeden wektor został skopiowany w celu oszczędzenia czasu obliczeń. Po obliczeniu wektorów program w zależności od wartości zmiennej *way* wykonywał dane podejście. Wartość 0 brała pod uwagę podejście podstawowe z schematem różnicowych rzędu pierwszego, wartość 1 brała podejście drugie, które wykorzystywało sztuczny punkt, a wartość 2 bierze brała schemat różnicowy w przód (w tył) wyższego rzędu. Jeśli wartość *way* była niepoprawna, to program zwraca błąd.

Pierwsze dwa podejścia rozwiązują układ równań metodą Thomasa, a ostatnie musiało użyć metody wbudowanej w bibliotekę *numpy*. Niestety wykorzystanie schematów wyższych rzędów wiąże się z utratą trójprzekątniowości.

```python
def solve(self) -> float:
    '''
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.'''
    if not all([self.set_ab, self.set_n]) or not any(self.derivative):
        raise ValueError("Cannot solve equation without setting parameters")
    v1 : np.ndarray = (-2)*np.ones(self.n+1)
    v2 : np.ndarray = np.ones(self.n)
    v3 : np.ndarray = v2.copy()
    B : np.ndarray = (self.h**2)*np.array(list(map(self.f, self.unknown_x)))

    match self.way:
        case 0:
            if self.derivative[0]:
                v1[0] = -1
                B[0] = self.h*self.val_a
                B[-1] = B[-1] - self.val_b
            else:
                v1[-1] = 1
                v3[-1] = -1
                B[-1] = self.h*self.val_b
                B[0] = B[0] - self.val_a
            print(f"{np.diag(v1) + np.diag(v2, 1) + np.diag(v3, -1)}\n{B}")
            sol = ThomasSolve(v3,v1,v2,B)
            return sol
        case 1:
            if self.derivative[0]:
                v1[0] = -1
                B[0] = 1/2*self.h**2*self.f(self.x[0]) + self.h*self.val_a
                B[-1] = B[-1] - self.val_b
            else:
                v1[-1] = -1
                v3[-1] = 1
                B[-1] = 1/2*self.h**2*self.f(self.x[-1]) - self.h*self.val_b
                B[0] = B[0] - self.val_a
            print(f"{np.diag(v1) + np.diag(v2, 1) + np.diag(v3, -1)}\n{B}")
            sol = ThomasSolve(v3,v1,v2,B)
            return sol
        case 2:
            A = np.diag(v1)+np.diag(v2,1)+np.diag(v2,-1)
            if self.derivative[0]:
                A[0][0] = -3; A[0][1] = 4; A[0][2] = -1
                B[0] = 2*self.h*self.val_a
                B[-1] = B[-1] - self.val_b
            else:
                A[-1][-1] = 3; A[-1][-2] = -4; A[-1][-3] = 1
                B[-1] = 2*self.h*self.val_b
                B[0] = B[0] - self.val_a

            sol = LA.solve(A, B)
            return sol
        case _:
            raise ValueError("Nie ma takiej opcji")
```

Rysunek 3: Kod metody rozwiązującej równanie

3.4 Reszta metod

Reszta metod została zaimplementowana w ten sam sposób jak przy zagadnieniu Dirichleta, więc opis został pominięty.

4 Rozwiązywanie podstawowego równania różniczkowego II rzędu z częściowym zagadnieniem Neumanna

Podstawowe równanie różniczkowe II rzędu z częściowym zagadnieniem Neumanna jest postaci

$$\begin{cases} y'' = f \\ y'|_{x=a} = \tilde{y}_a \\ y|_{x=b} = y_b \end{cases} \quad x \in [a,b]$$

To znaczy, że w zadaniach będzie tylko jeden brzeg jako pochodna.

4.1 Zadanie 1

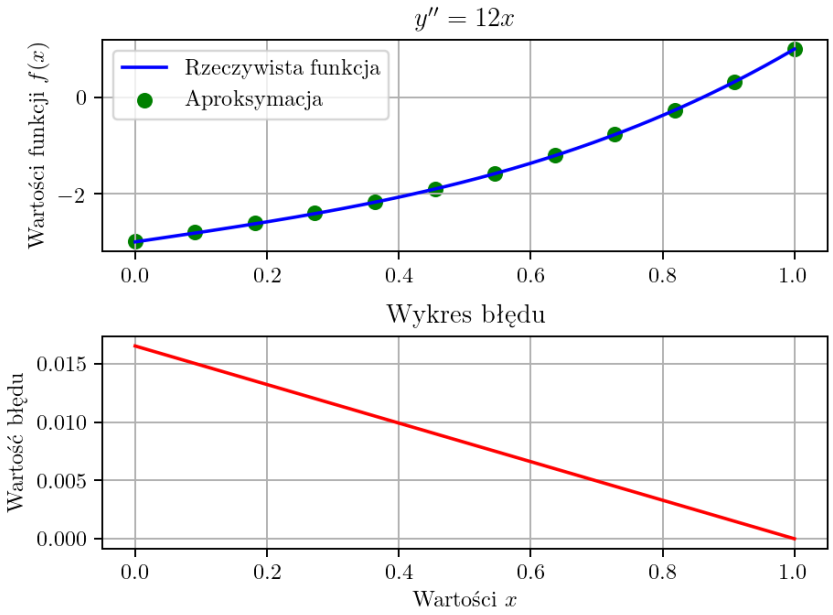
Mamy zagadnienie Neumanna postaci

$$\begin{cases} y'' = 12x \\ y'|_{x=0} = 2 \\ y|_{x=1} = 1 \end{cases} \quad x \in [0,1]$$

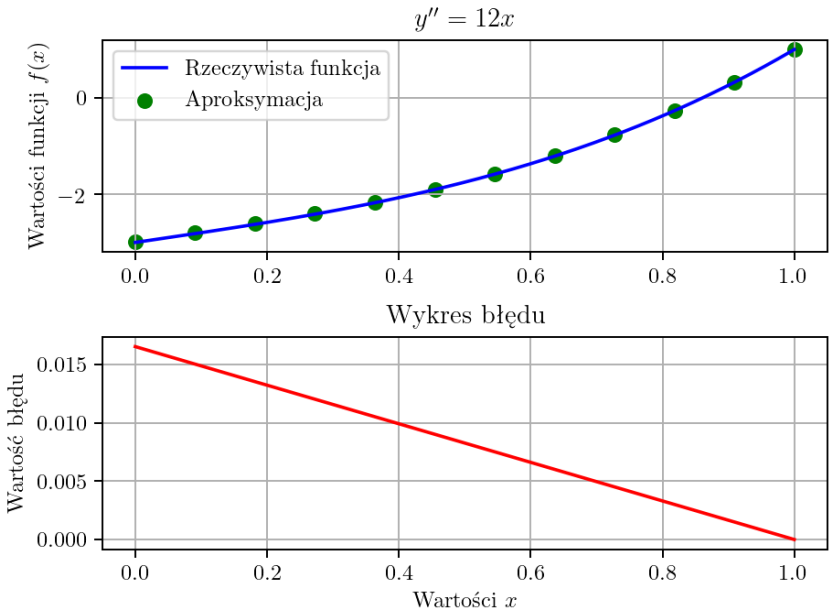
Rozwiązanie analityczne

$$y = 2x^3 + 2x - 3$$

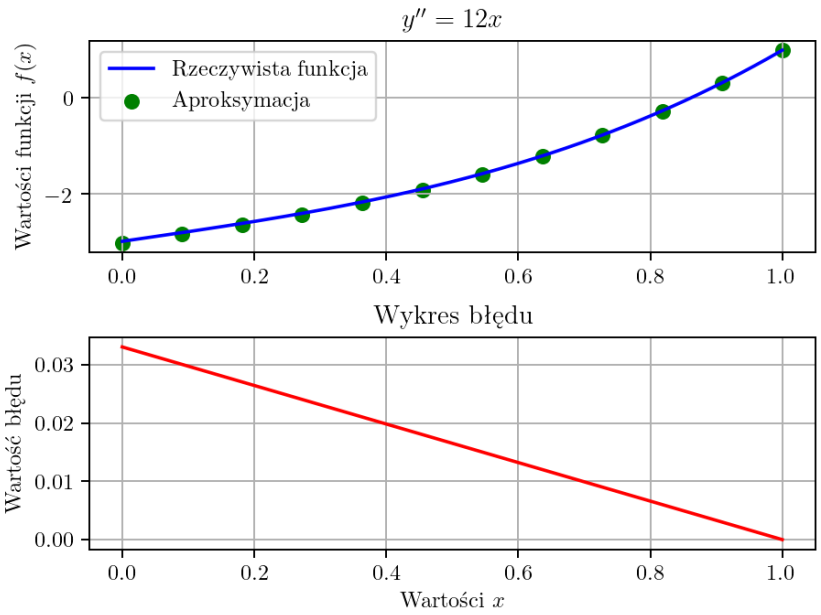
Rysunki 4, 5, 6 zawierają wyniki dla $n = 10$ i osobnych podejść, a rysunek 7 dla $n = 3\,000\,000$



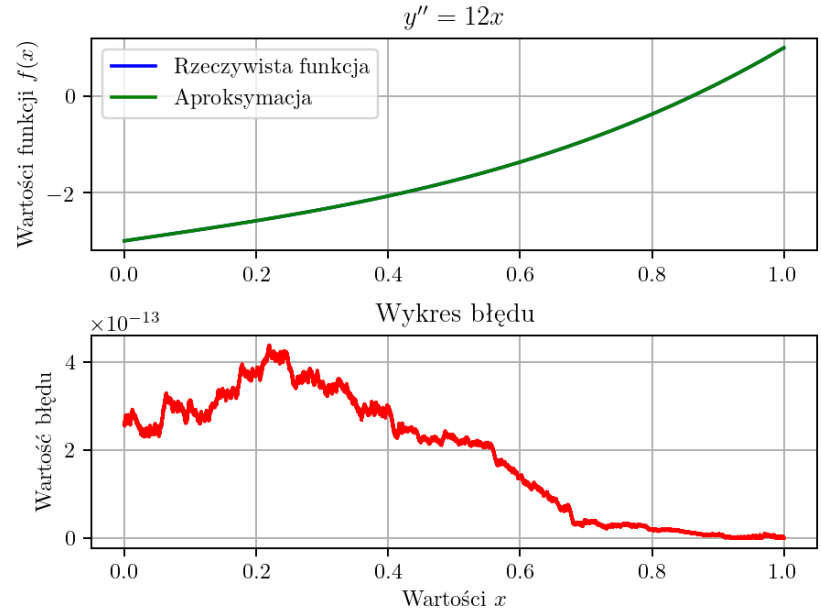
Rysunek 4: Wykres z rozwiązaniem dla $n = 10$ (metoda ze schematem pierwszego rzędu)



Rysunek 5: Wykres z rozwiązaniem dla $n = 10$ (metoda ze sztucznym punktem)



Rysunek 6: Wykres z rozwiązaniem dla $n = 10$ (metoda ze schematem wyższego rzędu)



Rysunek 7: Wykres z rozwiązaniem dla $n = 3\,000\,000$ (metoda ze schematem ze sztucznym punktem)

Na rysunku 8 przedstawiony jest kod wejściowy programu. W dalszej części będzie on pominięty, ponieważ będą się zmieniać tylko dane.

```
def main() -> None:
    f =     lambda x: 12*x
    n = 10
    a = [0, 2]
    b = [1, 1]
    neumann = Neumann(f, 2)

    neumann.setAB(a, b)
    neumann.addSolution(lambda x: 2*x**3+2*x-3)
    neumann.setLabel(r"$y''= 12x$")
    neumann.setN(n)
    neumann.setLeft()
    print(neumann)

if __name__ == '__main__':
    main()
```

Rysunek 8: Kod wejściowy

4.2 Zadanie 2

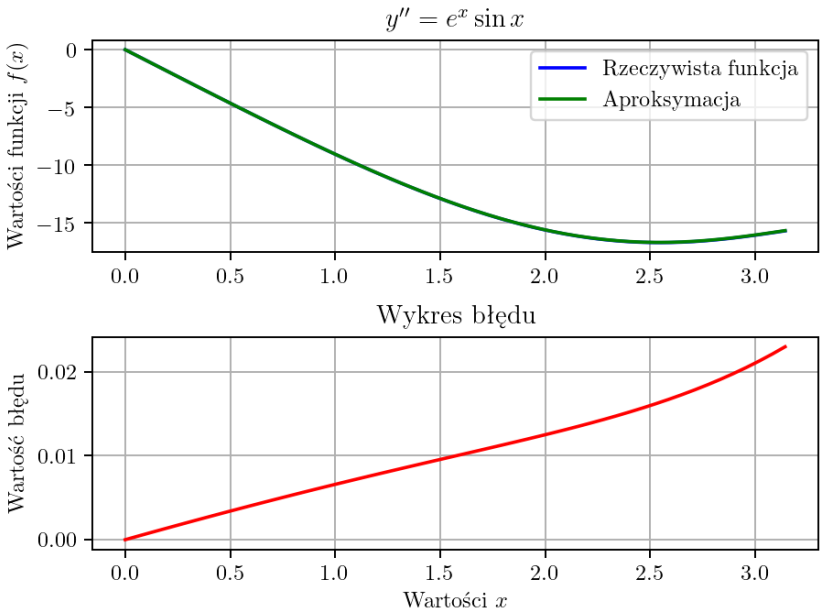
Mamy zagadnienie Neumanna postaci

$$\begin{cases} y'' = e^x \sin x \\ y|_{x=0} = 0 \\ y'|_{x=\pi} = e \end{cases} \qquad x \in [0,\pi]$$

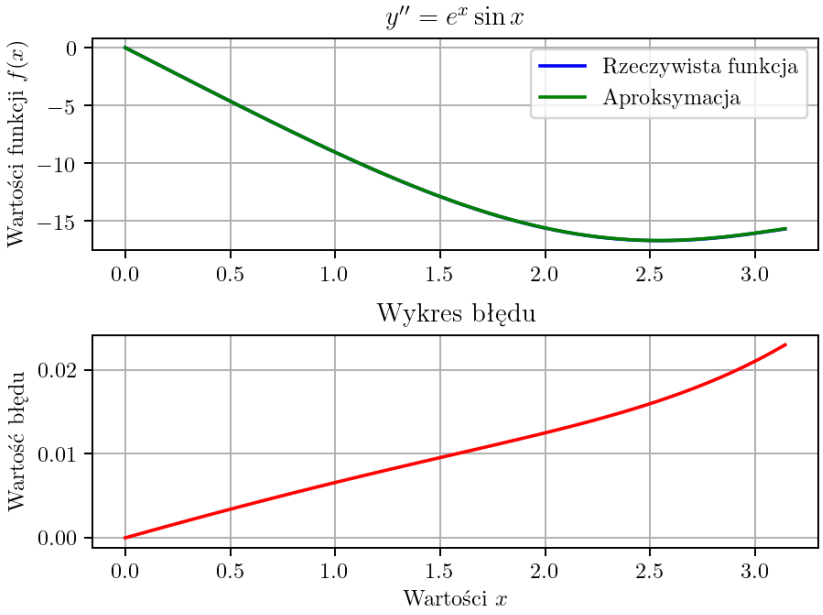
Rozwiązanie analityczne

$$y = \tfrac{1}{2}(1 + 2ex - e^{\pi}x - e^x \cos x)$$

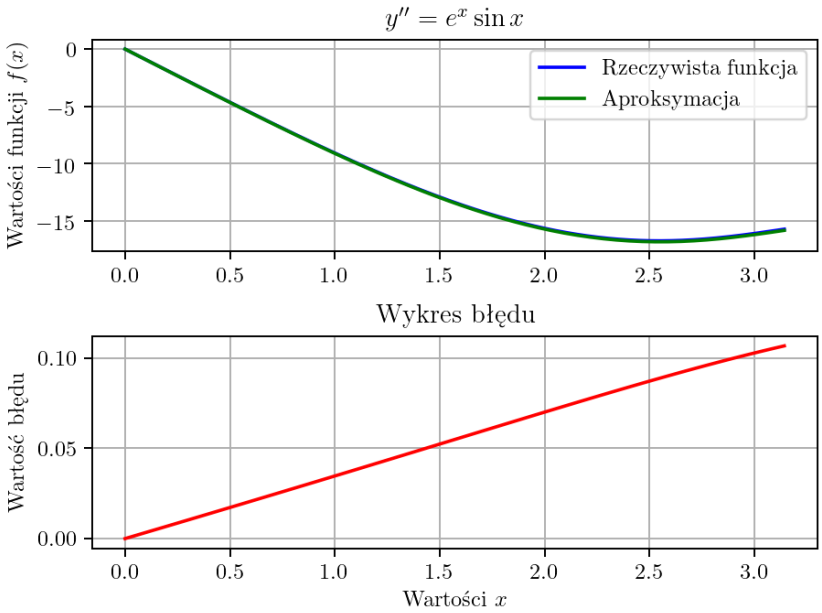
Rysunki 9, 10, 11 zawierają wyniki dla $n = 50$ i osobnych podejść.



Rysunek 9: Wykres z rozwiązaniem dla $n = 50$ (metoda ze schematem pierwszego rzędu)



Rysunek 10: Wykres z rozwiązaniem dla $n = 50$ (metoda ze sztucznym punktem)



Rysunek 11: Wykres z rozwiązaniem dla $n = 50$ (metoda ze schematem wyższego rzędu)

5 Rozwiązywanie ogólnego równania różniczkowego II rzędu z częściowym zagadnieniem Neumanna

Ogólne równanie różniczkowe II rzędu z częściowym zagadnieniem Neumanna jest postaci

$$\begin{cases} \alpha y'' + \beta y' + \gamma y = f \\ y'|_{x=a} = \tilde{y}_a \\ y|_{x=b} = y_b \end{cases} \qquad x \in [a,b]$$

Do tego rodzaju problemu program został przerobiony tak, aby był w stanie poradzić sobię z takim zagadnieniem. Jedyne co zostało zmienione to funkcja *solve* znajdująca się na rysunku 12 oraz konstruktor klasy z rysunku 13.

```python
class Neumann_all:
    '''
    Solves second order differencial equations with Neumann boundary conditions.
    '''
    def __init__(self,alpha ,beta, gamma, f) -> None:
        '''
        Initiation of problem given by equation: alpha*y''+beta*y'+gamma*y = f(x).

        Args:
            alpha (lambda function): alpha coefficient
            beta (lambda function): beta coefficient
            gamma (lambda function): gamma coefficient
            f (lambda function): function f'''
        self.f = f
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma

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

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

        self.derivative : list[bool] = [False, False]

        self.n = 1
        self.a = 0
        self.b = 2
        self.val_a = 0
        self.val_b = 0
        self.h = 0

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

Rysunek 12: Kod konstruktora klasy

```python
def solve(self) -> float:
    '''
    Solves the problem.

    Returns:
        float: solution'''
    if not all([self.set_ab, self.set_n]) or not any(self.derivative):
        raise ValueError("Cannot solve equation without setting parameters")
    alpha = np.array(list(map(self.alpha, self.unknown_x)))
    beta = np.array(list(map(self.beta, self.unknown_x)))
    gamma = np.array(list(map(self.gamma, self.unknown_x)))
    v1 : np.ndarray = (-2)*alpha + gamma*self.h**2
    v2 : np.ndarray = alpha + 1/2*self.h*beta
    v3 : np.ndarray = alpha - 1/2*self.h*beta
    v2 = v2[0:-1]
    v3 = v3[1:len(v1)+1]

    B : np.ndarray = (self.h**2)*np.array(list(map(self.f, self.unknown_x)))

    if self.derivative[0]:
        v1[0] = -1
        v2[0] = 1
        B[0] = self.h*self.val_a
        B[-1] = B[-1] - self.val_b*(self.alpha(self.b)+self.h*self.beta(self.b)/2)
    else:
        v1[-1] = 1
        v3[-1] = -1
        B[-1] = self.h*self.val_b
        B[0] = B[0] - self.val_a*(self.alpha(self.a)-self.h*self.beta(self.a)/2)
    print(f"{np.diag(v1) + np.diag(v2, 1) + np.diag(v3, -1)}\n{B}")
    sol = ThomasSolve(v3, v1, v2, B)
    return sol
```

Rysunek 13: Kod metody *solve*

5.1 Zadanie 1

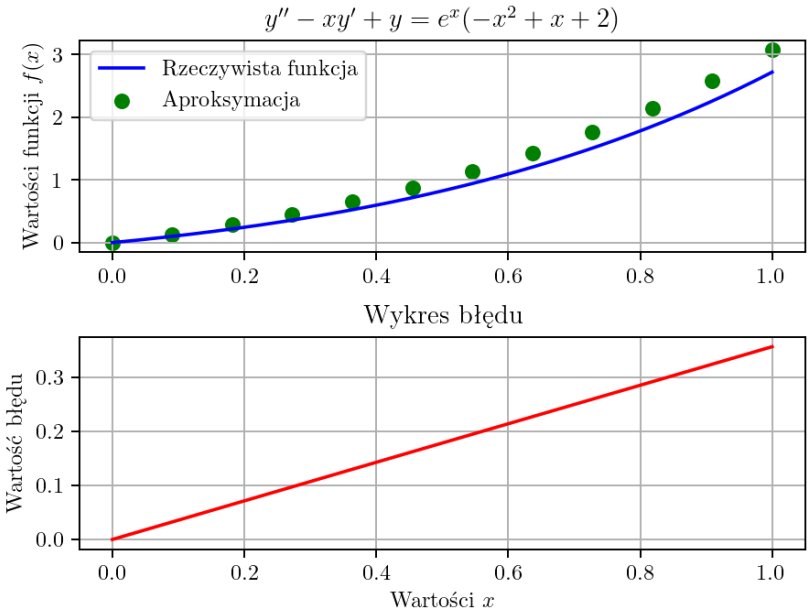
Mamy zagadnienie Neumanna postaci

$$\begin{cases} y'' - xy' + y = e^x(-x^2 + x + 2) & \\ y|_{x=0} = 0 & x \in [0,1] \\ y'|_{x=1} = 2e & \end{cases}$$

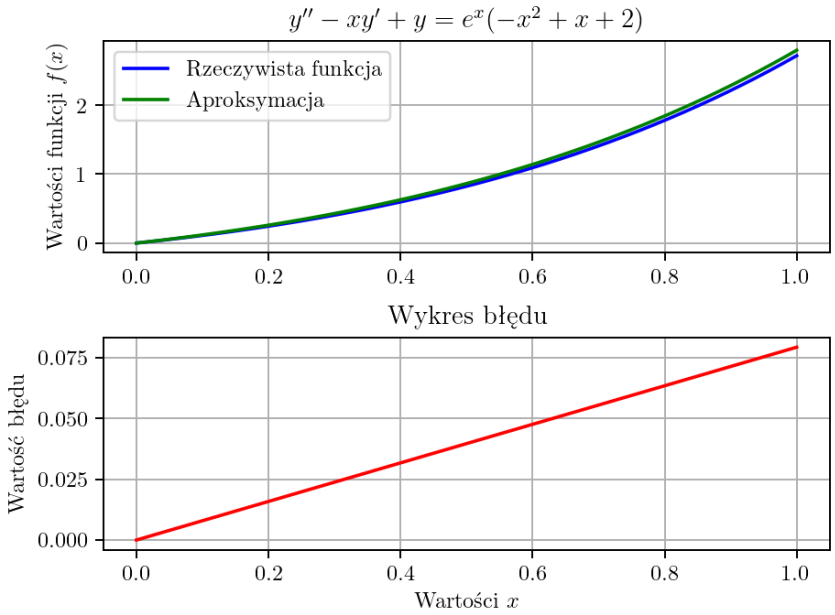
Rozwiązanie analityczne

$$y = xe^x$$

Rysunek 14 zawiera rozwiązanie dla $n = 10$, a rysunek 15 dla $n = 50$



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



Rysunek 15: Wykres z rozwiązaniem dla $n = 50$

6 Zestawienie wyników i wnioski dla zagadnienia Neumanna

6.1 Zestawienie wyników

Tabela 1 zawiera zestawienie błędów różnych metod dla każdego z zadań.

Tabela 1: Zestawienie wyników dla zagadnienia Neumanna

n	Zadanie 1			Zadanie 2			Zadanie 3
	Metoda 1	Metoda 2	Metoda 3	Metoda 1	Metoda 2	Metoda 3	
10	1,65E-02	1,65E-02	3,31E-02	4,91E-01	4,91E-01	1,71E+00	3,56E-01
100	1,96E-04	1,96E-04	3,92E-04	5,86E-03	5,86E-03	2,82E-02	4,02E-02
500	7,97E-06	7,97E-06	1,59E-05	2,38E-04	2,38E-04	1,18E-03	8,13E-03
1000	2,00E-06	2,00E-06	3,99E-06	5,97E-05	5,97E-05	2,97E-04	4,07E-03
5000	8,00E-08	8,00E-08	1,60E-07	2,39E-06	2,39E-06	1,19E-05	8,15E-04
10000	2,00E-08	2,00E-08	4,00E-08	5,97E-07	5,97E-07	2,99E-06	4,08E-04
50000	8,00E-10	8,00E-10		3,80E-08	3,80E-08		8,16E-05
100000	2,00E-10	2,00E-10		4,55E-08	4,55E-08		4,08E-05
500000	7,93E-12	7,93E-12		1,71E-05	1,71E-05		2,18E-06
1000000	2,05E-12	2,05E-12		1,83E-05	1,83E-05		4,31E-05

6.2 Wnioski

Różnice między podejściem pierwszym, a drugim nie zostały ujęte na wynikach przedstawionych w tabeli 1. Ma to sens, ponieważ w obu zadaniach funkcja f przyjmuje wartość 0 na brzegach z pochodną przez co warunki są identyczne dla obu podejść. Innymi słowy korekta postaci $\frac{1}{2}h^2 f_0$(równoznacznie $\frac{1}{2}h^2 f_{n+1}$) nie ma żadnego wpływu na rozwiązanie. Co ciekawe metoda wykorzystująca "lepszy" schemat różnicowy wyższego rzędu, osiągał gorsze wyniki.

Utrata trójprzekątniowości w rozwiązaniu trzecim mocno skreśla tę metodę, ponieważ nie jesteśmy w stanie dojść do bardzo dużego n'a. Metoda Thomasa pozwala dwóm pierwszym metodom wyliczenie rozwiązania dla milionowej liczby węzłów.

W zadaniu 3. metoda wykazała mocną zbieżność. Błąd był najmniejszy dla 500 000 węzłów.

Dla ciekawości został wygenerowany wynik dla 3 milionów węzłów. Znajduje się on na rysunku 7. Błąd jest na tyle mały, że jego wykres aż traci stabilność. Wraz z zwiększeniem liczby węzłów zjawisko to się pogłębia.

7 Zagadnienie brzegowe Robina dla ogólnego równania różniczkowego II rzędu

Zagadnieniem Robina nazywamy warunki brzegowe równania różniczkowego dane w następujący sposób

$$\begin{cases} \alpha y'' + \beta y' + \gamma y = f \\ p_1 y|_{x=a} + q_1 y'|_{x=a} = \hat{y}_a \quad x \in [a,b] \\ p_2 y|_{x=b} + q_2 y|_{x=b} = \hat{y}_b \end{cases}$$

Zagadnienie to uogólnia zagadnienie Dirichleta oraz Neumanna. Łatwo zauważyć, że dla odpowiednich wartości zmiennych p i q dostajemy jakieś kombinacje tych zagadnień. Główna część macierzy pozostanie bez zmian. Jedyne co się zmieni to pierwszy i ostatni wiersz. Korzystamy z

$$y'(a) \approx \frac{-y_0+y_1}{h}, \quad y'(b) \approx \frac{-y_n+y_{n+1}}{h}$$

Następnie tworzymy układ równań

$$\begin{cases} p_1 y_0 + q_1 \frac{-y_0+y_1}{h} = \hat{y}_a \\ p_2 y_0 + q_2 \frac{-y_n+y_{n+1}}{h} = \hat{y}_b \end{cases}$$

Po przekształceniach otrzymujemy

$$\text{dla } x_0 : y_0(p_1 h - q_1) + y_1 q_1 = \hat{y}_a h$$

$$\text{dla } x_{n+1} : -y_n q_2 + y_{n+1}(p_2 h + q_2) = \hat{y}_b h$$

Więc postać macierzowa jest postaci

$$A = \begin{bmatrix} p_1 h - q_1 & q_1 & 0 & \dots & & & \\ \alpha_1 - \frac{1}{2}\beta_1 h & -2\alpha_1 + h^2\gamma_1 & \alpha_1 + \frac{1}{2}\beta_1 h & 0 & \dots & & \\ 0 & \alpha_2 - \frac{1}{2}\beta_2 h & -2\alpha_2 + h^2\gamma_2 & \alpha_2 + \frac{1}{2}\beta_2 h & 0 & \dots & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & \alpha_n - \frac{1}{2}\beta_n h & -2\alpha_n + h^2\gamma_n & \alpha_n + \frac{1}{2}\beta_n h \\ 0 & \dots & & \dots & 0 & -q_2 & p_2 h + q_2 \end{bmatrix}_{n+2 \times n+2}$$

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix} \mathbf{f} = \begin{bmatrix} 0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ 0 \end{bmatrix} \mathbf{g} = \begin{bmatrix} \hat{y}_a \cdot h \\ 0 \\ \vdots \\ \hat{y}_b \cdot h \end{bmatrix}$$

Teraz możemy wyliczyć wartości w punktach węzłowych poprzez rozwiązanie układu równań:

$$A\mathbf{y} = \mathbf{f} - \mathbf{g}$$

8 Program do rozwiązywania równania z zagadnieniem Robina

Program został napisany w taki sam sposób jak przy rozwiązywaniu zagadnienia Neumanna. W tym przypadku niepotrzebne były metody, które służyły do wyboru strony, po której była pochodna. Różnice dotyczą tylko konstruktora klasy, metody *setAB* oraz metody *solve*.

8.1 Konstruktor klasy

Do konstruktora klasy dodane zostały zmienne odpowiedzialne za współczynniki przy warunkach brzegowych. Znajduje się on na rysunku 16.

```
class Robin:
    '''
    Solves second order differencial equations with Robin boundary conditions.
    '''
    def __init__(self, alpha, beta, gamma, f) -> None:
        '''
        Initiation of problem given by equation: alpha*y''+beta*y'+gamma*y = f(x).

        Args:
            alpha (lambda function): alpha coefficient
            beta (lambda function): beta coefficient
            gamma (lambda function): gamma coefficient
            f (lambda function): function f'''
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.f = f

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

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

        self.n : int = 1
        self.a : float = 0
        self.b : float = 2
        self.val_a : float = 0
        self.val_b : float = 0
        self.der_a : float = 0
        self.der_b : float = 0
        self.coef_a : float = 0
        self.coef_b : float = 0
        self.h : float = 0

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

Rysunek 16: Konstruktor klasy

8.2 Zmiana w metodzie *setAB*

Zmiana w metodzie *setAB* dotyczyła rozszerzenia list o dwa dodatkowe argumenty. Teraz w poprzed tę metodę można dodać każdą wartość w następujący sposób

$$\text{dla brzegu: } p_1 y(a) + q_1 y'(a) = \hat{y}_a$$

$$\text{brzeg} = [a, \hat{y}_a, q_1, p_1]$$

Implementacja takiego przypisania znajduje się na rysunku 17.

```python
def setAB(self, a : list, b : list) -> None:
    '''
    Sets boundary conditions.

    Args:
        a (list[float]): Four element list containing value of left boundary argument,
        value and coefficients next to derivative and y expresion.
        b (list[float]): Four element list containing value of right boundary argument,
        value and coefficients next to derivative and y expresion.'''
    self.a = a[0]
    self.b = b[0]
    self.val_a = a[1]
    self.val_b = b[1]
    self.der_a = a[2]
    self.der_b = b[2]
    self.coef_a = a[3]
    self.coef_b = b[3]
    self.set_ab = True
    self.setH()
```

Rysunek 17: Metoda *setAB*

8.3 Zmiana w metodzie *solve*

Metoda *solve* implementuje rozwiązanie zgodne z analitycznymi obliczeniami, które zostały pokazane we wstępie. Rysunek 18. przedstawia tę implementację.

```python
def solve(self) -> float:
    '''
    Solves the problem.

    Returns:
        float: solution depending on solve type variable.'''
    if not all([self.set_ab, self.set_n]):
        raise ValueError("Cannot solve equation without setting parameters")
    alpha = np.array(list(map(self.alpha, self.x)))
    beta = np.array(list(map(self.beta, self.x)))
    v1 : np.ndarray = -2*alpha + self.h**2*np.array(list(map(self.gamma, self.x)))
    v2 : np.ndarray = alpha+1/2*self.h*beta
    v3 : np.ndarray = alpha-1/2*self.h*beta
    v2 = v2[0:-1]
    v3 = v3[1:len(v1)+1]
    B : np.ndarray = (self.h**2)*np.array(list(map(self.f, self.x)))
    v1[0] = self.coef_a*self.h - self.der_a
    v2[0] = self.der_a
    v1[-1] = self.coef_b*self.h + self.der_b
    v3[-1] = -self.der_b
    B[0] = self.val_a*self.h
    B[-1] = self.val_b*self.h
    sol : np.ndarray = ThomasSolve(v3,v1,v2,B)
    return sol
```

Rysunek 18: Metoda *solve*

9 Rozwiązywanie ogólnego równania różniczkowego II rzędu z zagadnieniem Robina

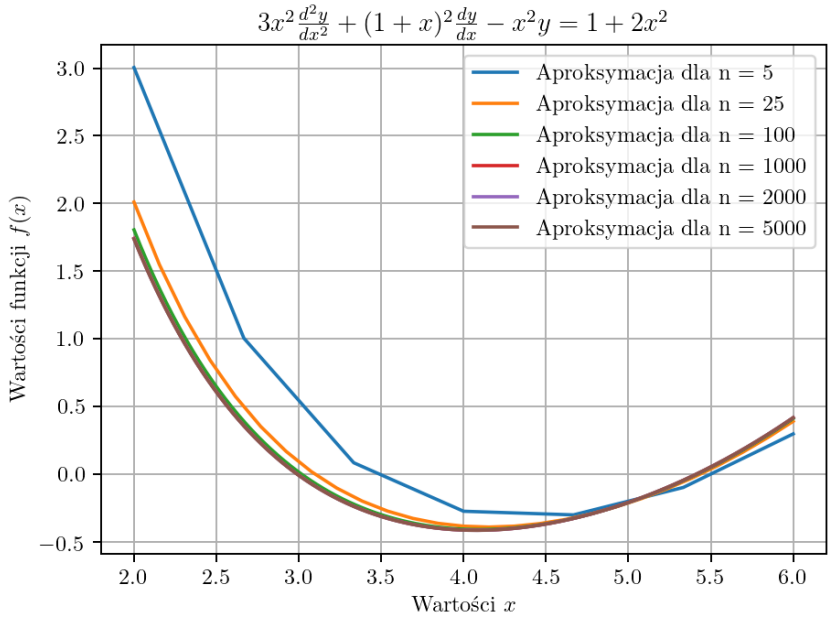
Mamy zagadnienie postaci

$$\begin{cases} 3x^2y'' + (1+x)^2y' - x^2y = 1 + 2x^2 \\ y'|_{x=2} = -3 \\ (2y - y')|_{x=6} = 0 \end{cases} \qquad x \in [0{,}6]$$

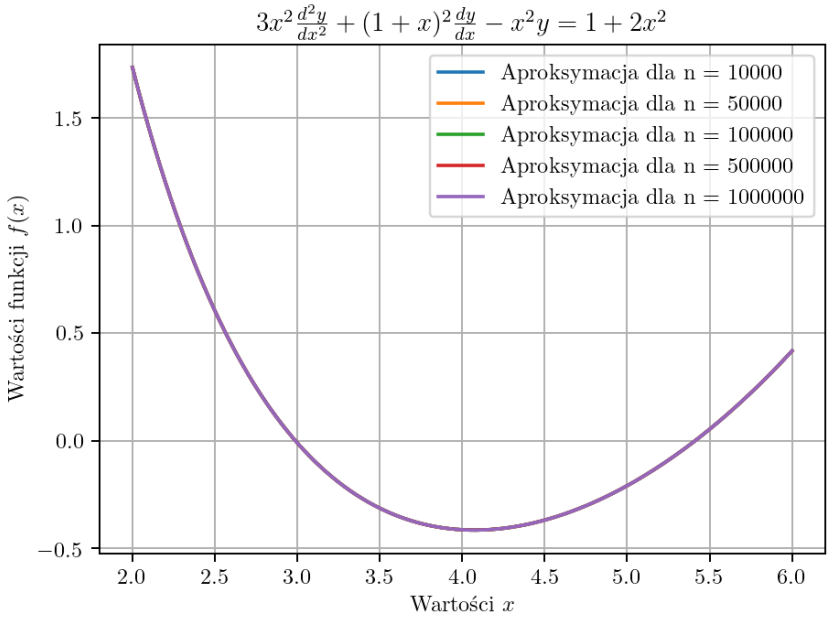
Dane wejściowe programu znajdują się na rysunku 19. Zestawione wyniki dla $n = 5$, 25, 100, 1000, 2000, 5000 są na rysunku 20. a dla n =10000, 50000, 100000, 500000, 1000000 na rysunku 21.

```python
from Robin import Robin
import matplotlib.pyplot as plt
def main() -> None:
    alpha =     lambda x: 3*x**2
    beta =      lambda x: (1+x)**2
    gamma =     lambda x: -x**2
    f =         lambda x: 1+2*x**2

    a = [2, -3, 1, 0]
    b = [6, 0, -1, 2]

    robin = Robin(alpha, beta, gamma, f)

    robin.setAB(a, b)
    robin.plt_config()
    robin.setLabel(r"$3x^2\frac{d^2y}{dx^2}+(1+x)^2\frac{dy}{dx}-x^2y= 1+2x^2$")
    for n in [10_000, 50_000, 100_000, 500_000, 1_000_000]:
        x = robin.setN(n)
        sol = robin.solve()
        label = f"Aproksymacja dla n = {n}"
        plt.plot(x, sol, label=label)
    plt.grid(True)
    plt.ylabel(r"Wartości funkcji $f(x)$")
    plt.xlabel(r"Wartości $x$")
    plt.title(robin.label)
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

Rysunek 19: Dane wejściowe do programu



Rysunek 20: Wykres z rozwiązaniem dla małych n



Rysunek 21: Wykres z rozwiązaniem dla dużych n

9.1 Wnioski

Dla liczby węzłów większej niż 10000 wykresy się praktycznie pokrywały. Niestety bez rozwiązania analitycznego ciężko jest sprawdzić czy nasze rozwiązanie na pewno jest prawidłowe.

Dużym plusem tego zagadnienia jest to, że możemy je wykorzystać do modelowania innych zagadnień. Dzięki temu mamy możliwość rozwiązania każdego równania różniczkowego II rzędu z dowolnymi warunkami brzegowymi.

Kolejnym plusem jest możliwość skorzystania z metody Thomasa do rozwiązywania układu trójprzekątniowego. Dzięki niej możemy liczyć rozwiązania dla macierzy milion na milion.

Algorytm został także sprawdzony na innych zagadnieniach. Zwracał on wyniki podobne do innych metod. Największy problem z jakim się spotkałem podczas badania algorytmów była próba rozwiązania zagadnienia:

$$y'' - 2y' + y = 0$$

Jest to proste jednorodne równanie różniczkowe II rzędu o stałych współczynnikach. Za wartości brzegowe obrałem zagadnienie częściowe Neumanna postaci

$$\begin{cases} y'|_{x=0} = 2 \\ y|_{x=1} = 2e \end{cases}$$

Rozwiązaniem analitycznym jest

$$y = e^x + xe^x$$

Równanie jest bardzo proste z perspektywy analitycznej. Ze strony numerycznej zarówno algorytm rozwiązujący zagadnienie Robina jak i Neumanna nie podołały z takim równaniem.