Zadanie 3

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:

Zadanie polegało na rozwiązaniu układu z macierzą trójdiagonalną z dodatkową wartością w dolnym rogu. Do rozwiązania tego zadania użyłem metody Shermana–Morrisona. Algorytm polega na rozwiązaniu dwóch równań $Ay = b$ i $Az = u$ z ta samą macierzą trójdiagonalną bez elementu w rogu macierzy. A następie obliczenie x = y-$\frac{v*y}{1+v*z}$*z. Aby rozwiązać te równania uzylem dwukrotnie algorytmu z poprzedniego zadania (algorytm Thomasa O(n)).

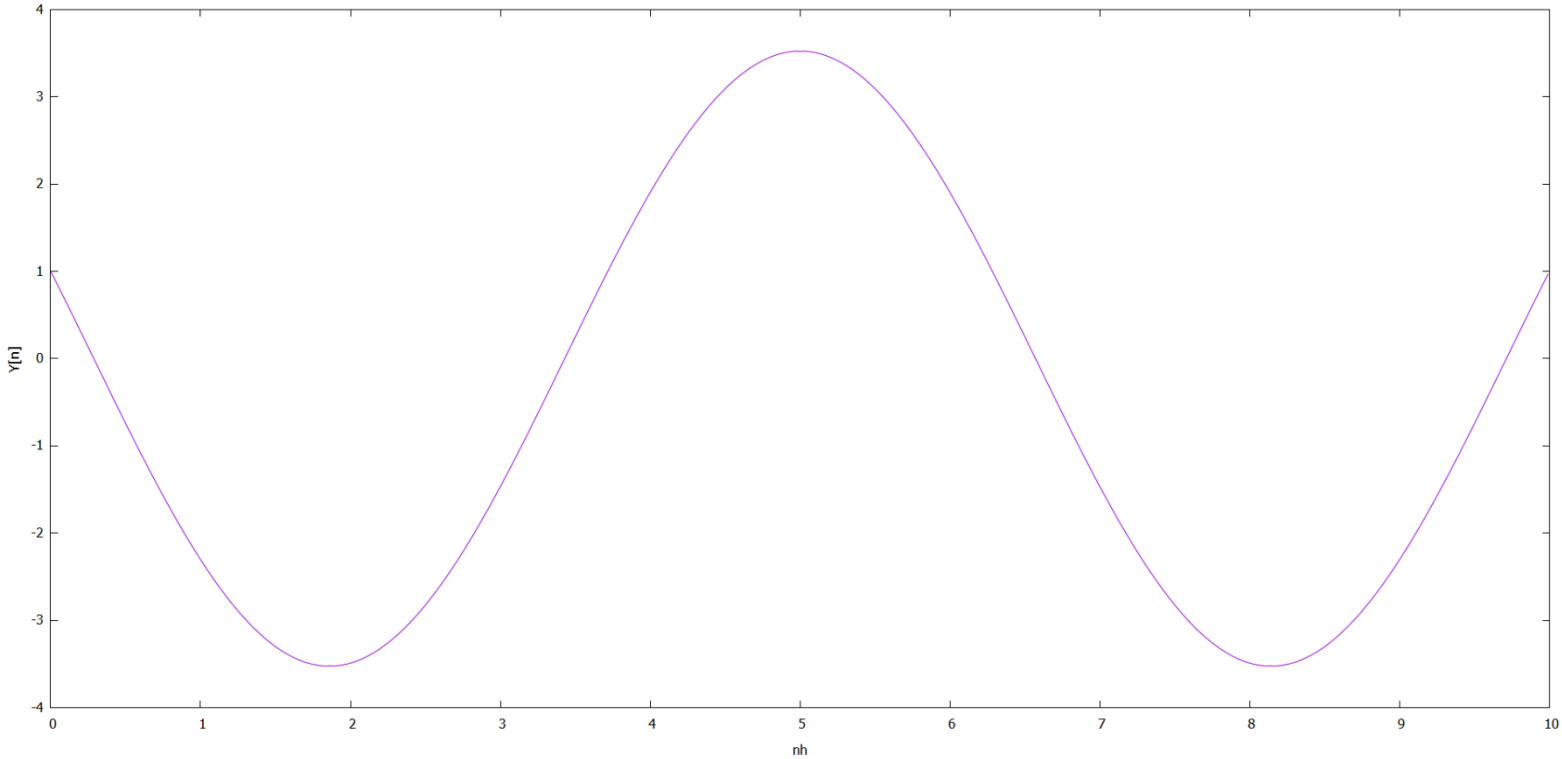
Std::cout zastąpiłem mniej zasobożernymi operacjami printf.

Zamiast wektorów (drogie operacje push) użyłem zwykłych tablic.

No i brak pow(), tylko zwykle mnożenie.

Graficzne przedstawienie wyników działania programu:

Zadanie 3



Kod programu:

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


using namespace std;
const double h = 0.01;
const int N = 1000;
const double alfa = 1;
const double beta = 0;

int main()
{
	double diagA[N];
	double diagB[N];
	double diagC[N];
	double d[N];
	double d1[N];
	double c[N];
	double x[N];
	double x1[N];
	double r[N];


	double bb[N];
	double u[N];
	double z[N];
	double f;
	double gamma;

	//diagonala a
	for (int i = 1; i <= N - 1; i++)
	{
		diagA[i] = 1.0;
	}

	diagA[0] = 0;

	//diagonala b
	diagB[0] = 1;

	for (int i = 1; i < N - 1; i++)
	{
		diagB[i] = (-2.0 + h * h);
	}

	diagB[N - 1] = -2;

	//diagonala c

	for (int i = 1; i < N - 1; i++)
	{
		diagC[i] = 1.0;
	}

	diagC[0] = 0;
	diagC[N - 1] = 0;

	//wektor F
```

```
	r[0] = 1.0;
	for (int i = 1; i < N; i++)
	{
		r[i] = 0;
	}

	//cialo metody Shermana-Morisona
	gamma = -diagB[0];
	bb[0] = diagB[0] - gamma;
	bb[N - 1] = diagB[N - 1] - (alfa * beta) / gamma;

	for (int i = 1; i < N - 1; i++)
	{
		bb[i] = diagB[i];
	}
	for (int i = 0; i < N; i++) {
	}

	// Ax = r
	c[0] = diagC[0] / bb[0];

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

	d[0] = r[0] / bb[0];

	for (int i = 1; i <= N - 1; i++)
	{
		d[i] = (r[i] - (diagA[i] * d[i - 1])) / (bb[i] - (diagA[i] * c[i - 1]));
	}

	x[N - 1] = d[N - 1] / N * N;


	for (int i = N - 2; i >= 0; i--)
	{
		x[i] = (d[i] - (c[i] * x[i + 1])) / N * N;
	}

	u[0] = gamma;
	u[N - 1] = alfa;

	for (int i = 1; i < N - 1; i++)
	{
		u[i] = 0;
	}

	// Az = u

	d1[0] = u[0] / bb[0];

	for (int i = 1; i <= N - 1; i++)
	{
		d1[i] = (u[i] - (diagA[i] * d1[i - 1])) / (bb[i] - (diagA[i] * c[i -
1]));
	}

	z[N - 1] = d1[N - 1] / N * N;
```

```
	for (int i = N - 2; i >= 0; i--)
	{
		z[i] = (d1[i] - (c[i] * z[i + 1])) / N * N;
	}

	f = (x[0] + beta * z[N - 1] / gamma) / (1.0 + z[0] + beta * z[N - 1] / gamma);

	for (int i = 0; i <= N - 1; i++) {
		x1[i] = x[i] - f * z[i];
	}

	for (int i = 0; i <= N - 1; i++) {
		printf("%f \n", x1[i]);
	}

	return 0;
}
```