ZADANIE N6

Program implementuje metody iteracyjne i rozwiązuje macierz z zadania N5. Program zwraca ilość iteracji oraz czas w jakim wykonały sie dane metody.

```
#define N 1001 // wykorzystywana jako rozmiar macierzy


double Diagonal = 2.000001; // wartość na diagonali w macierzy
double epsilon = 1e-10; //zmienna potrzebna do ustalenia dokładności wyniku
bool close = false; //zmienna której używam podczas obliczania ilości wykonań programu
```

W każdej z użytych funkcji będziemy korzystać z tego samego członu do obliczania iteracji programu:

```
35          diff = 0.0;
36          for(int i = 0; i < N; i++){
37              if(diff < fabs(relaxation[i] - vectorN[i])){
38                  diff = fabs(relaxation[i] - vectorN[i]);
39              }
40          }
41
42
43          if(diff < epsilon){
44              close = true;
45
46          }
47
48          for (int i = 0; i < N; i++) {
49              vectorN[i] = relaxation[i];
50          }
51          *il = *il + 1;
52      }
53  }
```

W zależności od obliczanej metody nazwa tablicy będzie się zmieniać (relaxation, jacobi, gausSeidle, sor). Zmienna diff służy do sprawdzania ile wynosić będzie obecna różnica między kolejnymi

iteracjami (sprawdzamy dokładność wyniku). wykorzystujemy fabs() żeby uzyskać wartość bezwględną i następnie za pomocą if sprawdzamy czy osiągnęliśmy już szukaną dokładność. Jeśli tak program

ustawia parametr close jako true co zakończy pętle while i zwróci wartość il. Pętla na samym dole przepisuje nowe wyniki do tablicy po czym na samym końcu powiększamy parametr il o 1.

Metoda relaksacyjna

```
25          Ax[0] = vectorN[0];
26          Ax[N - 1] = vectorN[N - 1];
27          for(i = 1; i < N - 1; i++)
28              Ax[i] = Diagonal * vectorN[i] - vectorN[i - 1] - vectorN[i + 1];
29
30          for(i = 0; i < N; i++)
31              if(i == 0 || i == N - 1){
32                  relaxation[i] += gamma * (1.0 - Ax[i]);
33                  }
34              else{
35                  relaxation[i] -= gamma * Ax[i];
36              }
```

Pierwsze program przypisuje zmiennej relaxation wartości vectoraN, następnie żeby dostać wartość naszego Ax[N] mnożymy wartość diagonalną razy vectorN w zależności od danej pozycji i oraz odejmujemy od tej wartości poprzednik i następnik vectora. Jako ostatnie w pętli. W nastepnej pętli program sprawdza czy wartość na której się “znajdujemy” nie jest przypadkiem wartością końcową lub początkową. Jeśli nie to relaxation[i] przyjmuje wartości gammy * odejmowanie 1 – Ax[i].

Metoda Jacobiego

```
62      while(!close) {
63          double jacobi[N] = {0.0};
64          jacobi[0] = 1.0;
65          jacobi[N - 1] = 1.0;
66
67          for (int i = 1; i < N - 1; i++) {
68              jacobi[i] = (vectorN[i - 1] + vectorN[i + 1]) / Diagonal;
69          }
70          diff = 0.0;
71          for(int i = 0; i < N; i++){
72              if(diff < fabs(jacobi[i] - vectorN[i])){
73                  diff = fabs(jacobi[i] - vectorN[i]);
74              }
75          }
76
```

Program ustala na sztywno wartości srodkowe oraz pierwszą i ostatnią jaką ma nadać tablicy jacobi[N]. Następnie iterujemy tablice na każdej pozycji wstawiając wynik sumy poprzednika i następnika dzielonych przez wartość diagonalną.

Meroda Gausa Seidle'a

```
93      while(!close){
94          double gausSeidle[N] = {0.0};
95          gausSeidle[0] = 1;
96          gausSeidle[N - 1] = 1;
97
98          for(int i=1; i < N - 1; i++){
99              gausSeidle[i] = vectorN[i+1];
100         }
101
102         for(int i = 1; i < N; i++){
103             gausSeidle[i] = (gausSeidle[i] + gausSeidle[i-1]) / Diagonal;
104         }
```

W metodzie gaussa seidla każdy element tablicy dostaje pierwsze następnik vector[N]. W późniejszy, kroku program nadaje tablicy gausSeidle[i] wartość sumy tejże pozycji razem z jej poprzednikiem, dzielonych przez wartość diagonalną.

Metoda SOR

```
130     while(!close) {
131         for(int i = 0; i < N; i++)
132             sor[i] = vectorN[i] * (1.0 - omega);
133
134         sor[0] += (omega / Diagonal);
135         sor[N - 1] += (omega / Diagonal);
136         for(int i = 1; i < N - 1; i++) {
137             sor[i] += (omega / Diagonal)* (sor[i - 1] + vectorN[i + 1]);
138         }
```

W ostatniej już metodzie każdemu elementowi tablicy sor[i] nadajemy wartość vector[N] mnożoną razy różnicę 1 i omegi. Następnie elementy ostatni i pierwszy traktujemy w taki sposób że będą otrzymywać sumę po wszystkich iteracjach wartości $\frac{\omega}{a_{ii}}$. Ostatnim już krokiem jest przeprowadzenie podobnej sumy dla elementów od i = 1 aż do przedostatniej pozycji w tablicy. Różnicą jest to że w tym przypadku powyższą wartość mnożymy jeszcze razy sor[i – 1] + vectorN[i+1].

Używamy jeszcze poniższej pętli do zerowania tablicy podczas wypisywania

```
void SetAsZero (double vector[N]){
   for(int i = 0; i < N; i++){
      vector[i] = 0.0;
   }
}
```

```
double vecZero[N] = {0.0};
int il = 0;

   clock_t start = clock();
   Jacobi(vecZero,&il);
   clock_t end = clock();
   printf("dla jakobiego przeszlo %i razy, czas wykonywania programu wyniosl: %.3f  \n",
il,(double)(end - start)/CLOCKS_PER_SEC);

   SetAsZero(vecZero);
```

Powyżej załączyłem tylko jedno wypisywanie ponieważ wszystkie inne sa analogiczne. Program włącza zegar po czym wykonuje daną funkcje. Następuje zakończenie działania zegara i program wypisuje

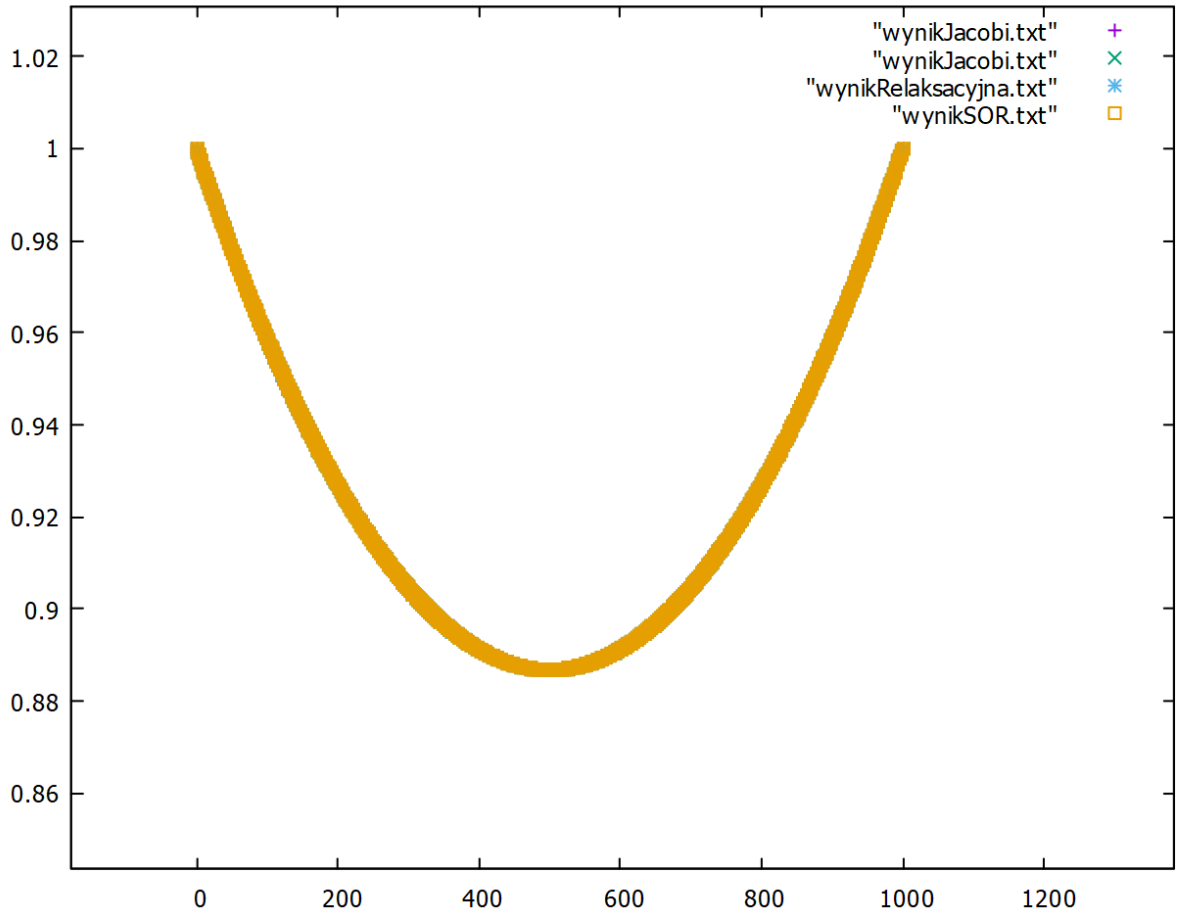
ilość iteracji oraz czas potrzebny do wykonania programu. Na sam koniec nastepuje wyczyszczenie tablicy z której korzystamy i schemat jest powtarzany dla następnych funkcji.

Ilość przejść otrzymanych w programie prezentuje się następująco:

```
PS C:\Users\Aslave2137\Desktop\Nauka\Numerki> cd "c:\Users\Aslave2137\Desktop\Nauka\Numerki\Zadanie 6"
PS C:\Users\Aslave2137\Desktop\Nauka\Numerki\Zadanie 6> & .\"main.exe"
dla Relaksacyjnej przeszlo 2033235 razy, czas wykonywania programu wyniosl: 16.551
dla jakobiego przeszlo 2160402 razy, czas wykonywania programu wyniosl: 9.759
dla gaussa przeszlo 1081980 razy, czas wykonywania programu wyniosl: 10.203
dla SOR przeszlo 4069 razy, czas wykonywania programu wyniosl: 0.035
PS C:\Users\Aslave2137\Desktop\Nauka\Numerki\Zadanie 6>
```

\

Wyniki programu przedstawiłem na jednym wykresie. Dzięki temu da się zauważyć że są one niemalże identyczne (jedyne różnice są wręcz marginalne i występują dopiero na piątym miejscu po przecinku).



Poniżej załączam jeszcze przybliżenie które dobrze pokazuje jak marginalne są to różnice: