Sprawozdanie - laboratorium 2

Rozwiązywanie układów równań liniowych metodami iteracyjnymi – transport ciepła

Krystsina Mironenka

1. Wstęp teorytyczny:

Jednym z podstawowych problemów inżynierskich jest analiza transportu ciepła, zwłaszcza w konstrukcjach pracujących w ekstremalnych warunkach temperaturowych. Takim przypadkiem jest projektowanie pieców do termicznej utylizacji zagrożeń biologicznych, gdzie wymagana jest wysoka i stabilna temperatura pracy rzędu 1000°C. Aby prawidłowo zaprojektować izolację cieplną takiego pieca, konieczne jest określenie rozkładu temperatury w ścianie pieca w stanie ustalonym. Problem ten można opisać przy pomocy jednowymiarowego stacjonarnego równania przewodnictwa cieplnego:

$$\frac{d}{dx}\left(\lambda(x)\frac{dT}{dx}\right)=0$$

gdzie: T(x) – temperatura, λ(x) – współczynnik przewodzenia ciepła zależny od położenia. Warunki brzegowe zakładają temperaturę 1000°C od strony wnętrza pieca oraz 100°C od strony zewnętrznej. Aby rozwiązać ten problem numerycznie, konieczne jest przeprowadzenie dyskretyzacji, czyli zamiany ciągłego równania różniczkowego na układ równań liniowych. W tym celu dzielimy analizowany obszar (ścianę pieca) na skończoną liczbę węzłów. Przy zastosowaniu metody różnic skończonych uzyskujemy trójdiagonalny układ równań liniowych, który można zapisać w postaci macierzowej At=b, gdzie: A – macierz współczynników, t – wektor temperatur w węzłach, b – wektor wynikający z warunków brzegowych. Rozwiązywanie takich układów metodami iteracyjnymi jest często bardziej efektywne niż stosowanie metod bezpośrednich, zwłaszcza dla dużych macierzy. W niniejszym ćwiczeniu zastosowano metodę największego spadku, która pozwala na stopniowe przybliżanie rozwiązania przy wykorzystaniu iteracyjnego obliczania wektora reszt i wektora temperatur. Kolejne przybliżenia są wyznaczane zgodnie ze wzorami:

$$\vec{r_k}=\vec{b}-A\vec{t_k}$$

$$\alpha_k=\frac{\vec{r_k}^T\vec{r_k}}{\vec{r_k}^T A\vec{r_k}}$$

$$\vec{t_{k+1}}=\vec{t_k}+\alpha_k\vec{r_k}$$

Proces iteracyjny jest kontynuowany do momentu osiągnięcia zadanej dokładności (normy euklidesowej wektora reszt) lub przekroczenia maksymalnej liczby iteracji. Poprawność wyników można ocenić na podstawie rozkładu temperatury oraz zbieżności normy wektora reszt w kolejnych iteracjach.

2. Zadanie:

W celu rozwiązania zagadnienia stacjonarnego transportu ciepła przez ścianę pieca zastosowano metodę numeryczną, opartą na iteracyjnej metodzie największego spadku. Analizowany problem został upraszczony do postaci jednowymiarowej, a następnie zdyskretyzowany – ścianę pieca podzielono na n=9 równych segmentów, w których obliczano temperaturę. Na podstawie funkcji opisującej współczynnik przewodzenia ciepła w zależności od położenia (funkcja lambda_f), obliczono wartości współczynników λ w każdym węźle. Następnie utworzono trójdiagonalną macierz A, która opisuje zależności pomiędzy temperaturami w sąsiednich węzłach. Wektor prawej strony b skonstruowano, uwzględniając zadane warunki brzegowe – temperatury 1000°C po stronie wewnętrznej i 100°C po stronie zewnętrznej ściany. Wartości te wpłynęły na pierwszy i ostatni element wektora b, zgodnie z równaniem różniczkowym przewodzenia ciepła. Rozwiązanie układu równań At=b przeprowadzono metodą największego spadku. Polegała ona na iteracyjnym poprawianiu przybliżenia wektora temperatur t poprzez obliczanie wektora reszt, wyznaczanie optymalnego kroku iteracji α i aktualizowanie wartości temperatury. Proces ten powtarzano aż do osiągnięcia zadowalającej dokładności, określonej jako norma euklidesowa wektora reszt mniejsza niż 10−6, lub do przekroczenia maksymalnej liczby iteracji (60 000). Obliczone wartości temperatur zostały zapisane do pliku tekstowego, co umożliwia dalszą analizę wyników i wizualizację rozkładu temperatury w ścianie pieca. Całość została zaimplementowana z wykorzystaniem biblioteki GSL, umożliwiającej wygodną obsługę macierzy i wektorów oraz precyzyjne obliczenia numeryczne.

3. Wyniki:

Dla budowania macierzy A otrzymaliśmy następny kod:

```
// Funkcja budująca macierz A i wektor b na podstawie funkcji lambda_f
void build_matrix(gsl_matrix *A, gsl_vector *b, int n)
{
	double h = 1.0 / (n + 1);
	for (int i = 0; i < n; i++)
	{
		double xi = (i + 1) * h;
		double lam_1 = lambda_f(xi - 0.5 * h);
		double lam_2 = lambda_f(xi + 0.5 * h);

		gsl_matrix_set(A, i, i, -lam_1 - lam_2);
		if (i < n - 1)
			gsl_matrix_set(A, i, i + 1, lam_2);
		if (i > 0)
			gsl_matrix_set(A, i, i - 1, lam_1);
	}

	gsl_vector_set(b, 0, -lambda_f(0.5 * h) * TL); // Ustawienie wartości na lewym końcu
	gsl_vector_set(b, n - 1, -lambda_f(1 - 0.5 * h) * TR); // Ustawienie wartości na
prawym końcu
}
```

Dla n = 9, powstała trójdiagonalną macierz A:

-0.6 0.3 0 0 0 0 0 0 0
0.3 -0.6 0.3 0 0 0 0 0 0
0 0.3 -0.6 0.3 0 0 0 0 0
0 0 0.3 -0.5 0.2 0 0 0 0
0 0 0 0.2 -0.4 0.2 0 0 0
0 0 0 0 0.2 -0.4 0.2 0 0
0 0 0 0 0 0.2 -0.3 0.1 0
0 0 0 0 0 0 0.1 -0.2 0.1
0 0 0 0 0 0 0 0.1 -0.2

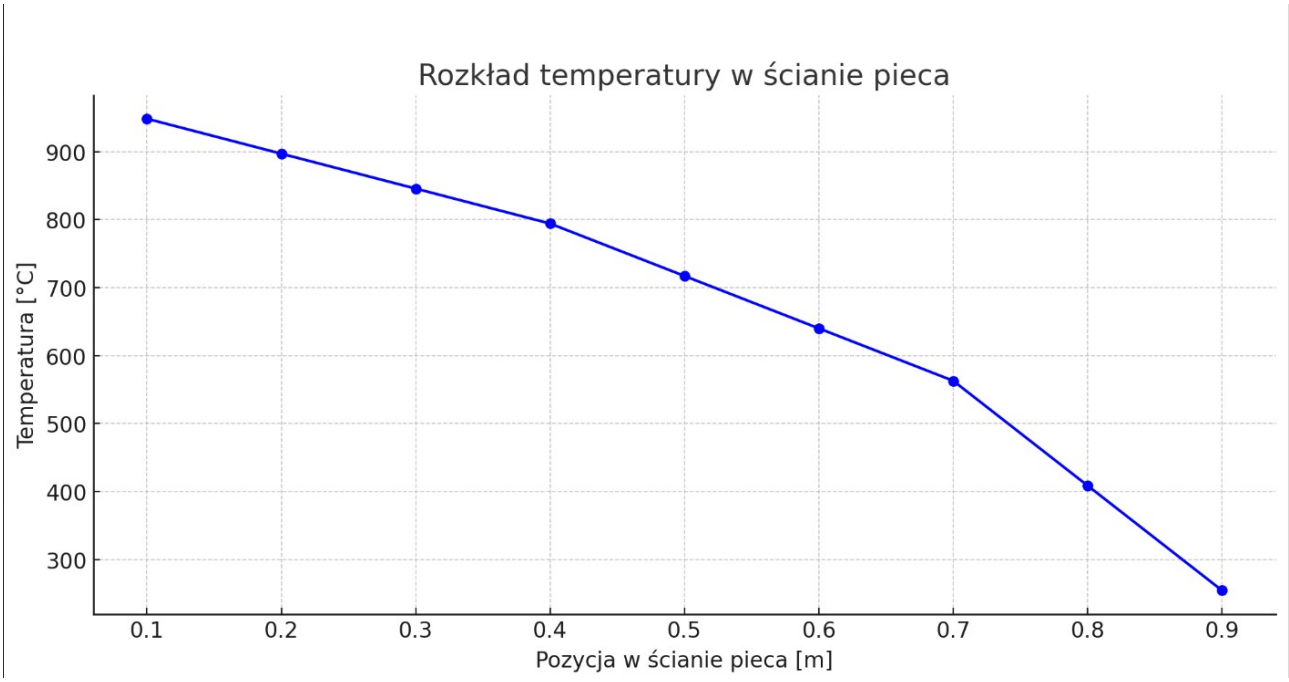
Dla rozwiązania zadania zostałą zaimplimentowana funkcja realizująca metodę największych spadków:

```
void steepest_descent(const gsl_matrix *A, const gsl_vector *b, gsl_vector *t, ofstream &outFile)
{
	std::ofstream outFile1("wektor_reszt.txt");
	std::ofstream outFile2("wektor_temp.txt");
	int n = b->size;
	gsl_vector *r = gsl_vector_calloc(n); // Wektor reszty
	gsl_vector *Ar = gsl_vector_calloc(n); // Wektor Ax
	double alpha, rr, rAr; // Zmienna alpha (współczynnik krokowy), rr (iloczyn wektora r), rAr
	(iloczyn r i Ar)

	for (int iter = 0; iter < MAX_ITER; iter++)
	{
	matrix_vector_mult(A, t, Ar); // Obliczenie Ax
	gsl_vector_memcpy(r, b); // r = b
	daxpy(-1.0, Ar, r); // r = b - Ax

	rr = vector_product(r, r); // Iloczyn skalarny r z r
	matrix_vector_mult(A, r, Ar); // Obliczenie A * r
	rAr = vector_product(r, Ar); // Iloczyn skalarny r z Ar

	alpha = rr / rAr; // Obliczenie współczynnika krokowego alpha
	daxpy(alpha, r, t); // t = t + alpha * r
	outFile1 << iter << " " << 0.5 * log10(rr) << std::endl;
	outFile2 << iter << " " << sqrt(vector_product(t, t)) << std::endl;

	// Warunek zakończenia iteracji
	if (sqrt(rr) < EPS)
		break;
	}

	gsl_vector_free(r);
	gsl_vector_free(Ar);
}
```

W którejw każdej iteracji zapisujemy wartości normy euklidesowej wektora reszt $\log(\|r_k\|_2)$ oraz normy wektora rozwiązań $\|t_k\|_2$.

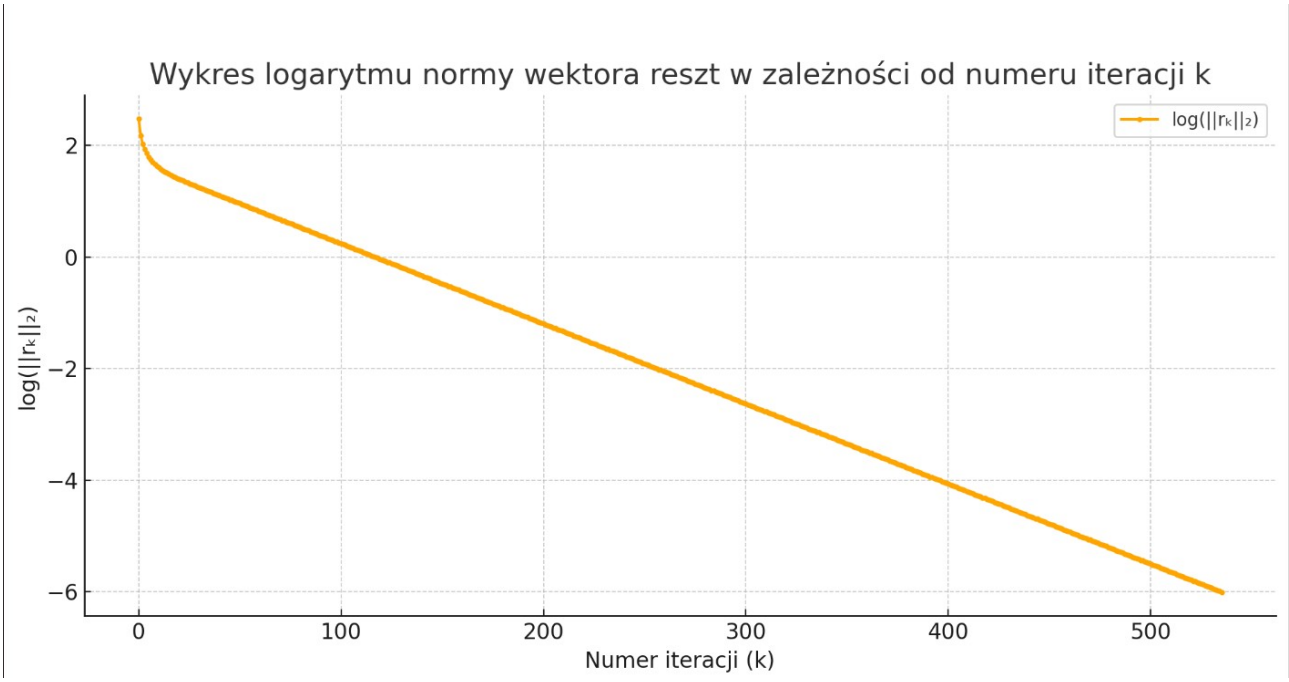
Otrzymane rozwiązanie:

0.1	948.571
0.2	897.143
0.3	845.714
0.4	794.286
0.5	717.143
0.6	640
0.7	562.857
0.8	408.571
0.9	254.286



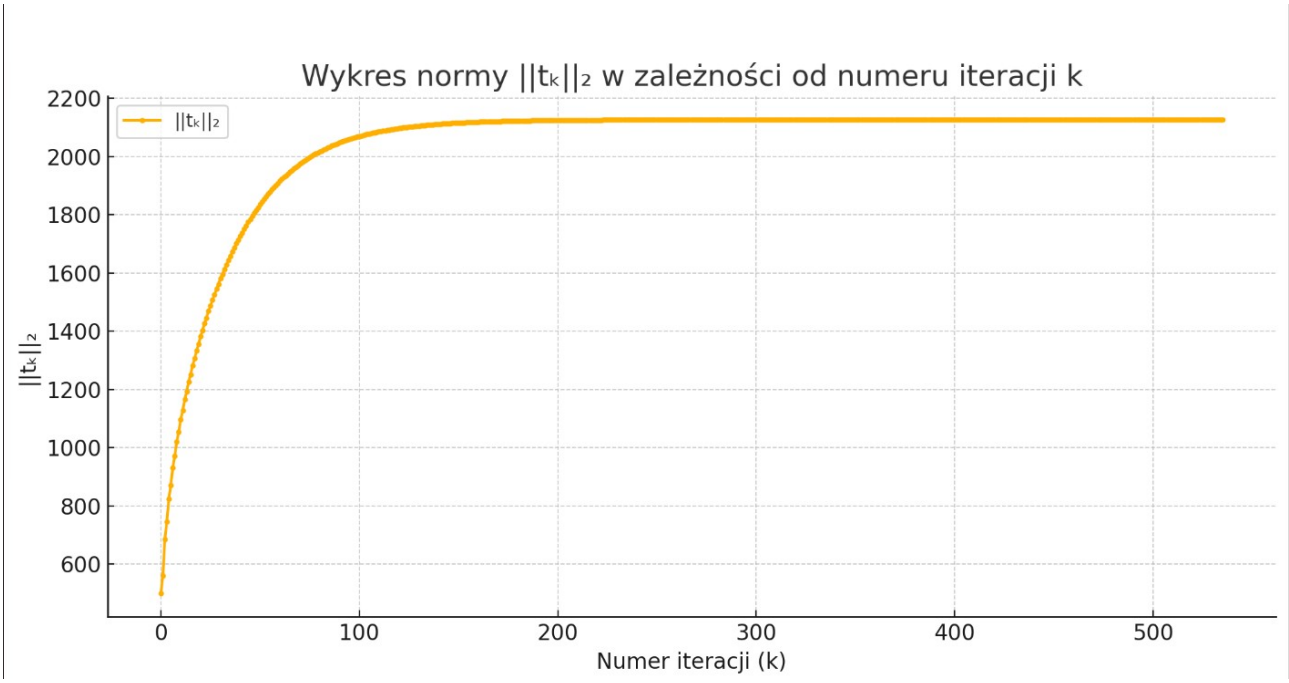
Rysunek 1. Rozkład temperatury w ścianie pieca

Wykres przedstawia rozkład temperatury w ścianie pieca w funkcji odległości od wewnętrznej do zewnętrznej strony ściany. Jak widać, temperatura stopniowo maleje wraz ze wzrostem odległości, co jest zgodne z oczekiwaniami fizycznymi dotyczącymi przenikania ciepła.



Rysunek 2. Wykres logarytmu normy wektora reszt

Wykres pokazuje logarytm normy wektora reszt $\log(\|r_k\|_2)$. Tutaj widzimy monotonny spadek wartości, co świadczy o poprawnym działaniu metody iteracyjnej – reszta systematycznie maleje, co wskazuje na zbieżność metody do rozwiązania.



Rysunek 3. Wykres normy temperatury

Wykres przedstawia normę wektora rozwiązań $\|t_k\|_2$ w zależności od numeru iteracji k. Obserwujemy, że norma początkowo szybko rośnie, a następnie osiąga stabilizację, co oznacza, że rozwiązanie zbiega się do pewnej wartości.

Oba wykresy wskazują, że rozwiązanie iteracyjne przebiega poprawnie i spełnia warunki zadane w ćwiczeniu.

4. Wnioski:

Na podstawie wyników uzyskanych metodą największego spadku można stwierdzić, że konstrukcja pieca spełnia założenia — temperatura na końcu ściany zbiega do ~100°C, zgodnie z warunkiem brzegowym. Metoda wykazuje dobrą zbieżność: norma wektora reszt szybko maleje, co widoczne jest na wykresie $\log(\|rk\|)_2$. Proces zbieżności stabilizuje się już po około 500 iteracjach. W przypadku zwiększenia liczby węzłów (n = 99), otrzymujemy dokładniejszy rozkład temperatury, ale jednocześnie wydłuża się czas obliczeń oraz liczba potrzebnych iteracji. Porównanie wykresów pokazuje większą płynność i szczegółowość temperatury dla większej siatki. Dodatkowo, wartość początkowa wektora t ma wpływ na szybkość zbieżności. Jeżeli wektor początkowy znajduje się bliżej rozwiązania (np. stała wartość 500), metoda dochodzi do wyniku szybciej niż w przypadku startu od zera lub losowych wartości. Jednak w każdym przypadku końcowy rezultat pozostaje taki sam — różni się tylko czas obliczeń i liczba iteracji.