Zadanie 6

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 rozwiązanie (wektor x).

Metoda rozwiązania/ dyskusja:

Zadanie 6 polegalo na rozwiązaniu równania z zadania 6 z poprzedniego zestawu za pomocą opisanych w zestawie gradientów sprzężonych.

Równanie było następujące: AX=B

Macierz A: $\begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}$ Wektor b: $\begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix}$ i należy znaleźć wektor x.

Std::cout zastąpiłem mniej zasobożernymi operacjami printf.

Zamiast wektorów (drogie operacje push) użyłem zwykłych tablic.

No i brak pow(), tylko zwykle mnożenie.

Metoda Gradientów Sprzężonych :

Jest algorytmem numerycznym służącym do rozwiązywania niektórych układów równań liniowych. Pozwala rozwiązać te, których macierz jest symetryczna i dodatnio określona. Metoda gradientu sprzężonego jest metodą iteracyjną. Aby skorzystać z metody gradientów sprzężonych postępujemy zgodnie z poniższymi wzorami (wzory z zestawu):

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k},$$

$$r_{k+1} = r_k - \alpha_k A p_k,$$

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k},$$

$$p_{k+1} = r_{k+1} + \beta_k p_k.$$

Wynik działania programu: Szukany wektor X= $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$ z dokładnością do 8 miejsca po przecinku.

Opis:

Na sam początek przyjąłem wg algorytmu (https://pl.wikipedia.org/wiki/Metoda_gradientu_sprz%C4%99%C5%BConego) początkową wartość p=r, a r = b – A*x0, x0 to wektor zerowy. Najpierw według pierwszego wzoru wyliczamy alfa i pierwszą przybliżoną wartość x i z każdym kolejnym krokiem wstawiając kolejne wartości za wektor p , r i beta (z powyższych wzorów) będziemy uzyskiwać coraz to dokładniejszy wynik.

Kod programu:

```
#include <iostream>

using namespace std;
int const n = 5;
double const epsilon = 1e-8;

//double norma(double x[])
//{
//     int norm = 0;
//     double xn;
//     for (int i = 0; i < 3; i++)
//     {
//          xn = x[i] * x[i];
//          norm += xn;
//     }
//     return sqrt(norm);
//
//}

int main()
{


       //macierz A
       double A[3][3] = { 4, -1, 0,
                                     -1, 4, -1,
                                      0, -1, 4};
       //wektor b
       double b[3] = { 2, 6 , 2 };
       double x[3] = { 0,0,0 };
       //double xn[3] = { 0,0,0 };
       double r[3] = { 2,6,2 };
       double p[3] = { 2,6,2 };
       double r_r[3];
       double p_p[3];
       double alfa;
       double beta;
       int counter = 1;
```

```
	while (counter<n){//while (norma(x) -norma(xn)) < epsilon) {

		alfa = (r[0] * r[0] + r[1] * r[1] + r[2] * r[2]) /
			((p[0] * A[0][0] + p[1] * A[0][1] + p[2] * A[0][2]) * p[0] +
			(p[0] * A[1][0] + p[1] * A[1][1] + p[2] * A[1][2]) * p[1] +
				(p[0] * A[2][0] + p[1] * A[2][1] + p[2] * A[2][2]) * p[2]);

		for (int i = 0; i < 3; i++) {
			x[i] = x[i] + alfa * p[i];
		}

		for (int i = 0; i < 3; i++) {
			r_r[i] = r[i] - ((A[0][i] * p[0] + A[1][i] * p[1] + A[2][i] *
p[2]) * alfa);
		}

		beta = (r_r[0] * r_r[0] + r_r[1] * r_r[1] + r_r[2] * r_r[2]) /
			(r[0] * r[0] + r[1] * r[1] + r[2] * r[2]);

		for (int i = 0; i < 3; i++) {
			p[i] = r_r[i] + beta * p[i];

			r[i] = r_r[i];
		}



	    counter++;

	}

	for (int i = 0; i < 3; i++) {
		printf("%.8f\n",x[i]);

	}

	return 0;
}
```