HW #3 Computational

Kevin Corcoran

April 24, 2021

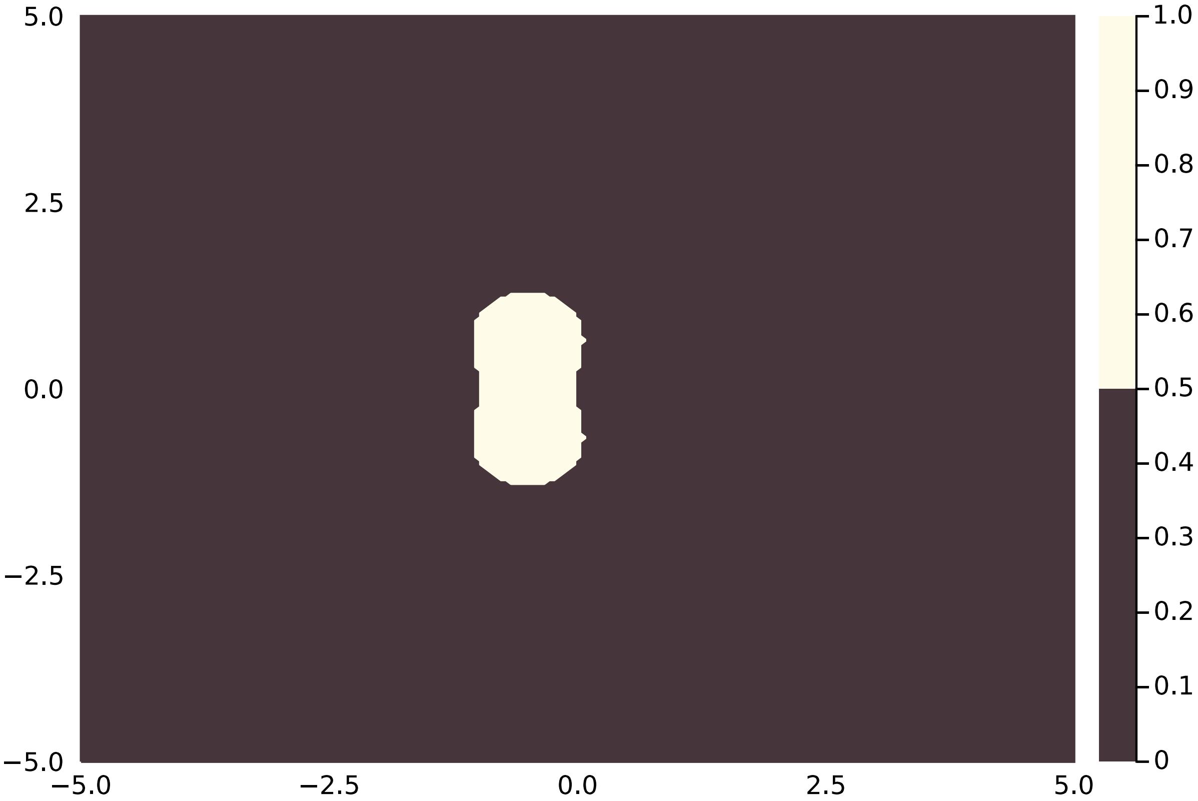
1 Including required packages

```
using Plots
using LaTeXStrings
theme(:mute)

using Pkg
Pkg.activate("RAS")
include("code/RAS.jl") # Makes sure the module is run before using it
using .RAS: RAS_stabf, RASrk

Pkg.activate("DiffyQ")
include("code/DiffyQ.jl") # Makes sure the module is run before using it
using .DiffyQ: s2_DIRK, BackwardEuler_n
```

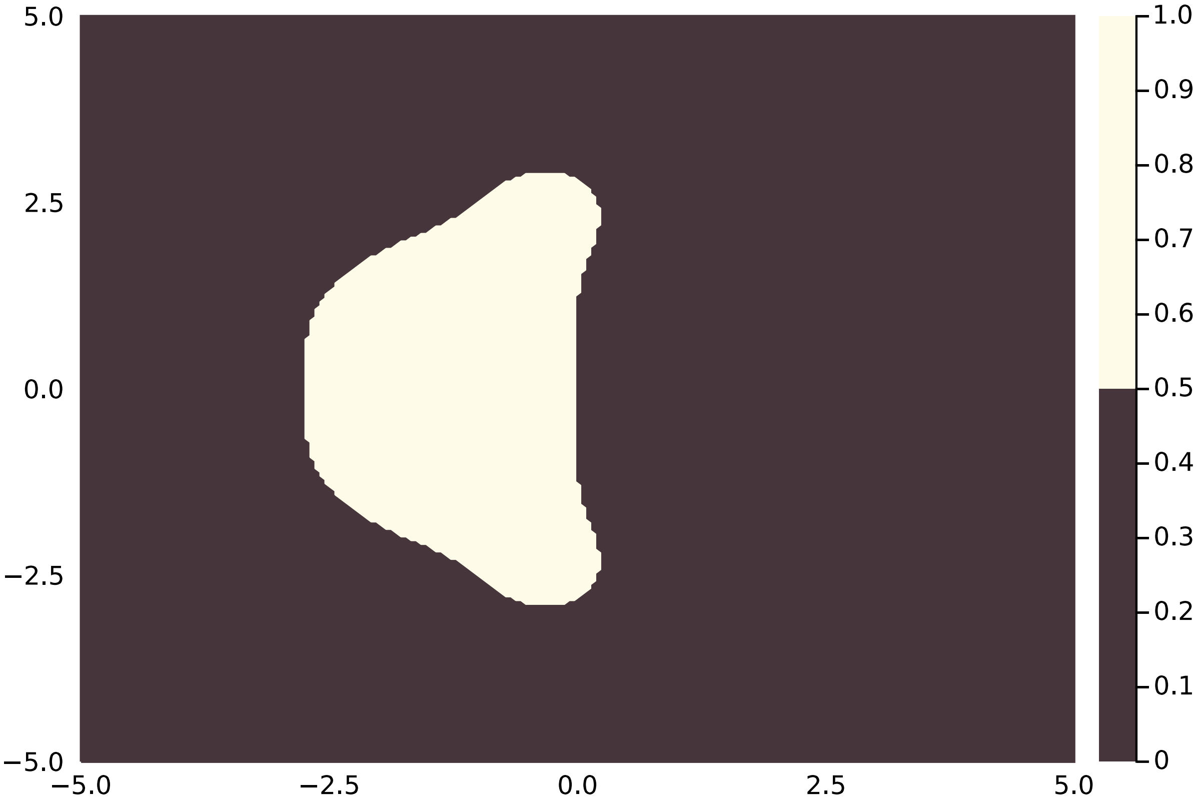
2 Problem 4: Plot Region of Absolute Stability

Note that ”light” color is the region of stability

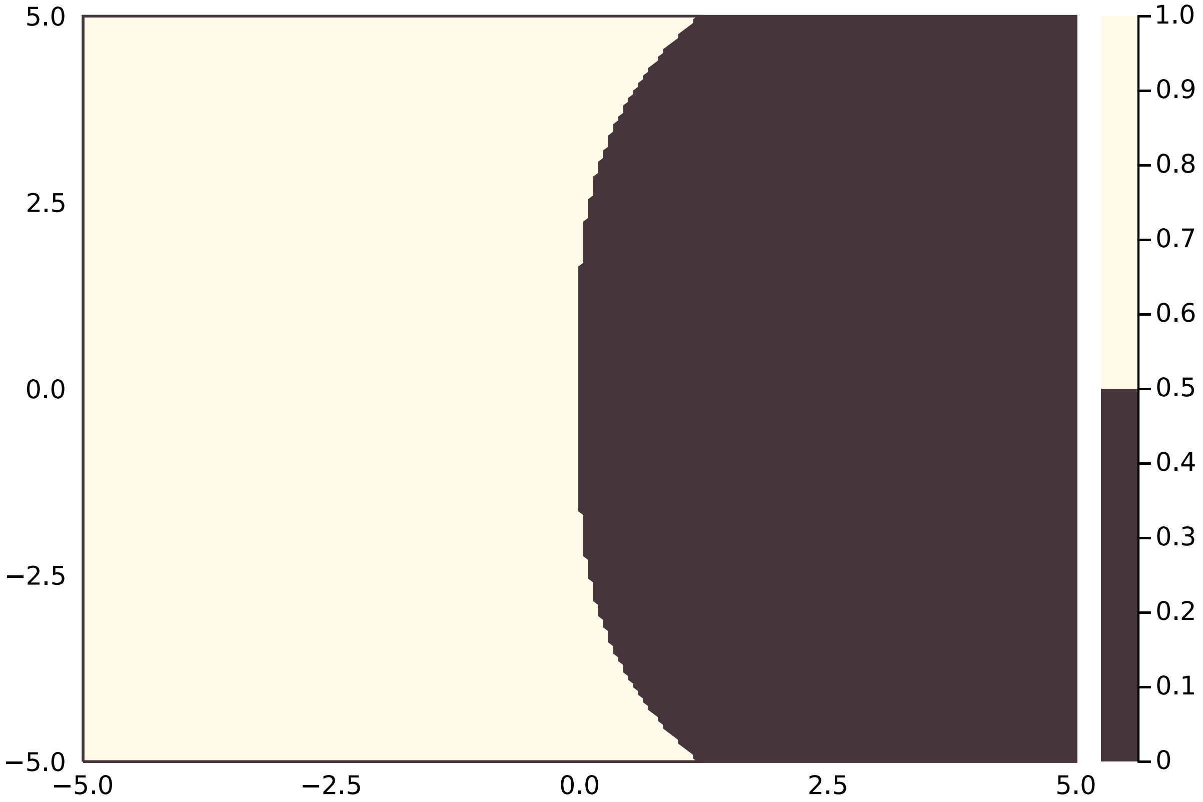
```
# stability function for Heun's
Φ(z) = 1.0 + z + z^2
xs, Z = RAS_stabf(Φ)

contourf(xs, xs, Z, levels = 1)
```



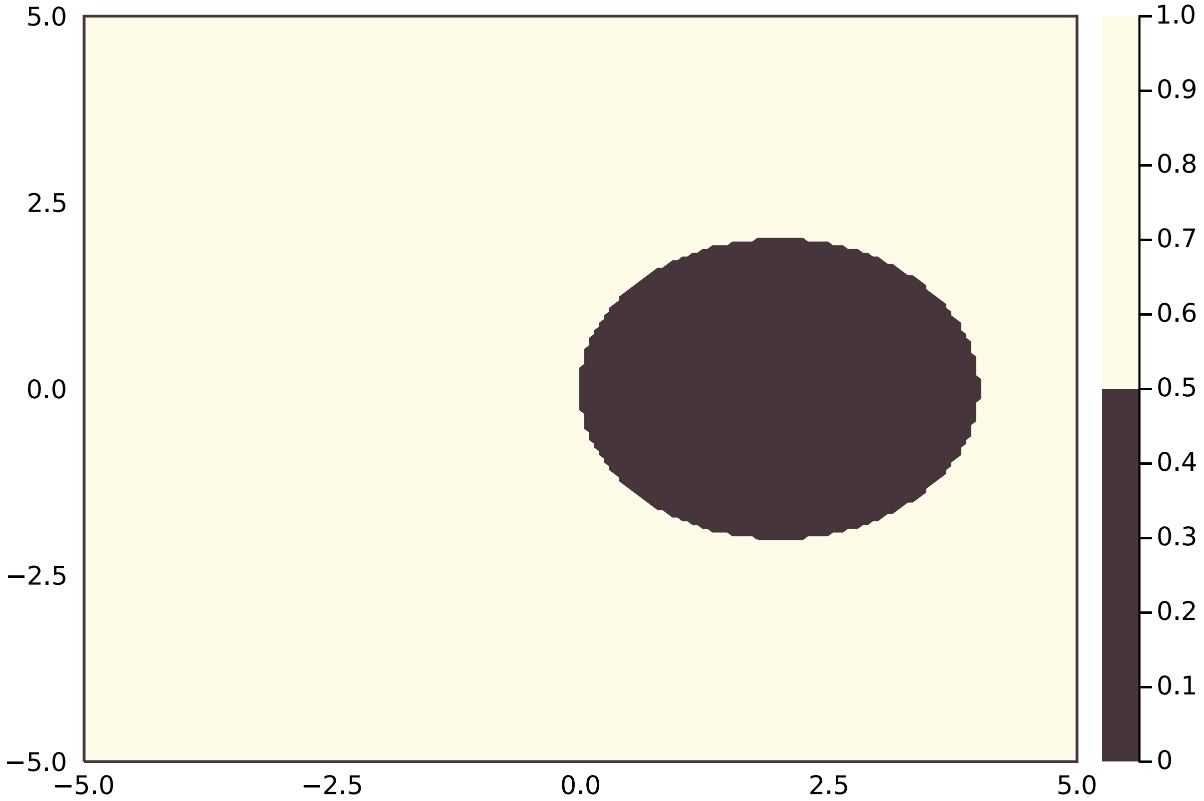
```
# RK4
A = [0 0 0 0
    1/2 0 0 0
    0 1/2 0 0
    0 0 1 0]
b = [1/6, 1/3, 1/3, 1/6]

xs, Z = RASrk(A,b)
# plotly()
contourf(xs,xs,Z, levels = 1)
```



```
# 2s-DIRK
α = 1-1/sqrt(2)
A = [α 0
    1-α α]
b = [1-α, α]

xs, Z = RASrk(A,b)
contourf(xs,xs,Z, levels = 1)
```



```
# 2s-DIRK
α = 0.5
A = [α 0
    1-α α]
b = [1-α, α]

xs, Z = RASrk(A,b)
contourf(xs,xs,Z, levels = 1)
```

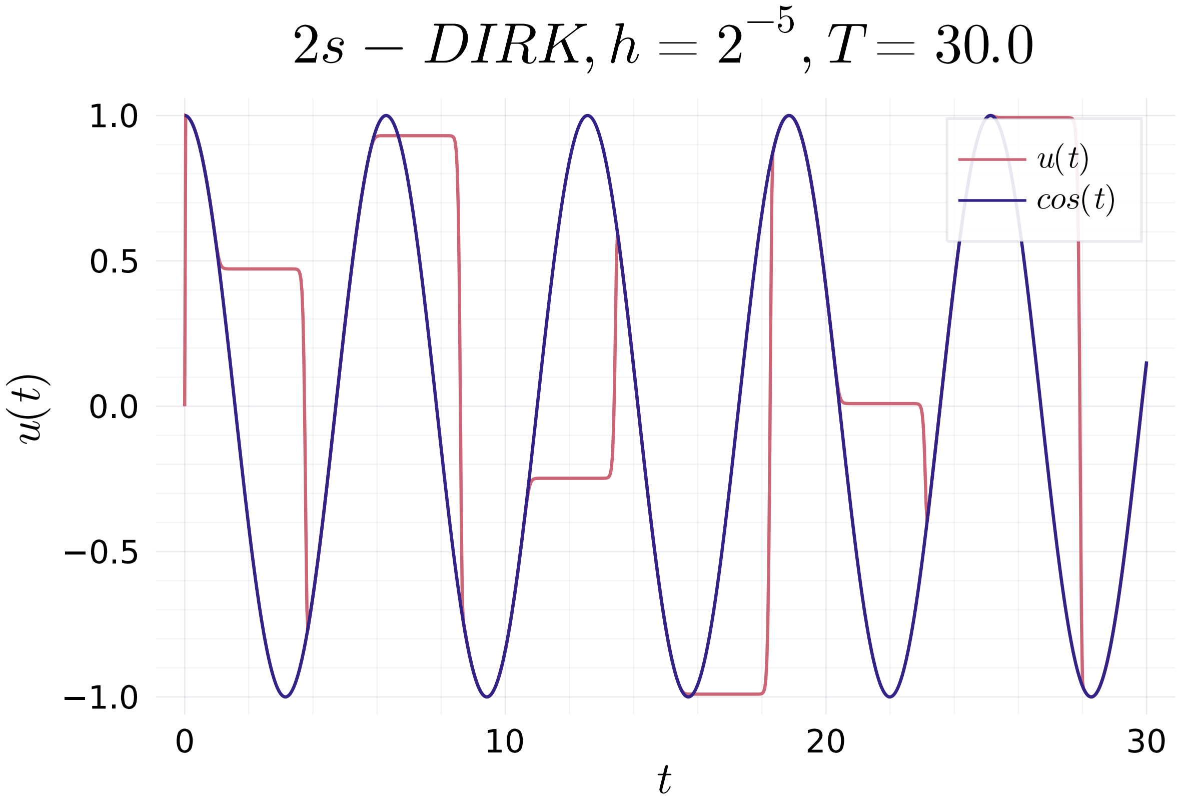


3 Problem 5

```
f(u,t,μ) = -(0.5*exp(20*cos(1.3*t)) * sinh(u-cos(t)));

α = 1 - 1/sqrt(2);
T = 30.0; h = 2.0^(-5); N = Int(T/h);
u0 = 0.0;

u = s2_DIRK(f, N, T, u0, α);
```

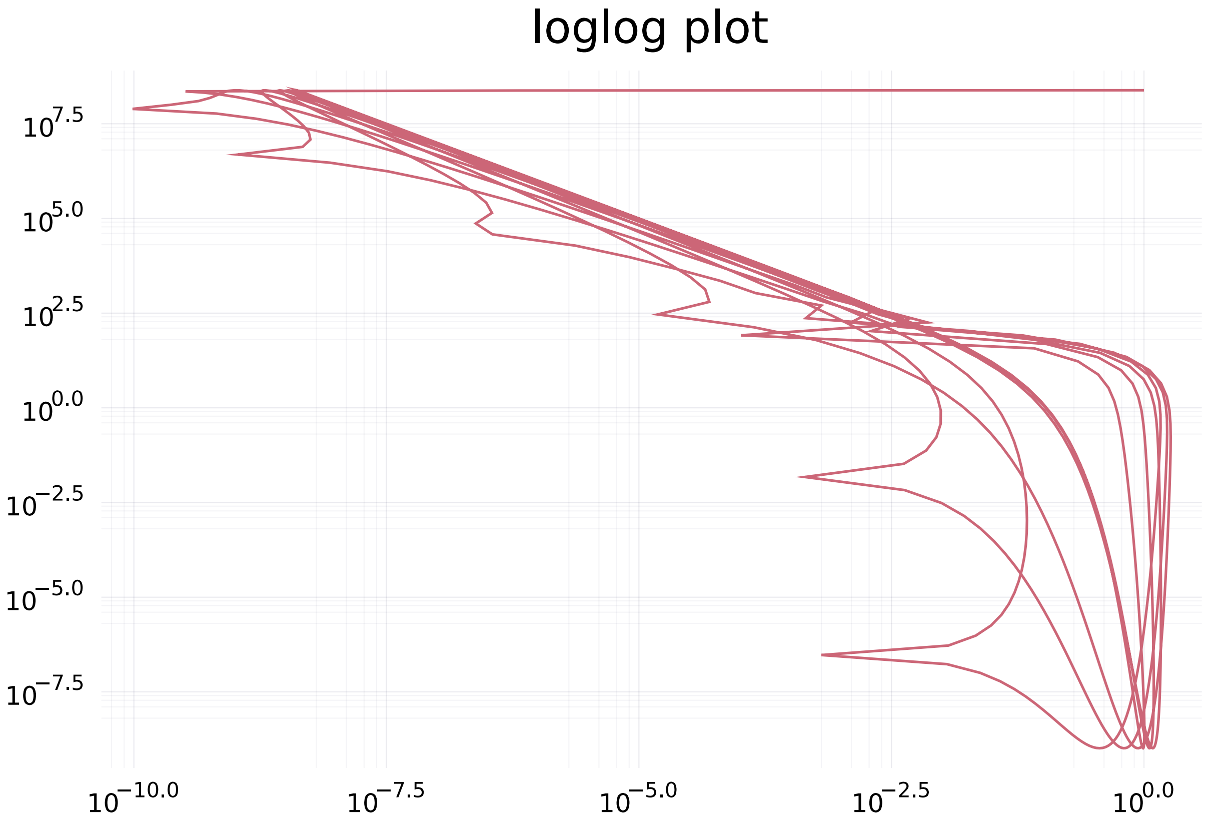
3.1 Part 1

```
tList = collect(0:N)*(T/N)
plot(tList, u, label = L"u(t)", thickness_scaling =1.25)
xlabel!(L"t")
ylabel!(L"u(t)")
title!(latexstring("2s-DIRK,h=2^{-5},T=",T))
plot!(tList, cos.(tList), label = L"cos(t)")
```



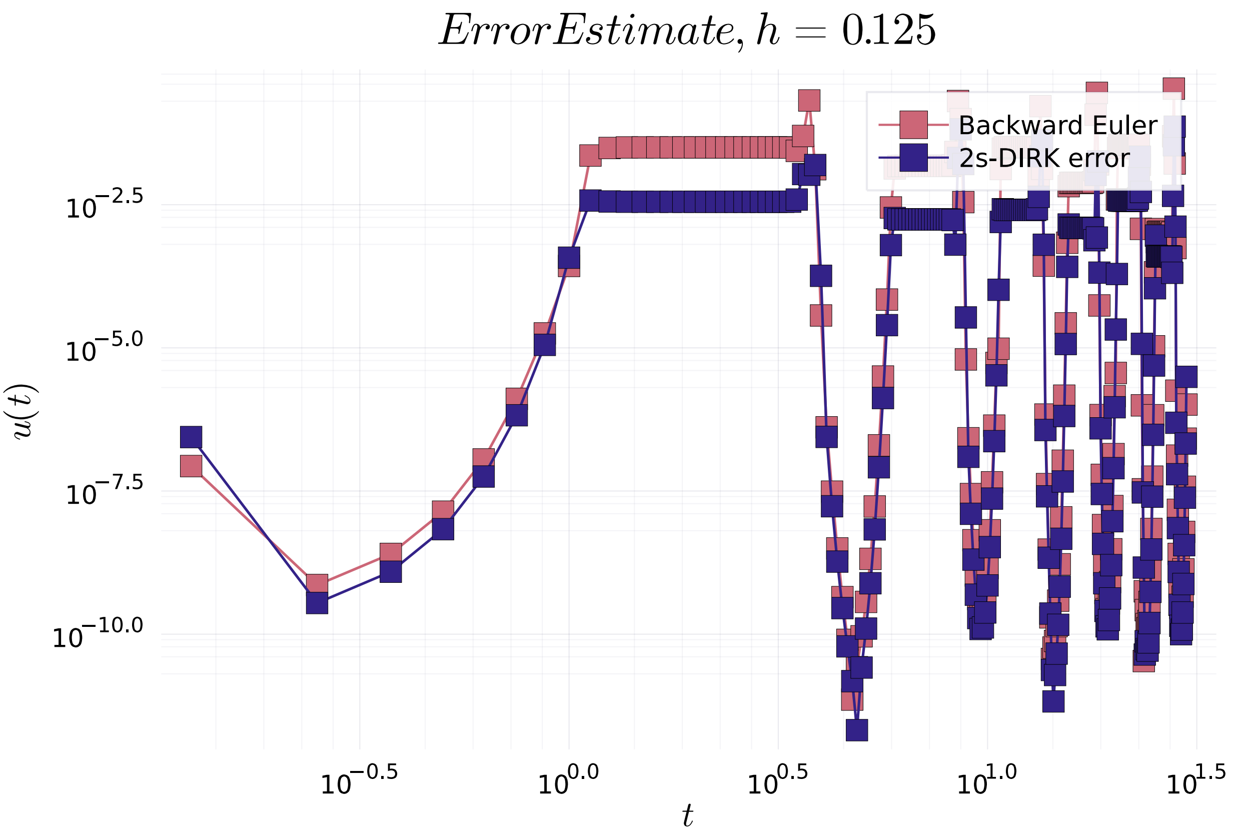
3.2 Part 2

```
g(t) = 0.5*exp(20*cos(1.3*t))
plot(abs.(u - cos.(tList)), g.(tList), xaxis=:log, yaxis=:log, legend = false)
title!("loglog plot")
```

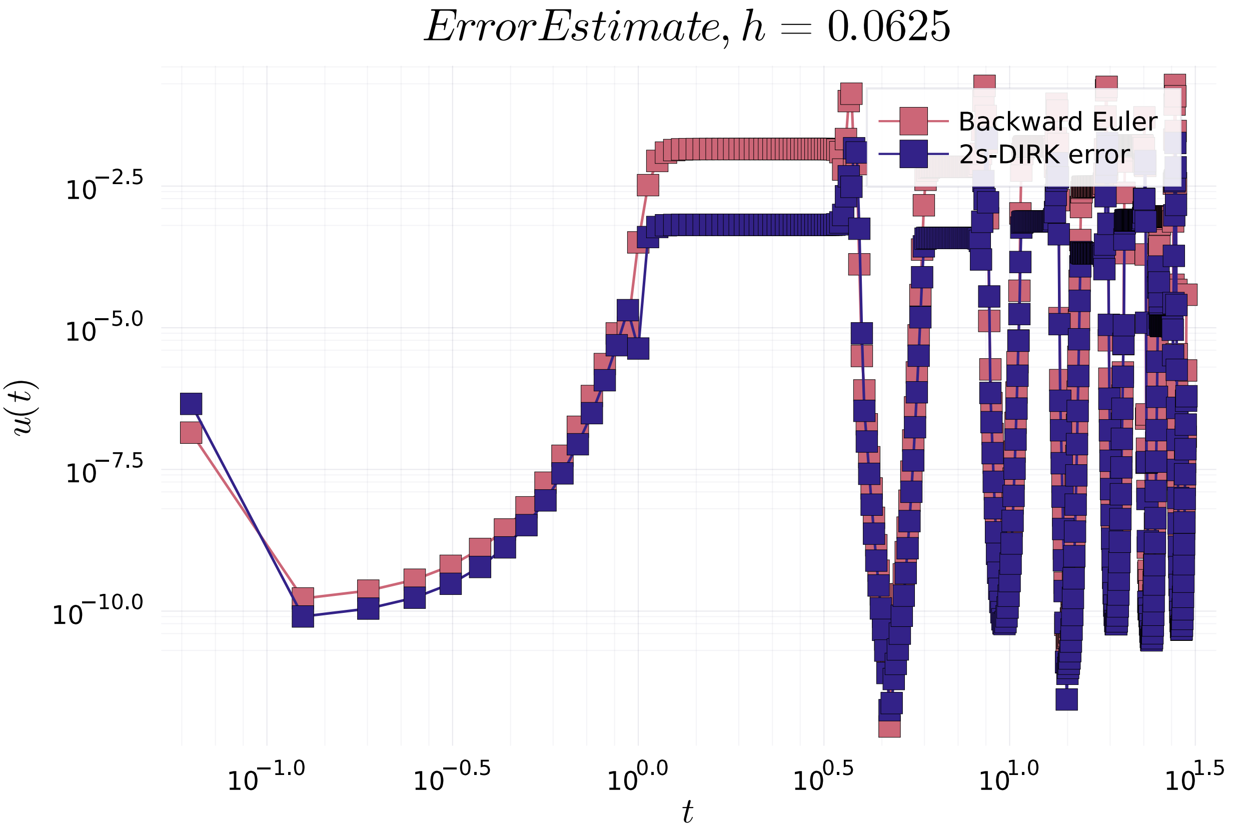


4 Problem 6

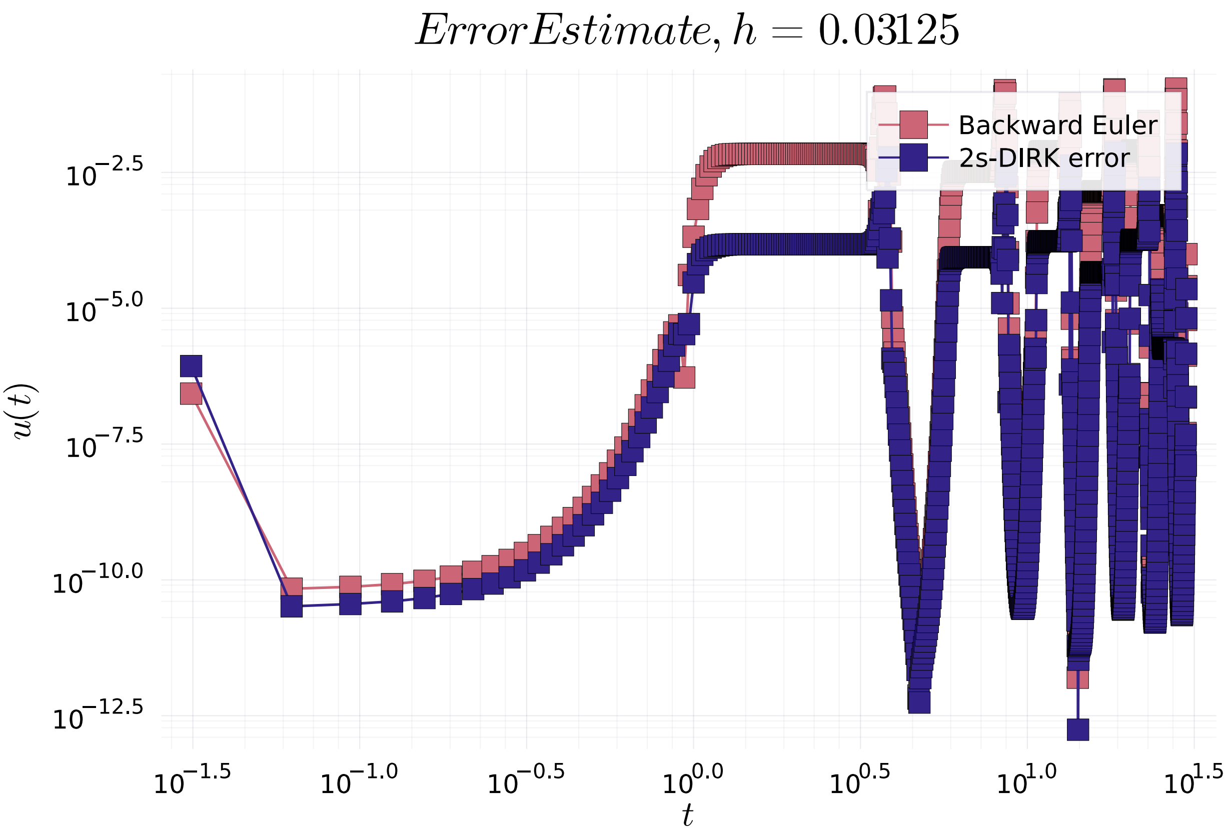
```
hs = 1 ./(2 .^(3:8))
for i = 1 : length(hs)
    N = Int(T/hs[i])
    tList = collect(0:N)*(T/N)

    ## Backward Euler
    u_euler = BackwardEuler_n(f, N, T, u0)
    u_eexact = BackwardEuler_n(f,2*N,T,u0)
    euler_error = abs.(u_euler[1:N] - u_eexact[1:2:2*N])./(1-0.5^1) # first order method
    p1 = plot(tList[2:N], euler_error[2:N], label = "Backward Euler", xaxis=:log,
yaxis=:log, marker = (:square,5))
    xaxis!(L"t")
    yaxis!(L"u(t)")
    title!(latexstring("Error Estimate,h=",hs[i]))

    ## s2-DIRK
    u_s2_DIRK = s2_DIRK(f, N, T, u0, α)
    u_Dexact = s2_DIRK(f, 2*N, T, u0, α)
    DIRK_error = abs.(u_s2_DIRK[1:N] - u_Dexact[1:2:2*N])./(1-0.5^2) # second order
method
    p2 = plot!(tList[2:N], DIRK_error[2:N], label = "2s-DIRK error", xaxis=:log,
yaxis=:log, marker = (:square,5))
    display(p2)
end
```



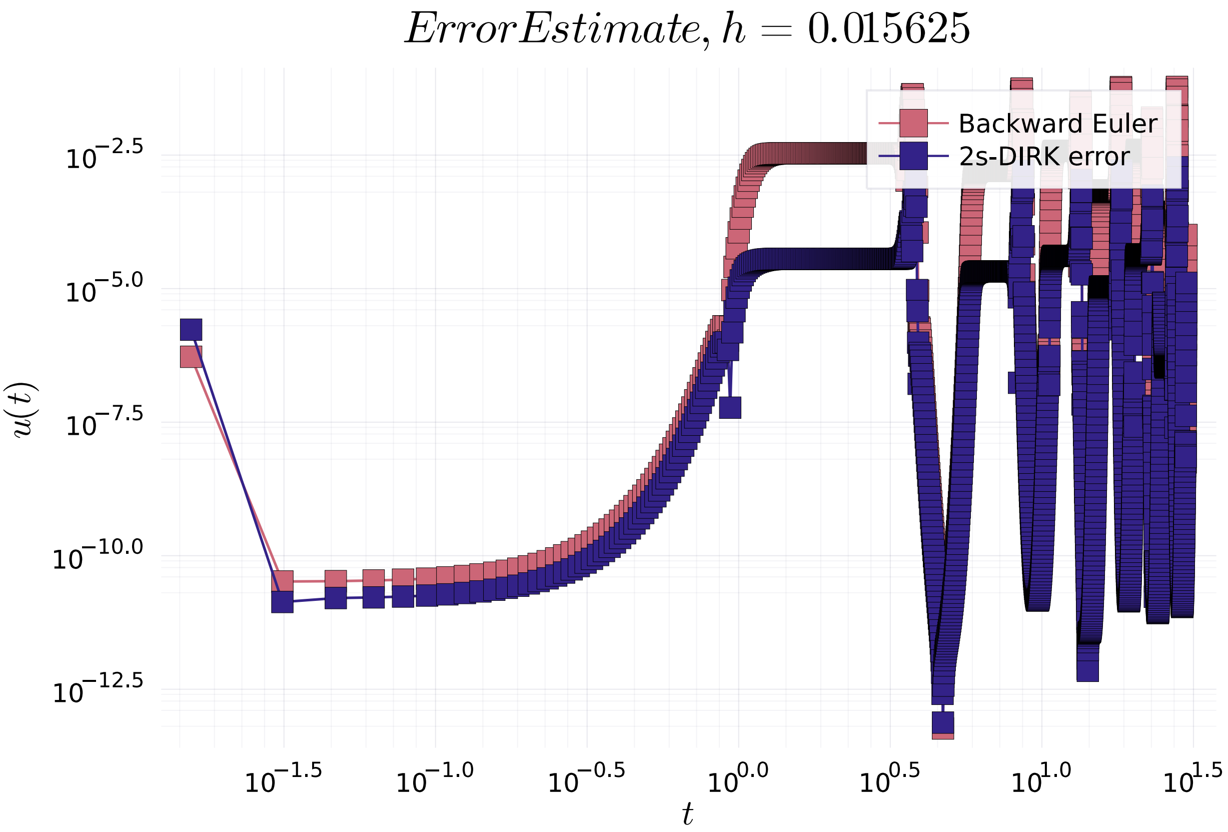
$ErrorEstimate, h = 0.125$



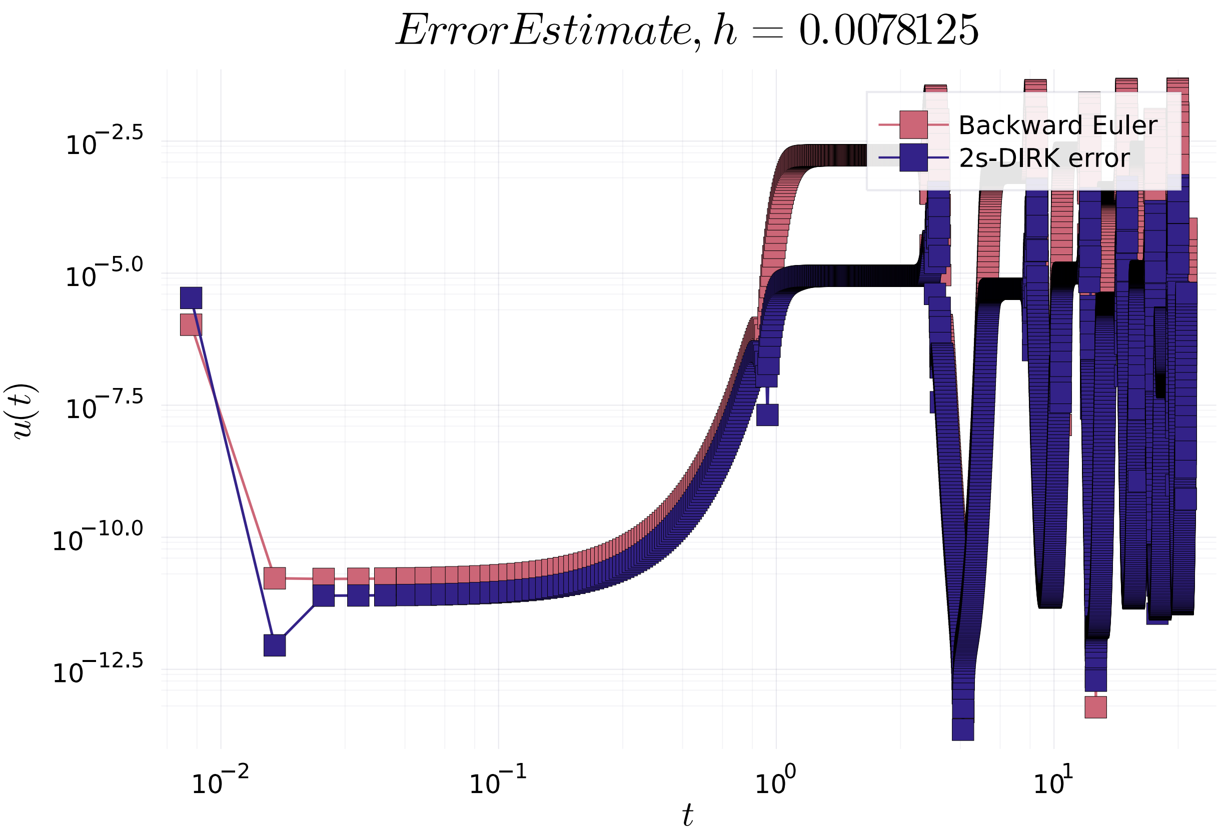
$ErrorEstimate, h = 0.0625$

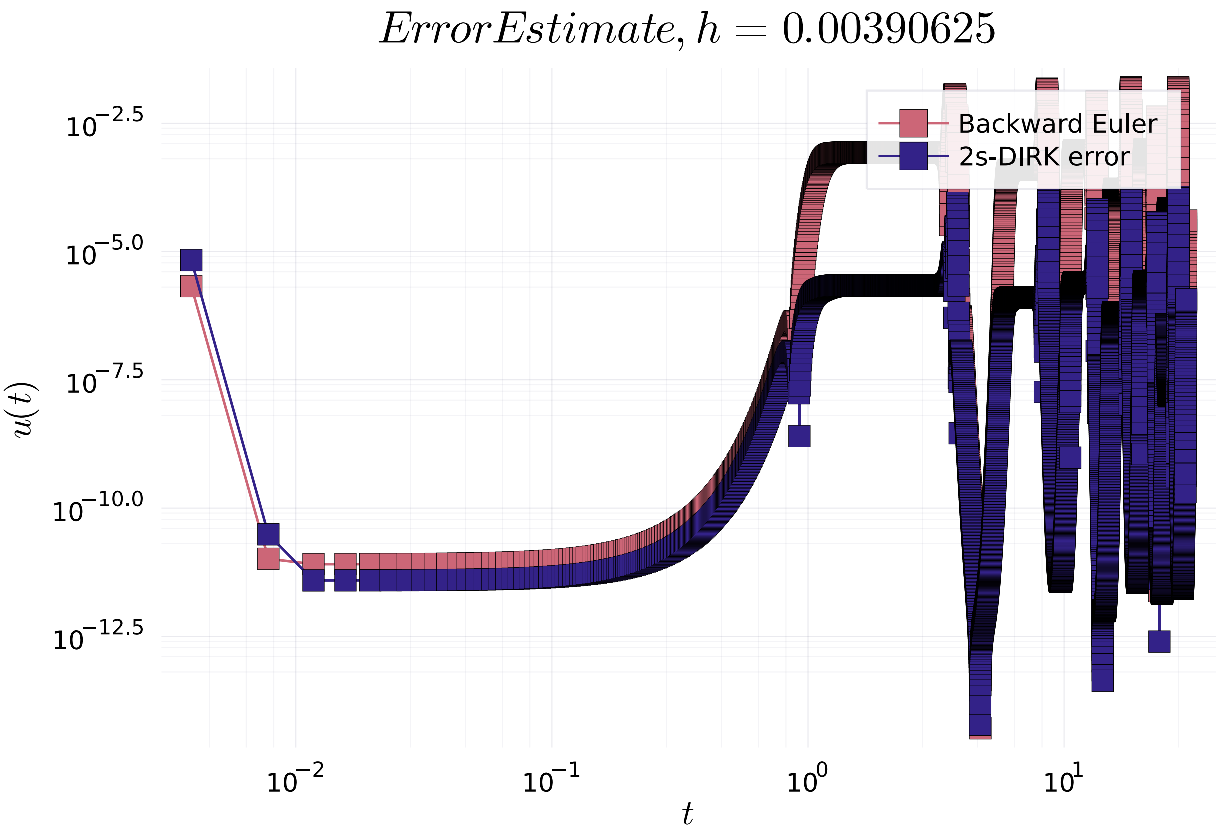


$Error Estimate, h = 0.03125$



$Error Estimate, h = 0.015625$

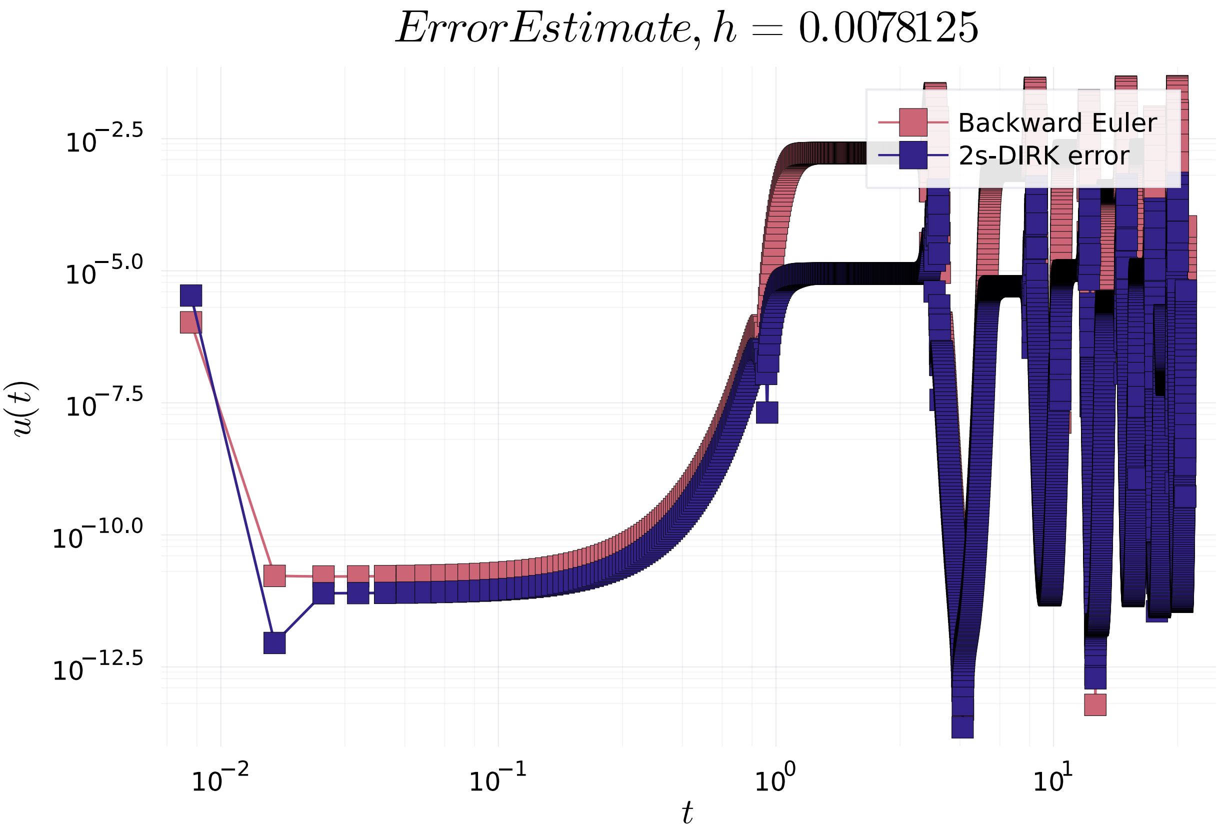




4.0.1 Part 2

```
h = 2.0^(-7)
N = Int(T/h)
tList = collect(0:N)*(T/N)
```

```
## Backward Euler
u_euler = BackwardEuler_n(f, N, T, u0)
u_eexact = BackwardEuler_n(f,2*N,T,u0)
euler_error = abs.(u_euler[1:N] - u_eexact[1:2:2*N])./(1-0.5^1) # first order method
p1 = plot(tList[2:N], euler_error[2:N], label = "Backward Euler", xaxis=:log,
yaxis=:log, marker = (:square,5))
xaxis!(L"t")
yaxis!(L"u(t)")
title!(latexstring("Error Estimate,h=",h))

## s2-DIRK
u_s2_DIRK = s2_DIRK(f, N, T, u0, α)
u_Dexact = s2_DIRK(f, 2*N, T, u0, α)
DIRK_error = abs.(u_s2_DIRK[1:N] - u_Dexact[1:2:2*N])./(1-0.5^2) # second order method
p2 = plot!(tList[2:N], DIRK_error[2:N], label = "2s-DIRK error", xaxis=:log, yaxis=:log,
marker = (:square,5))
display(p2)
```


$ErrorEstimate, h = 0.0078125$
Backward Euler
2s-DIRK error
$10^{-2.5}$
$10^{-5.0}$
$10^{-7.5}$
$10^{-10.0}$
$10^{-12.5}$
$u(t)$
10^{-2}
10^{-1}
10^{0}
10^{1}
t