W zadaniu poproszono mnie o zaimplementowanie metody gradientów sprzężonych dla poniższego układu

$$\begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix}$$

Macierz współczynników będę nazywać macierzą A, a rozwiązań b.
Metoda gradientów sprzężonych to metoda iteracyjna, którą warto wybierać wtedy, gdy macierz jest rzadka i duża - wtedy najlepiej sie sprawdza.

Macierz A musi być symetryczna i dodatnio określona (to znaczy że przemnożona z dwóch stron macierz A przez dowolny niezerowy wektor $x$ i $x^T$musi być większa od 0).

$xAx^T > 0$

Korzystamy z tego, że funkcja

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

ma dokładnie jedno minimum i jest to też minimum globalne. Minimum to leży w punkcie który spełnia: $\nabla f = 0$

Funkcja ta osiąga minimum w punkcie, w którym zachodzi $Ax - b = 0 \Leftrightarrow Ax = b$

Rozwiązywanie układu równań liniowych z macierzą symetryczną, dodatnio określoną jest równoważne poszukiwaniu minimum dodatnio określonej formy kwadratowej.

Czasami jest więc nam łatwiej policzyć to minimum i zastosować to jako metodę bardziej optymalną poszukiwania rozwiązania układu równań liniowych.

Mówimy, że dwa niezerowe wektory u i v są sprzężone (względem A) jeśli $u^T Av = 0$

Więc, dwa wektory są sprzężone jeśli są ortogonalne względem tego iloczynu skalarnego.

Przypuśćmy, że {pk} jest ciągiem n wzajemnie sprzężonych kierunków. Wtedy pk tworzą bazę Rn, wektor x* będący rozwiązaniem Ax = b możemy przedstawić w postaci:

$x_* = \sum_{i-1}^{n} \alpha_i p_i$

Współczynnik $\alpha_i$ możemy wyznaczyć przekształcając powyższe wyrażenie

$p_k^T Ax_* = \sum_{i-1}^{n} \alpha_i p_k^T Ap_i = p_k^T b$

$\alpha_k = \frac{p_k^T b}{p_k^T Ap_k}$

W moim programie korzystam z algorytmu w którym w pętli poszukuje rozwiązania $x_N$

$$
\begin{aligned}
&r_1 = b - Ax_1, p_1 = r_1 \\
while||r_k|| > \epsilon \\
&\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \\
&r_{k+1} = r_k - \alpha_k A p_k \\
&\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
&p_{k+1} = r_{k+1} + \beta_k p_k \\
&\boldsymbol{x_{k+1} = x_k + \alpha_k p_k}
\end{aligned}
$$

wówczas spełnione są zależności:

$$
\begin{cases}
r_i^T r_j = 0 & i > j \\
r_i^T p_j = 0 & i > j \\
p_i^T A p_j = 0 & i > j
\end{cases}
$$

Jeżeli $r_N = 0$ to $x_N$ jest ścisłym rozwiązaniem równania $Ax = b$

Metoda gradientu sprzężonego ma zbiegać w n krokach, gdzie n to rozmiar macierzy, a więc w powyższym przypadku maksymalnie metoda powinna zbiegać w 3 krokach.

W programie zaimplementowałam powyższy algorytm. Utworzyłam dwie funkcję, jedną do obliczania normy, a drugą do liczenia mnożenia wektorów skalarnie dla czytelności programu. Mnożenie macierzy z wektorem wykonuje się w funkcji main, gdyż występował problem z wskaźnikiem do statycznej tablicy wielowymiarowej. Tworzę tablicę dwuwymiarową A ze współczynnkami oraz wektor rozwiązań b. Następnie wyznaczam wektory r i p z równań $r = b - Ax$ i $r = p$ i zapisuję je w tablicach. Obliczam normę do warunku w pętli while. Norma wektora r ma być mniejsza od błędu aby zakończyć obliczenia.

Metoda zbiegła w 2 krokach, a więc tak jak przewidywano - w mniej niż 3 krokach.

Odpowiedź:

Wektorem szukanym jest tablica x o wartościach:

$$x_1 = 1$$
$$x_2 = 2$$
$$x_3 = 1$$

Analiza błędów i złożoność:
Koszt obliczeń tą metodą wynosi O($N^3$). Metoda zbiega po N krokach, zatem koszt wynosi O($N \cdot jedenKrok$), Koszt jednego kroku jest zdominowany przez obliczanie iloczynu $A \cdot p_k$, gdyż jest to składnik najbardziej złożony czasowo/pamięciowo w moim zadaniu. Gdy macierz jest pełna, obliczanie tego iloczynu jest złożoności O($N^2$), a więc $O(N \cdot N^2) = O(N^3)$.

Błędy metody:

Dokładność tej metody jest w tym wypadku większa niż $10^{-8}$, po podstawieniu pod niewiadome powyższych wartości i po wyliczeniu rozwiązania wychodzi mi dokładnie wektor b - więc w tym przykładzie dokładność może być nawet równa dokładności double - $10^{-16}$ .

Instrukcje uruchomienia programu:

1. Aby uruchomić program w języku c na linuxie należy przejść do katalogu w którym znajdują się pliki i wpisać w terminalu komendę:
   –> make all
2. A następnie
   –> make run

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double mnozenieWektor(double *tab, double *tab2,int size){
double sum=0;
for(int i=0;i<size;++i){
sum+=tab[i]*tab2[i];
}
return sum;
}

double liczenieNormy(double *tab,int size){
double norma = 0;
double temp=0;
int i=0;
for(i;i<size;++i){
temp=tab[i]*tab[i];
norma+=temp;
}
norma=sqrt(norma);
return norma;
}


int main(){


double A[3][3]={{4.0,-1.0,0.0},{-1.0,4.0,-1.0},{0.0,-1.0,4.0}};

double b[]={2.0,6.0,2.0};

int N = sizeof(b)/sizeof(double);
int i=0,j=0,k=0;
double norma;

double r[N],p[N],alfa,beta,x[N];
double x0[]={0.0,0.0,0.0};
double sum[N];
double mnoz[N];
double e=0.00000001;
for(int i=0;i<N;++i){ //r=b-Ax
for(j=0;j<N;++j){

sum[i]+=A[i][j]*x0[j];

}
r[i]=b[i]-sum[i];
p[i]=r[i];
}
void mnozenieMacierz(){
```

```
for(i=0;i<N;++i){
mnoz[i]=0;
for(j=0;j<N;++j){
mnoz[i]+=A[i][j]*p[j];
}
}

}
norma=liczenieNormy(r,N);
printf("Norma wynosi: %f\n",norma);
int counter=0;
while(e<norma){
counter++;
mnozenieMacierz();

alfa=mnozenieWektor(r,r,N)/mnozenieWektor(p,mnoz,N);
double rkrk=mnozenieWektor(r,r,N);
for(i=0;i<N;++i){
x[i]+=alfa*p[i];
r[i]-=alfa*mnoz[i];
}

norma=liczenieNormy(r,N);

beta=mnozenieWektor(r,r,N)/rkrk;

for(i=0;i<N;++i){
p[i]=r[i]+beta*p[i];
}

}
printf("%.16f\n",x[0]);

printf("%.16f\n",x[1]);

printf("%.16f\n",x[2]);
printf("%d\n",counter);

double odp[N];
i=0;j=0;
for(i;i<3;i++){
for(j;j<3;j++){
odp[i]+=A[i][j]*x[j];
}
j=0;
odp[i]=b[i]-odp[i];
printf("ODP %.16f\n",odp[i]);
}
return 0;

}
```