CIS 410/510 HW 3 - Brett Sumser

The problem we will be attempting to solve for our group project will be focused on heat diffusion rate from liquids into different materials of cups. Different cup materials will be compared in their rate of heat diffusion when containing heated liquids. This problem has practical applications towards material science and many applications where fast diffusion of heat is critical.

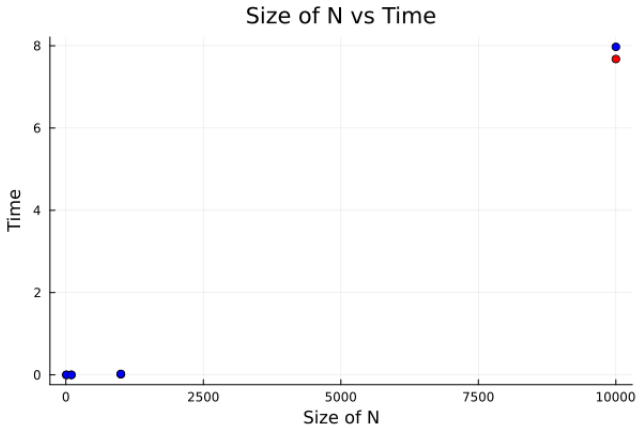
Our plan of action will be to start in 1d as detailed in the project guidelines, and eventually simplify calculation of 3d space into a 2d cylindrical system without rotation. As a more in depth goal we could possibly extend this project to compare more complicated solids of rotation, or even find a shape to minimize the rate of heat loss.

$\frac{\Delta Q}{\Delta t} = -k\frac{A}{d}(T - T_{room})$

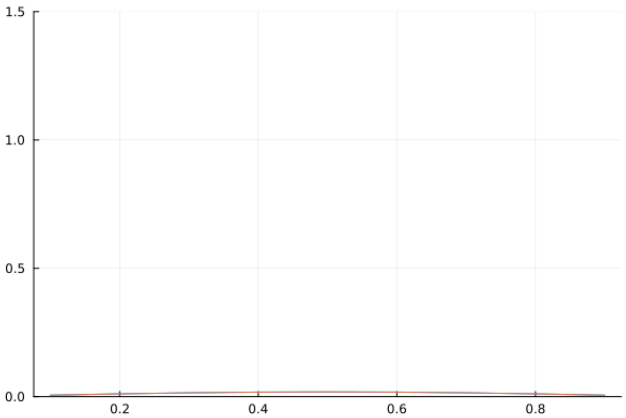
Where k is the thermal conductivity of the cup wall material, A is the surface area of the cup wall, d is the thickness of the cup wall, and T and Troom are the current temperature of the coffee and the temperature of the room respectively. In our system we'll be assuming the cup is a circle. With the boundary condition being 27 degrees on most of the wall but 40 degrees on a more narrow partition of the wall, this could be interpreted as the cup is heated from one side.

For our spatial discretization, we will be using finite differences in polar coordinates. Then, for our temporal stepping, we'll be using the explicit Euler's method, in which the equation above will be used to calculate the next time step. This equation can calculate the temperature change of the inner fluid where the adjacent nodes are interpreted as walls with a different k value from the coffee cup's walls.

For the second part of this assignment, we compared the time it takes to perform an LUP solve on a dense vs sparse matrix of different sizes N. Based on the below graph, the matrices had similar performance to each other up until $N = 10000$. At that point the sparse matrix starts taking noticibly more time to calculate the LP solve. I attempted to use a larger size for N, but ran into memory errors. Based on the data I collected, I would assume that the time difference would become even more drastic at larger values of N.

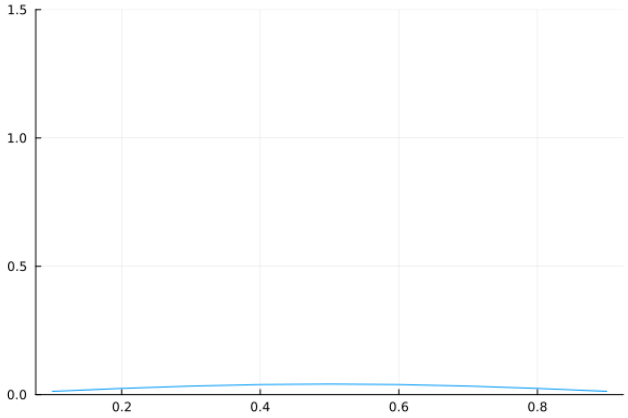


For the third part of this assignment, we solved the heat equation numerically by arriving at an initial value problem and using the forward euler method to solve. The numerical solution is in line with the exact solution extremely closely.



For part 3c, we used different time steps to examine for which Δt the solution was numerically stable. When

$\Delta t = 10^-1$ or 10^-2, the graph would start out similarly to how it developes in 3b, but would quickly become numerically unstable, not representing the actual solution. However for a $\Delta t = 10^-3$, the graph would track the actual solution of $u(x,t) = e^{2t}\sin(x)$ nicely. I would assume that numerical stability is extended for smaller values of Δt



```
using LinearAlgebra
using Plots
using Printf # for formatting text output

"""
    var_set(N)

Initializes banded sparse matrix, and vector b of size N


A = -2 1 0 .... 0
     0-2 1 0 .. 0
     0 0-2 1 0  0
"""
function var_set(N)
    A = Matrix{Float64}(undef, N, N)
    A = zeros(N,N)

    A_d = Matrix{Float64}(undef, N, N)
    A_d = ones(N,N)
    for i = 1:N
        A_d[i,i] = -2
    end

    b = rand(N,1)
    b = b[:]

    for i = 1:N
        b[i] = 1
    end

    for i = 1:N
        A[i,i] = -2
        if i != N
            A[i, i+1] = 1
            A[i+1, i] = 1
        end
    end

    return (A, A_d, b)
end


"""
    LU_solve(A, b)

Uses Julia linear algebra package to solve Ax = b
given Matrix A and vector b
"""
function LU_solve(A, b)
    F = lu(A)
    # display(F)

    x = F\b

    # display(x)
    return x
end
```

```
function main()
    dense_time = []
    sparse_time = []
    testSizes = [10, 100, 1000, 10000]
    @printf("starting loop for values")
    for i = 1:size(testSizes,1)
        (A, A_d, b) = var_set(testSizes[i])
        temp = @timed x = LU_solve(A, b)
        push!(sparse_time, temp[2])
        new_temp = @timed x = LU_solve(A_d, b)
        push!(dense_time, new_temp[2])
        @printf("done at N = %d\n", testSizes[i])
    end
    scatter(testSizes, dense_time, color="red", xlabel="Size of N", ylabel="Time", tit
    scatter!(testSizes, sparse_time, color="blue", legend=false)
    savefig("denseSparsePlot.png")
end

main()
```

```
using LinearAlgebra
using Plots
using Printf # for formatting text output

"""
    var_set(N)

Initializes banded sparse matrix, and vector b of size N


A = -2 1 0 .... 0
     0-2 1 0 .. 0
     0 0-2 1 0  0
"""
function var_set(N)
    A = Matrix{Float64}(undef, N, N)
    A = zeros(N,N)

    A_d = Matrix{Float64}(undef, N, N)
    A_d = ones(N,N)
    for i = 1:N
        A_d[i,i] = -2
    end

    b = rand(N,1)
    b = b[:]

    for i = 1:N
        b[i] = 1
    end

    for i = 1:N
        A[i,i] = -2
        if i != N
            A[i, i+1] = 1
            A[i+1, i] = 1
```

```
        end
    end

    return (A, A_d, b)
end


"""
    LU_solve(A, b)

Uses Julia linear algebra package to solve Ax = b
given Matrix A and vector b
"""
function LU_solve(A, b)
    F = lu(A)
    # display(F)

    x = F\b

    # display(x)
    return x
end

function main()
    dense_time = []
    sparse_time = []
    testSizes = [10, 100, 1000, 10000]
    @printf("starting loop for values")
    for i = 1:size(testSizes,1)
        (A, A_d, b) = var_set(testSizes[i])
        temp = @timed x = LU_solve(A, b)
        push!(sparse_time, temp[2])
        new_temp = @timed x = LU_solve(A_d, b)
        push!(dense_time, new_temp[2])
        @printf("done at N = %d\n", testSizes[i])
    end
    scatter(testSizes, dense_time, color="red", xlabel="Size of N", ylabel="Time", tit
    scatter!(testSizes, sparse_time, color="blue", legend=false)
    savefig("denseSparsePlot.png")
end

main()


using Plots

# write forward Euler for the IVP system y' = f(t, y)
# where y is a vector in R^n (i.e. has n components)


function my_forward_euler_linear(t0, Tf,  t , y0, A, b)

    # y0 has N components
    N = length(y0)
    display(y0)
    show(stdout, N)

    M = Integer(Tf/ t )  # M+1 total temporal nodes
```

```
    t = Vector{Float64}(undef, M+1)
    y = Matrix{Float64}(undef, N, M+1)

    # fill in the initial condition:
    t[1] = t0
    y[:, 1] = y0

    for n = 1:M # take M time steps
        y[:, n+1] = y[:, n] +  t *(A*y[:, n] + b(t[n]))
        t[n+1] = t[n] +  t
    end

    return (t, y)
end

function my_forward_euler(t0, Tf,  t , y0, f)

    # y0 has N components
    N = size(y0)

    M = Integer(Tf/ t )  # M+1 total temporal nodes

    t = Vector{Float64}(undef, M+1)
    y = Matrix{Float64}(undef, N, M+1)

    # fill in the initial condition:
    t[1] = t0
    y[:, 1] = y0

    for n = 1:M # take N time steps
        y[:, n+1] = y[:, n] +  t *f(t[n],y[:, n])
        t[n+1] = t[n] +  t
    end

    return (t, y)
end

function var_set(N)
    A = Matrix{Float64}(undef, N, N)
    A = rand(N,N)

    y = rand(N,1)
    y = y[:]

    for i = 1:N
        y[i] = 0
    end

    return (A, y)
end


#=
# Write forward Euler to solve the linear system IVP:
# y' = Ay + b on 0     t     Tf
# with initial y0
# Do this on particular example, where A = [-3 13; -5 -1]
# y0 = [3; -10]
t0 = 0
```

```
y0 = [3; -10]
A = [-3 13;-5 -1]
Tf = 10
 t  = 1
N = Integer(Tf/ t ) # N+1 total temporal nodes
y = Matrix{Float64}(undef, 2, N+1)
t = Vector{Float64}(undef, N+1)
t[1] = 0
# fill in initial condition:
y[:, 1] = y0
for n = 1:N  # take N time steps
    y[:, n+1] = y[:, n] +  t *A*y[:, n]  # Forward Euler
    t[n+1] = t[n] +  t
end
#plot(t, y[1, :])  # plot first component of solution vector
#plot!(t, y[2, :])  # plot second component
# plot initial condition:
p = plot([y[1, 1]], [y[2, 1]], marker=(:circle,5), color = :blue, legend = false)
for n = 2:N+1
    p = plot!([y[1, n]], [y[2, n]], marker=(:circle,5), color = :blue, legend = false)
    display(p)
    sleep(1)
end
=#
```