[ ]:
```julia
using QuadGK
using ForwardDiff
using LinearAlgebra

function KF(f, p, a, b, α, β)

    x1, x2, x3 = a, (a + b) / 2, b
    nodes = [x1, x2, x3]

    # Моменты
    μ0 = quadgk(x -> p(x), a, b; rtol=1e-12)[1]
    μ1 = quadgk(x -> x * p(x), a, b; rtol=1e-12)[1]
    μ2 = quadgk(x -> x^2 * p(x), a, b; rtol=1e-12)[1]

    # Альтернативный способ: через явные формулы
    z1, z2, z3 = x1, x2, x3
    A1 = (μ2 - μ1*(z2 + z3) + μ0*z2*z3) / ((z2 - z1)*(z3 - z1))
    A2 = -(μ2 - μ1*(z1 + z3) + μ0*z1*z3) / ((z2 - z1)*(z3 - z2))
    A3 = (μ2 - μ1*(z2 + z1) + μ0*z2*z1) / ((z3 - z2)*(z3 - z1))

    A = [A1, A2, A3]  # заменяет решение СЛАУ

    approx_integral = sum(A[i] * f(nodes[i]) for i in 1:3)
    exact_integral = quadgk(x -> f(x) * p(x), a, b; rtol=1e-12)[1]

    # Методическая и точная погрешность
    f3(x) = ForwardDiff.derivative(x -> ForwardDiff.derivative(
                x -> ForwardDiff.derivative(f, x), x), x)

    m_error = method_error(f3, p, a, b, α, β, nodes)
    abs_error = abs(exact_integral - approx_integral)

    return A, approx_integral, exact_integral, m_error, abs_error
end
```

[ ]: KF (generic function with 2 methods)

[ ]:
```julia
function method_error(f_n::Function, p, a, b, α, β, nodes)

    n = length(nodes)
    ω(x) = prod(x - i for i in nodes)
    fun(x) = abs(p(x) * ω(x))

    M_n = maximum([abs(f_n(x)) for x in range(a, b, length=100)])

    I = quadgk(fun, a, b)[1]
```

```
    return M_n * I / factorial(n)
end
```

[ ]: method_error (generic function with 2 methods)

[ ]:
```
function comp_KF(f, p, a, b, α, β, n)
    h = (b - a) / n
    total = 0.0

    for i in 0:n-1
        xl = a + i*h
        xr = xl + h
        approx = KF(f, p, xl, xr, α, β)[2]
        total += approx
        #println("Проверка $n: $total")
    end

    return total
end
```

[ ]: comp_KF (generic function with 2 methods)

[ ]:
```
function adapt_KF(f, p, a, b, α, β; eps=1e-6, max_n = 1_000_000, n_0␣
↪= 1)
    results = []
    ns = n_0 * [1, 2, 4]

    for n in ns
        push!(results, comp_KF(f, p, a, b, α, β, n))
    end

    while true
        n_new = ns[end]*2
        I_new = comp_KF(f, p, a, b, α, β, n_new)
        push!(results, I_new)
        push!(ns, n_new)

        # Эйткен (по 3 последним)
        L = 2  # во сколько раз уменьшается шаг
        S1, S2, S3 = results[end-2], results[end-1], results[end]

        r = (S3 - S2) / (S2 - S1)
        m = -log(abs(r)) / log(L)
        println("Скорость сходимости m ≈ $m")

        # Ричардсон
        err = abs(results[end] - results[end-1]) / (2^m - 1)
```

```
        println("n = $n_new, I ≈ $(results[end]), ошибка ≈ $err")

        if abs(err) < eps
            return results[end]
        end

        if n_new > max_n
            error("Не удалось достичь требуемой точности с n ≤␣
↪$max_n")
        end
    end
end
```

[ ]: adapt_KF (generic function with 2 methods)

[ ]:
```
function hopt_KF(f, p, a, b, α, β; eps=1e-6)
    ns = [1, 2, 4]
    results = [comp_KF(f, p, a, b, α, β, n) for n in ns]
    hs = [(b - a) / n for n in ns]
    L = 2

    S1, S2, S3 = results
    h1, h2, h3 = hs

    Δ1 = S2 - S1
    Δ2 = S3 - S2

    if iszero(Δ2 - Δ1)
        error("Недостаточная точность на грубых сетках — невозможно␣
↪оценить скорость сходимости.")
    end

    # Эйткен
    r = Δ2 / Δ1
    m = -log(abs(r)) / log(L)
    println("Скорость сходимости m ≈ $m")

    if abs(Δ2) < 1e-14
        error("Разность между S3 и S2 слишком мала: err = 0, hopt =␣
↪∞")
    end

    # Ричардсон
    err = abs(Δ2) / (L^m - 1)
    println("Ошибка на грубой сетке ≈ $err")

    hopt = h3 * (eps / err)^(1 / m)
```

```julia
    if !isfinite(hopt) || hopt <= 0
        error("Получен неверный hopt: $hopt.")
    end

    nopt = Int(ceil((b - a) / hopt))
    nopt += nopt % 2  # сделать чётным
    println("Оптимальный шаг hopt ≈ $hopt, nopt = $nopt")

    return nopt, hopt
end
```

[ ]: hopt_KF (generic function with 2 methods)

[ ]:
```julia
using QuadGK
using LinearAlgebra
using Polynomials

# Вычисление моментов
function get_moments(p, a, b, α, β, maxk)
    μ = zeros(maxk+1)
    for k in 0:maxk
        μ[k+1] = quadgk(x -> x^k * p(x,a,b,α,β), a, b; rtol=1e-12)[1]
    end
    return μ
end

# Построение матрицы Якоби из моментов
function build_J(μ)
    n = 3
    H = [μ[i+j-1] for i in 1:n, j in 1:n]
    H1 = [μ[i+j] for i in 1:n, j in 1:n]

    λ, V = eigen(H \ H1)

    nodes = sort(λ)
    return nodes
end

# Вычисление весов
function get_weights(nodes, μ)
    n = length(nodes)
    A = zeros(n)

    # Система для весов
    V = [nodes[j]^(i-1) for i in 1:n, j in 1:n]
    A = V \ μ[1:n]
```

```
    return A
end

function gauss_KF(f, p, a, b, α, β)
    μ = get_moments(p, a, b, α, β, 5)
    nodes = build_J(μ)
    weights = get_weights(nodes, μ)

    approx = sum(weights[i] * f(nodes[i]) for i in 1:3)
    return weights, approx
end
```

[ ]: gauss_KF (generic function with 2 methods)

```
[ ]: f(x) = 4.5 * cos(7x) * exp(-2x/3) + 1.4 * sin(1.5x) * exp(-x/3) + 3
a = 2.1
b = 3.3
α = 2/5
β = 0
p(x) = (x - a)^(-α) * (b - x)^(-β)

A, approx, exact, m_err, abs_err = KF(f, p, a, b, α, β)

println("Коэффициенты квадратурной формулы: ", A)
println("Приближённое значение интеграла: ", approx)
println("Точное значение интеграла: ", exact)
println("Методическая погрешность: ", m_err)
println("Точная погрешность: ", abs_err)
println()

approx = adapt_KF(f, p, a, b, α, β)
println("Приближённое значение интеграла с точностью 1e-6: ", approx)
println()

nopt = hopt_KF(f, p, a, b, α, β)[1]
println()

approx = adapt_KF(f, p, a, b, α, β, n_0 = nopt)
println("Приближённое значение интеграла с точностью 1e-6 с nopt: ",
  ↪approx)
println()

approx = gauss_KF(f, p, a, b, α, β)[2]
println("Приближённое значение интеграла по формулам Гаусса: ",
  ↪approx)
```

```
Коэффициенты квадратурной формулы: [0.6257375278022516, 1.
  ↪0726929055094603,
```

```
0.1609039358264127]
Приближённое значение интеграла: 5.410653820993932
Точное значение интеграла: 4.461512704342535
Методическая погрешность: 4.821231045047852
Точная погрешность: 0.949141116651397

Скорость сходимости m ≈ 6.833717584236995
n = 8, I ≈ 4.46120525773206, ошибка ≈ 2.343137960005599e-6
Скорость сходимости m ≈ -0.008328282769484933
n = 16, I ≈ 4.461471719370319, ошибка ≈ -0.04629208040399076
Скорость сходимости m ≈ 2.860082149629102
n = 32, I ≈ 4.461508419210803, ошибка ≈ 5.862063687647671e-6
Скорость сходимости m ≈ 3.2422474222710287
n = 64, I ≈ 4.461512297591612, ошибка ≈ 4.582927707834388e-7
Приближённое значение интеграла с точностью 1e-6: 4.461512297591612

Скорость сходимости m ≈ 4.927307901635068
Ошибка на грубой сетке ≈ 0.0010268972504532102
Оптимальный шаг hopt ≈ 0.07343956623110538, nopt = 18

Скорость сходимости m ≈ 3.4059937448142055
n = 144, I ≈ 4.461512680227917, ошибка ≈ 2.5699793730270305e-8
Приближённое значение интеграла с точностью 1e-6 с nopt: 4.
  ↪461512680227917

Приближённое значение интеграла по формулам Гаусса: 4.608624872632719
```