***Zadanie 2***

**Opis:**

Wykres wygenerowany za pomocą programu GNUplot. Program wykonany w języku C++

**Kompilacja/uruchomienie programu:**

Aby skompilować program, którego kod znajduje się na końcu tego dokumentu można skopiować go do wybranego IDE np.: Visual Studio, Code Block itd. Bądź uruchomić z poziomu konsoli wybranymi komendami, tak jak zwykły program w C++. Program po takim uruchomieniu wypisze w słupku rozwiązania układu równań.

**Metoda rozwiązania/ dyskusja:**

Cały ciężar zadania polegał na rozwiązaniu układu, w którego skład wchodziła macierz trójdiagonalną. Do tego typu zadań można użyć eliminacji Gaussa, lecz złożoność obliczeniowa tej metody to aż O(n^3),rozkład LU też mógłby zostać użyty lecz wygład tej macierzy nie jest odpowiedni do tej metody (dużo zer) , stąd zdecydowałem się na użycie algorytmu Thomasa - O(n). Pomogły mi w jego implementacji informacje z tej strony: https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm.

Aby uniknąć niepotrzebnych dzieleń wyciągnąłem czynnik $\frac{1}{h^2}$ przed macierz. Przemnożyłem na końcu programu.

Std::cout zastąpiłem mniej zasobożernymi operacjami printf.

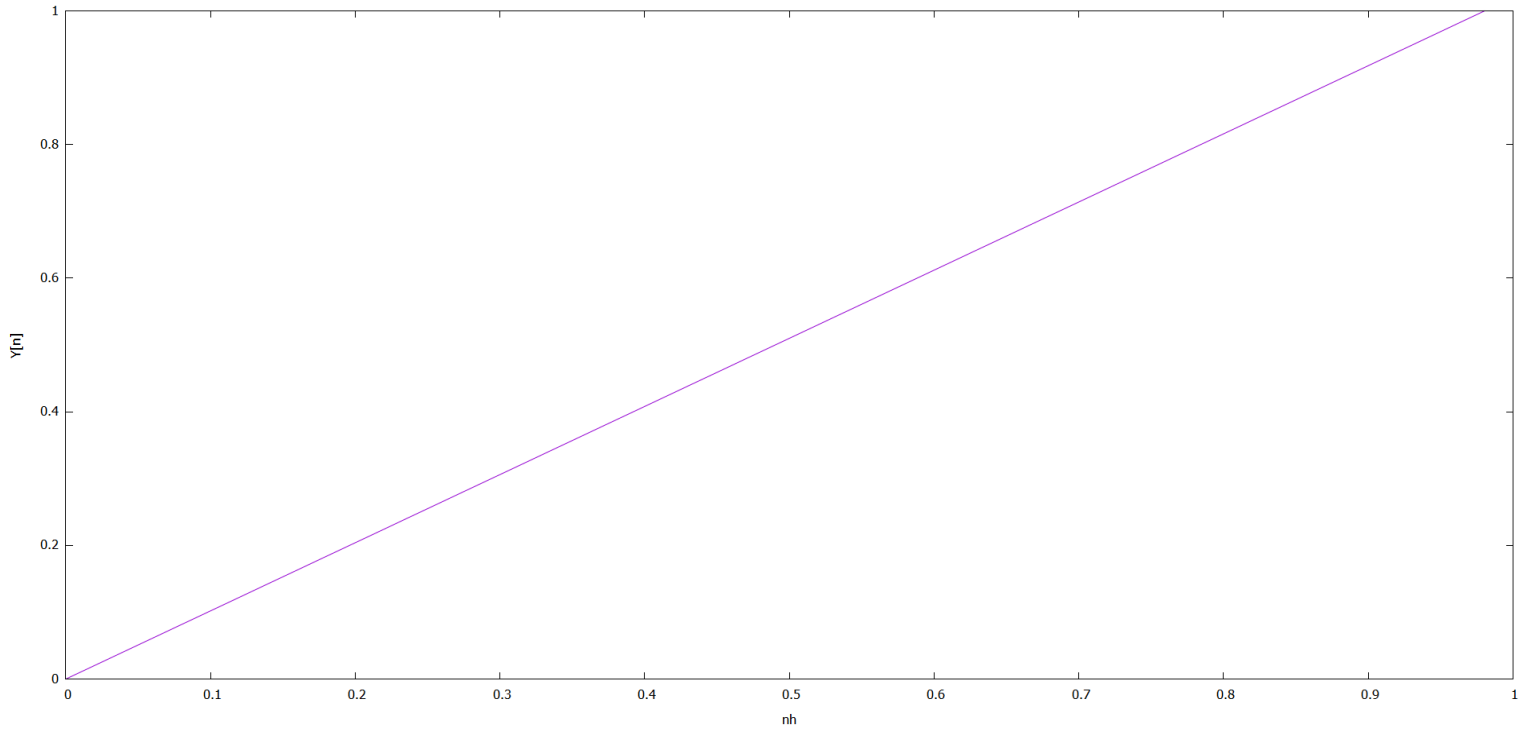
Zamiast wektorów (użytych w poprzednim zadaniu w zestawie(przykład błędny)-drogie operacje push) użyłem zwykłych tablic. Za pomocą ich wykonałem trzy diagonale A B i C i wektory f oraz y.

Aby móc użyć algorytmu Thomasa niezbędne jest wyliczenie współczynników c i d (tak nazwanych w Wikipedii).

Wynik otrzymuje korzystając z back substitution.

**Graficzne przedstawienie wyników działania programu:**

Zadanie 2



**Kod programu:**

```
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <math.h>


using namespace std;

int const N = 51;
double const h = 1.0 / N;
double const H = (1.0 / (h * h));

int main()
{
	double wek_f[N];
	double wspol_c[N];
	double wspol_d[N];
	double wek_y[N];
	double diagA[N];
	double diagB[N];
	double diagC[N];

	diagC[0] = 0;
	//wypelnienie jedynkami diagonali c
	for (int i = 1; i < N - 1; i++)
	{
		diagC[i] = 1.0;
	}


	diagC[N - 1] = 0;
	diagA[N - 1] = 0;
	// diagonala a po wyciagnieciu 1/h^2 przed macierz
	diagA[0] = 0;
	for (int i = 1; i < N - 1; i++)
	{
		diagA[i] = 1.0;
	}






	//wstawianie do diagonali B
	for (int i = 1; i < N - 1; i++)
	{
		diagB[i] = -2.0;
	}

	diagB[0] = (1.0/ (H));
	diagB[N - 1] = (1.0/(H));






	for (int i = 0; i <= N - 1; i++)
	{
```

```
		if (i == N - 1)
			wek_f[i] = 1.0;
		else
			wek_f[i] = 0;
	}

	wspol_c[0] = 0;

	for (int i = 1; i <= N - 1; i++)
	{
		wspol_c[i] = diagC[i] / (diagB[i] - (diagA[i] * wspol_c[i - 1]));
	}

	wspol_d[0] = 0;

	for (int i = 1; i <= N - 1; i++)
	{
		wspol_d[i] = (wek_f[i] - (diagA[i] * wspol_d[i - 1])) / (diagB[i] –
		(diagA[i] * wspol_c[i - 1]));
	}

	wek_y[N - 1] = wspol_d[N - 1] * (1.0/H);

	for (int i = N - 2; i >= 0; i--)
	{
		wek_y[i] = (wspol_d[i] - (wspol_c[i] * wek_y[i + 1])) ;
	}


	//wydrukowanie wynikow
	for (int i = 0; i <= N - 1; i++)
	{
		printf("%f \n", wek_y[i]);
	}
	printf("\n\n%f, %f \n", diagB[0], diagB[N - 1]);


	return 0;
}
```