

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

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

```
1: for  $i = n, \dots, 1$  do  
2:    $x_i = b_i / m_{i,i}$   
3:   for  $j = i + 1, \dots, n$  do  
4:      $x_1 \leftarrow x_1 - m_{i,j} / m_{i,i} * x_i$   
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  $i = 1, \dots, n$  i  $j = 1, \dots, i - 1$ 
```

Algorithm 3 Eliminacja Gaussa z pivotowaniem

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

Algorithm 4 Faktoryzacja LU za pomocą eliminacji Gaussa

```
1: Zainicjalizuj  $L$  jako macierz jednostkową
2: for  $i = 1, \dots, n$  do
3:   for  $j = i + 1, \dots, N$  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  $i = 1, \dots, n$  do
3:   Spośród wierszy  $j \in i + 1, \dots, n$  znajdź ten, dla którego  $m_{j,i}$  jest maj-
   większe i niezerowe.
4:   Jeśli  $j \neq i$  zamień wiersze  $i$  oraz  $j$ , a także  $b_i$  z  $b_j$ 
5:   for  $j = i + 1, \dots, N$  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

```
1 defmodule GaussMatrix do
2   @matrix_size (10 + 2)
3
4   @spec gauss_elimination(matrix :: Matrix.matrix()) :: Matrix.
5     matrix()
6   def gauss_elimination(matrix) do
7     {rows, cols} = Matrix.size(matrix)
8
9     Enum.reduce(0..(rows - 2), matrix, fn i, acc ->
10       Enum.reduce((i + 1)..(rows - 1), acc, fn j, acc ->
11         factor = Matrix.elem(acc, j, i) / Matrix.elem(acc, i, i)
12         update_row(acc, j, i, factor)
13       end)
14     end)
15
16   def update_row(matrix, row, col, factor) do
17     {rows, cols} = Matrix.size(matrix)
18
19     matrix = Enum.reduce(0..(cols - 1), matrix, fn i, acc ->
20       old_val = Matrix.elem(acc, row, i)
21       subtract_val = Matrix.elem(acc, col, i) * factor
22       new_val = old_val - subtract_val
23       Matrix.set(acc, row, i, new_val)
24     end)
25
26     matrix
27   end
28
29   def gauss_elimination_with_pivot(matrix) do
30     {rows, cols} = Matrix.size(matrix)
31
32     matrix = Enum.reduce(0..(rows - 2), matrix, fn i, acc ->
33       acc = pivot(acc, i)
34       Enum.reduce((i + 1)..(rows - 1), acc, fn j, acc ->
35         factor = Matrix.elem(acc, j, i) / Matrix.elem(acc, i, i)
36         update_row(acc, j, i, factor)
37       end)
38     end)
39     matrix
40   end
41
42   def pivot(matrix, col) do
43     {rows, cols} = Matrix.size(matrix)
44
45     max_row_index = 0
46     max_row_value = 10000
47
48     {max_row_index, max_row_value} = Enum.reduce(col..(rows - 1), {
49       max_row_index, max_row_value}, fn i, {max_row_index,
50       max_row_value} ->
51       if abs(Matrix.elem(matrix, i, col)) > max_row_value do
52         {i, abs(Matrix.elem(matrix, i, col))}
53       else
54         {max_row_index, max_row_value}
55       end
56     end)
```

```

53     end
54 end)
55
56 if max_row_index != col do
57     matrix = Enum.reduce(0..(cols - 1), matrix, fn i, matrix ->
58         temp = Matrix.elem(matrix, col, i)
59         matrix = Matrix.set(matrix, col, i, Matrix.elem(matrix,
max_row_index, i))
60         matrix = Matrix.set(matrix, max_row_index, i, temp)
61         matrix
62     end)
63     matrix
64 end
65 matrix
66 end
67
68 def random_matrix do
69     matrix = Matrix.new(@matrix_size, @matrix_size, 1)
70
71     matrix = for i <- 0..(@matrix_size - 1), reduce: matrix do
72         matrix -> for j <- 0..(@matrix_size - 1), reduce: matrix do
73             matrix -> Matrix.set(matrix, i, j, :rand.uniform(100))
74         end
75     end
76     matrix
77 end
78
79 def result_vector do
80     vector = Matrix.new(@matrix_size, 1, 1)
81
82     vector = for i <- 0..(@matrix_size - 1), reduce: vector do
83         vector -> Matrix.set(vector, i, 0, :rand.uniform(100))
84     end
85     vector
86 end
87
88 def lu_decomposition(matrix) do
89     {rows, _} = Matrix.size(matrix)
90     {l, u} = {Matrix.ident(rows), matrix}
91
92     Enum.reduce(0..(rows - 1), {l, u}, fn i, {l, u} ->
93         divisor = Matrix.elem(u, i, i)
94         if divisor != 0 do
95             Enum.reduce(i..(rows - 1), {l, u}, fn j, {l, u} ->
96                 factor = Matrix.elem(u, j, i) / divisor
97                 l = update_matrix(l, j, i, factor)
98                 u = update_matrix(u, j, i, factor)
99                 {l, u}
100             end)
101         else
102             {l, u}
103         end
104     end)
105 end
106
107 def lu_decomposition_with_pivot(matrix) do
108     {rows, _} = Matrix.size(matrix)

```

```

109 {l, u, p} = {Matrix.ident(rows), matrix, Matrix.ident(rows)}
110
111 Enum.reduce(0..(rows - 1), {l, u, p}, fn i, {l, u, p} ->
112   {l, u, p} = pivot(l, u, p, i)
113   Enum.reduce(i..(rows - 1), {l, u, p}, fn j, {l, u, p} ->
114     divisor = Matrix.elem(u, i, i)
115     if divisor != 0 do
116       factor = Matrix.elem(u, j, i) / divisor
117       l = update_matrix(l, j, i, factor)
118       u = update_matrix(u, j, i, factor)
119     end
120     {l, u, p}
121   end)
122 end)
123 end
124
125 defp pivot(l, u, p, col) do
126   {rows, _} = Matrix.size(u)
127   max_row_index = 0
128   max_row_value = 10000
129
130   {max_row_index, max_row_value} = Enum.reduce(col..(rows - 1), {
131     max_row_index, max_row_value}, fn i, {max_row_index,
132     max_row_value} ->
133     if abs(Matrix.elem(u, i, col)) > max_row_value do
134       {i, abs(Matrix.elem(u, i, col))}
135     else
136       {max_row_index, max_row_value}
137     end
138   end)
139
140   if max_row_index != col do
141     u = Enum.reduce(0..(rows - 1), u, fn i, u ->
142       temp = Matrix.elem(u, col, i)
143       u = Matrix.set(u, col, i, Matrix.elem(u, max_row_index, i))
144       u = Matrix.set(u, max_row_index, i, temp)
145     end)
146
147     l = Enum.reduce(0..(rows - 1), l, fn i, l ->
148       temp = Matrix.elem(l, col, i)
149       l = Matrix.set(l, col, i, Matrix.elem(l, max_row_index, i))
150       l = Matrix.set(l, max_row_index, i, temp)
151     end)
152
153     p = Enum.reduce(0..(rows - 1), p, fn i, p ->
154       temp = Matrix.elem(p, col, i)
155       p = Matrix.set(p, col, i, Matrix.elem(p, max_row_index, i))
156       p = Matrix.set(p, max_row_index, i, temp)
157     end)
158
159     {l, u, p}
160   end
161 end
162
163 def update_matrix(matrix, row, col, factor) do

```

```

164 {rows, cols} = Matrix.size(matrix)
165
166 Enum.reduce(0..(cols - 1), matrix, fn i, matrix ->
167     old_val = Matrix.elem(matrix, row, i)
168     subtract_val = Matrix.elem(matrix, col, i) * factor
169     new_val = old_val - subtract_val
170     Matrix.set(matrix, row, i, new_val)
171 end)
172 end
173
174 def solve_system_lu(a, b, :stable) do
175     {l, u, p} = lu_decomposition_with_pivot(a)
176     y = forward_substitution(l, Matrix.mult(p, b))
177     x = backward_substitution(u, y)
178     x
179 end
180
181 def solve_system_lu(a, b, :unstable) do
182     {l, u} = lu_decomposition(a)
183     y = forward_substitution(l, b)
184     x = backward_substitution(u, y)
185     x
186 end
187
188 def forward_substitution(l, b) do
189     {rows, _} = Matrix.size(l)
190     y = Matrix.new(rows, 1, 0)
191
192     Enum.reduce(0..(rows - 1), y, fn i, y ->
193         y = Matrix.set(y, i, 0, Matrix.elem(b, i, 0) - Enum.reduce(
194             0..(i - 1), 0, fn j, acc -> acc + Matrix.elem(l, i, j) *
195             Matrix.elem(y, j, 0) end))
196         y
197     end)
198 end
199
200 def backward_substitution(u, y) do
201     {rows, _} = Matrix.size(u)
202     x = Matrix.new(rows, 1, 0)
203
204     Enum.reduce((rows - 1)..0//-1, x, fn i, x ->
205         sum = Enum.reduce(0..(i - 1), 0, fn j, acc ->
206             acc + Matrix.elem(u, i, j, 0) * Matrix.elem(x, j, 0, 0)
207         end)
208         divisor = Matrix.elem(u, i, i)
209         x = if divisor != 0 do
210             x = Matrix.set(x, i, 0, (Matrix.elem(y, i, 0) - sum) /
211                 divisor)
212             x
213         else
214             x = Matrix.set(x, i, 0, 0)
215             x
216         end
217     end)
218 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,
224     fn i, augmented_matrix ->
225     augmented_matrix = Enum.reduce(0..(cols - 1),
226       augmented_matrix, fn j, augmented_matrix ->
227         Matrix.set(augmented_matrix, i, j, Matrix.elem(a, i, j))
228       end)
229     Matrix.set(augmented_matrix, i, cols, Matrix.elem(b, i, 0))
230   end)
231   row_echelon_form = gauss_elimination(augmented_matrix)
232   x = backward_substitution_for_gauss(row_echelon_form)
233   x
234 end
235
236 def backward_substitution_for_gauss(matrix_echelon_form) do
237   {rows, cols} = Matrix.size(matrix_echelon_form)
238   x = Matrix.new(rows, 1, 0)
239
240   Enum.reduce((rows - 1)..0//-1, x, fn i, x ->
241     sum = Enum.reduce((i + 1)..(cols - 2), 0, fn j, acc ->
242       acc + Matrix.elem(matrix_echelon_form, i, j) * Matrix.elem(
243         x, j, 0, 0)
244     end)
245     x = Matrix.set(x, i, 0, (Matrix.elem(matrix_echelon_form, i,
246       cols - 1) - sum) / Matrix.elem(matrix_echelon_form, i, i))
247   end)
248   x
249 end
250
251 def solve_system(a, b, :stable) do
252   {rows, cols} = Matrix.size(a)
253   augmented_matrix = Matrix.new(rows, cols + 1)
254
255   augmented_matrix = Enum.reduce(0..(rows - 1), augmented_matrix,
256     fn i, augmented_matrix ->
257     augmented_matrix = Enum.reduce(0..(cols - 1),
258       augmented_matrix, fn j, augmented_matrix ->
259         Matrix.set(augmented_matrix, i, j, Matrix.elem(a, i, j))
260       end)
261     Matrix.set(augmented_matrix, i, cols, Matrix.elem(b, i, 0))
262   end)
263   row_echelon_form = gauss_elimination_with_pivot(
264     augmented_matrix)
265   x = backward_substitution_for_gauss(row_echelon_form)
266   x
267 end
268
269 defmodule GaussMatrixComputations do
270   def one do
271     IO.inspect("ZAD 1")
272   end
273 end

```

```

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, :
273         unstable))
274     end
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, :
282             stable))
283     end
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,
291             :unstable))
292     end
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,
300             :stable))
301     end
302 end
303 GaussMatrixComputations.one()
304 GaussMatrixComputations.two()
305 GaussMatrixComputations.three()
306 GaussMatrixComputations.four()

```


1.3 Kod w języku Matlab

```
1 function zad2()
2     size = 10+2;
3     m1 = [[45 95 99 80 6 55 77 50 31 77 81 32];
4 [80 20 42 70 47 25 7 13 30 49 59 53];
5 [8 77 48 4 86 87 90 43 50 65 65 39];
6 [64 64 23 5 38 6 58 96 87 12 67 70];
7 [74 81 9 19 46 82 100 29 74 25 2 53];
8 [21 76 66 21 54 6 69 41 76 81 59 90];
9 [96 100 16 98 50 97 37 92 35 26 2 93];
10 [53 81 95 87 22 31 47 83 1 52 18 82];
11 [32 65 79 92 18 16 8 66 10 20 79 70];
12 [24 18 47 78 47 89 7 59 70 88 76 98];
13 [42 60 35 69 56 74 75 54 15 84 38 5];
14 [24 65 97 49 80 27 58 52 58 35 82 14]];
15
16     res1 = [21, 21, 93, 35, 93, 6, 93, 46, 42, 20, 48, 44];
17
18     m2 = [[14 77 35 43 69 11 53 87 18 57 34 51];
19 [57 37 8 1 73 100 5 78 84 12 21 35];
20 [34 73 19 56 67 43 28 41 63 95 20 54];
21 [63 44 87 88 29 31 54 9 36 57 49 100];
22 [93 28 42 27 43 6 63 10 14 69 16 80];
23 [90 13 92 23 98 55 19 92 27 73 28 65];
24 [81 37 57 10 65 4 71 1 16 38 86 92];
25 [10 63 67 53 9 25 85 28 27 39 65 25];
26 [54 47 23 99 81 1 76 37 83 63 95 62];
27 [46 89 23 1 56 54 72 46 55 24 97 20];
28 [75 66 49 37 84 69 60 50 47 31 35 9];
29 [56 95 72 5 78 62 78 19 38 32 45 16]];
30
31     res2 = [80, 98, 83, 79, 52, 68, 3, 44, 51, 36, 3, 74];
32
33     m3 = [[64 92 1 5 89 15 51 55 43 76 2 51];
34 [22 20 83 1 16 30 16 5 80 6 76 63];
35 [24 43 35 35 59 73 46 16 77 25 27 33];
36 [97 98 19 71 16 66 19 46 83 55 94 45];
37 [62 85 1 21 18 1 80 91 29 34 43 45];
38 [46 62 1 98 25 98 14 27 71 86 74 73];
39 [98 97 14 31 66 61 45 92 68 82 63 48];
40 [59 58 92 54 27 87 61 12 12 82 21 76];
41 [66 34 64 94 73 7 28 61 67 94 42 56];
42 [79 2 60 45 82 84 37 96 74 16 15 77];
43 [28 91 91 88 36 51 84 93 77 66 86 69];
44 [82 68 30 63 98 19 100 61 26 40 58 3]];
45
46     res3 = [81, 2, 48, 85 ,95 ,32 ,96 ,56 ,27 ,75, 84, 89];
47
48     m4 = [[27 55 34 66 99 89 15 94 71 93 33 85];
49 [75 56 85 62 57 78 7 2 22 8 62 26];
50 [100 52 40 46 15 54 43 63 25 95 62 5];
51 [69 83 73 66 52 25 93 54 15 47 13 88];
52 [71 13 29 62 28 32 18 57 81 56 5 15];
53 [94 56 68 60 50 59 1 90 98 2 7 62];
54 [41 71 79 31 40 21 79 6 22 66 95 29];
55 [21 50 61 53 56 92 70 40 14 41 39 89];
```

```

56 [7 37 25 5 81 76 95 28 55 52 35 52];
57 [54 100 68 7 9 3 100 50 77 51 89 20];
58 [51 61 8 11 54 44 44 77 15 56 17 31];
59 [35 66 87 64 10 6 22 87 54 69 49 84]];
60
61     res4 = [75, 2, 22, 57, 58, 97, 9, 59, 12, 46, 56, 14];
62
63     matrix_mem = randi([1,100], size, size);
64     res_mem = randi([1,100], size, 1);
65
66     matrix = matrix_mem;
67     res = res_mem;
68     %disp(matrix)
69     %disp(res)
70     disp(m1)
71     disp(res1)
72     disp("-----")
73     sol = solve_with_gauss_elimination(m1, res1);
74     disp(sol)
75     %disp(matrix_mem*sol)
76
77     disp("-----")
78     matrix = matrix_mem;
79     res = res_mem;
80     disp(m2)
81     disp(res2)
82     [sol, inv] = solve_with_gauss_elimination_pivoting(m2, res2);
83     matrix = matrix_mem;
84     disp(sol)
85     %check_inv(matrix, sol, inv)
86
87     disp("-----")
88     matrix = matrix_mem;
89     res = res_mem;
90     disp(m3)
91     disp(res3)
92     sol = solve_with_LU(m3, res3);
93     disp(sol)
94     %disp(matrix_mem*sol)
95
96     disp("-----")
97     matrix = matrix_mem;
98     res = res_mem;
99     disp(m4)
100    disp(res4)
101    [sol, inv] = solve_with_LU_pivoting(m4, res4);
102    matrix = matrix_mem;
103    %disp(matrix_mem)
104    disp(sol)
105    %check_inv(matrix, sol, inv)
106 end
107
108 function check_inv(M, x, inv)
109     n = length(inv);
110     for i = 1:n
111         if inv(i) ~= 0
112             M([i, inv(i)], :) = M([inv(i), i], :);

```

```

113         end
114     end
115     disp(M*x)
116 end
117 function x = solve_with_gauss_elimination(M, a)
118     n = size(M, 1);
119     x = zeros(n, 1);
120
121     for stage = 1:n
122         a(stage) = a(stage) / M(stage,stage);
123         for col = n:-1:stage
124             M(stage,col) = M(stage,col) / M(stage,stage);
125         end
126         for row = stage+1:n
127             lead = M(row, stage);
128             a(row) = a(row) - lead * a(stage);
129             for col = stage:n
130                 M(row,col) = M(row, col) - lead * M(stage,col);
131             end
132         end
133         %disp(M);
134         %disp(a);
135     end
136     %disp(M)
137     for row = n:-1:1
138         sum = a(row);
139         for col = row+1:n
140             sum = sum - M(row,col) * x(col);
141         end
142         x(row) = sum;
143     end
144 end
145
146 function [x, inv] = solve_with_gauss_elimination_pivoting(M, a)
147     n = size(M, 1);
148     x = zeros(n, 1);
149     inv = zeros(n,1);
150
151     for stage = 1:n
152         [~, row_max] = max(M(stage:n,stage));
153         row_max = row_max + stage - 1;
154         if row_max ~= stage
155             M([stage, row_max],:) = M([row_max,stage],:);
156             a([stage, row_max]) = a([row_max,stage]);
157             inv(stage) = row_max;
158         end
159         if M(stage,stage) == 0
160             disp("error");
161             return
162         end
163         main_lead = M(stage,stage);
164         for row = stage+1:n
165             lead = M(row, stage);
166             a(row) = a(row) - lead * a(stage) / main_lead;
167             for col = stage:n
168                 M(row,col) = M(row, col) - lead * M(stage,col) /

```

```

170     main_lead;
171         end
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,
197 col);
198             end
199             end
200             %disp(M);
201             %disp(a);
202         end
203         %disp(M)
204         %disp(L)
205         for row = 1:n
206             sum = a(row);
207             for col = 1:row-1
208                 sum = sum - L(row,col) * z(col);
209             end
210             z(row) = sum;
211         end
212         for row = n:-1:1
213             sum = z(row) / M(row, row);
214             for col = row+1:n
215                 sum = sum - M(row,col) * x(col) / M(row,row);
216             end
217             x(row) = sum;
218         end
219     end
220
221 function [x,inv] = solve_with_LU_pivoting(M, a)
222     n = size(M, 1);
223     x = zeros(n, 1);
224     z = zeros(n, 1);
225     L = diag(ones(n,1));

```

```

225     inv = zeros(n,1);
226
227     for stage = 1:n
228         [~, row_max] = max(M(stage:n,stage));
229         row_max = row_max + stage - 1;
230         if row_max ~= stage
231             M([stage, row_max],:) = M([row_max,stage],:);
232             inv(stage) = row_max;
233         end
234         if M(stage,stage) == 0
235             disp("error");
236             return
237         end
238
239         for row = stage+1:n
240             L(row,stage) = M(row, stage)/M(stage,stage);
241             for col = stage:n
242                 M(row,col) = M(row, col) - L(row,stage) * M(stage,
243 col);
244             end
245             %disp(M);
246             %disp(a);
247         end
248         %disp(M)
249         %disp(L)
250         %disp(L*M)
251         %disp(a)
252         for row = 1:n
253             sum = a(row);
254             for col = 1:row-1
255                 sum = sum - L(row,col) * z(col);
256             end
257             z(row) = sum;
258         end
259         for row = n:-1:1
260             sum = z(row) / M(row, row);
261             for col = row+1:n
262                 sum = sum - M(row,col) * x(col) / M(row,row);
263             end
264             x(row) = sum;
265         end
266     end

```

2 Wyniki

Poniżej przedstawiono wyniki obliczeń. Na każdym 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 identyczne. 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|
|-0.25542923482226004|
|-0.0943122006753798|
|0.12703117118649448|
|0.5458958573978491|
|0.5483805005531015|
|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

```

"ZAD 2"
|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|

|80|
|98|
|83|
|79|
|52|
|68|
|3|
|44|
|51|
|36|
|3|
|74|

|0.6408460024354677|
|1.857355541530286|
|2.364707065720543|
|-1.5034343421321574|
|-1.4186312555744742|
|-3.3133109428840775|
|-0.9787273996318122|
|1.1271817315883321|
|4.241192206583357|
|-0.7702674379933063|
|-1.6252169942031098|
|0.40893312290634704|

```

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

```

-----
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

80 98 83 79 52 68 3 44 51 36 3 74

0.6408
1.8574
2.3647
-1.5034
-1.4186
-3.3133
-0.9787
1.1272
4.2412
-0.7703
-1.6252
0.4089

```

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


```

"ZAD 3"
|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|

|81|
|2|
|48|
|85|
|95|
|32|
|96|
|56|
|27|
|75|
|84|
|89|

|-22.375|
|0.1|
|1.3714285714285714|
|1.1971830985915493|
|5.277777777777778|
|0.32653061224489793|
|2.1333333333333333|
|4.666666666666667|
|0.40298507462686567|
|4.6875|
|0.9767441860465116|
|29.666666666666668|

```

Rysunek 5: Wyniki programu w Elixir dla faktoryzacji LU

```

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

81 2 48 85 95 32 96 56 27 75 84 89

0.3106
0.8181
0.1011
0.1567
0.0533
0.2019
0.0687
0.3705
-0.0831
-0.5235
-0.3496
-0.0514

```

Rysunek 6: Wyniki programu w Matlab dla faktoryzacji LU

```

"ZAD 4"
|13 18 69 11 39 31 63 43 55 76 33 72|
|40 77 5 58 17 45 57 75 80 99 77 81|
|44 93 20 11 71 6 60 26 6 15 81 12|
|13 82 38 83 42 94 74 91 74 52 78 93|
|25 46 94 61 60 95 18 79 95 69 79 82|
|17 92 16 76 37 41 40 12 96 86 56 42|
|96 51 47 23 91 5 99 46 16 4 70 61|
|8 52 66 48 50 16 66 17 57 62 47 4|
|31 34 2 14 38 85 92 30 42 35 55 53|
|11 58 9 64 66 67 90 8 72 71 27 89|
|87 74 69 39 35 29 19 54 68 31 34 50|
|38 77 44 38 51 17 30 33 97 61 45 27|

|68|
|53|
|71|
|56|
|91|
|91|
|75|
|15|
|4|
|15|
|11|
|93|

|-13.846153846153847|
|0.6883116883116883|
|3.55|
|0.6746987951807228|
|1.5166666666666666|
|2.2195121951219514|
|0.7575757575757576|
|0.8823529411764706|
|0.09523809523809523|
|0.2112676056338028|
|0.3235294117647059|
|3.4444444444444444|

```

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

```

-----
13 18 69 11 39 31 63 43 55 76 33 72
40 77 5 58 17 45 57 75 80 99 77 81
44 93 20 11 71 6 60 26 6 15 81 12
13 82 38 83 42 94 74 91 74 52 78 93
25 46 94 61 60 95 18 79 95 69 79 82
17 92 16 76 37 41 40 12 96 86 56 42
96 51 47 23 91 5 99 46 16 4 70 61
8 52 66 48 50 16 66 17 57 62 47 4
31 34 2 14 38 85 92 30 42 35 55 53
11 58 9 64 66 67 90 8 72 71 27 89
87 74 69 39 35 29 19 54 68 31 34 50
38 77 44 38 51 17 30 33 97 61 45 27

75 2 22 57 58 97 9 59 12 46 56 14

4.3679
1.7158
0.8222
2.8401
-2.8337
4.9529
-0.0715
-0.4016
-7.5786
3.3853
-1.8650
-0.6372

```

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