

// Autor: Daniel Szarek

//=====

Zawartość folderu z zadaniem:

- main.cpp – Kod z rozwiązaniem zadania napisanym w języku C++
- Makefile – Do uruchomienia programu main.cpp (Makefile oferuje możliwość uruchomienia programu za pomocą komendy 'make run', usunięcia plików po uruchomieniu programu komendą 'make clean' oraz możliwość utworzenia paczki .tar.gz z zawartością foldera z zadaniem za pomocą komendy 'make tar')
- Opracowanie.pdf – Ten dokument z opracowaniem zadania N3
- wykres.gnu – Skrypt do programu gnuplot do utworzenia wykresu podanego w opracowaniu
- wykres.png – Gotowy wykres, zrobiony przez skrypt wykres.gnu
- wyniki.txt – Plik tekstowy z wynikami programu main.cpp

Do zrealizowania zadania wykorzystałem GSL - GNU Scientific Library, w Makefile zawarłem komendy do uruchomienia zadania na swojej maszynie w systemie Ubuntu, w celu uruchomienia zadania na swoim sprzęcie należy w prawidłowy sposób zainstalować bibliotekę GSL oraz zmienić ścieżkę INCLUDEPATH1 dla swojej maszyny.

Treść Zadania:

N3 Zadanie numeryczne

Rozwiązać układ $(N + 1) \times (N + 1)$ równań postaci

$$\begin{cases} y_0 &= 1 \\ (D_2 y)_n + y_n &= 0, \quad n = 1 \dots (N - 1) \\ y_N &= 0, \end{cases} \quad (1)$$

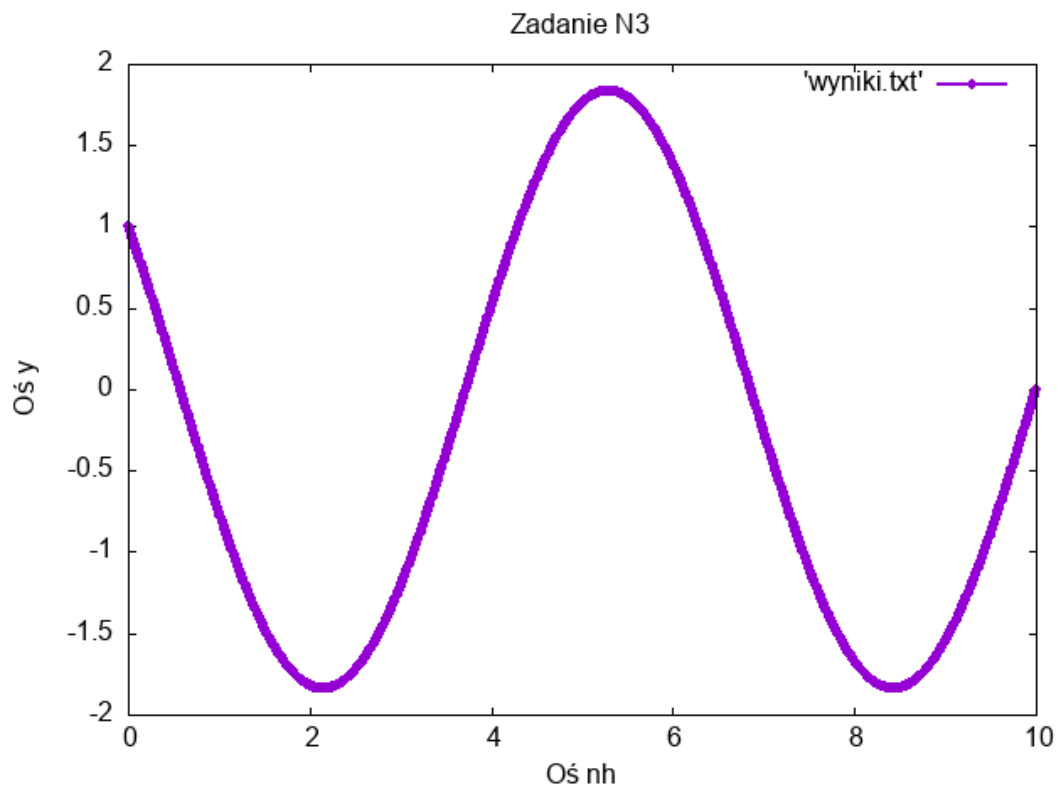
gdzie $N = 1000$, $h = 0.01$, a

$$(D_2 y)_n = \frac{y_{n-1} - 2y_n + y_{n+1}}{h^2}. \quad (2)$$

Rozwiązanie przedstawić graficznie (nh, y_n) .

Uwaga: najważniejsze w rozwiązaniu zadanie będzie dobranie odpowiedniego algorytmu i optymalizacja (dla bardzo dużego N).

Rozwiązanie w postaci graficznej:



Omówienie:

W celu rozwiązania w pierwszej kolejności musimy się skupić na wizualizacji macierzy A , związanej z reprezentacją kolejnych zmiennych y_i podanej w postaci równań.

$$\begin{cases} y_0 &= 1 \\ (D_2 y)_n + y_n &= 0, \quad n = 1 \dots (N-1) \\ y_N &= 0, \end{cases}$$

Opracowanie macierzy zaczynam od $y_0 = 1$, następnie podstawiam kolejne wartości zgodnie z powyższym wzorem, nie zapominając o wartości $(D_2 y)_n$.

$$(D_2 y)_n = \frac{y_{n-1} - 2y_n + y_{n+1}}{h^2}.$$

$$y_0 = 1, y_{1000} = 0, \text{ bo } N=1000$$

Obliczamy pierwszą wiersz macierzy:

$$\frac{y_0 - 2y_1 + y_2}{h^2} + y_1 = \frac{y_0}{h^2} - \frac{2y_1}{h^2} + y_1 + \frac{y_2}{h^2}$$

Stąd otrzymujemy:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Obliczamy drugą wiersz macierzy:

$$\frac{y_1 - 2y_2 + y_3}{h^2} + y_2 = \frac{y_1}{h^2} - \frac{2y_2}{h^2} + y_2 + \frac{y_3}{h^2}$$

Stąd otrzymujemy:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 & \dots \\ 0 & \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Widzimy zależność dla kolejnych elementów i będziemy otrzymywać takie same wyniki, z tą różnicą, że indeksy będą przesuwane w prawo. Otrzymamy koniec końców macierz w poniższej postaci.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & \vdots & \ddots & \vdots & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \frac{1}{h^2} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Widzimy, że macierz będzie przyjmowała postać macierzy trójdzielnej z elementami a_{12} , a_{N+1N} różnymi 0.

Zapisujemy wektory rozwiązania y oraz wyników który nazwałem b .

$$y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

W celu rozwiązania układu równań $N+1$ na $N+1$ dla macierzy diagonalnej stosuje metodę Thomasa, znaną również jako algorytm eliminacji Gaussa dla macierzy trójdzielnych.

Jeżeli układ równań jest trójdzielny, spełnia warunki brzegowe, jest dobrze uwarunkowany, jednorodny, a liczba równań jest odpowiednio duża to metoda Thomasa oferuje efektywną i stabilną numerycznie procedurę rozwiązania układu równań. Nasz układ równań spełnia te kryteria.

W swoim programie zaimplementowałem metodę Thomasa w funkcji `algorytmThomasa`.

Swoją implementację oparłem na podstawie źródła z Wikipedii:

Method [\[edit \]](#)

The forward sweep consists of the computation of new coefficients as follows, denoting the new coefficients with primes:

$$c'_i = \begin{cases} \frac{c_i}{b_i}, & i = 1, \\ \frac{c_i}{b_i - a_i c'_{i-1}}, & i = 2, 3, \dots, n-1 \end{cases}$$

and

$$d'_i = \begin{cases} \frac{d_i}{b_i}, & i = 1, \\ \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, & i = 2, 3, \dots, n. \end{cases}$$

The solution is then obtained by back substitution:

$$\begin{aligned} x_n &= d'_n, \\ x_i &= d'_i - c'_i x_{i+1}, \quad i = n-1, n-2, \dots, 1. \end{aligned}$$

Link do źródła: https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

Algorytm Thomasa jest bardzo efektywny w przypadku trójdzielnych układów równań, ponieważ wymaga tylko $O(n)$ operacji, gdzie n to liczba niewiadomych.

Dzięki bibliotece GSL w celu rozwiązania zadania można było skorzystać z gotowej funkcji `gsl_linalg_solve_tridiag` która jako argumenty przyjmuje kolejno macierze: diagonalną, subdiagonalną, supdiagonalną, rozwiązań oraz y . Poniżej zdjęcia z dokumentacji oraz bezpośrednio z biblioteki GSL z objaśnieniem.

```
int gsl_linalg_solve_tridiag(const gsl_vector *diag, const gsl_vector *e, const gsl_vector *f, const gsl_vector *b, gsl_vector *x)
```

This function solves the general N -by- N system $Ax = b$ where A is tridiagonal ($N \geq 2$). The super-diagonal and sub-diagonal vectors `e` and `f` must be one element shorter than the diagonal vector `diag`. The form of A for the 4-by-4 case is shown below,

$$A = \begin{pmatrix} d_0 & e_0 & 0 & 0 \\ f_0 & d_1 & e_1 & 0 \\ 0 & f_1 & d_2 & e_2 \\ 0 & 0 & f_2 & d_3 \end{pmatrix}$$

```

/* Linear solve for a nonsymmetric tridiagonal system.

 * The input vectors represent the NxN matrix as follows:
 *
 *      diag[0]  abovediag[0]      0      ...
 * belowdiag[0]      diag[1]  abovediag[1]  ...
 *      0  belowdiag[1]      diag[2]      ...
 *      0      0  belowdiag[2]      ...
 *      ...      ...      ...      ...
 */
int gsl_linalg_solve_tridiag (const gsl_vector * diag,
                              const gsl_vector * abovediag,
                              const gsl_vector * belowdiag,
                              const gsl_vector * b,
                              gsl_vector * x);

```