Реализуем функцию для LU-разложения матрицы A с выбором ведущего элемента по столбцу или по всей матрице (по умолчанию - по столбцу)

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

[ ]:
```julia
function LU_decomp(A::Matrix{Float64}, pivoting::Symbol=:column)
    n = size(A, 1) # размер матрицы [n, n]
    L = Matrix{Float64}(I, n, n)
    U = copy(A)
    P = Matrix{Float64}(I, n, n)
    Q = Matrix{Float64}(I, n, n)
    row_swaps = 0
    col_swaps = 0

    for k in 1:n-1
        if pivoting == :column
            pivot_row = k + argmax(abs.(U[k:end, k])) - 1

            # При необходимости меняем местами строки
            if pivot_row != k
                row_swaps += 1
                U[[k, pivot_row], :] = U[[pivot_row, k], :]
                P[[k, pivot_row], :] = P[[pivot_row, k], :]

                if k > 1
                    L[[k, pivot_row], 1:k-1] = L[[pivot_row, k], 1:
↪k-1]
                end
            end

        elseif pivoting == :full
            submatrix = abs.(U[k:end, k:end])
            ind = argmax(submatrix)
            i, j = Tuple(CartesianIndices(size(submatrix))[ind])
            pivot_row, pivot_col = i + k - 1, j + k - 1

            # При необходимости меняем местами строки
            if pivot_row != k
                row_swaps += 1
                U[[k, pivot_row], :] = U[[pivot_row, k], :]
                P[[k, pivot_row], :] = P[[pivot_row, k], :]

                if k > 1
                    L[[k, pivot_row], 1:k-1] = L[[pivot_row, k], 1:
↪k-1]
                end
            end
```

```
            # При необходимости меняем местами столбцы
            if pivot_col != k
                col_swaps += 1
                U[:, [k, pivot_col]] = U[:, [pivot_col, k]]
                Q[:, [k, pivot_col]] = Q[:, [pivot_col, k]]
            end
        end

        for i in k+1:n
            L[i, k] = U[i, k] / U[k, k]
            U[i, k:n] .-= L[i, k] * U[k, k:n]
        end
    end

    return L, U, P, Q, row_swaps, col_swaps
end
```

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

[ ]:
```
function LU_rang(A::Matrix{Float64}, pivoting::Symbol = :column)
    m, n = size(A) # размер матрицы [m, n]
    U = copy(A)
    L = Matrix{Float64}(I, m, m)
    P = Matrix{Float64}(I, m, m)
    Q = Matrix{Float64}(I, n, n)

    r = 0
    k = 1
    last_col = n

    while k <= min(m, last_col)
        if pivoting == :column
            pivot_row = k - 1 + argmax(abs.(U[k:end, k]))

            if isapprox(U[pivot_row, k], 0.0; atol=1e-12)
                if k != last_col
                    U[:, [k, last_col]] = U[:, [last_col, k]]
                    Q[:, [k, last_col]] = Q[:, [last_col, k]]
                end
                last_col -= 1
                continue
            end

            # При необходимости переставляем строки
            if pivot_row != k
                U[[k, pivot_row], :] = U[[pivot_row, k], :]
```

```
                P[[k, pivot_row], :] = P[[pivot_row, k], :]
                if k > 1
                    L[[k, pivot_row], 1:k-1] = L[[pivot_row, k], 1:
↪k-1]
                end
            end

        elseif pivoting == :full
            sub = abs.(U[k:end, k:end])
            ind = argmax(sub)
            i, j = Tuple(CartesianIndices(sub)[ind])
            pivot_row, pivot_col = i + k - 1, j + k - 1

            if isapprox(U[pivot_row, pivot_col], 0.0; atol=1e-12)
                k += 1
                continue
            end

            # При необходимости переставляем строки
            if pivot_row != k
                U[[k, pivot_row], :] = U[[pivot_row, k], :]
                P[[k, pivot_row], :] = P[[pivot_row, k], :]
                if k > 1
                    L[[k, pivot_row], 1:k-1] = L[[pivot_row, k], 1:
↪k-1]
                end
            end

            # При необходимости переставляем столбцы
            if pivot_col != k
                U[:, [k, pivot_col]] = U[:, [pivot_col, k]]
                Q[:, [k, pivot_col]] = Q[:, [pivot_col, k]]
            end
        end

        for i in k+1:m
            L[i, k] = U[i, k] / U[k, k]
            U[i, k:end] .-= L[i, k] * U[k, k:end]
        end

        r += 1
        k += 1
    end

    L = L[:, 1:r]
    U = U[1:r, :]
```

```
    return L, U, P, Q, r
end
```

[ ]: LU_rang (generic function with 2 methods)

[ ]:
```
function solveLU_ex(A::Matrix{Float64}, b::Vector{Float64}, pivoting::
  ↪Symbol = :column)
    L, U, P, Q, r = LU_rang(A, pivoting)
    _, _, _, _, r_ex = LU_rang(hcat(A, b), pivoting)

    if r != r_ex
        return nothing  # Система несовместна
    end

    m, n = size(A)

    # Прямой ход для Гаусса (только первые r уравнений)
    Pb = P * b
    y = zeros(r)
    for i in 1:r
        y[i] = Pb[i] - sum(L[i, 1:i-1] .* y[1:i-1])
    end

    # Обратный ход Гаусса (только первые r переменных)
    z = zeros(r)
    for i in r:-1:1
        if isapprox(U[i, i], 0)
            z[i] = 0.0
        else
            z[i] = (y[i] - sum(U[i, i+1:r] .* z[i+1:r])) / U[i, i]
        end
    end

    x_basic = vcat(z, zeros(n - r))  # свободные переменные = 0
    x = Q * x_basic

    return x
end
```

[ ]: solveLU_ex (generic function with 2 methods)

Реализуем с помощью LU-разложения дополнительные функции.

[ ]:
```
# Нахождение определителя
function detLU(A::Matrix{Float64}, pivoting::Symbol=:column)
    _, U, P, _, row_swaps, col_swaps = LU_decomp(A, pivoting)
    result = prod(diag(U))
```

```
    if pivoting == :column
        return (-1)^row_swaps * result
    end

    if pivoting == :full
        return (-1)^(row_swaps + col_swaps) * result
    end
end

# Решение СЛАУ Ax = b
function solveLU(A::Matrix{Float64}, b::Vector{Float64}, pivoting::
  ↪Symbol=:column)
    L, U, P, Q = LU_decomp(A, pivoting)

    # При необходимости переставляем строки
    Pb = P * b

    # Прямой ход Гаусса
    n = length(Pb)
    y = zeros(Float64, n)

    for i in 1:n
        y[i] = Pb[i] - sum(L[i,1:i-1] .* y[1:i-1])
    end

    # Обратный ход Гаусса
    x = zeros(Float64, n)

    for i in n:-1:1
        x[i] = (y[i] - sum(U[i,i+1:end] .* x[i+1:end])) / U[i,i]
    end

    # При необходимости переставляем столбцы
    if Q != Matrix{Float64}(I, size(Q)...)
        x = Q * x
    end

    return x
end

# Вариант с уже проведенным LU-разложением
function solveLU(L::Matrix{Float64}, U::Matrix{Float64}, P::
  ↪Matrix{Float64}, Q::Matrix{Float64}, b::Vector{Float64})
    # При необходимости переставляем строки
    Pb = P * b
```

```
    # Прямой ход Гаусса
    n = length(Pb)
    y = zeros(Float64, n)

    for i in 1:n
        y[i] = Pb[i] - sum(L[i,1:i-1] .* y[1:i-1])
    end

    # Обратный ход Гаусса
    x = zeros(Float64, n)

    for i in n:-1:1
        x[i] = (y[i] - sum(U[i,i+1:end] .* x[i+1:end])) / U[i,i]
    end

    # При необходимости переставляем столбцы
    if Q != Matrix{Float64}(I, size(Q)...)
        x = Q * x
    end

    return x
end

# Обратная матрица
function invLU(A::Matrix{Float64}, pivoting::Symbol=:column)
    L, U, P, Q = LU_decomp(A, pivoting)

    n = size(L, 1)
    E = Matrix{Float64}(I, n, n)
    invA = zeros(Float64, n, n)

    for i in 1:n
        e = E[:, i]
        invA[:, i] = solveLU(L, U, P, Q, e)
    end

    return invA
end

# Число обусловленности
function condLU(A::Matrix{Float64}, pivoting::Symbol=:column)
    invA = invLU(A, pivoting)
    return norm(A) * norm(invA)
end

# Число обусловленности (более точный вариант)
function condS(A::Matrix{Float64}, pivoting::Symbol=:column)
```

```
    S = svd(A).S
    return maximum(S) / minimum(S)
end
```

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

Выполним проверку написанной функции с разными примерами.

[ ]:
```
using Printf

# Проверка LU-разложения
function check(A, L, U, P, Q)
    if Q == Matrix{Float64}(I, size(Q)...)
        println("Проверка: LU ≈ P*A: ", isapprox(L * U, P * A))
    else
        println("Проверка: LU ≈ P*A*Q: ", isapprox(L * U, P * A * Q))
    end
end

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

# Основная демонстрация
function demo(n=5, pivoting=:column)
    println("-"^50)
    println("Размерность матрицы: $n, выбор главного элемента:␣
  ↪$pivoting")
    A = randn(n, n)
    L, U, P, Q = LU_decomp(A, pivoting)

    print_matrix_str(A)
    println()
    print_matrix_str(L)
    println()
    print_matrix_str(U)
    check(A, L, U, P, Q)

    # Определитель
    detA = detLU(A)
    println("Определитель det(A): $detA ", isapprox(detA, det(A)))

    # Решение СЛАУ
    b = randn(n)
    x = solveLU(A, b)
```

```
    println("Проверка Ax ≈ b: ", isapprox(A * x, b))

    # Обратная матрица
    A_inv = invLU(A)
    E = Matrix{Float64}(I, n, n)
    println("Проверка A*A⁻¹ ≈ E: ", isapprox(A * A_inv, E))
    println("Проверка A⁻¹*A ≈ E: ", isapprox(A_inv * A, E))

    # Число обусловленности
    #A = [1. 1.; 3. 4.]
    cond_A = condLU(A)
    condS_A = condS(A)
    println("Число обусловленности condLU(A): $cond_A, condS(A):␣
  ↪$condS_A, cond(A): ", cond(A), " ", isapprox(cond_A, cond(A)))
    println("—"^50)
end

demo(5)
```

```
──────────────────────────────────────────────────
Размерность матрицы: 5, выбор главного элемента: column
     1.314     -0.304     -0.112      1.090     -0.696
    -1.844     -0.761      0.927      1.114     -0.719
     0.874      1.366     -0.906     -0.729     -1.133
     0.034     -0.069     -0.198      0.946      0.162
     0.891      0.725     -0.729      0.646      1.364

     1.000      0.000      0.000      0.000      0.000
    -0.474      1.000      0.000      0.000      0.000
    -0.018     -0.082      1.000      0.000      0.000
    -0.713     -0.842     -0.709      1.000      0.000
    -0.483      0.355      0.524      0.318      1.000

    -1.844     -0.761      0.927      1.114     -0.719
     0.000      1.005     -0.467     -0.201     -1.474
     0.000      0.000     -0.220      0.949      0.027
     0.000      0.000      0.000      2.388     -2.429
     0.000      0.000      0.000      0.000      2.298
Проверка: LU ≈ P*A: true
Определитель det(A): -2.237130824501221 true
Проверка Ax ≈ b: true
Проверка A*A⁻¹ ≈ E: true
Проверка A⁻¹*A ≈ E: true
Число обусловленности condLU(A): 30.458002229139982, condS(A):
21.82590430911236, cond(A): 21.82590430911235 false
──────────────────────────────────────────────────
```

[ ]:
```
# Основная демонстрация
function demo2(n=5, pivoting=:column)
    println("—"^50)
    println("Размерность матрицы: $n, выбор главного элемента:␣
  ↪$pivoting")
    A = randn(n, n)
    A[:, 1] = A[:, 2]
    #A[n-2, :] = A[n, :]
    #A[2, :] = A[n, :]

    L, U, P, Q, r = LU_rang(A, pivoting)

    print_matrix_str(A)
    println()
    print_matrix_str(L)
    println()
    print_matrix_str(U)
    println()
    println("Ранг матрицы: $r ", isapprox(r, rank(A)))
    check(A, L, U, P, Q)

    # Решение СЛАУ
    b = randn(n)
    x = solveLU_ex(A, b)
    if x !== nothing
        println("Проверка Ax ≈ b: ", isapprox(A * x, b))
    else
        println("Система несовместна")
    end
end

demo2(10)
```

```
——————————————————————————————————————————————————
Размерность матрицы: 10, выбор главного элемента: column
    -0.904     -0.904      0.352     -0.842     -0.777     -0.584     ␣
  ↪-0.003
0.414      0.884      0.308
    -0.858     -0.858     -0.816      0.113      0.250      1.801     ␣
  ↪-0.413
1.991     -0.205      0.137
     1.586      1.586      1.705      0.618      0.626     -0.147     ␣
  ↪1.105
-0.516     -0.775      0.054
    -1.183     -1.183     -0.353      1.473      0.808      1.915     ␣
  ↪1.665
1.218      0.688      1.148
```

```
     0.592      0.592     -0.720      0.397      0.111     -0.009
  ↪-0.582
-1.428     -0.555     -0.542
     2.125      2.125      1.783      0.452      1.535     -0.940
  ↪-1.084
1.087      0.797     -1.750
    -2.102     -2.102     -1.210      2.197      0.074     -0.860
  ↪-0.625
0.036     -0.291      0.445
    -1.572     -1.572     -0.004      0.649     -1.046     -1.777
  ↪0.292
-0.197      1.678     -1.446
     0.762      0.762      0.499      0.209      1.161     -0.266
  ↪-0.964
0.585     -0.793      0.391
     0.088      0.088     -1.254     -1.182     -0.305      0.893
  ↪0.088
-0.860      0.349      0.433

     1.000      0.000      0.000      0.000      0.000      0.000
  ↪0.000
0.000      0.000
    -0.740      1.000      0.000      0.000      0.000      0.000
  ↪0.000
0.000      0.000
     0.279      0.020      1.000      0.000      0.000      0.000
  ↪0.000
0.000      0.000
    -0.989      0.469      0.052      1.000      0.000      0.000
  ↪0.000
0.000      0.000
     0.746     -0.496     -0.826      0.450      1.000      0.000
  ↪0.000
0.000      0.000
    -0.557     -0.064     -0.582      0.891     -0.061      1.000
  ↪0.000
0.000      0.000
    -0.404      0.208      0.297      0.008     -0.647      0.893
  ↪1.000
0.000      0.000
    -0.425      0.159     -0.725     -0.288     -0.056     -0.286
  ↪0.168
1.000      0.000
     0.359     -0.372     -0.280      0.223     -0.143     -0.323
  ↪-0.386
0.516      1.000
```

```
     0.042     -0.184      0.873     -0.571     -0.569     -0.117
  ↪-0.674
-0.543     -0.169

     2.125     -1.750      1.783      0.452      1.535     -0.940
  ↪-1.084
1.087      0.797      2.125
     0.000     -2.740      1.315      0.984      0.090     -2.472
  ↪-0.510
0.607      2.267      0.000
     0.000      0.000     -1.243      0.252     -0.319      0.301
  ↪-0.270
-1.743     -0.822      0.000
     0.000      0.000      0.000      2.170      1.566     -0.645
  ↪-1.444
0.916     -0.525      0.000
     0.000      0.000      0.000      0.000     -1.443     -0.133
  ↪2.087
-2.877     -0.687      0.000
    -0.000     -0.000      0.000      0.000      0.000      1.977
  ↪2.284
-0.143      1.224     -0.000
     0.000      0.000      0.000      0.000      0.000      0.000
  ↪-1.342
1.082     -1.643      0.000
     0.000      0.000      0.000      0.000      0.000      0.000
  ↪0.000
-0.604      0.702     -0.000
     0.000      0.000      0.000      0.000      0.000      0.000
  ↪0.000
0.000     -1.046      0.000

Ранг матрицы: 9 true
Проверка: LU ≈ P*A*Q: true
Система несовместна
```