

✓ Opis działania programu (Zadanie 1)

Program służy do **numerycznego całkowania funkcji wielomianowej trzeciego stopnia**:

$$f(x)=Ax^3+Bx^2+Cx+D \quad f(x) = Ax^3 + Bx^2 + Cx + D \quad f(x)=Ax^3+Bx^2+Cx+D$$

Program pozwala użytkownikowi:

- podać współczynniki A, B, C, D,
- ustawić granice całkowania x_1 , x_2 ,
- wybrać liczbę przedziałów n ,
- wybrać jedną z 3 metod całkowania: prostokątów, trapezów lub Simpsona,
- oraz określić liczbę wątków do przeliczeń równoległych (OpenMP).

🔧 Najważniejsze funkcje

f(...)

Oblicza wartość funkcji $f(x)$ dla zadanych współczynników.

integrate_sequential(...)

Oblicza całkę w sposób **sekwencyjny**. W zależności od metody:

- **Metoda prostokątów** — dodaje wartości funkcji na początku każdego przedziału.
- **Metoda trapezów** — wykorzystuje średnie wartości funkcji na końcach przedziału.
- **Metoda Simpsona** — dokładniejsza, używa parabol do przybliżenia obszaru pod wykresem (działa tylko dla parzystych n).

integrate_parallel(...)

Wersja równoległa, z zastosowaniem dyrektyw OpenMP (`#pragma omp parallel for`) i mechanizmu `reduction(+:sum)` do sumowania wyników bez kolizji między wątkami.

⚙️ Jak wygląda optymalizacja? Co można by zmienić na obronie

1. Unifikacja funkcji `integrate_*`

- Obecnie są dwie funkcje (`sequential` i `parallel`), których kod jest bardzo podobny.
- Można je połączyć w jedną, z opcjonalnym argumentem `bool parallel`, który decyduje o tym, czy włączyć `#pragma omp`.

2. Użycie OpenMP w bardziej zaawansowany sposób

- Zamiast dzielić tylko pętle, można użyć zagnieżdżonych parallel sections, np. oddzielnie obliczyć wartości skrajne i środkowe (np. w trapezach).

3. Użycie lepszych metod integracji

- Można zaproponować adaptacyjne metody numeryczne (jak adaptacyjna Simpsona), które zwiększają dokładność przy mniejszej liczbie przedziałów.

4. Zoptymalizowanie $f(x)$

- Jeśli funkcja $f(x)$ jest często wywoływana, można zastosować minimalną optymalizację np. przez Horner's scheme:

```
return ((A * x + B) * x + C) * x + D;
```

5. Pomiar czasu

- Użycie `omp_get_wtime()` do pomiaru czasu działania – możesz zaproponować zapis wyników do pliku w celu późniejszej analizy.

6. Bezpieczne sprawdzanie danych wejściowych

- Aktualnie `scanf` może pozostawić dane w buforze, lepiej byłoby użyć `fgets` + `sscanf` lub systemowego sposobu walidacji.

Na co uważać na obronie

- Musisz umieć wytłumaczyć, czym się różnią metody:
 - prostokątów (lewa wartość),
 - trapezów (średnia dwóch końców),
 - Simpsona (paraboliczna aproksymacja).
- Pokaż różnicę w czasie wykonania dla sekwencyjnego i równoległego.
- Pokaż fragment kodu `#pragma omp parallel for reduction(+:sum)` i wytłumacz, że chodzi o **sumowanie równoległe bez błędów wyścigu**.

Opis działania programu (Zadanie 2)

Program służy do **rozwiązywania układu równań liniowych** metodą **eliminacji Gaussa** z częściowym wyborem elementu głównego (**pivoting**).

Przetwarza dane:

- z pliku **C.csv** zawierającego macierz rozszerzoną (czyli współczynniki + wyrazy wolne),
- rozwiązuje układ zarówno **sekwencyjnie**, jak i **równolegle (OpenMP)**,
- zapisuje wyniki do pliku **X_...csv** oraz dane pomiarowe do **datalogger.txt**.

Najważniejsze funkcje

read_matrix_from_csv(...)

- Wczytuje dane z pliku .csv. Pierwsza linia zawiera wymiar n , a kolejne n linii zawierają $n+1$ wartości (macierz rozszerzona układu).

gauss_sequential(...)

- Sekwencyjna implementacja eliminacji Gaussa:
 - Dla każdego wiersza r znajduje wiersz z największym elementem w kolumnie r i zamienia je miejscami (pivoting).
 - Wykonuje eliminację do postaci trójkątnej.
 - Następnie wykonuje **podstawienie wsteczne** do uzyskania wektora wynikowego X .

gauss_parallel(...)

- Wersja równoległa z OpenMP:
 - Równolegle wykonuje pętlę **eliminacji** wierszy (ale nie pivoting! – pivoting musi być sekwencyjny, bo zależy od kolejności i maksów).
 - Podstawienie wsteczne również jest sekwencyjne.

clone_matrix(...)

- Tworzy kopię macierzy, by wykonać obliczenia niezależnie w dwóch wersjach (dla sekwencyjnego i równoległego podejścia).

write_result_to_csv(...)

- Zapisuje wynikowy wektor X do pliku CSV, a nazwa zawiera czasy T_s i T_p .

log_results(...)

- Dodaje wpis do pliku datalogger.txt z czasami wykonania i liczbą wątków.



Część testowa (main)

Program:

- Wczytuje dane z pliku,
- Rozwiązuje układ równań:
 - najpierw **sekwencyjnie** (gauss_sequential),
 - potem **równoległe** (gauss_parallel),
- Mierzy czasy T_s i T_p ,
- Wylicza **przyspieszenie (speedup)** jako T_s / T_p ,
- Pętla pozwala testować wiele razy (z różną liczbą wątków).



Co może pojawić się na obronie

1. Wytłumaczenie algorytmu Gaussa

- Eliminacja elementów pod przekątną.
- Pivoting — po co się robi? (by uniknąć dzielenia przez zera i poprawić dokładność).

2. Dlaczego nie można równoległe pivotować?

- Bo wybór wiersza zależy od aktualnego stanu i kolejności. Trzeba go zrobić w jednym wątku.

3. Jak można zoptymalizować program?

◆ Propozycje zmian:

- **Równoległe podstawianie wsteczne** – bardziej złożone, ale możliwe z zależnościami (trudne do synchronizacji).
- **Użycie struktur danych zbliżonych do cache** – unikać podwójnego wskaźnika (`double**`), lepszy byłby `double*`.
- **Użycie SIMD / BLAS** – w większych przypadkach można zastosować zoptymalizowane biblioteki jak OpenBLAS.
- **Zoptymalizować `swap_rows()`** – przez memcpy zamiast pętli for.

4. Typowy test na obronie

- Pokaż, jak zmiana liczby wątków wpływa na wynik.
- Możesz dostać polecenie, żeby zmodyfikować `gauss_parallel` np. tak, by wyłączał pivoting.

- Możesz też zostać poproszony o dodanie pomiaru dokładności (czy $X_{seq} \approx X_{par}$).

✓ Opis działania programu (Zadanie 3)

Program wykonuje **równoległe mnożenie macierzy** z wykorzystaniem **MPI (Message Passing Interface)**.

Przetwarza dane:

- **Macierz A (rozmiar $m \times n$) i B (rozmiar $n \times p$)** wczytywane z plików A.csv i B.csv.
- Macierz wynikowa **C (rozmiar $m \times p$)** jest zapisywana do pliku CSV C_T1_Tp.csv, gdzie T1 i Tp to czasy wykonania sekwencyjnego i równoległego.

📁 Jak działa program

1. [Rank 0]:

- Wczytuje dane z plików.
- Oblicza wynik sekwencyjnie (dla porównania) → czas T1.
- Przygotowuje miejsce na wynik C.

2. [Wszystkie rangi]:

- Otrzymują przez MPI_Bcast informacje o wymiarach.
- Dzielona jest macierz A wierszami między procesy (MPI_Scatter).
- Macierz B przesyłana jest wszystkim (MPI_Bcast).

3. [Każdy proces]:

- Mnoży swoją część macierzy A z pełną macierzą B.
- Wynik umieszczany lokalnie w local_C.

4. [Rank 0]:

- Zbiera części C od każdego procesu (MPI_Gather).
- Zapisuje wynik i czasy do pliku, oblicza przyspieszenie T1/Tp.

🔧 Najważniejsze funkcje

`read_matrix_csv(...)`

- Wczytuje macierz z pliku .csv. Oczekuje pierwszych dwóch liczb jako rows i cols.

`multiply_sequential(...)`

- Sekwencyjne mnożenie dwóch macierzy: trój-pętla i-j-k, klasyczna forma.

main()

- Obsługuje całą logikę MPI: rozsyłanie, zbieranie, pomiary czasu, synchronizacja, alokacje i zwalnianie pamięci.

MPI_Scatter / MPI_Gather / MPI_Bcast

- Podstawowe mechanizmy komunikacji MPI: rozsyłanie i zbieranie danych.

Pytania, które mogą się pojawić na obronie

1. Jak działa mnożenie macierzy?

- Standardowe: $C[i][j] = \text{sum}(A[i][k] * B[k][j])$

2. Dlaczego macierz A musi być podzielna przez liczbę procesów?

- Bo każdy proces dostaje dokładnie m / size wierszy — nierówny podział nie jest tu obsługiwany.

3. Dlaczego B jest rozsyłana przez MPI_Bcast?

- Każdy proces potrzebuje **pełnej macierzy B**, bo każdy wykonuje swoje wiersze $A \times$ wszystkie kolumny B.

4. Które części można zoptymalizować?

◆ Propozycje optymalizacji:

- **Rozszerzenie na przypadki, gdy $m \% \text{size} \neq 0$** (dodanie dynamicznego podziału).
- **Użycie biblioteki BLAS** (np. OpenBLAS, Intel MKL) zamiast własnej implementacji.
- **Użycie nieblokujących operacji MPI (MPI_Isend, MPI_Irecv)** dla poprawy nakładania komunikacji z obliczeniami.
- **Użycie MPI_Scatterv i MPI_Gatherv** – umożliwia bardziej elastyczne dzielenie danych.

5. Dlaczego MPI_Barrier() jest potrzebne?

- Synchronizacja: zapewnia, że wszystkie procesy zaczynają/kończą dokładnie w tym samym momencie → precyzyjny pomiar Tp.

Wersja skrócona (co powiedzieć ustnie):

Program wczytuje dwie macierze, mnoży je najpierw sekwencyjnie (rank 0), a potem równoległe z użyciem MPI. Macierz A jest dzielona wierszami między procesy, B jest

nadawana do wszystkich. Każdy proces oblicza swoją część wyniku i odsyła ją do rank 0, który zapisuje wynik i czas. Pomiar przyspieszenia pokazuje korzyści z równoległości.

✓ Opis programu – Zadanie 4 (OpenMP)

Program sortuje wiersze macierzy za pomocą sortowania bąbelkowego (**Bubble Sort**) i porównuje **czas sortowania sekwencyjnego (TS)** i **równoległego (TP)** wykorzystując **OpenMP**.

🔧 Działanie programu

Program działa w dwóch trybach:

A Tryb A (Generowanie danych):

Uruchomienie:

program.exe m n min max output.csv P

- Tworzy macierz $m \times n$ z losowymi wartościami z przedziału $[min, max]$.
- Wykonuje sortowanie wierszy:
 - Sekwencyjnie (`omp_set_num_threads(1)`)
 - Równoległe z P wątkami (`omp_set_num_threads(P)`)
- Zapisuje wynikową (posortowaną) macierz do `output.csv`.

B Tryb B (Wczytanie z pliku):

Uruchomienie:

program.exe input.csv output.csv P

- Wczytuje macierz z pliku `input.csv`.
- Wykonuje analogicznie sortowanie sekwencyjne i równoległe.
- Zapisuje wynik.

✚ Najważniejsze funkcje

◆ `sort_row(...)`

- Prosty algorytm **Bubble Sort** — sortuje jeden wiersz rosnąco.

◆ `read_matrix_csv(...)`

- Wczytuje macierz z pliku `.csv`, analizując liczbę wierszy i kolumn.

◆ `generate_matrix(...)`

- Losuje wartości w podanym zakresie.

♦ **omp parallel for**

- Kluczowy fragment przyspieszający sortowanie – każdy wiersz sortowany niezależnie.

Pomiar czasów

```
double start_seq = omp_get_wtime();
```

```
// sortowanie sekwencyjne
```

```
double end_seq = omp_get_wtime();
```

```
double start_par = omp_get_wtime();
```

```
// sortowanie równoległe
```

```
double end_par = omp_get_wtime();
```

Czasy są wypisywane na ekran: TS i TP — do porównań efektywności.

Propozycje optymalizacji (na obronę)

1. Użycie lepszego algorytmu sortowania

- Bubble Sort ma złożoność $O(n^2)$. Można użyć:
 - `std::sort` w C++
 - QuickSort / MergeSort
- Przykład zamiany:

```
qsort(row, n, sizeof(int), compare_function);
```

2. Użycie przydziału pamięci jednowymiarowej

- Zamiast tablicy wskaźników: `int**`, można użyć `int*` i indeksować ręcznie.

3. Zrównoleglenie także generowania danych:

```
#pragma omp parallel for
```

```
for (int i = 0; i < m; i++)
```

```
    for (int j = 0; j < n; j++)
```

```
        matrix[i][j] = rand() % (max - min + 1) + min;
```

(choć `rand()` nie jest bezpieczny w wielu wątkach – można zastąpić `rand_r` lub C++ `std::mt19937`).

4. Dodanie pomiaru zużycia CPU/RAM (dla bardziej zaawansowanej prezentacji).

5. Unifikacja sortowania

- Zamiast kopiować macierz i sortować dwukrotnie, można napisać osobne funkcje `sort_matrix_seq(...)` i `sort_matrix_parallel(...)`.

Co możesz powiedzieć na obronie?

Program sortuje każdy wiersz macierzy osobno. W wersji sekwencyjnej robi to na jednym wątku, a w równoległej używa OpenMP i przydziela każdy wiersz do osobnego wątku. Wykorzystano bubble sort, który jest prosty, ale mało wydajny. Do optymalizacji można zastąpić go szybszym algorytmem sortowania lub przekształcić tablicę do jednowymiarowej.

Opis programu – Zadanie 4 (MPI)

Cel:

Program sortuje wszystkie elementy dużej macierzy w sposób rozproszony (zrównoleglony) za pomocą **MPI**. Każdy proces dostaje fragment danych, sortuje lokalnie, a potem wyniki są **scalane hierarchicznie w stylu drzewa (merge tree)**.

Główne funkcjonalności

1. Wczytywanie lub generowanie danych

- **Rank 0:**
 - Wczytuje macierz z pliku **lub** generuje losowo, jeśli podano: `m n min max output.csv`.
 - Przekształca macierz do wektora (1D).
 - Oblicza rozmiar danych `total = rows * cols`.

2. Dystrybucja danych

- `MPI_Scatter`: dzieli dane po równo na wszystkie procesy (`local_data`).
- Każdy proces sortuje **swój fragment lokalnie**: `std::sort(...)`.

3. Scalanie drzewiaste (merge tree)

- Wykorzystuje wzorec **"binary reduction tree"**:
 - Procesy parzyste zbierają dane od sąsiednich nieparzystych i je **łączą** (merge).
 - W każdej rundzie zasięg łączenia się podwaja (`stride *= 2`).
 - W końcu **rank 0** ma całość posortowaną.

4. Zapis wyniku

- Tylko **rank 0** zapisuje posortowaną macierz do output.csv.
- Mierzy i wypisuje czas wykonania całej operacji.

Najważniejsze funkcje

Funkcja	Opis
load_matrix(...)	Wczytuje macierz z pliku CSV.
create_random_matrix(...)	Generuje losowe wartości float.
flatten(...)	Konwertuje macierz 2D → 1D (dla MPI_Scatter).
unflatten(...)	Konwertuje dane 1D z powrotem do 2D (po sortowaniu).
merge_vectors(...)	Łączy dwie posortowane tablice (klasyczny merge).
sort(...)	Standardowa sortowanie STL lokalnych danych.

Pomiar czasu





- Tylko **rank 0** mierzy i wypisuje:


```
double timer_start = MPI_Wtime();
```

```
...
```

```
double timer_end = MPI_Wtime();
```

Pomysły na optymalizację (do obrony)

Optymalizacja	Opis
 <i>Lepszy algorytm sortowania lokalnego</i>	Zamiast std::sort → np. parallel sort z Thrust lub TBB (jeśli dostępne).
 <i>Zbalansowanie danych</i>	Obsługa przypadków, gdy total % num_procs != 0. Teraz program załamie się, jeśli niepodzielne.
 <i>Zrównoleglenie merge'a</i>	Teraz merge działa sekwencyjnie — da się zrównoleglić merge wielu podtablic.
 <i>Użycie MPI_Gatherv</i>	Jeśli rozmiary lokalnych danych są różne, Gatherv zamiast Gather/merge.

Optymalizacja	Opis
 <i>Zapis do pliku binarnego</i>	Dla dużych danych szybciej niż CSV.

Co możesz powiedzieć na obronie?

Program implementuje sortowanie rozproszone z wykorzystaniem MPI. Dane są dzielone między procesy, sortowane lokalnie, a następnie łączone w sposób drzewiasty. Całość kończy się zapisem do pliku CSV. Optymalizacją może być wsparcie nierównych danych, bardziej wydajne łączenie danych czy alternatywny format zapisu.

Opis programu – Zadanie 5: Rasteryzacja linii (OpenMP)

Cel programu:

Program wykonuje **rasteryzację (rysowanie)** wielu odcinków (linii) na obrazie BMP o dużej rozdzielczości (8000x8000) z wykorzystaniem **równoległości z OpenMP**. Odcinki są odczytywane z pliku tekstowego.

Najważniejsze funkcje i komponenty

Funkcja / część	Opis
load_segments(...)	Wczytuje listę odcinków z pliku input.txt (każda linia: x1 y1 x2 y2) i zapisuje jako std::vector<Point>.
rasterize_line(...)	Implementacja algorytmu DDA (Digital Differential Analyzer) do rysowania linii piksel po pikselu.
save_bmp(...)	Zapisuje wygenerowany obraz do pliku BMP (w formacie 24-bit RGB).
main(...)	Parsuje argumenty wejściowe, ustawia liczbę wątków, mierzy czas wykonania i steruje całością.

Rasteryzacja – jak działa rasterize_line(...)

Program używa algorytmu **DDA**, który:

1. Wyznacza liczbę kroków jako $\max(dx, dy)$,
2. W każdej iteracji przesuwa się o stałą wartość x_inc, y_inc ,

3. Zaokrągla współrzędne i zapisuje biały piksel RGB = (255,255,255).

✅ Bardzo prosty, skuteczny sposób na rysowanie linii, ale:

- ❌ nie uwzględnia grubości linii,
- ❌ może powodować dziury przy bardzo stromych lub płaskich liniach.

🔧 OpenMP – równoległość

Równoległość osiągnięta dzięki:

```
#pragma omp parallel for
```

```
for (int i = 0; i < segments.size(); i++) {
```

```
    rasterize_line(image, WIDTH, HEIGHT, segments[i]);
```

```
}
```

- Każdy wątek przetwarza inny odcinek.
- Zapis do bufora image może powodować **warunki wyścigu**, jeśli dwa odcinki rysują na tym samym pikselu. Tu nie ma synchronizacji!

🕒 Pomiar czasu

Użycie `omp_get_wtime()` do zmierzenia:

- Czasu rasteryzacji wielu odcinków.

Przykład:

```
double t_start = omp_get_wtime();
```

```
// ... równoległa rasteryzacja ...
```

```
double t_end = omp_get_wtime();
```

📌 Optymalizacje, które można zaproponować na obronie:

Propozycja	Wyjaśnienie
💎 Bezpieczeństwo wątków (thread safety)	Obecnie nie ma blokad przy zapisie do <code>image[]</code> . Można użyć np. <code>omp critical</code> lub przydzielić osobne bufora dla każdego wątku i później je zmergować.
🎯 Ulepszony algorytm rasteryzacji	Zastosowanie Bresenhama zamiast DDA: jest bardziej wydajny (całkowitoliczbowy).

Propozycja

Wyjaśnienie



Culling

Odrzucanie linii całkowicie poza ekranem, by nie marnować czasu.



Pamięć obrazu jako 2D

Użycie `image[y][x][3]` zamiast 1D bufora może zwiększyć czytelność kodu.



Zapis do PNG / JPEG

Format BMP jest nieefektywny – bez kompresji. Biblioteki np. `stb_image_write` mogłyby zapisywać mniejsze pliki.



Dostosowanie liczby wątków

Dynamiczne dopasowanie liczby wątków do liczby linii (np. więcej linii = więcej wątków).



Co możesz powiedzieć na obronie?

Zadanie 5 realizuje równoległą rasteryzację linii za pomocą OpenMP. Wykorzystujemy prosty algorytm DDA do rysowania każdej linii na bitmapie. Każdy odcinek przetwarzany jest w oddzielnym wątku. Rasteryzacja jest szybka, ale może wymagać zabezpieczeń w przypadku kolizji na buforze obrazu. Program zapisuje wynik do pliku BMP, który można otworzyć graficznie.



Informacje ogólne



`struct Point { int x1, y1, x2, y2; };`

Prosty typ do reprezentacji odcinka.



`load_segments(...)`

Odczytuje odcinki z pliku tekstowego i zwraca wektor `std::vector<Point>`.



`rasterize_line(...)`

Rasteryzuje (czyli rysuje) linię na obrazie przy pomocy algorytmu DDA.



`save_bmp(...)`

Zapisuje bufor pikseli jako plik BMP (24-bit, bez kompresji).



`main(...)`

- Parsuje argumenty (`argc`, `argv`),
- Ustawia liczbę wątków (`omp_set_num_threads`),
- Tworzy bufor obrazu, rasteryzuje linie w pętli równoległej,
- Mierzy czas rasteryzacji i zapisuje wynik.

Biblioteki

- `<stdio.h>` – wejście/wyjście
- `<stdlib.h>` – `malloc`, `exit`
- `<omp.h>` – OpenMP
- `<math.h>` – funkcje matematyczne (`fabs`, `pow`)
- `<string.h>` – `strtok`
- `<time.h>` – opcjonalnie `clock`, `time`
- `<fstream>` – pliki
- `<iostream>` – standardowe I/O
- `<sstream>` – `getline`, parsowanie linii
- `<vector>` – przechowywanie danych
- `<algorithm>` – `sort`
- `<random>` – `uniform_real_distribution`
- `<cstring>` – `memset`

ARGUMENTY URUCHOMIENIA (z `argv[]`) – używane w zadaniach

- `input.txt` / `in.csv` – plik wejściowy z danymi (np. odcinki, macierze)
- `output.bmp` / `output.csv` – plik wynikowy (obraz lub dane)
- `m n` – liczba wierszy i kolumn (rozmiar macierzy)
- `min max` – zakres wartości do losowania
- `P` – liczba wątków (OpenMP)
- `A B C D` – współczynniki funkcji (dla całkowania)
- `x1 x2` – granice całkowania
- `n` – liczba przedziałów (np. w całkowaniu)
- `method` – metoda całkowania (1: prostokąty, 2: trapezy, 3: Simpsona)

KLUCZOWE FUNKCJE (unikalne, z całego zestawu zadań)

Obsługa plików i danych:

- `read_matrix_from_csv(...)` – wczytuje macierz z pliku CSV

- `write_result_to_csv(...)` – zapisuje wyniki i czasy do pliku CSV
- `save_bmp(...)` – zapisuje obraz w formacie BMP
- `save_matrix(...)` – zapisuje macierz float do pliku CSV
- `write_matrix_csv(...)` – zapisuje macierz int do CSV
- `read_matrix_csv(...)` – wczytuje macierz int z CSV
- `load_segments(...)` – wczytuje odcinki z pliku tekstowego
- `load_matrix(...)` – wczytuje macierz float z CSV
- `create_random_matrix(...)` – losowa macierz float
- `generate_matrix(...)` – losowa macierz int
- `log_results(...)` – zapisuje log z czasami do pliku txt



Operacje matematyczne i algorytmy:

- `f(...)` – funkcja wielomianowa $A \cdot x^3 + B \cdot x^2 + C \cdot x + D$
- `integrate_sequential(...)` – całkowanie sekwencyjne (trzy metody)
- `integrate_parallel(...)` – całkowanie równoległe z OpenMP
- `gauss_sequential(...)` – eliminacja Gaussa sekwencyjna
- `gauss_parallel(...)` – eliminacja Gaussa równoległa
- `multiply_sequential(...)` – mnożenie macierzy (1 proces)
- `rasterize_line(...)` – rasteryzacja linii metodą DDA
- `sort_row(...)` – sortowanie jednego wiersza (bubble sort)
- `merge_vectors(...)` – scalanie dwóch posortowanych wektorów
- `flatten(...)` – zamienia macierz 2D w wektor 1D
- `unflatten(...)` – odwrotnie: 1D → 2D



Obsługa macierzy i pamięci:

- `allocate_matrix(...)` – alokacja dynamiczna macierzy int
- `free_matrix(...)` – zwolnienie pamięci macierzy int
- `clone_matrix(...)` – kopiuje macierz (dla Gaussa)
- `swap_rows(...)` – zamienia dwa wiersze macierzy

