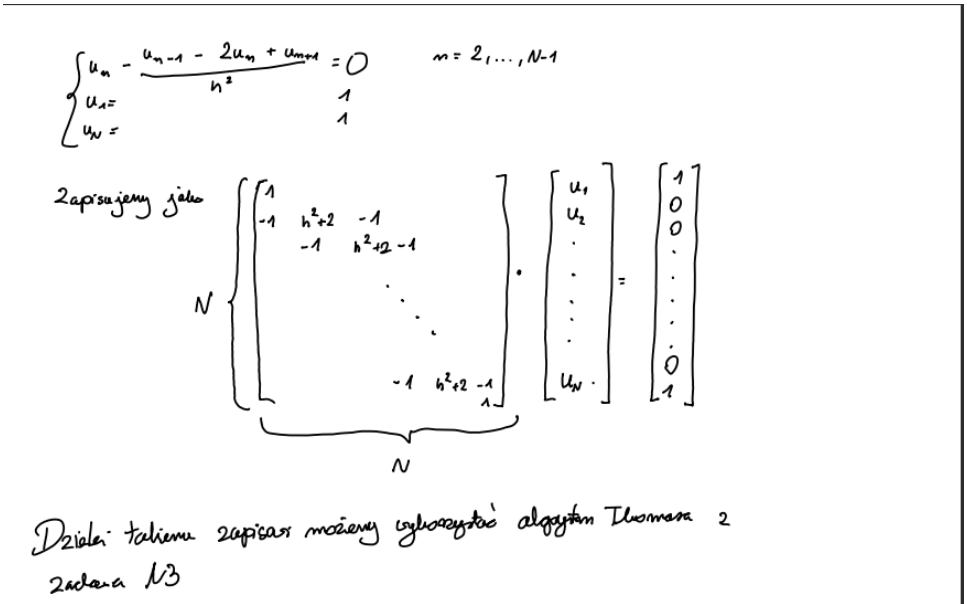
ZADANIE N5 pkt a)

W programie należy rozwiązać układ równań z danymi zadanymi w macierzy:

- N = 1001

- h = 1 / (N-1)

Dla ułatwienia obliczeń zapisujemy układ równań jako macierz.



W programie będziemy korzystać z tego samego kodu wykorzystującego algorytm Thomasa co w zadaniach N3 oraz N4

```
double d[N] = {0.0};

  double c[N - 1] = {};
  double u[N] = {0.0};
  double x[N];
  double diagonal = 2.000001;
```

Zmienne pozostają tak samo jak w poprzednich programach. Jedyna zmiana to zmienna diagonal której używam do przechowywania wartości na diagonali.

Dokładny opis algorytmu Thomasa zastosowanego w programie znajduję się w zadaniu N3. W N5 zastosowałem tylko inne zmienne i wywołałem go jednokrotnie

---

```
/*Zapełnianie macierzy wyników*/
    d[0] = 1.0;
    d[N - 1] = 1.0;
    /*--------------------------------*/
    c[0] = 0;
```

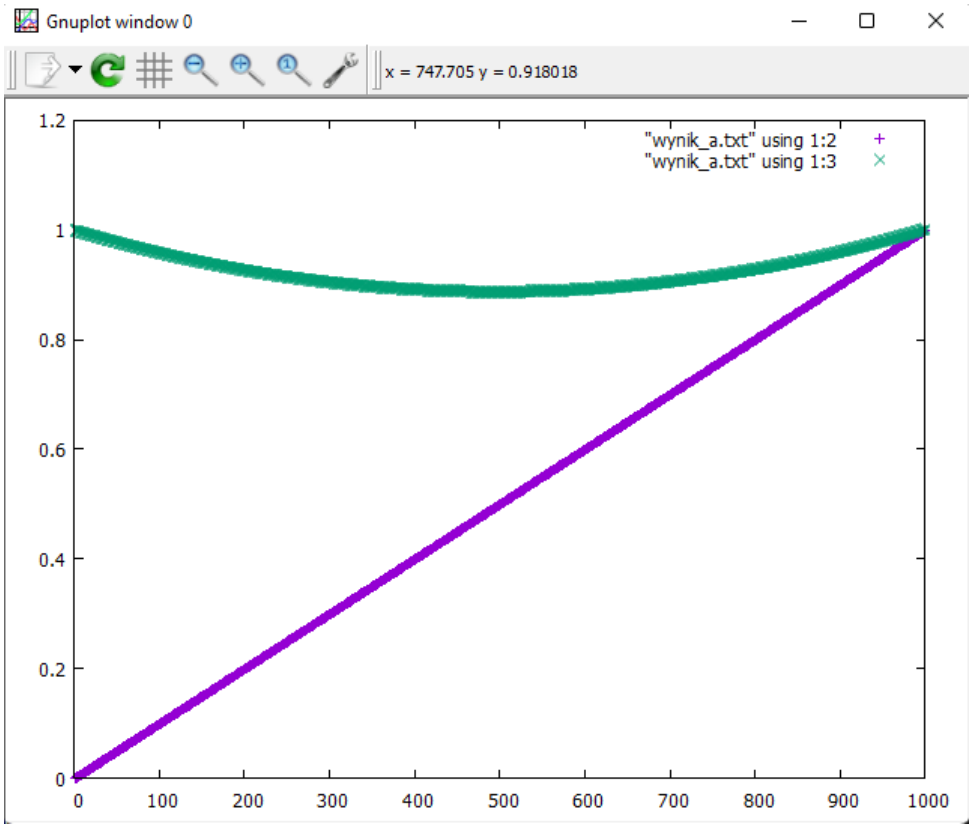
```
  for ( int i = 1; i < N ; i++){

      if(i < N - 1){
      c[i] = -1/(diagonal - (-1*c[i-1]));
      }
      if(i ==1){
      c[i] = -1/(diagonal - (0*c[i-1]));
      }

      d[i] = (d[i] - (-1 * d[i-1])/(diagonal - (-1*c[i-1])));
   }

   u[N - 1] = d[N - 1];

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

Powyżej wklejam kod implementujący algorytm Thomasa. Pierwsze linijki to wypełnianie wartości "wyjątkowych" tj. jedynek dla pierwszej i ostatniej pozycji wektora z odpowiedziami.

Jest to poprawa opisu programu. Okazało się że popełniłem kilka błędów w algorytmie Thomasa, zwykle były to za długie lub za krótkie pętle. W obecnej wersji program zwraca odpowiednie wyniki reprezentowane na wykresie przez zieloną linie.

Złożoność obliczeniowa kodu to O(n) a dokładniej O(2n) jeśli nie wliczamy wypisywania wyników. Rozpatrzenie tego programu jako szukanie rozwiązania dla macierzy trójdiagonalnej jest najszybszym sposobem na jaki wpadłem.

Poniżej wklejam wykres wyników z podpunktu a:



ZADANIE N5 pkt b)

W programie rozwiązujemy układ równań w sposób iteracyjny.

- N = 1001
- h = 1 / (N-1)
- 3u1 + 4u2 - u3 = 0

Program jest dość prosty. Do obliczeń używamy:

double results[N] - tablica o rozmiarze N = 1001 do której zapisujemy wyniki

double h = 0.001 - zmienna h o wartości zadanej w zadaniu

double h2 = h*h + 2 - zmienna h2 której będziemy używac w dalszej części programu

---

Obliczamy "ręcznie" pierwsze trzy pozycje tablicy:

```
results[0] = 1;
results[1] = 2 / (2 - h*h);
```

results[2] = 4 * results[1] - 3; (tutaj nastąpiła poprawa błędu na który Pan zwrócił uwagę)

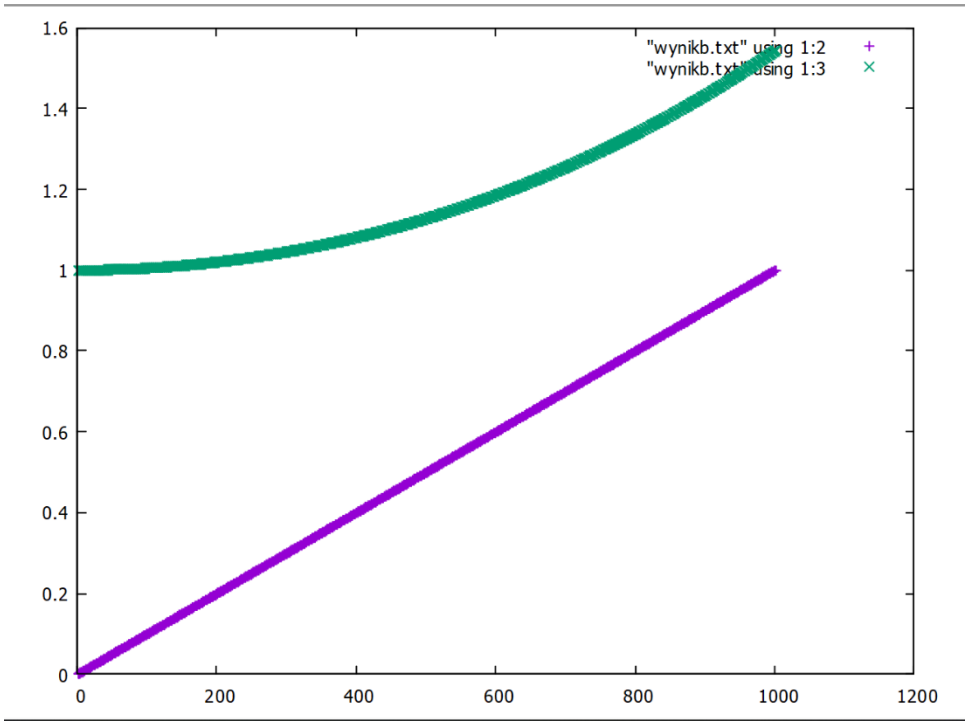
Następnie zaczynając od 3 pozycji tablicy wykonujemy pętle for która tworzy pozycję results[i] za pomocą mnożenia poprzedniego elementu * h2 przez - element[i - 2]:

```
for(int i = 3; i < N ; i++){
    results[i] = results[i - 1] * h2 - results[i - 2];
}
```

Za pomocą ostatniej pętli wypisujemy wyniki:

```
for(int i = 0 ; i < N ; i++){
    fprintf (file," %i %.16lf %.16lf \n",i+1 ,i*h, results[i] );
}
```

Poniżej wklejam wykres wyników z podpunktu b:



Złożoność obliczeniowa kodu to O(n) jeśli nie wliczamy wypisywania wyników. Poniżej wklejam obliczenia których poprzednio barkowało:

$$\begin{cases} u_m - \dfrac{u_{m-1} - 2u_m + u_{m+1}}{h^2} = 0 \\ u_1 = 1 \\ -3u_1 + 4u_2 - u_3 = 0 \end{cases}$$

$$\begin{cases} u_m h^2 - u_{m-1} + 2u_m - u_{m-1} = 0 \\ u_1 = 1 \\ -3 + 4u_2 - u_3 = 0 \end{cases}$$

$$\begin{cases} -u_{m-1} + u_m(h^2+2) - u_{m+1} = 0 \\ u_1 = 1 \\ 4u_2 - u_3 = 0 \end{cases}$$

Jedno z podstaw 2

$$\begin{cases} 4u_2 - u_3 = 3 \\ -1 + u_2(h^2+2) - u_3 = 0 \end{cases}$$

$$\begin{cases} 4u_2 - u_3 = 3 \\ u_2h^2 + 2u_2 - u_3 = 1 \end{cases} \; -$$

$$-u_2h^2 + 2u_2 = 3 - 1$$

$$\Leftarrow \begin{cases} u_2 = \dfrac{2}{2-h^2} \\ u_3 = 4u_2 - 3 \end{cases}$$ i podstawy do wcześniejszego równania

$$u_3 = 4\left(\frac{2}{2-h^2}\right) - 3$$

$$u_3 = \frac{8}{2-h^2} - 3$$

$$\begin{cases} u_{m+1} = -u_{m-1} + u_m(h^2+2) \\ u_1 = 1 \\ u_2 = \frac{2}{2-h^2} \\ u_3 = \frac{8}{2-h^2} - 3 \end{cases}$$