[ ]:
```julia
using LinearAlgebra, Random # Подключаем библиотеки
```

Реализуем QR-разложение с помощью двух методов: метод вращений и метод отражений

[ ]:
```julia
function QR_decomp(A::Matrix{Float64}, method::Symbol=:Хаусхолдер)
    m, n = size(A)
    R = copy(A)
    Q = Matrix{Float64}(I, m, m)

    if method == :Хаусхолдер
        # Метод отражений Хаусхолдера
        v_full = Vector{Float64}(undef, m)

        for k in 1:min(m-1, n)
            x = @view R[k:end, k]
            x_norm = norm(x)

            # пропускаем нулевые столбцы
            if isapprox(x_norm, 0.0)
                continue
            end

            e = zeros(length(x))
            e[1] = sign(x[1]) * x_norm
            v = @view v_full[1:length(x)]
            v .= x - e

            v_norm = norm(v)
            if isapprox(v_norm, 0.0)
                continue
            end
            v ./= v_norm

            # Обновляем R и Q
            R_sub = @view R[k:end, k:n]
            R_sub .-= 2 .* v .* (v' * R_sub)

            Q_sub = @view Q[:, k:end]
            Q_sub .-= 2 .* (Q_sub * v) .* v'
        end

    elseif method == :Гивенс
        # Метод вращений Гивенса
        for j in 1:n
            for i in m:-1:j+1
                a, b = R[i-1, j], R[i, j]
```

```
                r = hypot(a, b)

                if isapprox(r, 0.0)
                    continue
                end

                c = a / r
                s = -b / r

                # Обновляем R и Q
                G = [c -s; s c]
                R_sub = @view R[i-1:i, j:n]
                R_sub .= G * R_sub

                Q_sub = @view Q[:, i-1:i]
                Q_sub .= Q_sub * G'
            end
        end
    end

    return Q, R
end
```

[ ]:
```
QR_decomp (generic function with 2 methods)
```

[ ]:
```
using Printf

function print_matrix_str(A)
    for row in eachrow(A)
        println(join([@sprintf("%10.3f", x) for x in row], " "))
    end
end
```

[ ]:
```
print_matrix_str (generic function with 1 method)
```

[ ]:
```
# Решение СЛАУ Ax = b
function solveQR(A::Matrix{Float64}, b::Vector{Float64}, method::
  ↪Symbol=:Хаусхолдер)
    Q, R = QR_decomp(A, method)

    y = Q' * b

    n = length(b)
    x = zeros(Float64, n)
    for i in n:-1:1
        x[i] = (y[i] - sum(R[i,i+1:end] .* x[i+1:end])) / R[i,i]
    end
```

```
    return x
end
```

[ ]:
```
solveQR (generic function with 2 methods)
```

[ ]:
```
function demo1(n::Int=5, m::Int=n; method::Symbol=:Хаусхолдер)
    println("–"^50)
    println("Размерность матрицы: [$n, $m], метод: $method")

    A = randn(n, m)

    Q, R = QR_decomp(A, method)
    #print_matrix_str(Q)
    #println()
    #print_matrix_str(R)
    println("Проверка A ≈ Q * R ", isapprox(Q * R, A, atol=1e-8))

    # Проверка решения СЛАУ
    A = randn(n, n)
    b = randn(n)
    x = solveQR(A, b, method)
    println("Проверка Ax ≈ b: ", isapprox(A * x, b))

    println("–"^50)
end

demo1(10, method = :Гивенс)
```

```
--------------------------------------------------
Размерность матрицы: [10, 10], метод: Хаусхолдер
     0.474      0.570     -0.100     -0.394     -0.402      0.153
  ↪0.013
-0.107     -0.285     -0.087
     0.313     -0.211      0.227      0.149     -0.252     -0.086
  ↪0.320
-0.496      0.447     -0.406
    -0.001      0.315      0.012      0.593     -0.313      0.290
  ↪0.251
0.505      0.220      0.015
     0.222      0.253      0.720     -0.128      0.142     -0.363
  ↪-0.205
0.261      0.245      0.173
     0.296      0.184     -0.275      0.177      0.529     -0.192
  ↪-0.117
0.213     -0.030     -0.629
```

```
     0.189     -0.005     -0.141      0.373     -0.183      0.073
  ↪-0.814
-0.251      0.151      0.136
     0.250     -0.006     -0.127      0.414     -0.068     -0.615
  ↪0.250
-0.125     -0.409      0.348
    -0.360     -0.054     -0.146     -0.164     -0.552     -0.532
  ↪-0.187
0.269      0.060     -0.346
     0.481     -0.367     -0.408     -0.284     -0.030     -0.047
  ↪0.067
0.366      0.396      0.298
     0.285     -0.544      0.348      0.073     -0.182      0.218
  ↪-0.121
0.297     -0.509     -0.229

     2.968     -1.096     -2.961     -0.525     -0.070     -1.460
  ↪-0.840
-0.253     -0.477      0.668
     0.000      3.133      1.819     -0.298     -1.156     -0.476
  ↪-1.012
-2.807      0.610      0.234
    -0.000      0.000      3.220     -0.621      0.102     -0.653
  ↪-0.634
-0.722     -0.243      1.979
     0.000      0.000     -0.000     -2.404     -1.060      1.187
  ↪-0.312
1.165      0.520     -0.516
     0.000      0.000     -0.000     -0.000      2.619      1.044
  ↪-0.030
-1.161      1.035      0.289
     0.000      0.000      0.000      0.000      0.000      2.922
  ↪-0.054
-0.222     -1.606     -0.811
     0.000      0.000      0.000      0.000      0.000      0.000
  ↪2.631
-1.113     -1.434      0.136
    -0.000      0.000      0.000      0.000      0.000      0.000
  ↪-0.000
1.137     -1.864      0.326
     0.000      0.000      0.000     -0.000      0.000      0.000
  ↪0.000
-0.000      0.985     -0.081
     0.000      0.000      0.000      0.000      0.000      0.000
  ↪0.000
0.000      0.000      0.539
```

```
Проверка A ≈ Q * R true
Проверка Ax ≈ b: true
```

[ ]:
```julia
function solveIter(A::Matrix{Float64}, b::Vector{Float64}, method::
  ↪Symbol=:Якоби, max_iter::Int=10000, eps::Float64=1e-8)
    n = length(b)
    D = diag(A)
    B = A - Diagonal(D)
    x = zeros(n)

    if any(D .== 0)
        error("Матрица содержит нули на диагонали")
    end

    k_est = errorEst(A, b, x, method=method, eps=eps)

    if k_est != nothing
        println("Априорная оценка: $k_est итераций")
    else
        println("Априорная оценка: метод не сходится")
    end

    for k in 1:max_iter
        if method == :Якоби
            x_new = (b - B * x) ./ D
        elseif method == :Зейделя
            x_new = copy(x)
            for i in 1:length(x)
                x_new[i] = (b[i] - (B * x_new)[i]) / D[i]
            end
        else
            error("Неизвестный метод: $method")
        end

        if isapprox(norm(x_new - x), 0.0, atol=eps)
            println("Метод $method сошёлся за $k итераций")
            return x_new
        end

        x = copy(x_new)
    end

    println("Метод $method не сошёлся за $max_iter итераций")
    return nothing
end
```

[ ]: solveIter (generic function with 4 methods)

[ ]:
```
function errorEst(A::Matrix{Float64}, b::Vector{Float64}, x0::
  ↪Vector{Float64}=zeros(length(b)); method::Symbol=:Якоби, eps::
  ↪Float64=1e-8)
    n = length(b)
    D = Diagonal(diag(A))
    B = A - D
    L = tril(A, -1)
    U = triu(A, 1)

    # Матрица перехода
    if method == :Якоби
        T = -inv(D) * B
    elseif method == :Зейделя
        T = -inv(D + L) * U
    else
        error("Неизвестный метод: $method")
    end

    # Вычисление нормы
    if all(abs.(diag(A)) .> sum(abs.(A), dims=2) .- abs.(diag(A)))
        norm_T = norm(T, Inf)
    else
        norm_T = maximum(abs.(eigvals(T)))
    end

    if norm_T >= 1
        return nothing
    end

    # Оценка числа итераций
    initial_error = norm(b, Inf)
    k = ceil(log(eps / initial_error) / log(norm_T))

    return Int(k)
end
```

[ ]: errorEst (generic function with 2 methods)

[ ]:
```
function demo2(n::Int=5)
    println("="^50)

    # Матрица с диагональным преобладанием
    println()
    println("-"^50)
    A = rand(n, n) .* 2 .- 1
```

```
    A = A + n*I

    @assert all(abs.(diag(A)) .> sum(abs.(A), dims=2) .- abs.
↪(diag(A))) "Матрица не имеет диагональное преобладание!"

    print("Матрица с диагональным преобладанием\n")
    println("-"^50)
    println()

    b = randn(n)

    x = solveIter(A, b)
    if x != nothing
        println("Проверка Ax ≈ b: ", isapprox(A * x, b, atol=1e-6))
    end

    println()

    x = solveIter(A, b, :Зейделя)
    if x != nothing
        println("Проверка Ax ≈ b: ", isapprox(A * x, b, atol=1e-6))
    end

    # Матрица положительно определенная без диагонального преобладания
    println()
    println("-"^50)
    A = randn(n, n)
    A = A * A'

    @assert isposdef(A) "Матрица не положительно определенная!"
    @assert !all(abs.(diag(A)) .> sum(abs.(A), dims=2) .- abs.
↪(diag(A))) "Матрица имеет диагональное преобладание!"

    print("Матрица положительно определенная без диагонального␣
↪преобладания\n")
    println("-"^50)
    println()

    b = randn(n)

    x = solveIter(A, b)
    if x != nothing
        println("Проверка Ax ≈ b: ", isapprox(A * x, b, atol=1e-6))
    end

    println()
```

```
    x = solveIter(A, b, :Зейделя)
    if x != nothing
        println("Проверка Ax ≈ b: ", isapprox(A * x, b, atol=1e-6))
    end

    println()
    println("—"^50)
end

demo2(5)
```

```
——————————————————————————————————————————————————

--------------------------------------------------
Матрица с диагональным преобладанием
--------------------------------------------------

Априорная оценка: 13 итераций
Метод Якоби сошёлся за 14 итераций
Проверка Ax ≈ b: true

Априорная оценка: 14 итераций
Метод Зейделя сошёлся за 10 итераций
Проверка Ax ≈ b: true

--------------------------------------------------
Матрица положительно определенная без диагонального преобладания
--------------------------------------------------

Априорная оценка: метод не сходится
Метод Якоби не сошёлся за 10000 итераций

Априорная оценка: 492 итераций
Метод Зейделя сошёлся за 476 итераций
Проверка Ax ≈ b: true

——————————————————————————————————————————————————
```