**Autor**

**Imię i nazwisko:** Olaf Olejnik  
**Numer indeksu:** 122681  
**Uczelnia:** Społeczna Akademia Nauk, Łódź

**Opis programu**

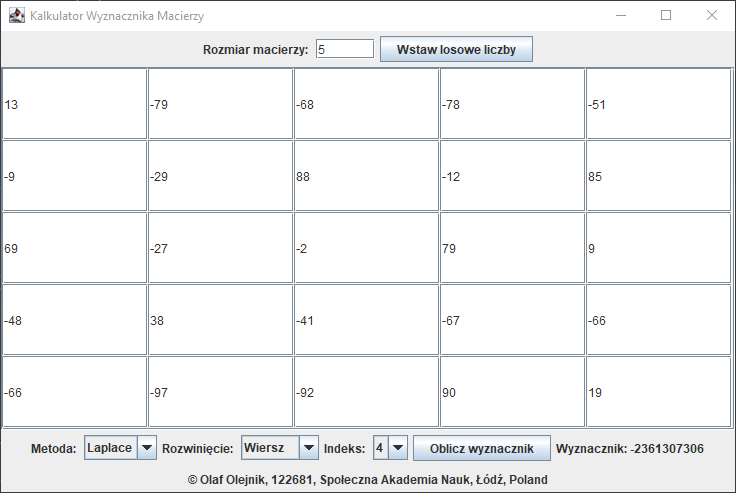
Kalkulator Wyznacznika Macierzy to aplikacja graficzna napisana w języku Java, umożliwiająca użytkownikowi obliczenie wyznacznika macierzy przy wykorzystaniu różnych metod matematycznych. Program został zaimplementowany z wykorzystaniem bibliotek Swing i oferuje intuicyjny interfejs graficzny.

**Funkcjonalności**

1. **Wybór rozmiaru macierzy:** Użytkownik może wybrać rozmiar macierzy w zakresie od 1x1 do 6x6 za pomocą pola tekstowego.
2. **Wprowadzenie danych:** Użytkownik może ręcznie wprowadzać wartości elementów macierzy lub automatycznie wygenerować liczby losowe z zakresu od -100 do 100.
3. **Obliczanie wyznacznika:** Program obsługuje trzy metody obliczeniowe:
   * **Metoda Laplace'a** (z rozwinięciem względem wierszy lub kolumn).
   * **Metoda eliminacji Gaussa.**
   * **Metoda Sarrusa** (dla macierzy 3x3).
4. **Dynamiczne dopasowanie interfejsu:** Wielkość okna oraz liczba pól tekstowych dostosowują się do wybranego rozmiaru macierzy.
5. **Wyświetlanie szczegółowych kroków:** Dla każdej metody obliczeniowej dostępny jest widok szczegółowy, pokazujący kolejne kroki obliczeń w osobnym oknie.
6. **Walidacja danych:** Aplikacja weryfikuje poprawność wprowadzanych danych i wyświetla odpowiednie komunikaty błędów.

**Jak korzystać**

1. **Uruchomienie programu:**
   * Program można uruchomić za pomocą klasy WyznacznikMacierzyGUI (metoda main).
2. **Konfiguracja macierzy:**
   * W polu "Rozmiar macierzy" wprowadź wartość od 1 do 6, aby określić rozmiar macierzy.
   * Wypełnij pola tekstowe danymi lub skorzystaj z przycisku "Wstaw losowe liczby".
3. **Wybór metody obliczeń:**
   * Z rozwijanej listy wybierz jedną z dostępnych metod obliczeniowych.
   * Dla metody Laplace'a wybierz wiersz lub kolumnę, a następnie wskaż ich indeks, a dla Sarrusa sposób rozszerzenia macierzy.
4. **Obliczenie wyznacznika:**
   * Kliknij przycisk "Oblicz wyznacznik", aby uzyskać wynik.
5. **Szczegółowe kroki obliczeń:**
   * W przypadku metody Laplace'a, Gaussa lub Sarrusa, szczegóły są wyświetlane w nowym oknie „Kroki”.

****A screenshot of a computer

Description automatically generated

# Uruchamianie programu i wstępna konfiguracja

## Uruchomienie aplikacji:

1. public static void main(String[] args) {

2. SwingUtilities.invokeLater(() -> new WyznacznikMacierzyGUI().stworzOkno());

3. }

* SwingUtilities.invokeLater zapewnia, że GUI zostanie utworzone i uruchomione w bezpiecznym wątku dla interfejsu graficznego (Event Dispatch Thread - EDT).

## Tworzenie głównego okna aplikacji

1. JFrame frame = new JFrame("Kalkulator Wyznacznika Macierzy");

* Tworzony jest obiekt klasy JFrame, który reprezentuje główne okno aplikacji.
* Konstruktor przyjmuje tytuł okna: "Kalkulator Wyznacznika Macierzy".

1. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

* Ustawienie, aby zamknięcie okna aplikacji spowodowało zakończenie działania programu (EXIT\_ON\_CLOSE).

1. frame.pack();

* Automatyczne dostosowanie rozmiaru okna do preferowanych rozmiarów komponentów w nim zawartych.

1. frame.setMinimumSize(new Dimension(750, 500));

* Ustawienie minimalnego rozmiaru okna na 750 pikseli szerokości i 500 pikseli wysokości.

1. frame.setResizable(true);

* Pozwolenie użytkownikowi na dynamiczne zmienianie rozmiaru okna.

# Interakcja z użytkownikiem, GUI

## Tworzenie paneli i komponentów

1. JPanel panel = new JPanel();

2. panel.setLayout(new BorderLayout());

* Tworzony jest główny panel (JPanel), który organizuje komponenty.
* Używany jest układ BorderLayout, który dzieli przestrzeń na pięć obszarów: północ, południe, wschód, zachód i centrum.

## Pole tekstowe do wprowadzania rozmiaru macierzy

1. JTextField rozmiarField = new JTextField(5);

* Tworzone jest pole tekstowe (JTextField) z widoczną szerokością na 5 znaków.

Dodanie filtra do walidacji danych:

1. ((PlainDocument) rozmiarField.getDocument()).setDocumentFilter(new DocumentFilter() {...});

1. Rzutowanie do PlainDocument:

* Metoda getDocument zwraca obiekt typu Document, który jest rzutowany do PlainDocument w celu zastosowania filtra.

1. Klasa DocumentFilter:

* Umożliwia kontrolę nad wprowadzaniem, modyfikacją i usuwaniem tekstu w polu tekstowym.

1. Metody nadpisane w filtrze:

* insertString:
  + Wywoływana przy wstawianiu nowych znaków.
  + Sprawdza poprawność danych poprzez isAllowedInput i wyświetla komunikat błędu w przypadku nieprawidłowych wartości.

1. @Override

2. public void insertString(FilterBypass fb, int offset, String string, AttributeSet attr) throws BadLocationException {

3. String currentText = fb.getDocument().getText(0, fb.getDocument().getLength());

4. String newText = new StringBuilder(currentText).insert(offset, string).toString();

5. if (isAllowedInput(newText)) {

6. super.insertString(fb, offset, string, attr);

7. } else {

8. pokazBlad(fb);

9. }

10. }

* replace:
  + Wywoływana przy zastępowaniu tekstu.
  + Działa analogicznie do insertString.

1. @Override

2. public void replace(FilterBypass fb, int offset, int length, String text, AttributeSet attrs) throws BadLocationException {

3. String currentText = fb.getDocument().getText(0, fb.getDocument().getLength());

4. String newText = new StringBuilder(currentText).replace(offset, offset + length, text).toString();

5. if (isAllowedInput(newText)) {

6. super.replace(fb, offset, length, text, attrs);

7. } else {

8. pokazBlad(fb);

9. }

10. }

* remove:
  + Wywoływana przy usuwaniu tekstu.
  + Nie stosuje żadnych ograniczeń.

1. @Override

2. public void remove(FilterBypass fb, int offset, int length) throws BadLocationException {

3. super.remove(fb, offset, length);

4. }

1. Metoda isAllowedInput:

* Dozwolone wartości: Tylko cyfry od 1 do 6.
* Unikalność: Każda cyfra może wystąpić tylko raz (np. "12" jest poprawne, ale "112" nie).
* Maksymalna długość: Do 6 znaków (np. "123456").

1. private boolean isAllowedInput(String newText) {

2. String currentText = "";

3. try {

4. currentText = rozmiarField.getText(0, rozmiarField.getDocument().getLength());

5. } catch (BadLocationException e) {

6. e.printStackTrace();

7. }

8. // Scal obecny tekst i nowy tekst

9. String combinedText = currentText + newText;

10. // Sprawdź, czy tekst składa się wyłącznie z unikalnych cyfr 1-6

11. return combinedText.matches("^[1-6]{1,6}$") && hasUniqueCharacters(combinedText);

12. }

1. Metoda pokazBlad:

Gdy użytkownik wprowadzi niedozwolone znaki, pole zostaje wyczyszczone, a użytkownik zobaczy komunikat błędu:

1. private void pokazBlad() {

2. JOptionPane.showMessageDialog(frame, "Dozwolone wartości to tylko liczby od 1 do 6!", "Błąd", JOptionPane.ERROR\_MESSAGE);

3. }

* Wyświetla okno dialogowe z komunikatem błędu w przypadku nieprawidłowych danych.

## Panel do wprowadzania danych macierzy

1. JPanel macierzPanel = new JPanel();

Tworzony jest panel (macierzPanel) do wyświetlania i edycji pól macierzy. Na razie nie ma przypisanego układu.

## Przycisk "Oblicz wyznacznik"

1. JButton obliczPrzycisk = new JButton("Oblicz wyznacznik");

Tworzony jest przycisk z napisem "Oblicz wyznacznik". Jego działanie zostanie przypisane później.

## Pole wyboru metody obliczeń

1. String[] metody = {"Laplace", "Gauss", "Sarrus"};

2. JComboBox<String> metodaCombo = new JComboBox<>(metody);

* Tworzona jest tablica zawierająca nazwy metod obliczania wyznacznika: "Laplace", "Gauss", "Sarrus".
* Na jej podstawie tworzony jest rozwijany wybór (JComboBox), umożliwiający użytkownikowi wybór jednej z metod.

## Pola wyboru dla metody Laplace'a (później też dla Sarrus’a, patrz punkt 19c))

1. JComboBox<String> wyborLaplaceCombo = new JComboBox<>(new String[] {"Wiersz", "Kolumna"});

2. JComboBox<Integer> indeksCombo = new JComboBox<>();

1. wyborLaplaceCombo:

* Tworzony rozwijany wybór dla użytkownika pozwalający na wskazanie, czy operacja w metodzie Laplace'a ma być przeprowadzona względem wierszy czy kolumn.

1. indeksCombo:

* Rozwijany wybór do wskazywania indeksu wiersza/kolumny. Jest pusty, ponieważ zostanie wypełniony danymi dopiero po określeniu rozmiaru macierzy.

## Etykieta wynikowa

1. JLabel wynikLabel = new JLabel("Wyznacznik: ");

* Tworzona etykieta (JLabel) do wyświetlania wyniku wyznacznika macierzy. Początkowo wyświetla napis "Wyznacznik: ".

## Górny panel z polem rozmiaru i przyciskiem losowym

1. JPanel topPanel = new JPanel();

2. topPanel.add(new JLabel("Rozmiar macierzy: "));

3. topPanel.add(rozmiarField);

* Tworzony jest panel (topPanel) do umieszczenia pola tekstowego rozmiarField.
* Dodawana jest etykieta opisująca pole: "Rozmiar macierzy: ".

## Przycisk "Wstaw losowe liczby"

1. JButton losowePrzycisk = new JButton("Wstaw losowe liczby");

* Tworzony jest przycisk umożliwiający wstawienie losowych wartości do pól macierzy.

Obsługa przycisku losowych liczb:

1. losowePrzycisk.addActionListener(e -> {

2. Component[] pola = macierzPanel.getComponents();

3. for (Component pole : pola) {

4. if (pole instanceof JTextField) {

5. int losowaLiczba = (int) (Math.random() \* 201) - 100;

6. ((JTextField) pole).setText(String.valueOf(losowaLiczba));

7. }

8. }

9. });

1. Pobieranie komponentów panelu macierzPanel:

* Wszystkie komponenty w panelu są iterowane za pomocą getComponents.

1. Sprawdzanie typu komponentu:

* Jeśli komponent jest polem tekstowym (JTextField), wstawiana jest losowa liczba.

1. Generowanie liczby:

1. int losowaLiczba = (int) (Math.random() \* 201) - 100;

* Losowa liczba z zakresu od -100 do 100 jest obliczana za pomocą Math.random.

1. Ustawienie wartości pola:

* Wartość losowa jest wstawiana do pola tekstowego za pomocą setText.

## Dodanie nasłuchiwacza zmian w polu rozmiarField

1. rozmiarField.getDocument().addDocumentListener(new javax.swing.event.DocumentListener() { ... });

1. Pobranie obiektu dokumentu pola tekstowego:

* rozmiarField.getDocument() zwraca obiekt dokumentu odpowiedzialny za przechowywanie tekstu w polu rozmiarField.

1. Dodanie DocumentListener:

* addDocumentListener rejestruje nasłuchiwacza, który monitoruje zmiany w zawartości pola tekstowego.

1. Interfejs DocumentListener:

* Definiuje trzy metody, które reagują na różne typy zmian:
  + insertUpdate – wywoływana, gdy użytkownik wstawia tekst.
  + removeUpdate – wywoływana, gdy tekst jest usuwany.
  + changedUpdate – wywoływana w przypadku zmian w atrybutach tekstu (tu nieistotne).

Każda z metod (insertUpdate, removeUpdate, changedUpdate) wywołuje metodę generujMacierz:

1. @Override

2. public void insertUpdate(javax.swing.event.DocumentEvent e) {

3. generujMacierz();

4. }

## Metoda generujMacierz

Ta metoda odpowiada za dynamiczne generowanie pól tekstowych na podstawie wprowadzonego rozmiaru macierzy.

1. Czyszczenie stanu komponentów

1. macierzPanel.removeAll();

2. indeksCombo.removeAllItems();

* macierzPanel.removeAll() usuwa wszystkie pola tekstowe z panelu macierzy, przygotowując go na dodanie nowych elementów.
* indeksCombo.removeAllItems() usuwa wszystkie elementy z rozwijanego wyboru indeksów.

1. Pobranie wartości z pola tekstowego

1. String input = rozmiarField.getText();

* Pobierany jest aktualny tekst wprowadzony przez użytkownika.

1. Sprawdzenie pustego pola

1. if (input.isEmpty()) {

2. frame.revalidate();

3. frame.repaint();

4. return;

5. }

* Jeśli pole tekstowe jest puste, metoda kończy swoje działanie bez dalszych zmian.

1. Walidacja wprowadzonej wartości

1. if (!input.matches("[1-6]")) {

2. throw new IllegalArgumentException("Dozwolone rozmiary macierzy to tylko 1, 2, 3, 4, 5, 6.");

3. }

* Sprawdzane jest, czy wprowadzony tekst pasuje do wyrażenia regularnego [1-6] (czyli czy jest cyfrą z zakresu 1–6).
* W przypadku niezgodności zgłaszany jest wyjątek IllegalArgumentException.

1. Tworzenie nowej macierzy

* Parsowanie rozmiaru:

1. int rozmiar = Integer.parseInt(input);

* + Wartość tekstowa jest zamieniana na liczbę całkowitą.
* Konfiguracja układu panelu macierzy:

1. macierzPanel.setLayout(new GridLayout(rozmiar, rozmiar));

* Panel macierzy otrzymuje układ GridLayout, który dzieli przestrzeń na siatkę o wymiarach rozmiar x rozmiar.
* Dodanie pól tekstowych:

1. for (int i = 0; i < rozmiar \* rozmiar; i++) {

2. JTextField pole = new JTextField(3);

3. pole.setText("0");

4. macierzPanel.add(pole);

5. }

* Iteracja przez wszystkie elementy macierzy (liczba pól tekstowych to rozmiar \* rozmiar).
* Dla każdego pola:
  + Tworzone jest pole tekstowe o szerokości 3 znaki.
  + Ustawiana jest domyślna wartość "0".
  + Pole tekstowe jest dodawane do panelu macierzy.
* Aktualizacja rozwijanego wyboru indeksów:

1. for (int i = 1; i <= rozmiar; i++) {

2. indeksCombo.addItem(i);

3. }

* Dodawane są kolejne liczby (od 1 do rozmiar) do rozwijanego wyboru indeksów.

## Dynamiczna zmiana rozmiaru okna

1. int newHeight = 100 + rozmiar \* 30;

2. int newWidth = 200 + rozmiar \* 30;

3. SwingUtilities.invokeLater(() -> {

4. frame.setSize(Math.max(newWidth, 500), Math.max(newHeight, 400));

5. frame.revalidate();

6. frame.repaint();

7. });

8.

* Wysokość (newHeight) i szerokość (newWidth) okna są obliczane dynamicznie na podstawie rozmiaru macierzy.
* Użycie SwingUtilities.invokeLater zapewnia, że zmiana rozmiaru okna jest wykonywana w bezpiecznym wątku GUI (EDT).
* Okno jest:
* Ustawiane na nowy rozmiar, przy czym minimalne wartości to 500x400.
* Aktualizowane za pomocą revalidate i repaint.

## Obsługa błędów

1. Niepoprawny format liczby:

1. catch (NumberFormatException ex) {

2. macierzPanel.removeAll();

3. frame.revalidate();

4. frame.repaint();

5. }

* Jeśli wprowadzona wartość nie jest liczbą, macierz jest czyszczona i interfejs odświeżany.

1. Niepoprawny zakres:

1. catch (IllegalArgumentException ex) {

2. JOptionPane.showMessageDialog(frame, ex.getMessage(), "Błąd", JOptionPane.ERROR\_MESSAGE);

3. rozmiarField.setText(""); // Czyści pole w przypadku błędu

4. }

* W przypadku zgłoszenia wyjątku IllegalArgumentException wyświetlany jest komunikat błędu.
* Pole tekstowe rozmiarField jest czyszczone, aby wymusić poprawne dane.

## Obsługa przycisku "Oblicz wyznacznik"

1. obliczPrzycisk.addActionListener(e -> { ... });

* Dodanie nasłuchiwacza zdarzeń:
* Metoda addActionListener dodaje reakcję na kliknięcie przycisku obliczPrzycisk.
* Wyrażenie lambda definiuje, co ma się stać po naciśnięciu przycisku.

## Pobranie danych macierzy

1. Pobranie komponentów panelu macierzy:

1. Component[] pola = macierzPanel.getComponents();

* macierzPanel.getComponents() zwraca wszystkie pola tekstowe utworzone w panelu macierzy.

1. Obliczenie rozmiaru macierzy:

1. int rozmiar = (int) Math.sqrt(pola.length);

* Zakłada się, że liczba komponentów jest kwadratem liczby wierszy i kolumn macierzy.
* Wyciągana jest pierwiastek kwadratowy z liczby komponentów.

1. Tworzenie tablicy do przechowywania danych:

1. double[][] macierz = new double[rozmiar][rozmiar];

* Tworzona jest tablica dwuwymiarowa, która będzie przechowywać wartości wprowadzone w polach tekstowych.

1. Pobranie wartości z pól tekstowych:

1. for (int i = 0; i < rozmiar; i++) {

2. for (int j = 0; j < rozmiar; j++) {

3. JTextField pole = (JTextField) pola[i \* rozmiar + j];

4. macierz[i][j] = Double.parseDouble(pole.getText());

5. }

6. }

* Dwie pętle for iterują po wierszach i kolumnach.
* Pole tekstowe jest rzutowane na JTextField.
* Tekst z pola jest konwertowany na liczbę zmiennoprzecinkową (Double.parseDouble).

## Wybór metody obliczeń

Pobranie wybranej metody:

1. String metoda = (String) metodaCombo.getSelectedItem();

* metodaCombo.getSelectedItem() zwraca aktualnie wybraną metodę obliczania wyznacznika.

## Obsługa metody Laplace'a

1. if ("Laplace".equals(metoda)) {

2. int indeks = (int) indeksCombo.getSelectedItem() - 1;

3. String wybor = (String) wyborLaplaceCombo.getSelectedItem();

4. if ("Wiersz".equalsIgnoreCase(wybor)) {

5. ...

6. } else if ("Kolumna".equalsIgnoreCase(wybor)) {

7. ...

8. }

9. }

1. Pobranie ustawień dla Laplace'a:

* indeksCombo.getSelectedItem() pobiera wybrany wiersz lub kolumnę.
* wyborLaplaceCombo.getSelectedItem() określa, czy obliczenia mają być wykonane względem wierszy czy kolumn.

1. Obliczenia względem wiersza lub kolumny:

* W zależności od wartości wybor, wywoływane są odpowiednie funkcje.

1. Asynchroniczność z użyciem SwingWorker:

1. SwingWorker<Void, Void> worker = new SwingWorker<>() { ... };

2. worker.execute();

* SwingWorker uruchamia obliczenia w tle, aby nie blokować interfejsu użytkownika.
* Metoda doInBackground wykonuje obliczenia, a done aktualizuje wynik w GUI.

## Obsługa metody Gaussa

1. else if ("Gauss".equals(metoda)) {

2. SwingWorker<Void, Void> worker = new SwingWorker<>() { ... };

3. worker.execute();

4. }

* Wykonywana jest eliminacja Gaussa.
* Wynik obliczany w doInBackground jest wyświetlany po zakończeniu obliczeń.

## Obsługa metody Sarrusa

1. else if ("Sarrus".equals(metoda)) {

2. if (macierz.length != 3 || macierz[0].length != 3) {

3. JOptionPane.showMessageDialog(frame, "Metoda Sarrusa działa tylko dla macierzy 3x3!", "Błąd", JOptionPane.ERROR\_MESSAGE);

4. return;

5. }

6. SwingWorker<Void, Void> worker = new SwingWorker<>() { ... };

7. worker.execute();

8. }

1. Walidacja rozmiaru macierzy:

* Sprawdzane jest, czy macierz jest rozmiaru 3x3. W przeciwnym razie wyświetlany jest komunikat błędu.

1. Obliczenia metodą Sarrusa:

* SwingWorker wykonuje obliczenia w tle.
* Wynik jest wyświetlany po zakończeniu obliczeń.

1. Pobranie ustawień z wyborLaplaceCombo.getSelectedItem()

1. String wybor = (String) wyborLaplaceCombo.getSelectedItem();

2. boolean dopiszWiersze = "Wiersz".equalsIgnoreCase(wybor);

* Jako, że funkcjonalność polegająca na możliwości wyboru rozszerzenia macierzy 3x3 poprzez dopisanie kolejnych dwóch kolumn lub wierszy identycznych do pierwszych została dodana później, to pobiera wybór ze string’u wyborLaplaceCombo.getSelectedItem().
* Parametr dopiszWiersze jest true, jeśli użytkownik wybrał "Wiersz", a false - jeśli "Kolumna".

## Obsługa błędów

1. catch (NumberFormatException ex) {

2. JOptionPane.showMessageDialog(frame, "Wprowadź poprawne liczby w macierzy!", "Błąd", JOptionPane.ERROR\_MESSAGE);

3. }

4. catch (IllegalArgumentException ex) {

5. JOptionPane.showMessageDialog(frame, ex.getMessage(), "Błąd", JOptionPane.ERROR\_MESSAGE);

6. }

* Obsługa wyjątków:
  + NumberFormatException – w przypadku, gdy wartości w polach tekstowych nie są liczbami.
  + IllegalArgumentException – w przypadku innych nieprawidłowych danych.

## Dolny panel (GUI)

1. Tworzenie dolnego panelu:

1. JPanel bottomPanel = new JPanel();

2.

3. Dodanie elementów GUI:

4.

5. bottomPanel.add(new JLabel("Metoda: "));

6. bottomPanel.add(metodaCombo);

7. ...

8. bottomPanel.add(obliczPrzycisk);

9. bottomPanel.add(wynikLabel);

* + Dodawane są etykiety, rozwijane pola wyboru (metodaCombo, wyborLaplaceCombo, indeksCombo) oraz przyciski (obliczPrzycisk).

1. Dodanie elementów GUI:

1. bottomPanel.add(new JLabel("Metoda: "));

2. bottomPanel.add(metodaCombo);

3. ...

4. bottomPanel.add(obliczPrzycisk);

5. bottomPanel.add(wynikLabel);

* + Dodawane są etykiety, rozwijane pola wyboru (metodaCombo, wyborLaplaceCombo, indeksCombo) oraz przyciski (obliczPrzycisk).

1. Nasłuchiwacz dla metodaCombo:

1. metodaCombo.addActionListener(e -> {

2. int rozmiar = (int) Math.sqrt(macierzPanel.getComponentCount()); // Oblicz rozmiar macierzy

3. boolean isLaplace = "Laplace".equals(metodaCombo.getSelectedItem());

4. boolean isSarrus = "Sarrus".equals(metodaCombo.getSelectedItem());

5.

6. if (isSarrus) {

7. if (rozmiar != 3) {

8. JOptionPane.showMessageDialog(frame, "Metoda Sarrusa działa tylko dla macierzy 3x3!", "Błąd", JOptionPane.ERROR\_MESSAGE);

9. metodaCombo.setSelectedIndex(0); // Przywróć domyślną metodę (Laplace)

10. } else {

11. rozmiarField.setEnabled(false); // Zablokuj zmianę rozmiaru macierzy

12. }

13. } else {

14. rozmiarField.setEnabled(true); // Odblokuj zmianę rozmiaru dla innych metod

15. }

16.

17. wyborLaplaceCombo.setEnabled(isLaplace || isSarrus);

18. indeksCombo.setEnabled(isLaplace);

19. });

* Ustawienia interfejsu są aktualizowane w zależności od wybranej metody:
  + - Przy metodzie Sarrusa rozmiar macierzy jest blokowany na 3x3, pole rozmiarField jest nieaktywne.
    - Inne metody pozwalają na dowolny rozmiar (1–6).

## Sygnatura i kontener nadrzędny

1. Panel sygnatury:

1. JPanel sygnaturaPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

2. JLabel sygnaturaLabel = new JLabel("© Olaf Olejnik, 122681, Społeczna Akademia Nauk, Łódź");

3. sygnaturaPanel.add(sygnaturaLabel);

1. Kontener dla dolnego panelu i sygnatury:

1. JPanel dolnyKontener = new JPanel();

2. dolnyKontener.setLayout(new BoxLayout(dolnyKontener, BoxLayout.Y\_AXIS));

3. dolnyKontener.add(bottomPanel);

4. dolnyKontener.add(sygnaturaPanel);

## Dodanie paneli do okna i ustawienia ramy

1. panel.add(topPanel, BorderLayout.NORTH);

2. panel.add(new JScrollPane(macierzPanel), BorderLayout.CENTER);

3. panel.add(dolnyKontener, BorderLayout.SOUTH);

* Górny panel (topPanel) znajduje się na górze (NORTH).
* Panel z macierzą (macierzPanel) jest w centrum (CENTER).
* Dolny kontener (dolnyKontener) jest na dole (SOUTH).

1. frame.add(panel);

2. frame.pack();

3. frame.setVisible(true);

* Główny panel jest dodawany do ramki (frame).
* pack() dostosowuje rozmiar okna do komponentów.
* setVisible(true) powoduje wyświetlenie okna użytkownikowi.

# Metody pokazKroki

## Metoda pokazKrokiLaplace

1. private void pokazKrokiLaplace(double[][] macierz, boolean poWierszach, int indeks) {

2. JTextArea krokiArea = new JTextArea(20, 50);

3. krokiArea.setEditable(false);

4.

5. JFrame krokiFrame = new JFrame("Kroki - Laplace");

6. krokiFrame.add(new JScrollPane(krokiArea));

7. krokiFrame.setSize(600, 400);

8. krokiFrame.setVisible(true);

9.

10. SwingWorker<Void, String> worker = new SwingWorker<>() {

11. @Override

12. protected Void doInBackground() {

13. obliczWyznacznikLaplace(macierz, krokiArea, poWierszach, indeks);

14. return null;

15. }

16. };

17.

18. worker.execute();

19. }

1. Tworzenie interfejsu do wyświetlania kroków:

* JTextArea krokiArea: Obszar tekstowy o wymiarach 20 wierszy i 50 kolumn, który będzie zawierał szczegóły obliczeń.
* krokiArea.setEditable(false): Zapewnia, że użytkownik nie może edytować tekstu w obszarze.
* JFrame krokiFrame: Tworzy nowe okno z tytułem "Kroki - Laplace".
* JScrollPane: Obudowuje krokiArea, umożliwiając przewijanie zawartości.

1. Asynchroniczne obliczenia z użyciem SwingWorker:

* doInBackground: Wywołuje metodę obliczWyznacznikLaplace, która wykonuje obliczenia w tle.
* worker.execute(): Uruchamia SwingWorker.

## Metoda pokazKrokiGauss

1. private void pokazKrokiGauss(double[][] macierz) {

2. JTextArea krokiArea = new JTextArea(20, 50);

3. krokiArea.setEditable(false);

4.

5. JFrame krokiFrame = new JFrame("Kroki - Gauss");

6. krokiFrame.add(new JScrollPane(krokiArea));

7. krokiFrame.setSize(600, 400);

8. krokiFrame.setVisible(true);

9.

10. SwingWorker<Void, String> worker = new SwingWorker<>() {

11. @Override

12. protected Void doInBackground() {

13. obliczWyznacznikGauss(macierz, krokiArea);

14. return null;

15. }

16. };

17.

18. worker.execute();

19. }

1. Podobieństwo do pokazKrokiLaplace:

* Tworzony jest interfejs użytkownika do wyświetlania kroków obliczeń.
* JTextArea i JScrollPane działają identycznie.

1. Asynchroniczność:

* doInBackground: Wywołuje metodę obliczWyznacznikGauss, która przeprowadza eliminację Gaussa w tle.

## Metoda pokazKrokiSarrus

1. private void pokazKrokiSarrus(double[][] macierz, boolean dopiszWiersze) {

2. JTextArea krokiArea = new JTextArea(20, 50);

3. krokiArea.setEditable(false);

4.

5. JFrame krokiFrame = new JFrame("Kroki - Sarrus");

6. krokiFrame.add(new JScrollPane(krokiArea));

7. krokiFrame.setSize(600, 400);

8. krokiFrame.setVisible(true);

9.

10. SwingWorker<Void, String> worker = new SwingWorker<>() {

11. @Override

12. protected Void doInBackground() {

13. obliczWyznacznikSarrus(macierz, krokiArea, dopiszWiersze);

14. return null;

15. }

16. };

17.

18. worker.execute();

19. }

1. Podobieństwo do poprzednich metod:

* Tworzy okno, w którym wyświetlane są szczegóły obliczeń metodą Sarrusa.

1. Parametr dopiszWiersze:

* Steruje sposobem rozszerzenia macierzy:
  + true: Rozszerzanie poprzez dodanie dwóch pierwszych wierszy.
  + false: Rozszerzanie poprzez dodanie dwóch pierwszych kolumn.

## Wspólne cechy metod pokazKroki

1. Tworzenie interfejsu:

* Każda metoda tworzy nowy JFrame do wyświetlenia kroków obliczeń.
* JScrollPane umożliwia wygodne przewijanie wyników.

1. Asynchroniczne obliczenia:

* SwingWorker zapewnia, że obliczenia są wykonywane w tle, a GUI pozostaje responsywne.
* Wynik obliczeń jest wyświetlany w JTextArea po zakończeniu obliczeń.

1. Nazwy okien:

* Tytuł okna ("Kroki - Laplace", "Kroki - Gauss", "Kroki - Sarrus") wskazuje, której metody dotyczą wyświetlane kroki.

# Metody obliczWyznacznik

## Metoda obliczWyznacznikLaplace

1. private double obliczWyznacznikLaplace(double[][] macierz, JTextArea krokiArea, boolean poWierszach, int indeks) {

* Parametry:
  + macierz: Dwuwymiarowa tablica liczb reprezentująca macierz wejściową.
  + krokiArea: Obszar tekstowy do wyświetlania kolejnych kroków.
  + poWierszach: Wskazuje, czy rozwinięcie ma być wykonywane względem wierszy (true) czy kolumn (false).
  + indeks: Wskazanie wiersza lub kolumny dla rozwinięcia.

1. Przypadek bazowy: macierz 1x1

1. if (n == 1) {

2. krokiArea.append("Wyznacznik dla macierzy 1x1: " + macierz[0][0] + "\n");

3. return macierz[0][0];

4. }

* Jeśli macierz ma rozmiar 1x1, jej wyznacznik to jedyny element w macierzy.
* Komunikat:

1. Wyznacznik dla macierzy 1x1: [wartość elementu]

1. Przypadek bazowy: macierz 2x2

1. else if (n == 2) {

2. BigDecimal bd1 = new BigDecimal(macierz[0][0]).setScale(5, RoundingMode.HALF\_UP).stripTrailingZeros();

3. ...

4. BigDecimal wynik = bd1.multiply(bd2).subtract(bd3.multiply(bd4));

5. krokiArea.append("Wyznacznik dla macierzy 2x2: " + wynik.stripTrailingZeros().toPlainString() + "\n");

6. return wynik.doubleValue();

7. }

* Wyznacznik macierzy 2x2 obliczany jest jako różnica iloczynów przekątnych.
* BigDecimal zapewnia precyzyjne obliczenia.
* Komunikat:

1. Wyznacznik dla macierzy 2x2: [wynik]

1. Ogólny przypadek dla większych macierzy

* Iteracja po elementach wiersza/kolumny:

1. for (int i = 0; i < n; i++) {

2. int row = poWierszach ? indeks : i;

3. int col = poWierszach ? i : indeks;

4. ...

5. }

* + Wybór indeksu wiersza lub kolumny zależy od wartości poWierszach.
* Pomijanie zer:

1. if (macierz[row][col] == 0) {

2. krokiArea.append("Element A(" + (row + 1) + "," + (col + 1) + ") = 0, pomijanie...\n");

3. continue;

4. }

5.

* Elementy zerowe są pomijane, aby zminimalizować obliczenia.
* Komunikat:

1. Element A(wiersz, kolumna) = 0, pomijanie...

* Tworzenie podmacierzy:

1. double[][] podmacierz = podmacierz(macierz, row, col);

* + Metoda pomocnicza podmacierz tworzy nową macierz, wykluczając wybrany wiersz i kolumnę (omówiona w kolejnej sekcji).
* Rekurencyjne obliczenia:

1. double wynikCzesciowy = Math.pow(-1, row + col) \* macierz[row][col] \* obliczWyznacznikLaplace(podmacierz, krokiArea, true, 0);

2. krokiArea.append("Częściowy wynik: " + formatujLiczbe(wynikCzesciowy) + "\n");

* Rekurencyjne wywołanie metody obliczWyznacznikLaplace.
* Komunikaty:

1. Obliczanie rozwinięcia dla elementu A(wiersz, kolumna): [wartość]

2. Podmacierz:

3. [podmacierz w formie tekstowej]

4. Częściowy wynik: [wartość]

* Sumowanie wyników:

1. wyznacznik += wynikCzesciowy;

## Metoda obliczWyznacznikGauss

1. private double obliczWyznacznikGauss(double[][] macierz, JTextArea krokiArea) {

* Parametry:
* macierz: Dwuwymiarowa tablica reprezentująca macierz wejściową.
* krokiArea: Obszar tekstowy, do którego dodawane są szczegóły kolejnych kroków obliczeń.

1. Tworzenie kopii macierzy

Obliczenia modyfikują macierz wejściową, dlatego tworzona jest jej kopia, aby zachować oryginalną macierz:

1. int n = macierz.length;

2. double[][] kopia = new double[n][n];

3. for (int i = 0; i < n; i++) {

4. for (int j = 0; j < n; j++) {

5. kopia[i][j] = macierz[i][j];

6. }

7. }

* Długość macierzy:
  + n = macierz.length oznacza liczbę wierszy/kolumn macierzy (kwadratowej).
* Iteracja i kopiowanie:
  + Zagnieżdżona pętla for iteruje po wszystkich elementach macierzy i kopiuje ich wartości do nowej tablicy kopia.

1. Inicjalizacja wyznacznika

1. double det = 1;

* Wyznacznik (det) jest początkowo ustawiony na 1.
* Zostanie pomnożony przez wartości na głównej przekątnej macierzy po redukcji Gaussa.

1. Eliminacja Gaussa

* Iteracja po kolumnach:

1. for (int i = 0; i < n; i++) { ... }

* + Pętla iteruje po każdej kolumnie (i) macierzy, prowadząc proces redukcji Gaussa.

1. Znajdowanie maksymalnego elementu w kolumnie

Zapewnia to numeryczną stabilność obliczeń i minimalizuje błędy związane z dzieleniem przez małe wartości:

1. int max = i;

2. for (int j = i + 1; j < n; j++) {

3. if (Math.abs(kopia[j][i]) > Math.abs(kopia[max][i])) {

4. max = j;

5. }

6. }

* Iteracja po wierszach:
  + j zaczyna od i + 1, ponieważ wiersze powyżej są już zredukowane.
* Aktualizacja indeksu wiersza z maksymalnym elementem:
  + Jeśli wartość bezwzględna elementu kopia[j][i] jest większa niż elementu w bieżącym max, indeks max zostaje zaktualizowany.

1. Zamiana wierszy

1. if (i != max) {

2. double[] temp = kopia[i];

3. kopia[i] = kopia[max];

4. kopia[max] = temp;

5. det \*= -1;

6. appendToArea(krokiArea, "Zamiana: Wiersz " + (i + 1) + " <-> Wiersz " + (max + 1));

7. }

* Sprawdzenie potrzeby zamiany:
  + Jeśli bieżący wiersz i nie zawiera maksymalnego elementu, zamiana jest konieczna.
* Zamiana wierszy:
  + Tymczasowy wskaźnik temp przechowuje wiersz i.
  + Wiersze i i max są zamieniane miejscami.
* Znak wyznacznika:
  + Zamiana wierszy powoduje zmianę znaku wyznacznika (det \*= -1).
* Komunikat w krokiArea:

1. Zamiana: Wiersz [i+1] <-> Wiersz [max+1]

1. Sprawdzenie zerowego elementu na przekątnej

Jeśli element na głównej przekątnej (kopia[i][i]) jest równy 0, wyznacznik całej macierzy jest równy 0:

1. if (kopia[i][i] == 0) {

2. appendToArea(krokiArea, "Wyznacznik równy 0 - zerowy element na przekątnej");

3. return 0;

4. }

* Komunikat w krokiArea:

1. Wyznacznik równy 0 - zerowy element na przekątnej

1. Redukcja wierszy

1. for (int j = i + 1; j < n; j++) {

2. double factor = kopia[j][i] / kopia[i][i];

3. appendToArea(krokiArea, "Wiersz " + (j + 1) + " = Wiersz " + (j + 1) + " - " + factor + " \* Wiersz " + (i + 1));

4. for (int k = i; k < n; k++) {

5. kopia[j][k] -= factor \* kopia[i][k];

6. }

7. }

* Iteracja po wierszach poniżej bieżącego:
  + Dla każdego wiersza poniżej i, elementy w kolumnie i są redukowane do 0.
* Obliczenie współczynnika:
  + factor to iloraz elementu poniżej pivotu przez pivot (kopia[j][i] / kopia[i][i]).
* Aktualizacja wierszy:
  + Każdy element w wierszu j jest zmniejszany o wartość pivotu pomnożoną przez factor.
* Komunikat w krokiArea:

1. Wiersz [j+1] = Wiersz [j+1] - [factor] \* Wiersz [i+1]

1. Obliczenie wyznacznika

1. det \*= kopia[i][i];

* Pivot (kopia[i][i]) jest mnożony do wyznacznika det.

1. Wyświetlenie i zwrócenie wyniku

1. appendToArea(krokiArea, "Wyznacznik: " + formatujLiczbe(det));

2. return det;

* Prezentacja wyniku:
  + Wyznacznik jest zaokrąglany i formatowany do czytelnej postaci za pomocą metody formatujLiczbe.
* Komunikat w krokiArea:

1. Wyznacznik: [wartość]

* Zwracanie wyniku:
  + Ostateczny wyznacznik jest zwracany jako liczba double.

## Metoda obliczWyznacznikSarrus

1. private double obliczWyznacznikSarrus(double[][] macierz, JTextArea krokiArea, boolean dopiszWiersze) {

* Parametry:
  + macierz: Dwuwymiarowa tablica reprezentująca macierz wejściową.
  + krokiArea: Obszar tekstowy, do którego są dodawane kolejne kroki obliczeń.
  + dopiszWiersze: Określa sposób rozszerzenia macierzy:
  + true: Dodawane są dwa pierwsze wiersze.
  + false: Dodawane są dwie pierwsze kolumny.

1. Walidacja rozmiaru macierzy

1. if (macierz.length != 3 || macierz[0].length != 3) {

2. throw new IllegalArgumentException("Metoda Sarrusa działa tylko dla macierzy 3x3.");

3. }

* Metoda Sarrusa jest ograniczona do macierzy 3x3.
* Jeśli rozmiar macierzy jest inny, zgłaszany jest wyjątek IllegalArgumentException z komunikatem:

1. Metoda Sarrusa działa tylko dla macierzy 3x3.

1. Tworzenie rozszerzonej macierzy

1. double[][] rozszerzona = new double[3][5];

2.

* Tworzona jest nowa tablica o rozmiarze 3x5, która będzie przechowywać rozszerzoną macierz.
* Rozszerzanie przez wiersze (dopiszWiersze == true):

1. if (dopiszWiersze) {

2. for (int i = 0; i < 3; i++) {

3. for (int j = 0; j < 3; j++) {

4. rozszerzona[i][j] = macierz[i][j];

5. if (j < 2) {

6. rozszerzona[i][j + 3] = macierz[i][j]; // Kopiowanie dodatkowych kolumn

7. }

8. }

9. }

10. }

* + Kopiowane są oryginalne wartości macierzy do kolumn 0, 1, 2.
  + Dwie pierwsze kolumny są kopiowane ponownie do kolumn 3 i 4.
* Rozszerzanie przez kolumny (dopiszWiersze == false):

1. else {

2. for (int i = 0; i < 3; i++) {

3. for (int j = 0; j < 3; j++) {

4. rozszerzona[i][j] = macierz[i][j];

5. }

6. }

7. for (int i = 0; i < 3; i++) {

8. rozszerzona[i][3] = macierz[i][0]; // Kopiowanie pierwszej kolumny

9. rozszerzona[i][4] = macierz[i][1]; // Kopiowanie drugiej kolumny

10. }

11. }

* + Kopiowane są oryginalne wartości macierzy do wierszy 0, 1, 2.
  + Dwa pierwsze wiersze są kopiowane ponownie do wierszy 3 i 4.

1. Obliczanie iloczynów przekątnych

1. double suma1 = rozszerzona[0][0] \* rozszerzona[1][1] \* rozszerzona[2][2] +

2. rozszerzona[0][1] \* rozszerzona[1][2] \* rozszerzona[2][3] +

3. rozszerzona[0][2] \* rozszerzona[1][3] \* rozszerzona[2][4];

4. krokiArea.append("Suma iloczynów przekątnych: " + suma1 + "\n");

* Pierwsza suma (suma1):
  + Obliczane są trzy iloczyny przekątnych:
    - Przekątna 1: rozszerzona[0][0] \* rozszerzona[1][1] \* rozszerzona[2][2].
    - Przekątna 2: rozszerzona[0][1] \* rozszerzona[1][2] \* rozszerzona[2][3].
    - Przekątna 3: rozszerzona[0][2] \* rozszerzona[1][3] \* rozszerzona[2][4].
  + Wynik tych sum jest zapisywany jako suma1.
* Komunikat w krokiArea:

1. Suma iloczynów przekątnych: [wartość suma1]

1. Obliczanie iloczynów przeciwprzekątnych

1. double suma2 = rozszerzona[0][4] \* rozszerzona[1][3] \* rozszerzona[2][2] +

2. rozszerzona[0][3] \* rozszerzona[1][2] \* rozszerzona[2][1] +

3. rozszerzona[0][2] \* rozszerzona[1][1] \* rozszerzona[2][0];

4. krokiArea.append("Suma iloczynów przeciwprzekątnych: " + suma2 + "\n");

* Druga suma (suma2):
  + Obliczane są trzy iloczyny przeciwprzekątnych:
    - Przeciwprzekątna 1: rozszerzona[0][4] \* rozszerzona[1][3] \* rozszerzona[2][2].
    - Przeciwprzekątna 2: rozszerzona[0][3] \* rozszerzona[1][2] \* rozszerzona[2][1].
    - Przeciwprzekątna 3: rozszerzona[0][2] \* rozszerzona[1][1] \* rozszerzona[2][0].
  + Wynik tych sum jest zapisywany jako suma2.
* Komunikat w krokiArea:

1. Suma iloczynów przeciwprzekątnych: [wartość suma2]

1. Obliczenie wyznacznika

1. double wynik = suma1 - suma2;

2. krokiArea.append("Wyznacznik (Suma1 - Suma2): " + formatujLiczbe(wynik) + "\n");

3. return wynik;

* Równanie:
  + Wyznacznik jest różnicą sum przekątnych i przeciwprzekątnych:

1. wyznacznik = suma1 - suma2

* Komunikat w krokiArea:

1. Wyznacznik (Suma1 - Suma2): [wartość wyznacznika]

* Zwracanie wyniku:
  + Ostateczny wynik jest zwracany jako liczba zmiennoprzecinkowa (double).

# Metody pomocnicze

## Metoda appendToArea

1. private void appendToArea(JTextArea krokiArea, String message) {

2. SwingUtilities.invokeLater(() -> krokiArea.append(message + "\n"));

3. }

* Umożliwia aktualizację obszaru tekstowego (JTextArea) w sposób bezpieczny dla wątku GUI.
* Operacje na GUI (np. krokiArea.append) muszą być wykonywane w wątku zdarzeń Swinga (Event Dispatch Thread - EDT).
* SwingUtilities.invokeLater zapewnia, że podana akcja zostanie wykonana w odpowiednim wątku.
* Dodanie wiadomości:
  + krokiArea.append(message + "\n") dodaje nową linię z tekstem message do obszaru tekstowego.
* Przykładowe użycie w metodzie Gaussa:

1. appendToArea(krokiArea, "Zamiana: Wiersz " + (i + 1) + " <-> Wiersz " + (max + 1));

* + Komunikat o zamianie wierszy zostanie dodany do interfejsu użytkownika w czasie rzeczywistym.

## Metoda formatujLiczbe

1. private String formatujLiczbe(double liczba) {

2. BigDecimal bd = new BigDecimal(liczba);

3. bd = bd.setScale(5, RoundingMode.HALF\_UP); // Zaokrąglij do 5 miejsc

4. return bd.stripTrailingZeros().toPlainString(); // Usuń zbędne zera i .0

5. }

* Poprawia czytelność wyników obliczeń przez zaokrąglenie do 5 miejsc po przecinku i usunięcie zbędnych zer.
* Klasa BigDecimal oferuje precyzyjne operacje arytmetyczne, eliminując problemy z zaokrąglaniem charakterystyczne dla typu double.

1. Konstruktor:

1. BigDecimal bd = new BigDecimal(liczba);

* Tworzy obiekt BigDecimal na podstawie liczby zmiennoprzecinkowej liczba.

1. Zaokrąglanie:

1. bd = bd.setScale(5, RoundingMode.HALF\_UP);

2.

* Zaokrągla liczbę do 5 miejsc dziesiętnych, stosując regułę zaokrąglania do najbliższej liczby (w przypadku remisu w górę).

1. Usuwanie zbędnych zer:

1. return bd.stripTrailingZeros().toPlainString();

* Usuwa niepotrzebne zera na końcu oraz .0 dla liczb całkowitych.

1. Przykładowe użycie:

* W metodzie Gaussa:

1. appendToArea(krokiArea, "Wyznacznik: " + formatujLiczbe(det));

* + Wyznacznik det jest formatowany przed wyświetleniem.

## Metoda podmacierz

1. private double[][] podmacierz(double[][] macierz, int wiersz, int kolumna) {

2. int n = macierz.length;

3. double[][] podmacierz = new double[n - 1][n - 1];

4.

5. int p = 0;

6. for (int i = 0; i < n; i++) {

7. if (i == wiersz) continue;

8. int q = 0;

9. for (int j = 0; j < n; j++) {

10. if (j == kolumna) continue;

11. podmacierz[p][q] = macierz[i][j];

12. q++;

13. }

14. p++;

15. }

16. return podmacierz;

17. }

* Tworzy podmacierz przez usunięcie określonego wiersza i kolumny z macierzy wejściowej.
* Parametry:
  + macierz: Oryginalna macierz, z której generowana jest podmacierz.
  + wiersz: Indeks wiersza do usunięcia.
  + kolumna: Indeks kolumny do usunięcia.

1. Rozmiar podmacierzy:

1. double[][] podmacierz = new double[n - 1][n - 1];

* Tworzona jest nowa tablica o wymiarach (n-1) x (n-1).

1. Iteracja po wierszach macierzy:

1. for (int i = 0; i < n; i++) {

2. if (i == wiersz) continue;

3. }

* Pętla iteruje po wszystkich wierszach macierzy wejściowej.
* Wiersz o indeksie wiersz jest pomijany (continue).

1. Iteracja po kolumnach macierzy:

1. for (int j = 0; j < n; j++) {

2. if (j == kolumna) continue;

3. podmacierz[p][q] = macierz[i][j];

4. q++;

5. }

* Pętla iteruje po wszystkich kolumnach bieżącego wiersza.
* Kolumna o indeksie kolumna jest pomijana (continue).
* Elementy pozostałych kolumn są kopiowane do podmacierzy.

1. Zmienne p i q:

* p: Indeks wiersza w podmacierzy.
* q: Indeks kolumny w podmacierzy.

1. Przykładowe użycie w metodzie Laplace'a:

1. double[][] podmacierz = podmacierz(macierz, row, col);

* Tworzy podmacierz dla bieżącego elementu podczas obliczania wyznacznika.