Zadanie 2 - eliminacja Gaussa i faktoryzacja LU

Mateusz Wejman, Andrzej Starzyk

Marzec/Kwiecień 2024

1 Wykonanie

Celem zadania była implementacja algorytmów eliminacji Gaussa oraz faktoryzacji LU w wersjach niestabilnej i stabilnej.

1.1 Pseudokody algorytmów

Niech M to macierz reprezentująca układ równań, n to jej rozmiar, a b to prawa strona układu.

Algorithm 1 Eliminacja Gaussa z 1 na przekątnej

1: **for** $i = 1, \ldots, n$ **do**
2: Podziel elementy w wierszu i-tym i b_i przez $m_{i,i}$
3: **for** $j = i + 1, \ldots, N$ **do**
4: Odejmij od elementów w wierszu j elementy w wierszu i przemnożone przez $m_{j,i}$
5: b_j -= $b_i * m_{j,i}$
6: **end for**
7: **end for**

Algorithm 2 Ustalenie wektora wynikowego na podstawie macierzy w postaci trójkątnej górnej

1: **for** $i = n, \ldots, 1$ **do**
2: $x_i = b_i / m_{i,i}$
3: **for** $j = i + 1, \ldots, n$ **do**
4: $x_1 - = m_{i,j} / m_{i,i}$
5: **end for**
6: **end for**
7: Powyższe obliczenia stają się znacznie prostsze, gdy elementy na przekątnej macierzy trójkątnej górnej wynoszą 1. Algorytm uzyskiwania wyniku dla macierzy trójkątnej dolnej jest niemal identyczny - wystarczy zmienić polecenia iteracji na $i = 1, \ldots, n$ i $j = 1, \ldots, i - 1$

Algorithm 3 Eliminacja Gaussa z pivotowaniem

1: **for** $i = 1, \ldots, n$ **do**
2: Spośród wierszy $j \in i+1, \ldots, n$ znajdź ten, dla którego $m_{j,i}$ jest największe i niezerowe.
3: Jeśli $j \neq i$ zamień wiersze i oraz j, a także b_i z b_j
4: **for** $j = i+1, \ldots, N$ **do**
5: Odejmij od elementów w wierszu j elementy w wierszu i przemnożone przez $m_{j,i}/m_{i,i}$.
6: b_j -= $b_i * m_{j,i}/m_{i,i}$
7: **end for**
8: **end for**

Algorithm 4 Faktoryzacja LU za pomocą eliminacji Gaussa

1: Zainicjalizuj L jako macierz jednostkową
2: **for** $i = 1, \ldots, n$ **do**
3: **for** $j = i+1, \ldots, N$ **do**
4: $L_{j,i} = m_{j,i}/m_{i,i}$
5: Odejmij od elementów w wierszu j elementy w wierszu i, przemnożone przez $m_{j,i}/m_{i,i}$.
6: **end for**
7: **end for**

Algorithm 5 Faktoryzacja LU za pomocą eliminacji Gaussa z pivotingiem

1: Zainicjalizuj L jako macierz jednostkową
2: **for** $i = 1, \ldots, n$ **do**
3: Spośród wierszy $j \in i+1, \ldots, n$ znajdź ten, dla którego $m_{j,i}$ jest największe i niezerowe.
4: Jeśli $j \neq i$ zamień wiersze i oraz j, a także b_i z b_j
5: **for** $j = i+1, \ldots, N$ **do**
6: $L_{j,i} = m_{j,i}/m_{i,i}$
7: Odejmij od elementów w wierszu j elementy w wierszu i, przemnożone przez $m_{j,i}/m_{i,i}$.
8: **end for**
9: **end for**

1.2 Kod w języku Elixir

```elixir
defmodule GaussMatrix do
  @matrix_size (10 + 2)

  @spec gauss_elimination(matrix :: Matrix.matrix()) :: Matrix.
    matrix()
  def gauss_elimination(matrix) do
    {rows, cols} = Matrix.size(matrix)

    Enum.reduce(0..(rows - 2), matrix, fn i, acc ->
      Enum.reduce((i + 1)..(rows - 1), acc, fn j, acc ->
        factor = Matrix.elem(acc, j, i) / Matrix.elem(acc, i, i)
        update_row(acc, j, i, factor)
      end)
    end)
  end

  def update_row(matrix, row, col, factor) do
    {rows, cols} = Matrix.size(matrix)

    matrix = Enum.reduce(0..(cols - 1), matrix, fn i, acc ->
      old_val = Matrix.elem(acc, row, i)
      subtract_val = Matrix.elem(acc, col, i) * factor
      new_val = old_val - subtract_val
      Matrix.set(acc, row, i, new_val)
    end)

    matrix
  end

  def gauss_elimination_with_pivot(matrix) do
    {rows, cols} = Matrix.size(matrix)

    matrix = Enum.reduce(0..(rows - 2), matrix, fn i, acc ->
      acc = pivot(acc, i)
      Enum.reduce((i + 1)..(rows - 1), acc, fn j, acc ->
        factor = Matrix.elem(acc, j, i) / Matrix.elem(acc, i, i)
        update_row(acc, j, i, factor)
      end)
    end)
    matrix
  end

  def pivot(matrix, col) do
    {rows, cols} = Matrix.size(matrix)

    max_row_index = 0
    max_row_value = 10000

    {max_row_index, max_row_value} = Enum.reduce(col..(rows - 1), {
    max_row_index, max_row_value}, fn i, {max_row_index,
    max_row_value} ->
      if abs(Matrix.elem(matrix, i, col)) > max_row_value do
        {i, abs(Matrix.elem(matrix, i, col))}
      else
        {max_row_index, max_row_value}
```

```
      end
    end)

    if max_row_index != col do
      matrix = Enum.reduce(0..(cols - 1), matrix, fn i, matrix ->
        temp = Matrix.elem(matrix, col, i)
        matrix = Matrix.set(matrix, col, i, Matrix.elem(matrix,
    max_row_index, i))
        matrix = Matrix.set(matrix, max_row_index, i, temp)
        matrix
      end)
      matrix
    end
    matrix
  end

  def random_matrix do
    matrix = Matrix.new(@matrix_size, @matrix_size, 1)

    matrix = for i <- 0..(@matrix_size - 1), reduce: matrix do
      matrix -> for j <- 0..(@matrix_size - 1), reduce: matrix do
        matrix -> Matrix.set(matrix, i, j, :rand.uniform(100))
      end
    end
    matrix
  end

  def result_vector do
    vector = Matrix.new(@matrix_size, 1, 1)

    vector = for i <- 0..(@matrix_size - 1), reduce: vector do
      vector -> Matrix.set(vector, i, 0, :rand.uniform(100))
    end
    vector
  end

  def lu_decomposition(matrix) do
    {rows, _} = Matrix.size(matrix)
    {l, u} = {Matrix.ident(rows), matrix}

    Enum.reduce(0..(rows - 1), {l, u}, fn i, {l, u} ->
      divisor = Matrix.elem(u, i, i)
      if divisor != 0 do
        Enum.reduce(i..(rows - 1), {l, u}, fn j, {l, u} ->
          factor = Matrix.elem(u, j, i) / divisor
          l = update_matrix(l, j, i, factor)
          u = update_matrix(u, j, i, factor)
          {l, u}
        end)
      else
        {l, u}
      end
    end)
  end

  def lu_decomposition_with_pivot(matrix) do
    {rows, _} = Matrix.size(matrix)
```

```
    {l, u, p} = {Matrix.ident(rows), matrix, Matrix.ident(rows)}

    Enum.reduce(0..(rows - 1), {l, u, p}, fn i, {l, u, p} ->
      {l, u, p} = pivot(l, u, p, i)
      Enum.reduce(i..(rows - 1), {l, u, p}, fn j, {l, u, p} ->
        divisor = Matrix.elem(u, i, i)
        if divisor != 0 do
          factor = Matrix.elem(u, j, i) / divisor
          l = update_matrix(l, j, i, factor)
          u = update_matrix(u, j, i, factor)
        end
        {l, u, p}
      end)
    end)
  end

  defp pivot(l, u, p, col) do
    {rows, _} = Matrix.size(u)
    max_row_index = 0
    max_row_value = 10000

    {max_row_index, max_row_value} = Enum.reduce(col..(rows - 1), {
    max_row_index, max_row_value}, fn i, {max_row_index,
    max_row_value} ->
      if abs(Matrix.elem(u, i, col)) > max_row_value do
        {i, abs(Matrix.elem(u, i, col))}
      else
        {max_row_index, max_row_value}
      end
    end)

    if max_row_index != col do
      u = Enum.reduce(0..(rows - 1), u, fn i, u ->
        temp = Matrix.elem(u, col, i)
        u = Matrix.set(u, col, i, Matrix.elem(u, max_row_index, i))
        u = Matrix.set(u, max_row_index, i, temp)
      end)

      l = Enum.reduce(0..(rows - 1), l, fn i, l ->
        temp = Matrix.elem(l, col, i)
        l = Matrix.set(l, col, i, Matrix.elem(l, max_row_index, i))
        l = Matrix.set(l, max_row_index, i, temp)
      end)

      p = Enum.reduce(0..(rows - 1), p, fn i, p ->
        temp = Matrix.elem(p, col, i)
        p = Matrix.set(p, col, i, Matrix.elem(p, max_row_index, i))
        p = Matrix.set(p, max_row_index, i, temp)
      end)

      {l, u, p}
    end

    {l, u, p}
  end

  def update_matrix(matrix, row, col, factor) do
```

```
    {rows, cols} = Matrix.size(matrix)

    Enum.reduce(0..(cols - 1), matrix, fn i, matrix ->
      old_val = Matrix.elem(matrix, row, i)
      subtract_val = Matrix.elem(matrix, col, i) * factor
      new_val = old_val - subtract_val
      Matrix.set(matrix, row, i, new_val)
    end)
  end

  def solve_system_lu(a, b, :stable) do
    {l, u, p} = lu_decomposition_with_pivot(a)
    y = forward_substitution(l, Matrix.mult(p, b))
    x = backward_substitution(u, y)
    x
  end

  def solve_system_lu(a, b, :unstable) do
    {l, u} = lu_decomposition(a)
    y = forward_substitution(l, b)
    x = backward_substitution(u, y)
    x
  end

  def forward_substitution(l, b) do
    {rows, _} = Matrix.size(l)
    y = Matrix.new(rows, 1, 0)

    Enum.reduce(0..(rows - 1), y, fn i, y ->
      y = Matrix.set(y, i, 0, Matrix.elem(b, i, 0) - Enum.reduce
    (0..(i - 1), 0, fn j, acc -> acc + Matrix.elem(l, i, j) *
    Matrix.elem(y, j, 0) end))
      y
    end)
  end

  def backward_substitution(u, y) do
    {rows, _} = Matrix.size(u)
    x = Matrix.new(rows, 1, 0)

    Enum.reduce((rows - 1)..0//-1, x, fn i, x ->
      sum = Enum.reduce(0..(i - 1), 0, fn j, acc ->
        acc + Matrix.elem(u, i, j, 0) * Matrix.elem(x, j, 0, 0)
      end)
      divisor = Matrix.elem(u, i, i)
      x = if divisor != 0 do
        x = Matrix.set(x, i, 0, (Matrix.elem(y, i, 0) - sum) /
    divisor)
        x
      else
        x = Matrix.set(x, i, 0, 0)
        x
      end

      x
    end)
  end
```

```
218
219   def solve_system(a, b, :unstable) do
220     {rows, cols} = Matrix.size(a)
221     augmented_matrix = Matrix.new(rows, cols + 1)
222
223     augmented_matrix = Enum.reduce(0..(rows - 1), augmented_matrix,
       fn i, augmented_matrix ->
224       augmented_matrix = Enum.reduce(0..(cols - 1),
      augmented_matrix, fn j, augmented_matrix ->
225         Matrix.set(augmented_matrix, i, j, Matrix.elem(a, i, j))
226       end)
227       Matrix.set(augmented_matrix, i, cols, Matrix.elem(b, i, 0))
228     end)
229     row_echelon_form = gauss_elimination(augmented_matrix)
230     x = backward_substitution_for_gauss(row_echelon_form)
231     x
232   end
233
234   def backward_substitution_for_gauss(matrix_echelon_form) do
235     {rows, cols} = Matrix.size(matrix_echelon_form)
236     x = Matrix.new(rows, 1, 0)
237
238     Enum.reduce((rows - 1)..0//-1, x, fn i, x ->
239       sum = Enum.reduce((i + 1)..(cols - 2), 0, fn j, acc ->
240         acc + Matrix.elem(matrix_echelon_form, i, j) * Matrix.elem(
      x, j, 0, 0)
241       end)
242       x = Matrix.set(x, i, 0, (Matrix.elem(matrix_echelon_form, i,
      cols - 1) - sum) / Matrix.elem(matrix_echelon_form, i, i))
243       x
244     end)
245   end
246
247   def solve_system(a, b, :stable) do
248     {rows, cols} = Matrix.size(a)
249
250     augmented_matrix = Matrix.new(rows, cols + 1)
251
252
253     augmented_matrix = Enum.reduce(0..(rows - 1), augmented_matrix,
       fn i, augmented_matrix ->
254       augmented_matrix = Enum.reduce(0..(cols - 1),
      augmented_matrix, fn j, augmented_matrix ->
255         Matrix.set(augmented_matrix, i, j, Matrix.elem(a, i, j))
256       end)
257       Matrix.set(augmented_matrix, i, cols, Matrix.elem(b, i, 0))
258     end)
259     row_echelon_form = gauss_elimination_with_pivot(
      augmented_matrix)
260     x = backward_substitution_for_gauss(row_echelon_form)
261     x
262   end
263 end
264
265 defmodule GaussMatrixComputations do
266   def one do
267     IO.inspect("ZAD 1")
```

```
268     matrix = GaussMatrix.random_matrix()
269     vector = GaussMatrix.result_vector()
270     Matrix.pretty_print(matrix)
271     Matrix.pretty_print(vector)
272     Matrix.pretty_print(GaussMatrix.solve_system(matrix, vector, :
        unstable))
273   end
274
275   def two do
276     IO.inspect("ZAD 2")
277     matrix = GaussMatrix.random_matrix()
278     vector = GaussMatrix.result_vector()
279     Matrix.pretty_print(matrix)
280     Matrix.pretty_print(vector)
281     Matrix.pretty_print(GaussMatrix.solve_system(matrix, vector, :
        stable))
282   end
283
284   def three do
285     IO.inspect("ZAD 3")
286     matrix = GaussMatrix.random_matrix()
287     vector = GaussMatrix.result_vector()
288     Matrix.pretty_print(matrix)
289     Matrix.pretty_print(vector)
290     Matrix.pretty_print(GaussMatrix.solve_system_lu(matrix, vector,
         :unstable))
291   end
292
293   def four do
294     IO.inspect("ZAD 4")
295     matrix = GaussMatrix.random_matrix()
296     vector = GaussMatrix.result_vector()
297     Matrix.pretty_print(matrix)
298     Matrix.pretty_print(vector)
299     Matrix.pretty_print(GaussMatrix.solve_system_lu(matrix, vector,
         :stable))
300   end
301 end
302
303 GaussMatrixComputations.one()
304 GaussMatrixComputations.two()
305 GaussMatrixComputations.three()
306 GaussMatrixComputations.four()
```

1.3 Kod w języku Matlab

```
function zad2()
    size  = 10+2;
    m1 = [[45 95 99 80 6 55 77 50 31 77 81 32];
[80 20 42 70 47 25 7 13 30 49 59 53];
[8 77 48 4 86 87 90 43 50 65 65 39];
[64 64 23 5 38 6 58 96 87 12 67 70];
[74 81 9 19 46 82 100 29 74 25 2 53];
[21 76 66 21 54 6 69 41 76 81 59 90];
[96 100 16 98 50 97 37 92 35 26 2 93];
[53 81 95 87 22 31 47 83 1 52 18 82];
[32 65 79 92 18 16 8 66 10 20 79 70];
[24 18 47 78 47 89 7 59 70 88 76 98];
[42 60 35 69 56 74 75 54 15 84 38 5];
[24 65 97 49 80 27 58 52 58 35 82 14]];

    res1 = [21, 21, 93, 35, 93, 6, 93, 46, 42, 20, 48, 44];

    m2 = [[14 77 35 43 69 11 53 87 18 57 34 51];
[57 37 8 1 73 100 5 78 84 12 21 35];
[34 73 19 56 67 43 28 41 63 95 20 54];
[63 44 87 88 29 31 54 9 36 57 49 100];
[93 28 42 27 43 6 63 10 14 69 16 80];
[90 13 92 23 98 55 19 92 27 73 28 65];
[81 37 57 10 65 4 71 1 16 38 86 92];
[10 63 67 53 9 25 85 28 27 39 65 25];
[54 47 23 99 81 1 76 37 83 63 95 62];
[46 89 23 1 56 54 72 46 55 24 97 20];
[75 66 49 37 84 69 60 50 47 31 35 9];
[56 95 72 5 78 62 78 19 38 32 45 16]];

    res2 = [80, 98, 83, 79, 52, 68, 3, 44, 51, 36, 3, 74];

    m3 = [[64 92 1 5 89 15 51 55 43 76 2 51];
[22 20 83 1 16 30 16 5 80 6 76 63];
[24 43 35 35 59 73 46 16 77 25 27 33];
[97 98 19 71 16 66 19 46 83 55 94 45];
[62 85 1 21 18 1 80 91 29 34 43 45];
[46 62 1 98 25 98 14 27 71 86 74 73];
[98 97 14 31 66 61 45 92 68 82 63 48];
[59 58 92 54 27 87 61 12 12 82 21 76];
[66 34 64 94 73 7 28 61 67 94 42 56];
[79 2 60 45 82 84 37 96 74 16 15 77];
[28 91 91 88 36 51 84 93 77 66 86 69];
[82 68 30 63 98 19 100 61 26 40 58 3]];

    res3 = [81, 2, 48, 85 ,95 ,32 ,96 ,56 ,27 ,75, 84, 89];

    m4 = [[27 55 34 66 99 89 15 94 71 93 33 85];
[75 56 85 62 57 78 7 2 22 8 62 26];
[100 52 40 46 15 54 43 63 25 95 62 5];
[69 83 73 66 52 25 93 54 15 47 13 88];
[71 13 29 62 28 32 18 57 81 56 5 15];
[94 56 68 60 50 59 1 90 98 2 7 62];
[41 71 79 31 40 21 79 6 22 66 95 29];
[21 50 61 53 56 92 70 40 14 41 39 89];
```

```
[7 37 25 5 81 76 95 28 55 52 35 52];
[54 100 68 7 9 3 100 50 77 51 89 20];
[51 61 8 11 54 44 44 77 15 56 17 31];
[35 66 87 64 10 6 22 87 54 69 49 84]];

    res4 = [75, 2, 22, 57, 58, 97, 9, 59, 12, 46, 56, 14];

    matrix_mem = randi([1,100], size, size);
    res_mem = randi([1,100], size, 1);

    matrix = matrix_mem;
    res = res_mem;
    %disp(matrix)
    %disp(res)
    disp(m1)
    disp(res1)
    disp("-------------------------------------------")
    sol = solve_with_gauss_elimination(m1, res1);
    disp(sol)
    %disp(matrix_mem*sol)

    disp("-------------------------------------------")
    matrix = matrix_mem;
    res = res_mem;
        disp(m2)
    disp(res2)
    [sol, inv] = solve_with_gauss_elimination_pivoting(m2, res2);
    matrix = matrix_mem;
    disp(sol)
    %check_inv(matrix,sol,inv)

    disp("-------------------------------------------")
    matrix = matrix_mem;
    res = res_mem;
        disp(m3)
    disp(res3)
    sol = solve_with_LU(m3, res3);
    disp(sol)
    %disp(matrix_mem*sol)

    disp("-------------------------------------------")
    matrix = matrix_mem;
    res = res_mem;
        disp(m4)
    disp(res4)
    [sol, inv] = solve_with_LU_pivoting(m4, res4);
    matrix = matrix_mem;
    %disp(matrix_mem)
    disp(sol)
    %check_inv(matrix, sol, inv)
end

function check_inv(M, x, inv)
    n = length(inv);
    for i = 1:n
        if inv(i) ~= 0
            M([i, inv(i)],:) =  M([inv(i), i],:);
```

```
        end
    end
    disp(M*x)
end
function x = solve_with_gauss_elimination(M, a)
    n = size(M, 1);
    x = zeros(n, 1);

    for stage = 1:n
        a(stage) = a(stage) / M(stage,stage);
        for col = n:-1:stage
            M(stage,col) = M(stage,col) / M(stage,stage);
        end
        for row = stage+1:n
            lead = M(row, stage);
            a(row) = a(row) - lead * a(stage);
            for col = stage:n
                M(row,col) = M(row, col) - lead * M(stage,col);
            end

        end
        %disp(M);
        %disp(a);
    end
    %disp(M)
    for row = n:-1:1
        sum = a(row);
        for col = row+1:n
            sum = sum - M(row,col) * x(col);
        end
        x(row) = sum;
    end
end

function [x, inv] = solve_with_gauss_elimination_pivoting(M, a)
    n = size(M, 1);
    x = zeros(n, 1);
    inv = zeros(n,1);

    for stage = 1:n
        [~, row_max] = max(M(stage:n,stage));
        row_max = row_max + stage - 1;
        if row_max ~= stage
            M([stage, row_max],:) = M([row_max,stage],:);
            a([stage, row_max]) = a([row_max,stage]);
            inv(stage) = row_max;
        end
        if M(stage,stage) == 0
            disp("error");
            return
        end
        main_lead = M(stage,stage);
        for row = stage+1:n
            lead = M(row, stage);
            a(row) = a(row) - lead * a(stage) / main_lead;
            for col = stage:n
                M(row,col) = M(row, col) - lead * M(stage,col) /
```

```
    main_lead;
170             end
171
172         end
173         %disp(M);
174         %disp(a);
175     end
176     %disp(M)
177     for row = n:-1:1
178         sum = a(row) / M(row,row);
179         for col = row+1:n
180             sum = sum - M(row,col) * x(col) / M(row,row);
181         end
182         x(row) = sum;
183     end
184 end
185
186 function x = solve_with_LU(M, a)
187     n = size(M, 1);
188     x = zeros(n, 1);
189     z = zeros(n, 1);
190     L = diag(ones(n,1));
191     %disp(M)
192     for stage = 1:n
193         for row = stage+1:n
194             L(row,stage) = M(row, stage)/M(stage,stage);
195             for col = stage:n
196                 M(row,col) = M(row, col) - L(row,stage) .* M(stage,
    col);
197             end
198         end
199         %disp(M);
200         %disp(a);
201     end
202     %disp(M)
203     %disp(L)
204     for row = 1:n
205         sum = a(row);
206         for col = 1:row-1
207             sum = sum - L(row,col) * z(col);
208         end
209         z(row) = sum;
210     end
211     for row = n:-1:1
212         sum = z(row) / M(row, row);
213         for col = row+1:n
214             sum = sum - M(row,col) * x(col) / M(row,row);
215         end
216         x(row) = sum;
217     end
218 end
219
220 function [x,inv] = solve_with_LU_pivoting(M, a)
221     n = size(M, 1);
222     x = zeros(n, 1);
223     z = zeros(n, 1);
224     L = diag(ones(n,1));
```

```
    inv = zeros(n,1);

    for stage = 1:n
        [~, row_max] = max(M(stage:n,stage));
        row_max = row_max + stage - 1;
        if row_max ~= stage
            M([stage, row_max],:) = M([row_max,stage],:);
            inv(stage) = row_max;
        end
        if M(stage,stage) == 0
            disp("error");
            return
        end

        for row = stage+1:n
            L(row,stage) = M(row, stage)/M(stage,stage);
            for col = stage:n
                M(row,col) = M(row, col) - L(row,stage) * M(stage,
    col);
            end
        end
        %disp(M);
        %disp(a);
    end
    %disp(M)
    %disp(L)
    %disp(L*M)
    %disp(a)
    for row = 1:n
        sum = a(row);
        for col = 1:row-1
            sum = sum - L(row,col) * z(col);
        end
        z(row) = sum;
    end
    for row = n:-1:1
        sum = z(row) / M(row, row);
        for col = row+1:n
            sum = sum - M(row,col) * x(col) / M(row,row);
        end
        x(row) = sum;
    end
end
```

2 Wyniki

Poniżej przedstawiono wyniki obliczeń. Na kążdym zdjęciu widać kolejno macierz układu równań, prawą stronę układu oraz wektor rozwiązań. Każda para zdjęć zawiera wyniki dla innego algorytmu.

3 Wnioski

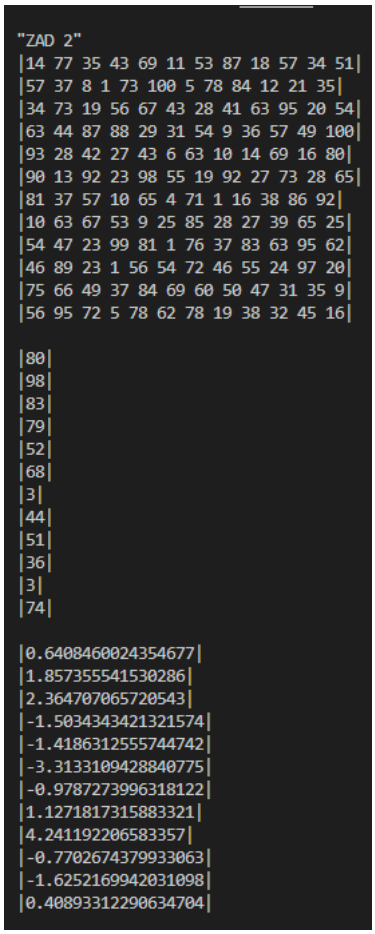
- Wyniki dla eliminacji Gaussa są identyczne z dokładnością do zaokrąglenia
- Wyniki faktoryzacji są zbliżone, jednak nie identyczne. Przyczyny mogą być różne - od arytmetyki zmiennoprzecinkowej, po różnice w specyfice języków programowania i wynikające z tego różniących się implementacje algorytmów.
- Wylosowanie macierzy nie ma wpływu na rozbieżności. Błędy wydają się być systematyczne.
- Za wyjątek od powyższego można by uznać macierz, gdzie na przykład dwa pierwsze wiersze są identycze. Wtedy algorytmy, szybko wykryją nierozwiązywalność.

```
"ZAD 1"
|45 95 99 80 6 55 77 50 31 77 81 32|
|80 20 42 70 47 25 7 13 30 49 59 53|
|8 77 48 4 86 87 90 43 50 65 65 39|
|64 64 23 5 38 6 58 96 87 12 67 70|
|74 81 9 19 46 82 100 29 74 25 2 53|
|21 76 66 21 54 6 69 41 76 81 59 90|
|96 100 16 98 50 97 37 92 35 26 2 93|
|53 81 95 87 22 31 47 83 1 52 18 82|
|32 65 79 92 18 16 8 66 10 20 79 70|
|24 18 47 78 47 89 7 59 70 88 76 98|
|42 60 35 69 56 74 75 54 15 84 38 5|
|24 65 97 49 80 27 58 52 58 35 82 14|

|21|
|21|
|93|
|35|
|93|
|6|
|93|
|46|
|42|
|20|
|48|
|44|

|-0.026605766395043653|
|-0.2554292348226004|
|-0.094312200675379 8|
|0.12703117118649448|
|0.545895857397 8491|
|0.548380500553 1015|
|0.9139869102156305|
|-0.0819570652490039|
|-0.6954923680412768|
|-0.9408876820374653|
|0.2735625723029889|
|0.5553767512029999|
```

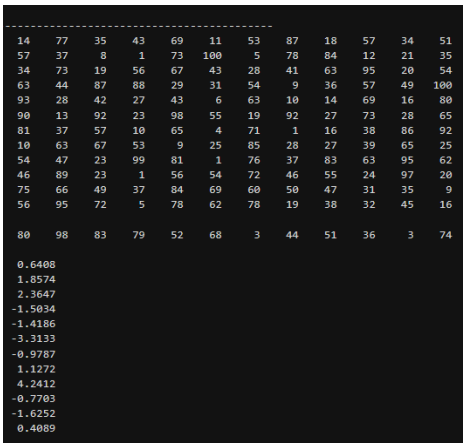
Rysunek 1: Wyniki programu w Elixir dla eliminacji Gaussa

```
   45    95    99    80     6    55    77    50    31    77    81    32
   80    20    42    70    47    25     7    13    30    49    59    53
    8    77    48     4    86    87    90    43    50    65    65    39
   64    64    23     5    38     6    58    96    87    12    67    70
   74    81     9    19    46    82   100    29    74    25     2    53
   21    76    66    21    54     6    69    41    76    81    59    90
   96   100    16    98    50    97    37    92    35    26     2    93
   53    81    95    87    22    31    47    83     1    52    18    82
   32    65    79    92    18    16     8    66    10    20    79    70
   24    18    47    78    47    89     7    59    70    88    76    98
   42    60    35    69    56    74    75    54    15    84    38     5
   24    65    97    49    80    27    58    52    58    35    82    14

   21    21    93    35    93     6    93    46    42    20    48    44

----------------------------------------

 -0.0266
 -0.2554
 -0.0943
  0.1270
  0.5459
  0.5484
  0.9140
 -0.0820
 -0.6955
 -0.9409
  0.2736
  0.5554

----------------------------------------
```

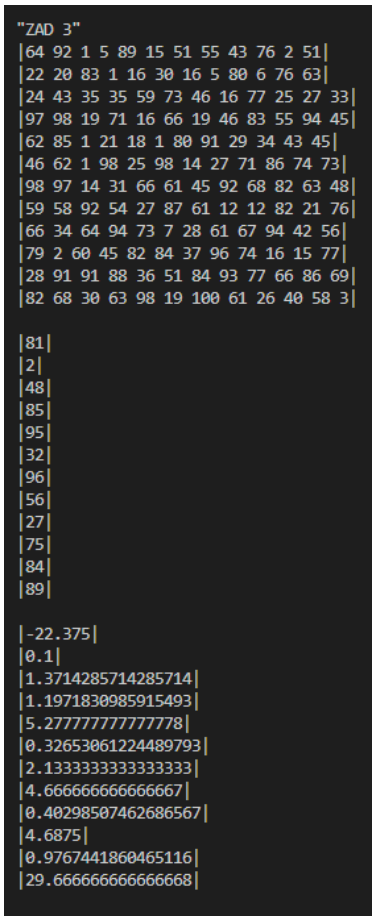
Rysunek 2: Wyniki programu w Matlab dla eliminacji Gaussa



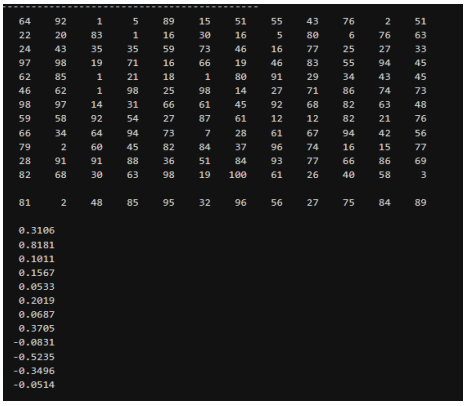
Rysunek 3: Wyniki programu w Elixir dla eliminacji Gaussa z pivitingiem



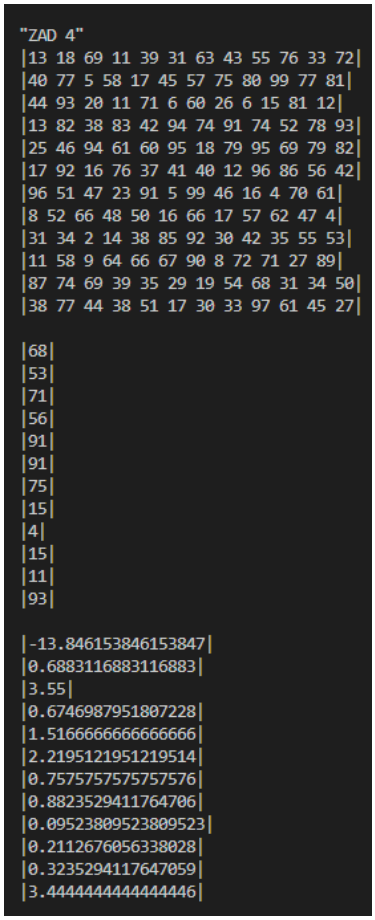
Rysunek 4: Wyniki programu w Matlab dla eliminacji Gaussa z pivitingiem



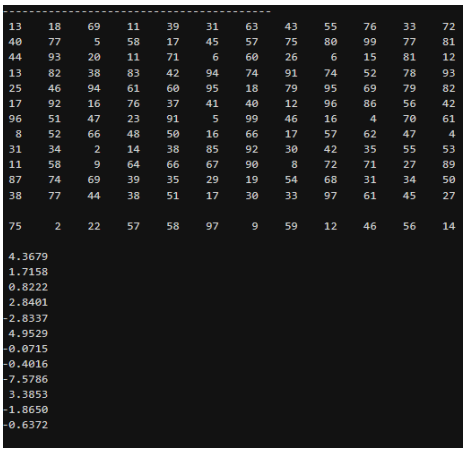
Rysunek 5: Wyniki programu w Elixir dla faktoryzacji LU



Rysunek 6: Wyniki programu w Matlab dla faktoryzacji LU



Rysunek 7: Wyniki programu w Elixir dla faktoryzacji LU z pivotingiem



Rysunek 8: Wyniki programu w Matlab dla faktoryzacji LU z pivotingiem