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 № 3

по курсу «Численные методы линейной алгебры»

«Метод Данилевского поиска собственных значений и векторов»

Студент группы ИУ9-72Б Шемякин В.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать метод Данилевского для нахождения собственных значений матрицы, а также выполнить проверку на правильность найденных значений (при помощи следа матрицы и проверки на ортогональность), а также добавить визуализацию характеристического многочлена на отрезках Гершгорина

2 Результаты

Исходный код программы представлен в листинге 1.

```
[language={},caption={main},label={lst:code1}]
using Random
using LinearAlgebra
using Plots

euclidean_norm(vec::AbstractVector) = sqrt(sum(abs2, vec))
identity_matrix(n::Int) = Matrix{Float64}(I, n, n)

function generate_symmetric_matrix(l::Real, r::Real, n::Int)
    A = rand(n, n) .* (r - l) .+ l
    (A + A') / 2
end

function danilevsky_algorithm(A)
    n = size(A, 1)
    B = Matrix{Float64}(I, n, n)
    P = copy(A)

    for k in n:-1:2
        B_k = Matrix{Float64}(I, n, n)
        for j in 1:n
            if j != k-1
                B_k[k-1, j] = -P[k, j] / P[k, k-1]
            else
                B_k[k-1, j] = 1.0 / P[k, k-1]
            end
        end
```

```
28
29          B_k_inverted = Matrix{Float64}(I, n, n)
30          for j in 1:n
31              B_k_inverted[k-1, j] = P[k, j]
32          end
33
34          P = B_k_inverted * P * B_k
35          B = B * B_k
36      end
37
38      return B, P
39  end
40
41  function union_intervals(intervals::Vector{Vector{Float64}})
42      isempty(intervals) && return Vector{Vector{Float64}}()
43      s = sort(intervals, by = x -> x[1])
44      merged = [copy(s[1])]
45      for v in s[2:end]
46          last = merged[end]
47          if last[2] >= v[1]
48              last[2] = max(last[2], v[2])
49          else
50              push!(merged, copy(v))
51          end
52      end
53      merged
54  end
55
56  function gershgorin_intervals_raw(A::Matrix{Float64})
57      n = size(A, 1)
58      ivs = Vector{Vector{Float64}}(undef, n)
59      for i in 1:n
60          r = sum(abs.(A[i, :])) - abs(A[i, i])
61          c = A[i, i]
62          ivs[i] = [c - r, c + r]
63      end
64      ivs
65  end
66
```

```
function gershgorin_intervals(A::Matrix{Float64})
    union_intervals(gershgorin_intervals_raw(A))
end

find_equation_coeffs(D::Matrix{Float64}) = vcat(1.0, -D[1, :])

function poly_eval(coeffs::AbstractVector{<:Real}, x::Real)
    acc = 0.0
    for c in coeffs
        acc = acc * x + c
    end
    acc
end

function find_real_roots_on_intervals(coeffs::Vector{Float64},
                                      intervals::Vector{Vector{Float64}};
                                      step::Real = 1e-2, tol::Real = 1e-8)
    roots = Float64[]
    for I in intervals
        a, b = I
        x = a
        prev_y = poly_eval(coeffs, x)
        while x < b - step/2
            x_next = min(x + step, b)
            y_next = poly_eval(coeffs, x_next)
            if prev_y == 0.0
                push!(roots, x)
            elseif y_next == 0.0
                push!(roots, x_next)
            elseif prev_y * y_next < 0
                left, right = x, x_next
                yl = prev_y
                while right - left > tol
                    mid = (left + right) / 2
                    ym  = poly_eval(coeffs, mid)
                    if yl * ym <= 0
                        right = mid
                    else
                        left = mid
```

```
106                         yl   = ym
107                     end
108                 end
109                 push!(roots, (left + right) / 2)
110             end
111             x = x_next
112             prev_y = y_next
113         end
114     end
115     sort!(unique!(round.(roots; digits=10)))
116     roots
117 end
118
119 function eigenvectors_from_Bs(Bs::Matrix{Float64},
        eigvals::AbstractVector{<:Real})
120     n = size(Bs, 1)
121     X = Matrix{Float64}(undef, n, length(eigvals))
122     for (k, ) in pairs(eigvals)
123         v = [^(n - i) for i in 1:n]
124         x = Bs * v
125         nrm = euclidean_norm(x)
126         nrm > 0 && (x ./= nrm)
127         X[:, k] = x
128     end
129     X
130 end
131
132 function check_trace(A::Matrix{Float64}, eigvals::AbstractVector{<:Real};
        tol=1e-5)
133     s = sum(eigvals)
134     traceA = LinearAlgebra.tr(A)
135     if abs(s - traceA) <= tol * max(1.0, abs(traceA))
136         println("  : (  = $(s), tr(A) = $(traceA))")
137     else
138         println("  :  (  = $(s), tr(A) = $(traceA))")
139     end
140 end
141
142 function check_orthogonality(X::Matrix{Float64}; tol=1e-6)
```

```
143     m = size(X, 2)
144     for i in 1:m-1, j in i+1:m
145         d = dot(X[:, i], X[:, j])
146         if abs(d) > tol
147             println(" :  (e$i, e$j = $d)")
148             return
149         end
150     end
151     println(" : ")
152 end
153 
154 function make_plot(coeffs::Vector{Float64},
155                                 merged_intervals::Vector{Vector{Float64}};
156                                 samples::Int = 2000,
157                                 roots::AbstractVector{<:Real} = Float64[],
158                                 raw_intervals::Vector{Vector{Float64}} =
      Vector{Vector{Float64}}())
159     a_min = minimum(I[1] for I in merged_intervals)
160     b_max = maximum(I[2] for I in merged_intervals)
161     xs = range(a_min, b_max; length=samples)
162     ys = [poly_eval(coeffs, x) for x in xs]
163 
164     plt = plot(xs, ys, legend=false, xlabel="", ylabel="p()",
165                 title = "    [$(round(a_min, digits=3)), $(round(b_max,
      digits=3))]")
166     hline!(plt, [0.0], linestyle=:dash)
167     vline!(plt, [a_min, b_max], linestyle=:dot)
168 
169     if !isempty(raw_intervals)
170         bounds = reduce(vcat, raw_intervals)
171         vline!(plt, bounds; linestyle=:dash, alpha=0.6, linewidth=1.8)
172     end
173 
174     if !isempty(roots)
175         scatter!(plt, roots, zeros(length(roots)); markersize=5)
176     end
177     display(plt)
178 end
179 
```

```
180
181 function main()
182     Random.seed!(42)
183     n = 5
184     A = generate_symmetric_matrix(-5, 5, n)
185
186     println("  A:")
187     show(stdout, "text/plain", round.(A; digits=4))
188     println("\n")
189
190     raw_intervals = gershgorin_intervals_raw(A)
191     println(" : ", raw_intervals)
192     intervals = union_intervals(copy(raw_intervals))
193     println("  (): ", intervals, "\n")
194     Bs, Df = danilevsky_algorithm(A)
195     coeffs = find_equation_coeffs(Df)
196     println("  : ", coeffs, "\n")
197     eig_vals = find_real_roots_on_intervals(coeffs, intervals; step=1e-3,
        tol=1e-10)
198     println(" : ", eig_vals, "\n")
199
200     make_plot(coeffs, intervals;
201     samples=3000, roots=eig_vals, raw_intervals=raw_intervals)
202     if length(eig_vals) < n
203         println(":     ,  n = $n.")
204         return
205     end
206     X = eigenvectors_from_Bs(Bs, eig_vals)
207
208     println("  ():")
209     show(stdout, "text/plain", round.(X; digits=6))
210     println("\n")
211
212     check_trace(A, eig_vals; tol=1e-8)
213     check_orthogonality(X; tol=1e-6)
214 end
215
216 main()
```

Результат запуска представлен на рисунке 1.

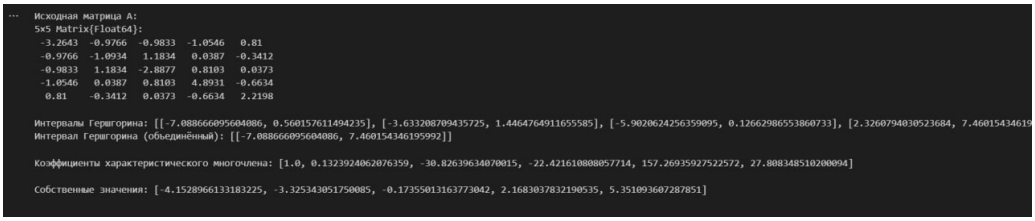

Рис. 1 — Результат

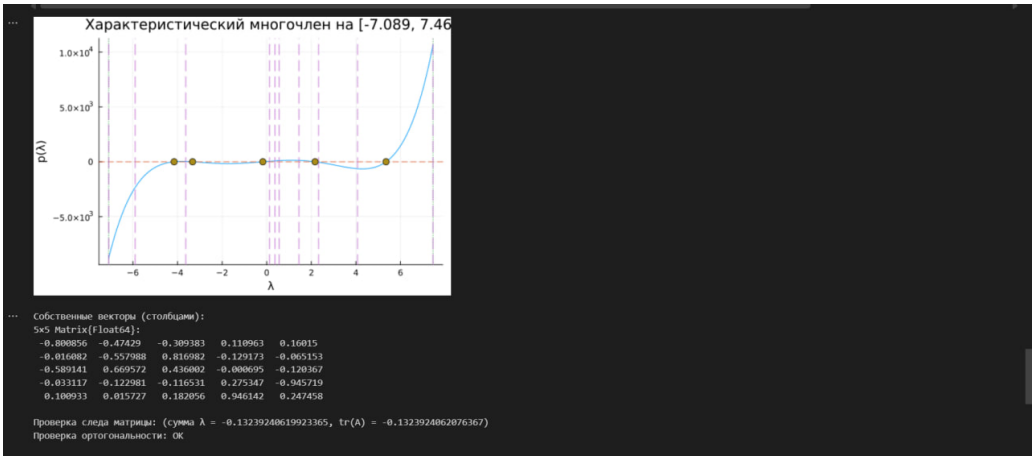

Рис. 2 — Результат 2

3 Вывод

В ходе лабораторной работы мне удалось реализовать метод Данилевского для нахождения собственных значений матрицы