Assignment 4: Programming in Julia

Chuyang Wang

December 8 2023

1 Codes

1.1 Main.jl

```julia
using LinearAlgebra
using Plots

struct SparseMatrixCSR{T} <: AbstractMatrix{T}
    rows::Vector{Int64}
    cols::Vector{Int64}
    values::Vector{T}
    function SparseMatrixCSR{T}(rows::Vector{Int64}, cols::Vector{Int64}, values::Vector{T})
        new{T}(rows, cols, values)
    end
end

import Base: size, getindex

function size(A::SparseMatrixCSR)
    (maximum(A.rows), maximum(A.cols))
end

function getindex(A::SparseMatrixCSR{T}, i::Int, j::Int) where T
    for k in 1:length(A.rows)
        if A.rows[k] == i && A.cols[k] == j
            return A.values[k]
        end
    end
    zero(T)
end

import LinearAlgebra: mul!

function mul!(out::AbstractVector, A::SparseMatrixCSR, x::AbstractVector)
    fill!(out, 0)
    for k in 1:length(A.values)
        out[A.rows[k]] += A.values[k] * x[A.cols[k]]
    end
    out
end

function iterate(A, b, u, a; tol = 1e-3, max_iters = 1000)
    iterations = 0
    norm_r = Inf
    while true
        r = b - A * u
        norm_r = norm(r)
        if norm_r < tol || iterations >= max_iters
            break
        end
        u += a * r
        iterations += 1
    end
    (u, iterations, norm_r)
end

function create_tridiagonal_matrix(n::Int)
    rows, cols, values = Int64[], Int64[], Float64[]
    h, a = 1.0 / n, 0.5 * h^2
    for i in 1:n+1
        if i > 1
            push!(rows, i)
            push!(cols, i-1)
            push!(values, -a)
        end
        push!(rows, i)
        push!(cols, i)
        push!(values, 2*a)
        if i < n+1
            push!(rows, i)
            push!(cols, i+1)
            push!(values, -a)
        end
    end
    SparseMatrixCSR{Float64}(rows, cols, values)
end

function test_and_time(n::Int)
    A_sparse = create_tridiagonal_matrix(n)
    b = [cos( * i * 1/n) for i in 0:n]
    initial_u = zeros(n+1)
    @time final_u_sparse, _, _ = iterate(A_sparse, b, initial_u, 0.5 * (1/n)^2)
    p = plot(0:n, initial_u, label="Initial u", title="Initial vs Final Solutions u", xlabe
    plot!(p, 0:n, final_u_sparse, label="Final u (Sparse)")
    display(p)
    savefig(p, "plot_result.png")
end

println("Starting test...")
test_and_time(100)
println("Test completed.")

n = 5
A_sparse = create_tridiagonal_matrix(n)
out = zeros(n + 1)
x = rand(n + 1)
b = [cos( * i / n) for i in 0:n]
u = zeros(n + 1)
a = 0.5 * (1 / n)^2

@code_warntype mul!(out, A_sparse, x)
@code_warntype iterate(A_sparse, b, u, a)
```

1.2 $Main_{o}de.jl$

```
using DifferentialEquations, LinearAlgebra, Plots, SparseArrays

struct SparseMatrixCSR{T} <: AbstractMatrix{T}
    rows::Vector{Int64}
    cols::Vector{Int64}
    values::Vector{T}
end

function size(A::SparseMatrixCSR)
    return (maximum(A.rows), maximum(A.cols))
end

function getindex(A::SparseMatrixCSR{T}, i::Int, j::Int) where T
    for k in 1:length(A.rows)
        if A.rows[k] == i && A.cols[k] == j
            return A.values[k]
        end
    end
    return zero(T)
end
```

```
function mul!(y::Vector{T}, A::SparseMatrixCSR{T}, x::Vector{T}) where T
    fill!(y, zero(T))
    for k in 1:length(A.values)
        y[A.rows[k]] += A.values[k] * x[A.cols[k]]
    end
    return y
end

function create_tridiagonal_matrix(n::Int)
    rows = []
    cols = []
    values = []
    h = 1.0 / n
    a = -2.0 * h^2
    for i = 1:n
        push!(rows, i)
        push!(cols, i)
        push!(values, a)
        if i > 1
            push!(rows, i)
            push!(cols, i-1)
            push!(values, h^2)
        end
        if i < n
            push!(rows, i)
            push!(cols, i+1)
            push!(values, h^2)
        end
    end
    return SparseMatrixCSR{Float64}(rows, cols, values)
end

function rhs!(du, u, p, t)
    mul!(du, p.A, u)
    du .*= -1
    du .+= p.b
end

n = 100
h = 1.0 / n
A = create_tridiagonal_matrix(n)
b = [cos( * i / n) for i in 1:n]
u0 = zeros(n)

p = (A = A, b = b, h = h)
```

```
tspan = (0.0, 1.0)

eigen_estimation = integrator -> 2 / (integrator.p.h^2)

prob = ODEProblem(rhs!, u0, tspan, p)

solvers = [
    Tsit5(),
    SSPRK33(),
    ROCK4(eigen_est = eigen_estimation)
]

animations = Dict()

for solver in solvers
    sol = solve(prob, solver, dt = h^2, saveat = 0.1, abstol = 1e-6, reltol = 1e-3)

    anim = @animate for (i, u) in enumerate(sol.u)
        plot(u, ylims=(-1, 1), label="u(t)", title="Solver: $(typeof(solver)) at time t=$(ro
    end

    animation_filename = "animation_$(typeof(solver)).gif"
    gif(anim, animation_filename, fps = 15)
    animations[typeof(solver)] = animation_filename

    println("Solver: ", typeof(solver))
    println("Number of accepted steps: ", sol.destats.naccept)
    println("Number of rejected steps: ", sol.destats.nreject)
    println("Number of function evaluations: ", sol.destats.nf)

    println("Solver: ", typeof(solver))
    println("Number of time steps: ", length(sol.t))
end

println("ODE solving and animations completed.")
```

2 Project.toml

```
    [deps]
DifferentialEquations = "0c46a032-eb83-5123-abaf-570d42b7fbaa"
LinearAlgebra = "37e2e46d-f89d-539d-b4ee-838fcccc9c8e"
Plots = "91a5bcdd-55d7-5caf-9e0b-520d859cae80"
SparseArrays = "2f01184e-e22b-5df5-ae63-d93ebab69eaf"
```

3 outputs

3.1 For main.jl

```
Starting test...
Creating tridiagonal matrix...
Running iterate with sparse matrix...
  0.010338 seconds (4.00 k allocations: 3.420 MiB)
Plotting results...
Test completed.
MethodInstance for LinearAlgebra.mul!(::Vector{Float64}, ::SparseMatrixCSR{Float64}, ::Vect
  from mul!(y::Vector{T}, A::SparseMatrixCSR{T}, x::Vector{T}) where T @ Main ~/homework-4/
Static Parameters
  T = Float64
Arguments
  #self#::Core.Const(LinearAlgebra.mul!)
  y::Vector{Float64}
  A::SparseMatrixCSR{Float64}
  x::Vector{Float64}
Locals
  @_5::Union{Nothing, Tuple{Int64, Int64}}
  k::Int64
Body::Vector{Float64}
1  %1  = Main.zero($(Expr(:static_parameter, 1)))::Core.Const(0.0)
         Main.fill!(y, %1)
   %3  = Base.getproperty(A, :values)::Vector{Float64}
   %4  = Main.length(%3)::Int64
   %5  = (1:%4)::Core.PartialStruct(UnitRange{Int64}, Any[Core.Const(1), Int64])
         (@_5 = Base.iterate(%5))
   %7  = (@_5 === nothing)::Bool
   %8  = Base.not_int(%7)::Bool
         goto #4 if not %8
2  %10 = @_5::Tuple{Int64, Int64}
         (k = Core.getfield(%10, 1))
   %12 = Core.getfield(%10, 2)::Int64
   %13 = Base.getproperty(A, :rows)::Vector{Int64}
   %14 = Base.getindex(%13, k)::Int64
   %15 = Base.getindex(y, %14)::Float64
   %16 = Base.getproperty(A, :values)::Vector{Float64}
   %17 = Base.getindex(%16, k)::Float64
   %18 = Base.getproperty(A, :cols)::Vector{Int64}
   %19 = Base.getindex(%18, k)::Int64
   %20 = Base.getindex(x, %19)::Float64
   %21 = (%17 * %20)::Float64
   %22 = (%15 + %21)::Float64
         Base.setindex!(y, %22, %14)
         (@_5 = Base.iterate(%5, %12))
   %25 = (@_5 === nothing)::Bool
   %26 = Base.not_int(%25)::Bool
       goto #4 if not %26
3        goto #2
4        return y

MethodInstance for iterate(::SparseMatrixCSR{Float64}, ::Vector{Float64}, ::Vector{Float64},
  from iterate(A, b, u, a; tol, max_iters) @ Main ~/homework-4/main.jl:43
Arguments
  #self#::Core.Const(iterate)
  A::SparseMatrixCSR{Float64}
  b::Vector{Float64}
  u::Vector{Float64}
  a::Float64
Body::Tuple{Vector{Float64}, Int64, Float64}
1  %1 = Main.:(var"#iterate#100")(0.001, 1000, #self#, A, b, u, a)::Tuple{Vector{Float64}, I
      return %1
```

3.2 *Formain − ode.jl*

```
    [ Info: Saved animation to /home/sunny/homework-4/animation_Tsit5{typeof(OrdinaryDiffEq.
Solver: Tsit5{typeof(OrdinaryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limiter
Number of accepted steps: 5
Number of rejected steps: 0
Number of function evaluations: 31
Solver: Tsit5{typeof(OrdinaryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limiter
Number of time steps: 11
[ Info: Saved animation to /home/sunny/homework-4/animation_SSPRK33{typeof(OrdinaryDiffEq.tr
Solver: SSPRK33{typeof(OrdinaryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limit
Number of accepted steps: 10001
Number of rejected steps: 0
Number of function evaluations: 30003
Solver: SSPRK33{typeof(OrdinaryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limit
Number of time steps: 11
[ Info: Saved animation to /home/sunny/homework-4/animation_ROCK4{var"#107#108"}.gif
Solver: ROCK4{var"#107#108"}
Number of accepted steps: 5
Number of rejected steps: 0
Number of function evaluations: 26
Solver: ROCK4{var"#107#108"}
Number of time steps: 11
ODE solving and animations completed.
```

4 Plot

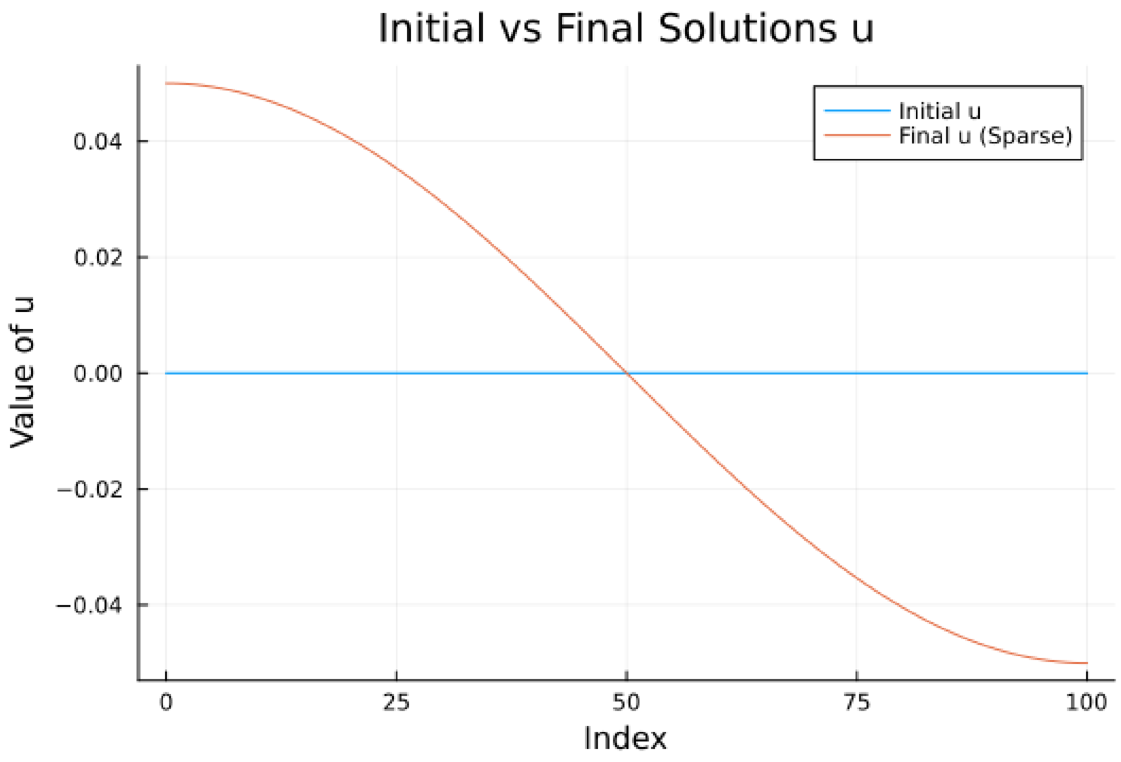

Figure 1: initial and final solutions u