ZADANIE 7

W programie mamy zaimplementować metodę gradientów sprzężonych:

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

#define N 1001

void conjugateGradients(double* vectorN) {
    double norm = 1.0;
    double h2 = 2.000001;
    double r[N], rNew[N], p[N], Ap[N], alpha, beta;
    double divident = 0.0;
    double divisor = 0.0;
```

Używamy powyższych zmiennych. Całość działania programu opiera się na schemacie z wykładu:

$\mathbf{A} \in \mathbb{R}^{N \times N}$ symetryczna, dodatnio określona, $\mathbf{x}_1$ — początkowe przybliżenie rozwiązania równania (12), $0 < \varepsilon \ll 1$.

$$
\begin{array}{l}
\mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_1,\ \mathbf{p}_1 = \mathbf{r}_1 \\
\textbf{while}\ \|\mathbf{r}_k\| > \varepsilon \\
\qquad \begin{aligned}
\alpha_k &= \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \\
\mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k \\
\beta_k &= \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k} \\
\mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k
\end{aligned} \\
\textbf{end}
\end{array}
\tag{13}
$$

```c
r[0] = 1.0 - vectorN[0];
r[N - 1] = 1.0 - vectorN[N - 1];

for(int i = 1; i < N - 1; i++)
    r[i] = vectorN[i - 1] + vectorN[i + 1] - h2 * vectorN[i];

for(int i = 0; i < N; i++)
    p[i] = r[i];
```

Pierwsze obliczamy fragment r = b - Ax. Ustawiamy element pierwszy i ostatni vectora jako 1.0 a następnie w pętli for nadajemy wartościom p[i] wartość r[i].

```
    for(int j = 0; j < 97592; j++) {
        Ap[0] = p[0];
        Ap[N - 1] = p[N - 1];

        for(int i = 1; i < N - 1; i++)
            Ap[i] = h2 * p[i] - p[i - 1] - p[i + 1];
```

Następnie “wchodzimy” w pętle która na wykładzie określona jest jako while. Do znalezienia optymalnej ilości przejść programu użyłem bardzo prostego fragmentu kodu:

```
        /*il++;
        if(vectorN[500]== 0.8868188924934421)
        {printf("%i\n",il);
            break;}*/
    }
}
```

Wystarczyło pierwsze przeiterować pętle milion razy i sprawdzić jaki wynik otrzymamy dla wartości w połowie N (jako że wyniki “schodzą się” do środka). Nastepnię po prostu sprawdziłem jaką wartość wyrzuca il i zaimplementowałem je w kodzie. Nie jest to może bardzo elegancki sposób liczenia tej wartości, ale skuteczny.

Po wejściu w pętle nadajemy wartość początkową i końcową wektora Apk. W kolejnej pętli for wektor przyjmuje dla pozycji i wartość 2.000001 * p[i] - poprzednik i następnik pozycji itej.

```
        for(int i = 0; i < N; i++){
            divident += r[i] * r[i];
            divisor += p[i] * Ap[i];
        }

        alpha = divident / divisor;
```

Kolejnym krokiem jest obliczanie alphy. Dla ułatwienia użyłem dwóch zmiennych: divisor oraz divident którym wartości są nadawane w pętli for.

```
        for(int i = 0; i < N; i++)
            rNew[i] = r[i] - alpha * Ap[i];

        for(int i = 0; i < N; i++){
            divident += rNew[i] * rNew[i];
            divisor += r[i] * r[i];
        }

        beta = divident / divisor;
```

Następnie program oblicza wartość r(k+1) przedstawioną tutaj jako zmienna rNew[i]. Poniżej mamy analogiczne do alphy obliczanie bety.

```c
            for(int i = 0; i < N; i++)
                vectorN[i] = vectorN[i] + alpha * p[i];

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

Ostatnie co dzieje się w funkcji to przypisanie danym p oraz vectorN ich nowych wartości.

```c
int main() {
    double x[N] = {0.0};
    FILE *data;
     data  = fopen("wynikGradient.txt","w");
    if(!data){
        perror("Error opening file");
        exit(1);
    }

    conjugateGradients(x);
    for(int i = 0; i < N; i++){
        printf("%.16lf\n", x[i]);
        fprintf(data, "%i\t %.16lf \n",i,x[i]);
        }
}
```

Na sam koniec program wypisuje otrzymane wyniki i zapisuje je w pliku “wynikGradient.txt”.