Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 2

по курсу «Численные методы линейной алгебры»

«Реализация метода Гаусса с перестановками»

Студент группы ИУ9-72Б Шемякин В.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать четыре варианта метода Гаусса с перестановками и научиться оценивать погрешность решения системы линейных уравнений для матриц произвольной размерности.

2 Результаты

Исходный код программы представлен в листинге 1.

```
using LinearAlgebra
using Plots
using Random

function check_diagonal_dominance(A::Matrix{Float64})
    n = size(A, 1)
    min_dominance = Inf
    is_dominant = true

    for i in 1:n
        diagonal_element = abs(A[i, i])
        row_sum = sum(abs(A[i, j]) for j in 1:n if j != i)
        dominance = diagonal_element - row_sum

        min_dominance = min(min_dominance, dominance)
        if dominance <= 0
            is_dominant = false
        end
    end

    if !is_dominant
        min_dominance = min(min_dominance, 0.0)
    end

    return is_dominant, min_dominance
end

function gauss_solve_vector(A::Matrix{Float64}, b::Vector{Float64})
    n = length(b)

    M = copy(A)
    r = copy(b)
```

```
33
34      for i in 1:n
35          if M[i, i] == 0
36              throw(ErrorException("
                                                                    $i,


                        ."))
37          end
38
39          for k in (i+1):n
40              if M[k, i] != 0
41                  factor = M[k, i] / M[i, i]
42                  for j in i:n
43                      M[k, j] -= factor * M[i, j]
44                  end
45                  r[k] -= factor * r[i]
46              end
47          end
48      end
49
50      x = zeros(n)
51
52      for i in n:-1:1
53          x[i] = r[i]
54          for k in (i+1):n
55              x[i] -= M[i, k] * x[k]
56          end
57          if M[i, i] == 0
58              throw(ErrorException("
    ."))
59          end
60          x[i] /= M[i, i]
61      end
62
63      return x
64  end
65
66  function gauss_solve_vector_column_pivot(A::Matrix{Float64}, b::
        Vector{Float64})
67      n = length(b)
68
69      M = copy(A)
70      r = copy(b)
71
72      col_perm = collect(1:n)
```

```
73
74      for i in 1:n
75          max_col = i
76          max_val = abs(M[i, i])
77
78          for j in (i+1):n
79              if abs(M[i, j]) > max_val
80                  max_val = abs(M[i, j])
81                  max_col = j
82              end
83          end
84
85          if max_col != i
86              for k in 1:n
87                  M[k, i], M[k, max_col] = M[k, max_col], M[k, i]
88              end
89              col_perm[i], col_perm[max_col] = col_perm[max_col],
    col_perm[i]
90          end
91
92          if abs(M[i, i]) < 1e-12
93              throw(ErrorException("
                                                    ."))
94          end
95
96          for k in (i+1):n
97              if M[k, i] != 0
98                  factor = M[k, i] / M[i, i]
99                  for j in i:n
100                     M[k, j] -= factor * M[i, j]
101                 end
102                 r[k] -= factor * r[i]
103             end
104         end
105     end
106
107     x_temp = zeros(n)
108
109     for i in n:-1:1
110         x_temp[i] = r[i]
111         for k in (i+1):n
112             x_temp[i] -= M[i, k] * x_temp[k]
113         end
114         x_temp[i] /= M[i, i]
115     end
```

```
116
117     x = zeros(n)
118     for i in 1:n
119         x[col_perm[i]] = x_temp[i]
120     end
121
122     return x
123 end
124
125
126 function gauss_solve_vector_row_pivot(A::Matrix{Float64}, b::Vector
        {Float64})
127     n = length(b)
128
129     M = copy(A)
130     r = copy(b)
131
132     for i in 1:n
133         max_row = i
134         max_val = abs(M[i, i])
135
136         for k in (i+1):n
137             if abs(M[k, i]) > max_val
138                 max_val = abs(M[k, i])
139                 max_row = k
140             end
141         end
142
143         if max_row != i
144             for j in 1:n
145                 M[i, j], M[max_row, j] = M[max_row, j], M[i, j]
146             end
147             r[i], r[max_row] = r[max_row], r[i]
148         end
149
150         if abs(M[i, i]) < 1e-12
151             throw(ErrorException("
                                                    ."))
152         end
153
154         for k in (i+1):n
155             if M[k, i] != 0
156                 factor = M[k, i] / M[i, i]
157                 for j in i:n
158                     M[k, j] -= factor * M[i, j]
```

```
159                 end
160                 r[k] -= factor * r[i]
161             end
162         end
163     end
164
165     x = zeros(n)
166
167     for i in n:-1:1
168         x[i] = r[i]
169         for k in (i+1):n
170             x[i] -= M[i, k] * x[k]
171         end
172         x[i] /= M[i, i]
173     end
174
175     return x
176 end
177
178 function gauss_solve_vector_combined(A::Matrix{Float64}, b::Vector{
        Float64})
179     n = length(b)
180
181     M = copy(A)
182     r = copy(b)
183
184     col_perm = collect(1:n)
185
186     for i in 1:n
187         max_row = i
188         max_col = i
189         max_val = abs(M[i, i])
190
191         for k in i:n
192             for j in i:n
193                 if abs(M[k, j]) > max_val
194                     max_val = abs(M[k, j])
195                     max_row = k
196                     max_col = j
197                 end
198             end
199         end
200
201         if max_row != i
202             for j in 1:n
```

```
203                 M[i, j], M[max_row, j] = M[max_row, j], M[i, j]
204             end
205             r[i], r[max_row] = r[max_row], r[i]
206         end
207
208         if max_col != i
209             for k in 1:n
210                 M[k, i], M[k, max_col] = M[k, max_col], M[k, i]
211             end
212             col_perm[i], col_perm[max_col] = col_perm[max_col],
    col_perm[i]
213         end
214
215         if abs(M[i, i]) < 1e-12
216             throw(ErrorException("
                                                    ."))
217         end
218
219         for k in (i+1):n
220             if M[k, i] != 0
221                 factor = M[k, i] / M[i, i]
222                 for j in i:n
223                     M[k, j] -= factor * M[i, j]
224                 end
225                 r[k] -= factor * r[i]
226             end
227         end
228     end
229
230     x_temp = zeros(n)
231
232     for i in n:-1:1
233         x_temp[i] = r[i]
234         for k in (i+1):n
235             x_temp[i] -= M[i, k] * x_temp[k]
236         end
237         x_temp[i] /= M[i, i]
238     end
239
240     x = zeros(n)
241     for i in 1:n
242         x[col_perm[i]] = x_temp[i]
243     end
244
245     return x
```

```
246 end
247
248 function compute_residual_errors(A::Matrix{Float64}, b::Vector{
    Float64}, x_approx::Vector{Float64})
249     residual_vec = A * x_approx - b
250
251     abs_residual = norm(residual_vec)
252
253     b_norm = norm(b)
254     rel_residual = b_norm == 0 ? abs_residual : abs_residual /
    b_norm
255
256     return abs_residual, rel_residual
257 end
258
259 function estimate_rounding_error(A::Matrix{Float64}, b::Vector{
    Float64}, x_approx::Vector{Float64})
260     residual_vec = A * x_approx - b
261
262     try
263         cond_number = cond(A)
264         machine_eps = eps(Float64)
265         round_error_est = cond_number * machine_eps * norm(x_approx
    )
266         residual_norm = norm(residual_vec)
267
268         return round_error_est, residual_norm
269     catch
270         return NaN, norm(residual_vec)
271     end
272 end
273
274 function estimate_perturbation_error(A::Matrix{Float64}, b::Vector{
    Float64}, x_ref::Vector{Float64},
275                                      deltaA::Matrix{Float64}, deltaB
    ::Vector{Float64})
276     try
277         cond_number = cond(A)
278
279         rel_deltaA = norm(deltaA) / norm(A)
280         rel_deltaB = norm(deltaB) / norm(b)
281
282         denominator = 1 - cond_number * rel_deltaA
283
284         if denominator > 0
```

```
285             theoretical_bound = cond_number * (rel_deltaA +
    rel_deltaB) / denominator
286
287             A_mod = A + deltaA
288             b_mod = b + deltaB
289
290             x_mod = A_mod \ b_mod
291             actual_perturb = norm(x_mod - x_ref) / norm(x_ref)
292
293             return theoretical_bound, actual_perturb
294         else
295             return Inf, NaN
296         end
297     catch
298         return NaN, NaN
299     end
300 end
301
302 function full_error_report(A::Matrix{Float64}, b::Vector{Float64},
    x_calc::Vector{Float64},
303                            x_exact::Vector{Float64})
304     diff = x_calc - x_exact
305     abs_err = norm(diff)
306     rel_err = abs_err / norm(x_exact)
307
308     println("
    : ", abs_err)
309     println("
                 : ", rel_err)
310
311     round_est, residual_norm = estimate_rounding_error(A, b, x_calc
    )
312     println("
    : ", round_est)
313
314     perturb_level = 1e-12
315     deltaA = (rand(Float64, size(A)) .- 0.5) * 2 * perturb_level *
    norm(A)
316     deltaB = (rand(Float64, size(b)) .- 0.5) * 2 * perturb_level *
    norm(b)
317
318     theory_bound, actual_perturb = estimate_perturbation_error(A, b
    , x_calc, deltaA, deltaB)
319     println("
                        : ", theory_bound)
```

```
320     println("
                    : ", actual_perturb)
321
322     try
323         cond_number = cond(A)
324         println("
                    : ", cond_number)
325     catch
326         println("                                                    :
                                              ")
327     end
328
329     return abs_err, rel_err, round_est, residual_norm, theory_bound
        , actual_perturb
330 end
331
332 function make_dominant_matrix(n::Int, dominance::Float64, scale::
        Float64 = 10.0)
333     A = rand(Float64, n, n) * scale
334
335     for i in 1:n
336         row_sum = sum(abs, A[i, :]) - abs(A[i, i])
337         target_diag = row_sum + max(dominance, 1e-10)
338         if target_diag > 0
339             A[i, i] = sign(A[i, i]) * target_diag
340         else
341             A[i, i] = sign(A[i, i]) * (abs(A[i, i]) + 1.0)
342         end
343     end
344
345     return A
346 end
347
348 make_random_vector(n::Int; scale=10.0) = rand(Float64, n) .* (2*
        scale) .- scale
349
350
351 function run_dominance_experiment(start_val, stop_val)
352     n = 10
353     num_points = 50
354
355     dom_levels = range(start_val, stop_val, length=num_points)
356
357     err_no_pivot = zeros(num_points)
358     err_col_pivot = zeros(num_points)
```

```
    err_row_pivot = zeros(num_points)
    err_combined = zeros(num_points)
    err_builtin = zeros(num_points)

    for (idx, dom) in enumerate(dom_levels)
        A = make_dominant_matrix(n, dom, 10.0)
        x_exact = make_random_vector(n)
        b = A * x_exact

        _, actual_dom = check_diagonal_dominance(A)

        try
            x_no_pivot = gauss_solve_vector(A, b)
            err_no_pivot[idx] = norm(x_exact - x_no_pivot)
        catch
            err_no_pivot[idx] = NaN
        end

        try
            x_col_pivot = gauss_solve_vector_column_pivot(A, b)
            err_col_pivot[idx] = norm(x_exact - x_col_pivot)
        catch
            err_col_pivot[idx] = NaN
        end

        try
            x_row_pivot = gauss_solve_vector_row_pivot(A, b)
            err_row_pivot[idx] = norm(x_exact - x_row_pivot)
        catch
            err_row_pivot[idx] = NaN
        end

        try
            x_combined = gauss_solve_vector_combined(A, b)
            err_combined[idx] = norm(x_exact - x_combined)
        catch
            err_combined[idx] = NaN
        end

        try
            x_builtin = A \ b
            err_builtin[idx] = norm(x_exact - x_builtin)
        catch
            err_builtin[idx] = NaN
        end
```

```
404     end
405
406     p = plot(dom_levels, err_no_pivot,
407             label="                          (
                                        )", linewidth=2, marker=:
    circle, markersize=3, color=:red)
408     plot!(p, dom_levels, err_row_pivot,
409             label="                          (
                    )", linewidth=2, marker=:square, markersize=3,
    color=:blue)
410     plot!(p, dom_levels, err_col_pivot,
411             label="                          (
                      )", linewidth=2, marker=:diamond, markersize=3,
     color=:green)
412     plot!(p, dom_levels, err_combined,
413             label="                          (
                                    )", linewidth=2, marker=:
    utriangle, markersize=3, color=:orange)
414     plot!(p, dom_levels, err_builtin,
415             label="                                  Julia", linewidth
    =2, marker=:cross, markersize=3, color=:purple)
416
417     xlabel!("
                              ")
418     ylabel!("                                                  ")
419     title!("

                 ")
420
421     return p
422 end
423
424 function main()
425     n = 4
426
427     println("                                                  A
                          x...")
428     A = rand(Float64, n, n) * 5.0
429     x_exact = rand(Float64, n) * 10.0
430
431     b = A * x_exact
432     x_builtin = A \ b
433
434     println("\n              A ($(n) $ (n)):")
435     display(A)
```

```
436     println("\n                                    x ($(n) 1 ):")
437     display(x_exact)
438     println("\n                                        b ($(n) 1 ):")
439     display(b)
440
441     methods = [
442         ("                          (
                                          )", gauss_solve_vector),
443         ("                          (
                    )", gauss_solve_vector_column_pivot),
444         ("                          (
                  )", gauss_solve_vector_row_pivot),
445         ("                          (                                  )
    ", gauss_solve_vector_combined)
446     ]
447
448     println("\n
                                :")
449     println("    " ^ 60)
450
451     for (method_name, method_func) in methods
452         try
453             println("\n$method_name:")
454             x_calc = method_func(A, b)
455
456             full_error_report(A, b, x_calc, x_exact)
457
458         catch e
459             println("              : ", e)
460         end
461     end
462
463     display(run_dominance_experiment(-20, 20))
464
465 end
466
467 main()
```

Результат запуска представлен на рисунках 1-3.

```
Матрица A (4×4):

4×4 Matrix{Float64}:
 3.78144  4.98659   3.33513  1.83076
 4.55779  0.397539  2.35406  0.63572
 4.63512  4.74567   2.52184  4.54378
 1.64791  2.3023    3.03132  2.07215


Истинное решение x (4×1):

4-element Vector{Float64}:
 9.377862826128212
 8.029377905414526
 3.923849063558956
 6.65034977695664


Вектор правой части b (4×1):

4-element Vector{Float64}:
 100.76282705471367
  59.39903972808405
 121.6853096594773
  59.614854900184945
```

Рис. 1 — Результат

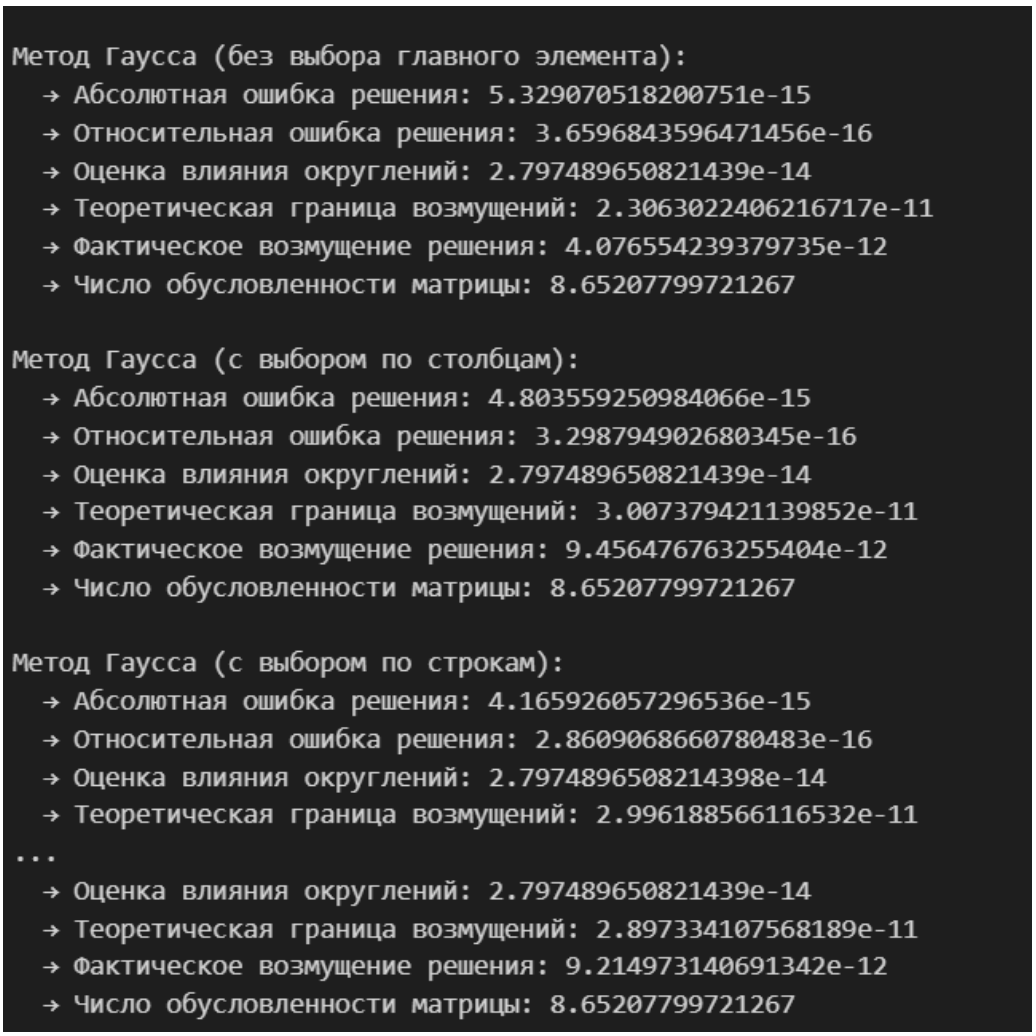

Рис. 2 — Результат

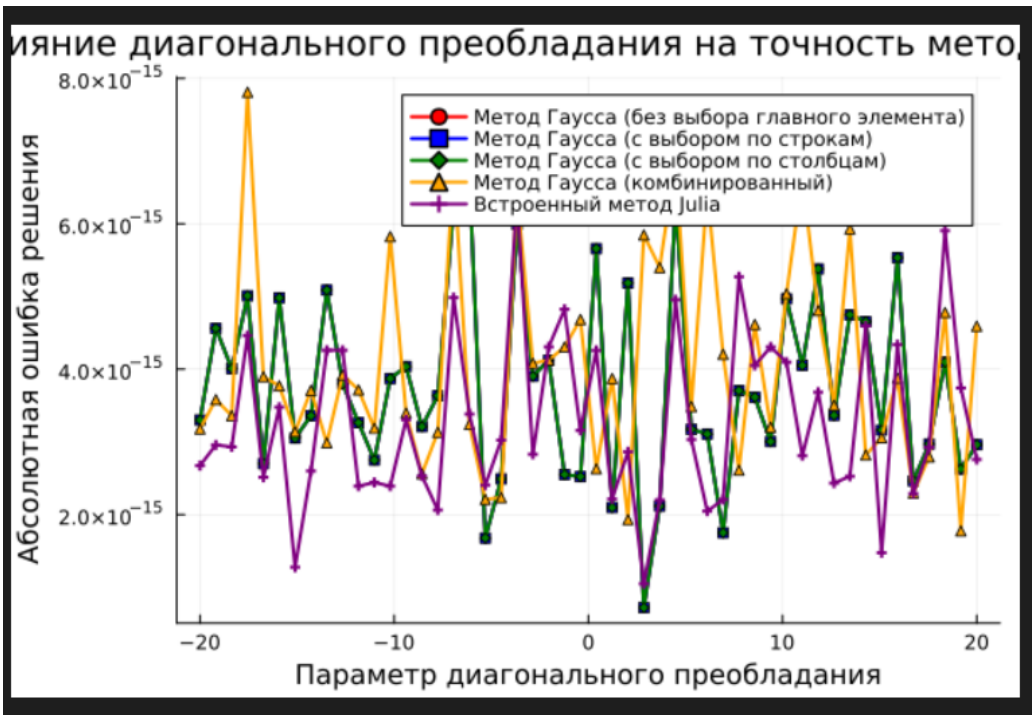

Рис. 3 — Результат