Assignment #1

Kevin Corcoran

April 7, 2021

Including packages

Most functions I've written in the DiffyQ.jl module. I did this in order to make the code, and this report a little cleaner. I have explicitely written the implicit methods in this report like backwards Euler and the Tapezoid method.

```
using Plots
using LaTeXStrings
# Load DiffyQ Module and required functions
using Pkg
Pkg.activate("DiffyQ")
include("code/DiffyQ.jl") # Makes sure the module is run before using it
using .DiffyQ: CompTrapezoid, CompSimpson, Newtons, Euler, Midpoint2Step
```

1 Problem 1

Assuming $Nh \leq T$ and

$$E_{n+1} \leq (1 + ch)E_n + h^2.$$

Show

$$E_N \leq \frac{h}{c}\left(e^{ct} - 1\right).$$

Expanding the recursive definition

$$\begin{aligned} E_{n+1} &\leq (1 + ch)E_n + h^2 \leq (1 + ch)\left((1 + ch)E_{n-1} + h^2\right) + h^2 \\ &\leq \dots \\ &\leq h^2 \sum_{k=0}^{n} (1 + ch)^k \end{aligned}$$

Setting n = N-1 and multiplying by $(1 + ch)^{-N}$

$$E_N \leq h^2 \sum_{k=0}^{N-1} (1 + ch)^k$$

$$(1 + ch)^{-N} E_N \leq h^2 \sum_{k=0}^{N-1} (1 + ch)^{-(k+1)}$$

Using the geometric series and simplifying

$$\begin{aligned}(1+ch)^{-N}E_N &\le h^2\sum_{k=0}^{N-1}(1+ch)^{-(k+1)}\\ &= h^2(1+ch)^{-1}\frac{1-(1+ch)^{-N}}{1-(1+ch)^{-1}}\\ &= \frac{h}{c}\left(1-(1+ch)^{-N}\right)\end{aligned}$$

Multiplying by $(1+ch)^N$ and using $1+ch \le e^{ch} \le e^{c\frac{T}{N}}$

$$\begin{aligned}E_N &\le \frac{h}{c}\left((1+ch)^N-1\right)\\ &\le \frac{h}{c}(e^{cT}-1)\end{aligned}$$

2 Problem 2

Solve

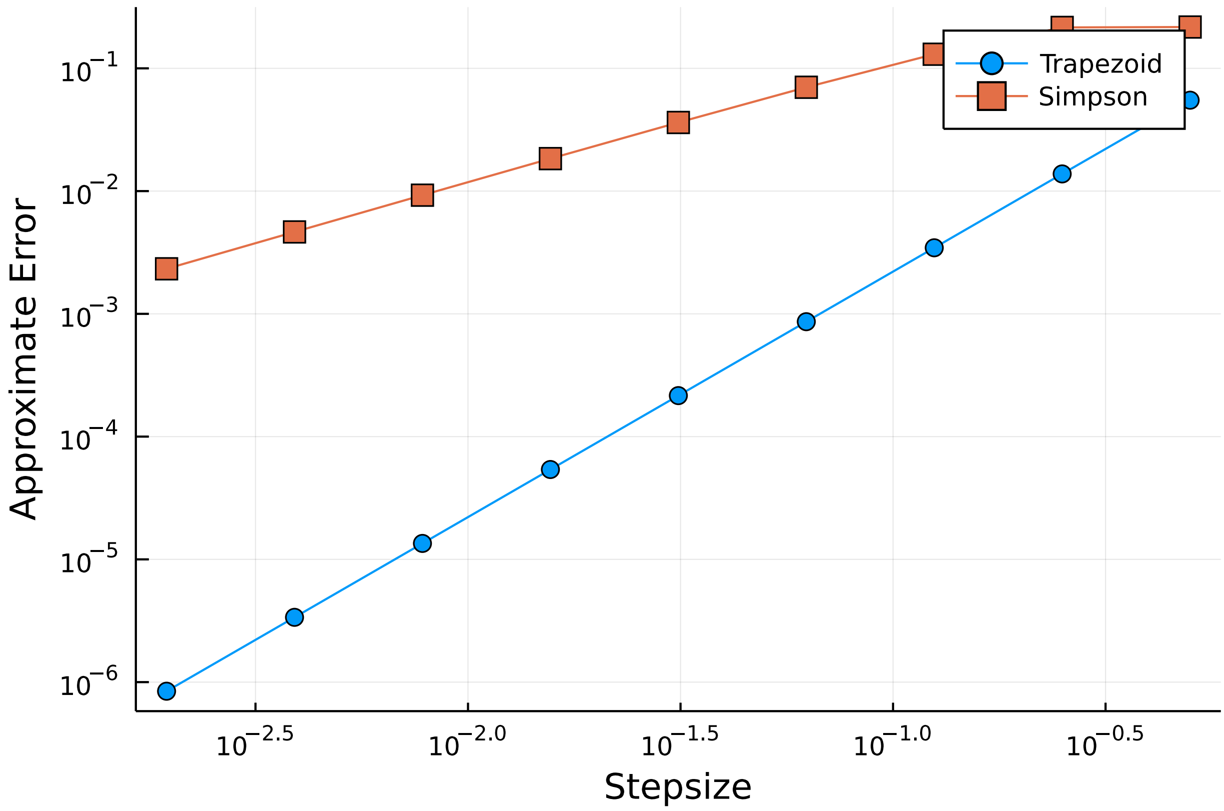
$$\int_1^3 \sqrt{2+\cos^3(x)}e^{\sin(x)}\,dx.$$

using the *Composite Simpson* and *Composite Trapezoid* Rule. Then estimate the error using formula $E(h) = \frac{|T(h)-T(h/2)|}{1-1/2^p}$. Finally, plot the result on a log log plot.

Run simulations for $N = 2^2, 2^3, \ldots, 2^{10}$

```
# function to be integrated from a to b
f(t) = sqrt(2 + cos(t)^3)*exp(sin(t))

a = 1; b = 3; #= N = 2^(10); =# T = b-a;

# Plotting error routine
NList = 2 .^(2:10)
errTrapeList = zeros(size(NList))
errSimpList = zeros(size(NList))
for i = 1 : length(NList)
    N = NList[i]

    utrape = CompTrapezoid(N,a,b,f)
    utexact = CompTrapezoid(2*N,a,b,f)

    usimps = CompSimpson(N,a,b,f)
    usexact = CompSimpson(2*N,a,b,f)

    # estimate error
    errTrapeList[i] = abs(utrape-utexact)./(1-(1/2^2))
    errSimpList[i] = abs(usimps-usexact)./(1-(1/2^4))
end

plot(T./NList, errTrapeList,label="Trapezoid",xaxis=:log,yaxis=:log, marker = (:dot,5),
add_marker = true)
plot!(T./NList, errSimpList,label="Simpson",xaxis=:log,yaxis=:log, marker = (:square,5),
add_marker = true)
```

```julia
xlabel!("Stepsize")
ylabel!("Approximate Error")
```



3 Problem 3

Implement Newton's method to solve the non-linear equation for 0

$x - \alpha + \beta \sinh\left(x - \cos(s-1)\right) = 0$

Defining variables

```julia
α = 0.9
β = 50000.0
# equation to solve
f(x,s) = x - α + β*sinh(x-cos(s-1))
# derivative of equation
df(x,s) = 1 + β*cosh(x-cos(s-1))
# initial guess
x0 = 2.0
```

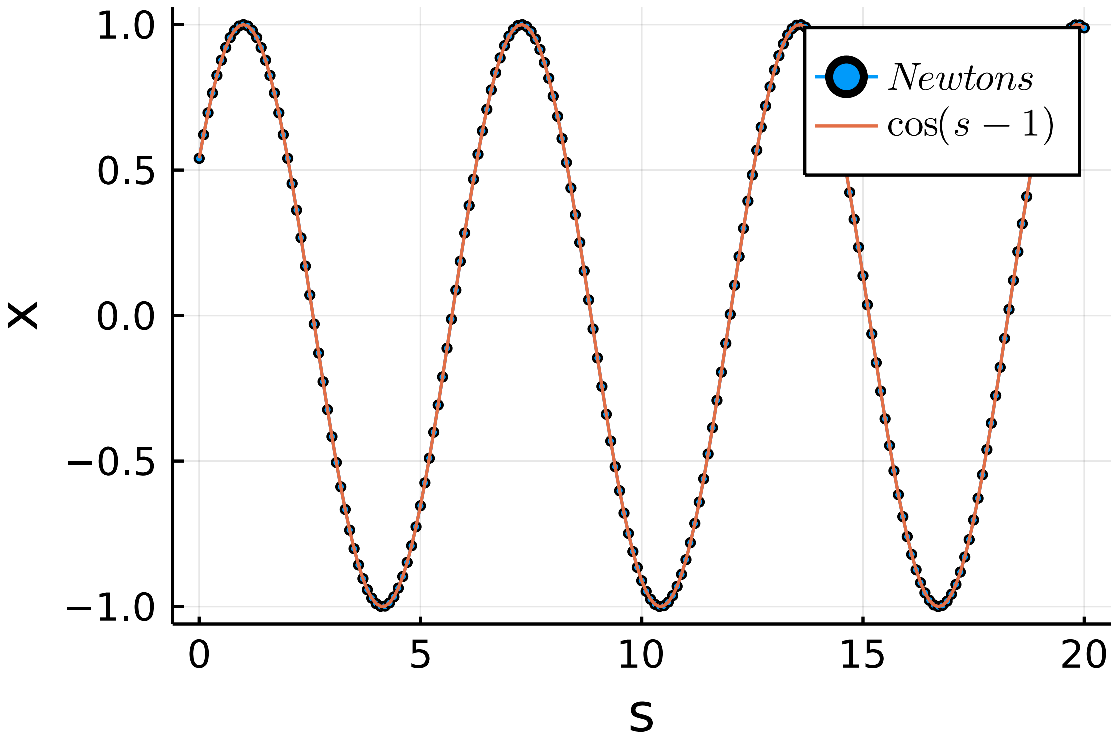
```
2.0
```

Solving equation for various values of s

```julia
ss = 0.0:0.1:20.0

xs = []
for s in ss
    push!(xs,Newtons(f,df,x0,s))
end
```

Plotting x vs s and cos(s-1) on the same plot

```
plot(ss,xs, label = L"Newtons",marker = (:dot,2), add_marker=true , thickness_scaling =
1.5)
plot!(ss, cos.(ss.-1), label = L"\cos(s-1)")
xlabel!("s")
ylabel!("x")
```



4 Problem 4

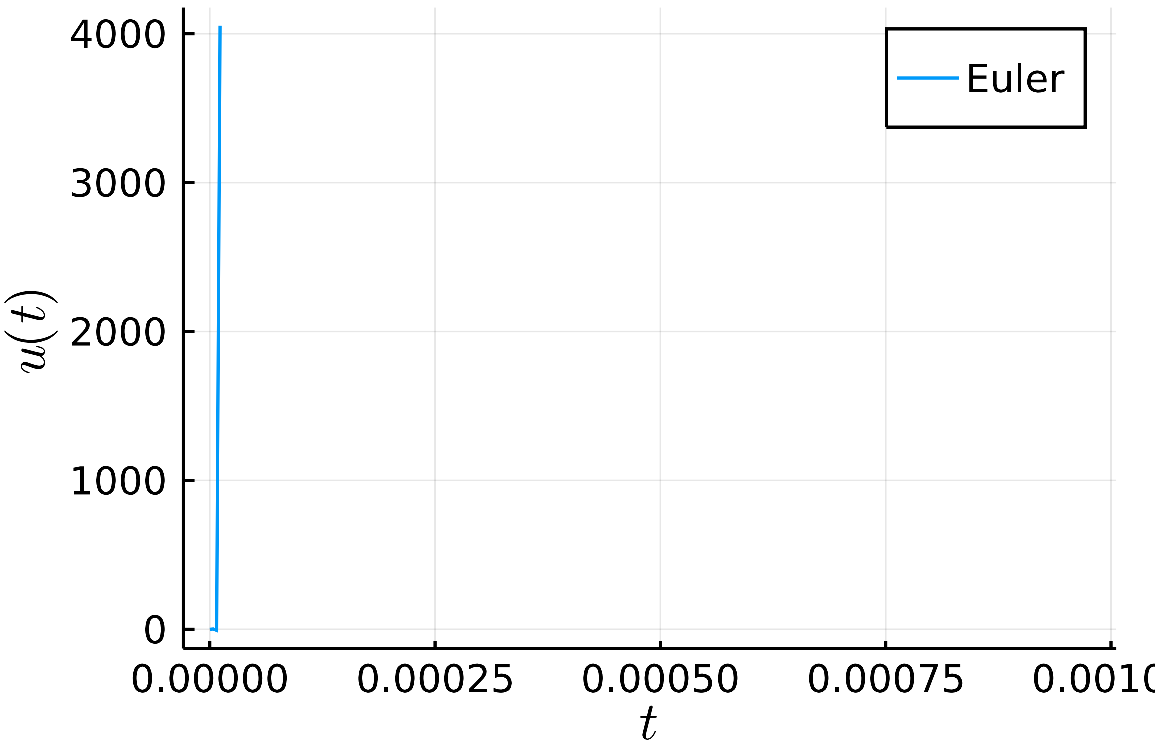
4.1 Part 1

Solve the initial value problem using Eulers method for $T = 2^{-10}$ and various values of h.

The solution when the stepsize is not small enough. We see that the solution is not bounded.

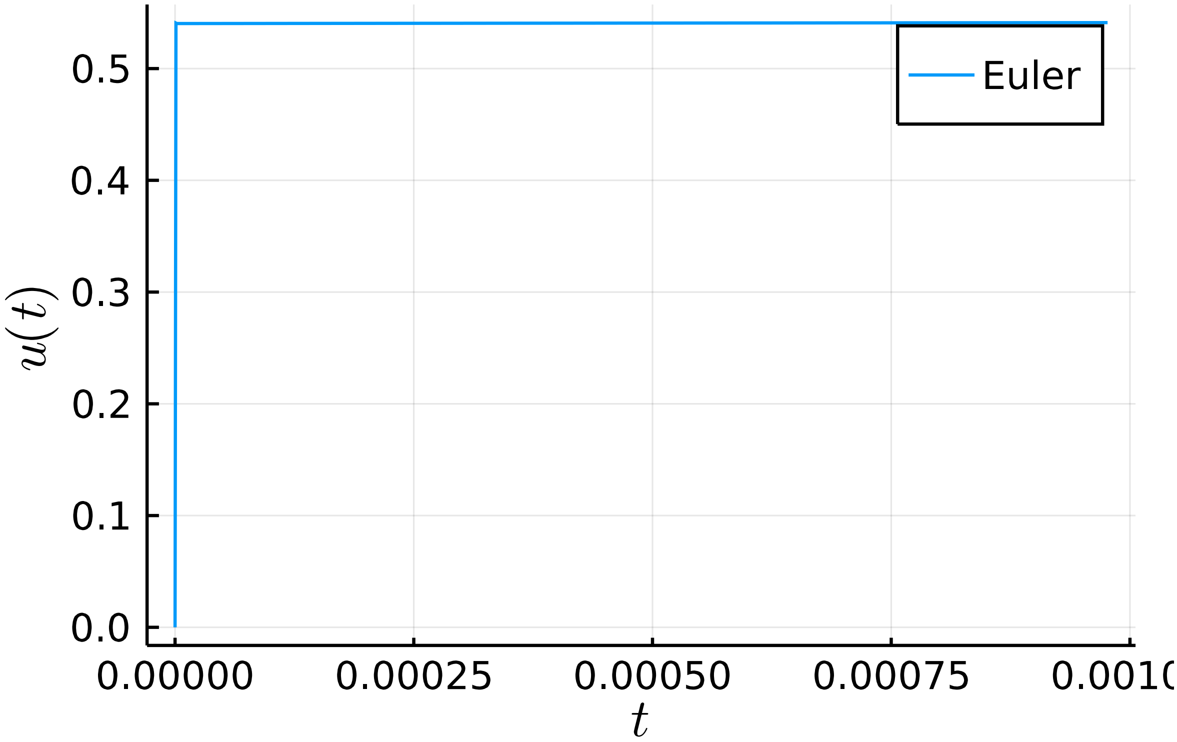
```
# Differential equation
func(u, t, λ) = -λ*sinh(u-cos(t-1))

# defining variables/initial conditions
T = 2.0^(-10.0); t0 = 0; u0 = 0; λ = 10.0^(6.0);

# Foward Euler
h0 = 2.0^(-18);
N = Int(T/(h0))
h = T/N

u = Euler(func,N,T,t0,u0,λ)
tList = collect(0:N)*(T/N)
```

```julia
plot(tList, u, label = "Euler")
xlabel!(L"t", thickness_scaling = 1.5)
ylabel!(L"u(t)")
```



Stepsize at $h = 2^{-20}$ the solution becomes bounded

```julia
# Foward Euler
h0 = 2.0^(-18);
N = Int(T/(h0*2.0^(-2)))
h = T/N

u = Euler(func,N,T,t0,u0,λ)
tList = collect(0:N)*(T/N)
plot(tList, u, label = "Euler")
xlabel!(L"t", thickness_scaling = 1.5)
ylabel!(L"u(t)")
```



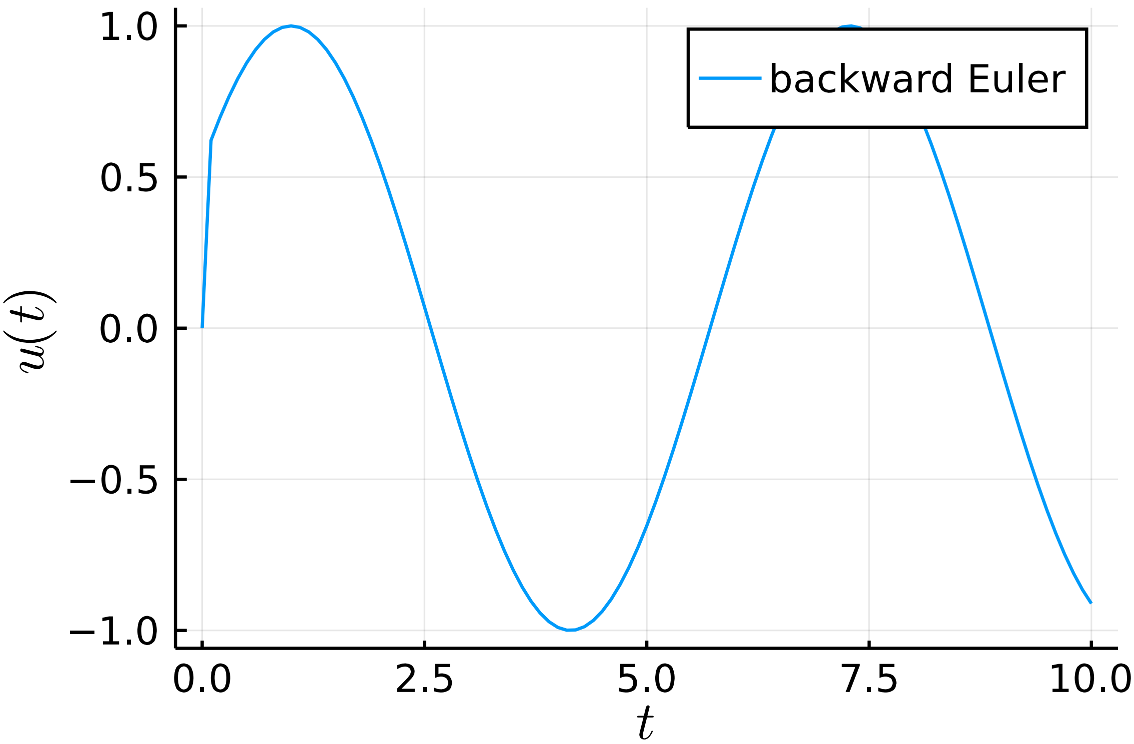
4.2 Part 2

For backward Euler I explicitely defined the function and derivative for Newtons method.

The solution for $T = 10$ using backward Euler

```
# Backward Euler
function BackwardEuler(N,T,t0,u0,λ)
    u = zeros(N+1)
    u[1] = u0
    h = T/N
    t = t0
    x0 = 2.0
    for i = 1:N
        t += h
        f(x, s) = u[i] + h*(-λ*sinh(x-cos(t-1)))
        df(x,s) = -λ*h*cosh(x-cos(t-1)) - 1

        u[i+1] = Newtons(f, df, x0, 1)
    end

    return u
end

T = 10.0; t0 = 0.0; u0 = 0; λ = 10.0^(6.0); h = 0.1;
N = Int(T/h)

# u = DiffyQ.BackwardEuler(N,T,t0,u0,λ)
u = BackwardEuler(N,T,t0,u0,λ)
tList = collect(0:N)*(T/N)
# tList = t0:h:T+t0
```

```
plot(tList, u, label = "backward Euler")
xlabel!(L"t", thickness_scaling = 1.5)
ylabel!(L"u(t)")
```



5 Problem 5

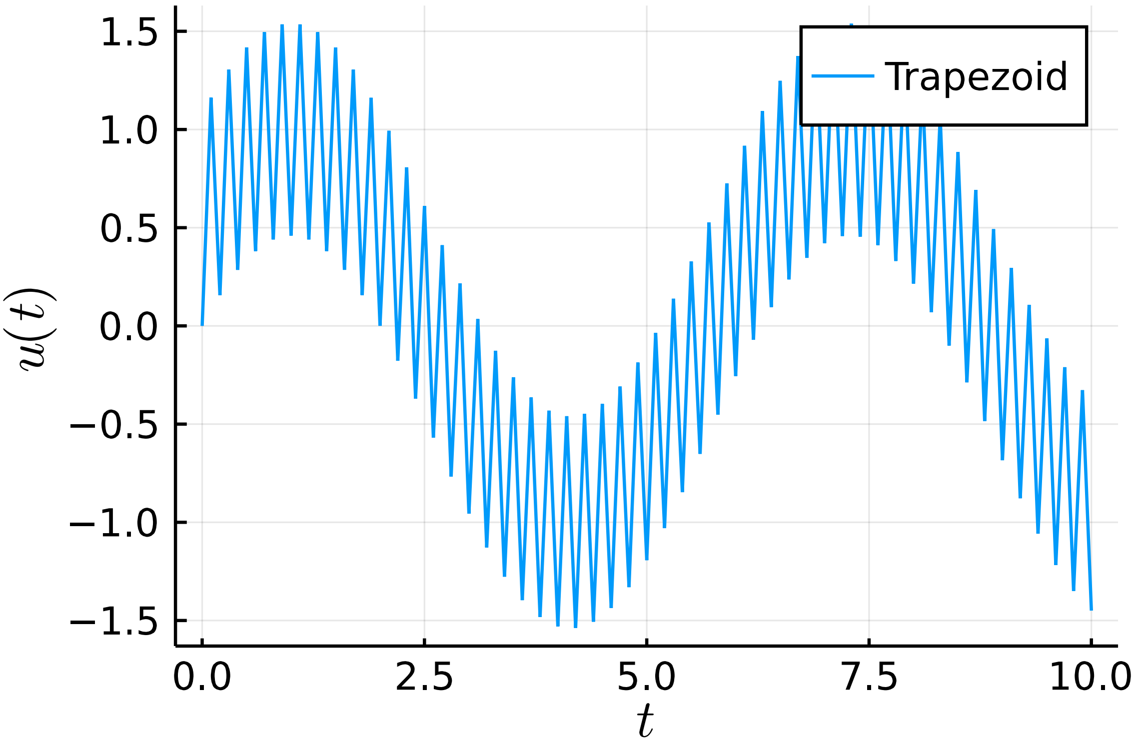
Like backward Euler the Trapezoid method is also implicit, so I explicitely define the method here.

5.1 Part 1

The solution for $T = 10$ and $h = 0.1$

```
# The differential equation to be solved
func(u, t, λ) = -λ*sinh(u-cos(t-1))

function Trapezoid(N,T,t0, u0, λ)
    # u = spzeros(N)
    u = zeros(N+1)
    u[1] = u0
    h = T/N
    t = t0
    for i = 1:N
        t += h
        # Use Newtons Method to solve nonlinear eqn for u[n+1]
        f(x,s) = u[i] + h/2 * (func(u[i],t-h,λ) - λ*sinh(x-cos(t-1))) - x
        df(x,s) = -λ*h/2*cosh(x-cos(t-1)) - 1
        u[i+1] = Newtons(f,df,2.0,1.0)
```

```
    end
    return u
end

T = 10; t0 = 0; u0 = 0; λ = 10.0^(6.0);

h = 0.1
N = Int(T/h)

u = Trapezoid(N,T,t0,u0,λ)
tList = collect(0:N)*(T/N)
plot(tList, u, label = "Trapezoid")
xlabel!(L"t", thickness_scaling = 1.5)
ylabel!(L"u(t)")
```

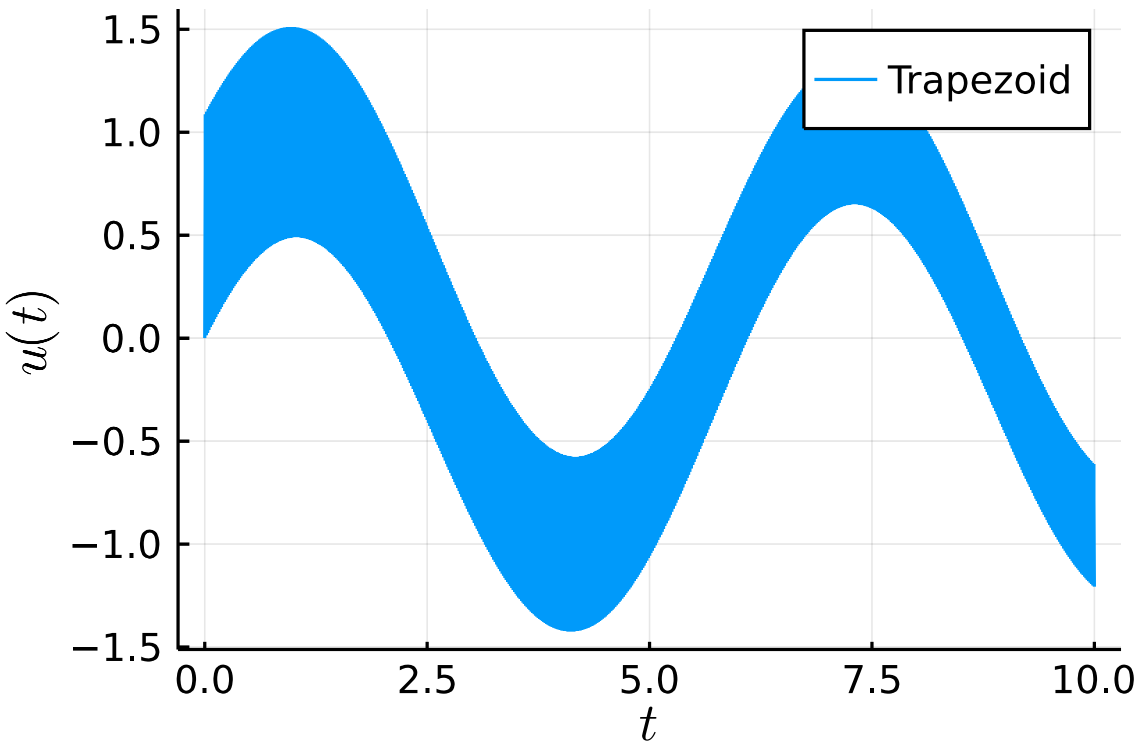


This method seems to oscillate around the solution.

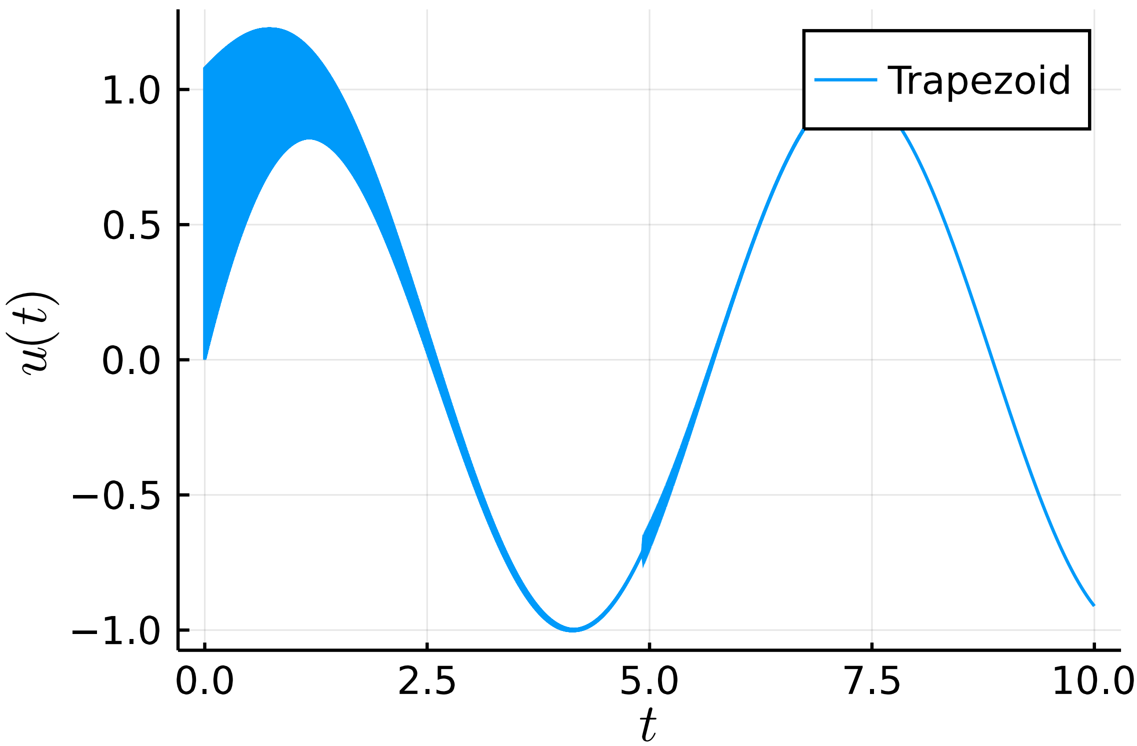
5.2 Part 2

Plotting smaller and smaller stepsizes, we can see how the solution behaves

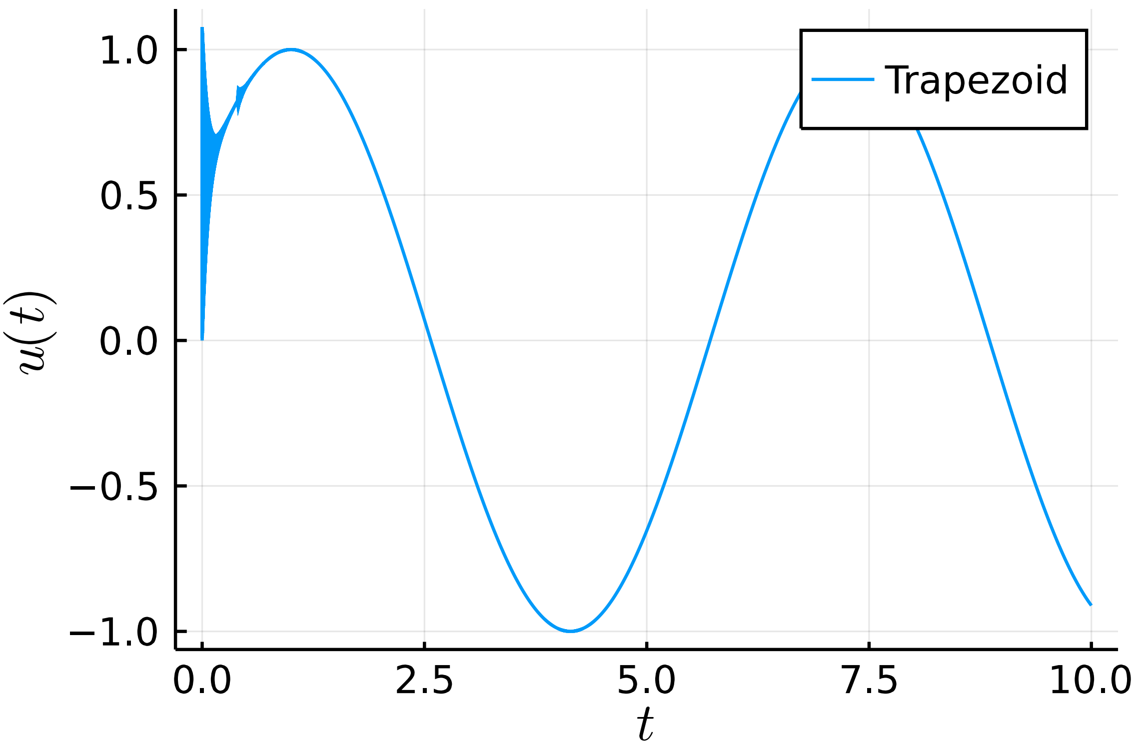
```
for i = 7:2:12
    local h = 2.0^(-i)
    local N = Int(T/h)
    local u = Trapezoid(N,T,t0,u0,λ)
    local tList = collect(0:N)*(T/N)
    display(plot(tList, u, label = "Trapezoid", xlabel = L"t", thickness_scaling = 1.5,
ylabel= L"u(t)"))
end
```



Trapezoid
$u(t)$
t



Trapezoid
$u(t)$
t



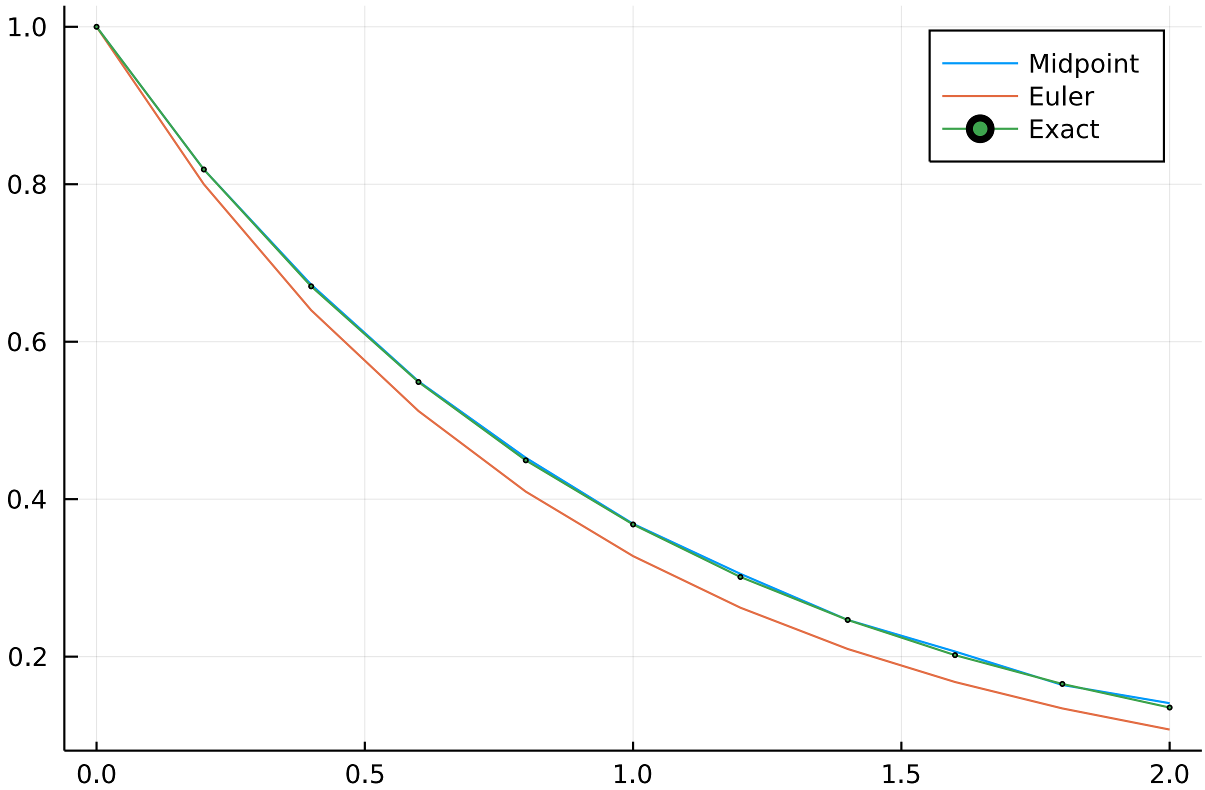
The oscillations dissipate and the method converges to the solution.

6 Problem 6

6.1 Part 1

Plotting Euler's method and 2-step Midpoint rule for $T = 2$ and $h = 0.2$

```
func(u,t, λ) = -u

T = 2; h = 0.2; t0 = 0;
N = Int(T/h)
uexact(t) = exp(-t)
u0 = 1
u1 = uexact(h)

umid = Midpoint2Step(func, N, T, t0, u0, u1)
ueul = Euler(func, N, T, t0, u0)

tList = collect(0:N)*(T/N)
plot(tList,umid, label = "Midpoint")
plot!(tList,ueul, label = "Euler")
plot!(tList,uexact.(tList), label = "Exact", marker = (:dot,1.5), add_marker=true)
```

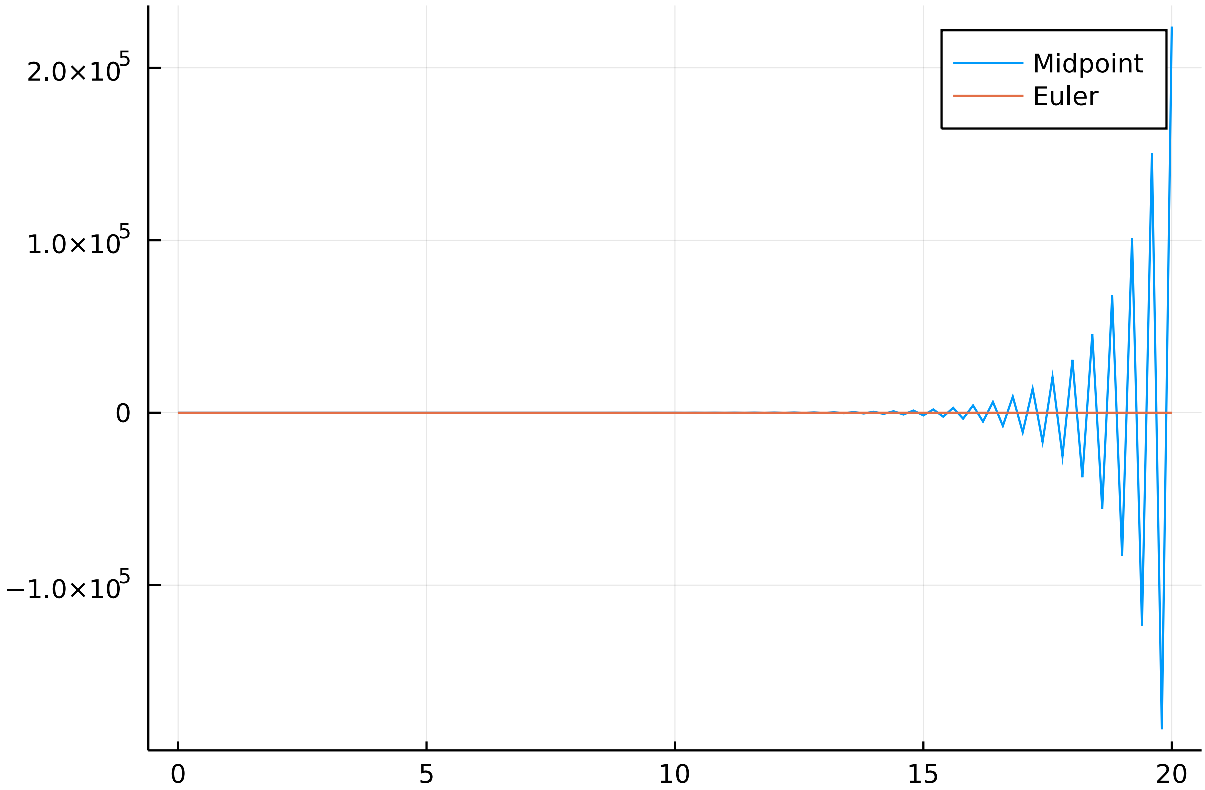


The midpoint method is more accurate in this time period.

6.2 Part 2

Ploting now over a longer time period $T = 20$

```
T = 20.0
N = Int(T/h)

umid = Midpoint2Step(func, N, T, t0, u0, u1)
ueul = Euler(func, N, T, t0, u0)

tList = collect(0:N)*(T/N)
plot(tList,umid, label = "Midpoint")
plot!(tList,ueul, label = "Euler")
```

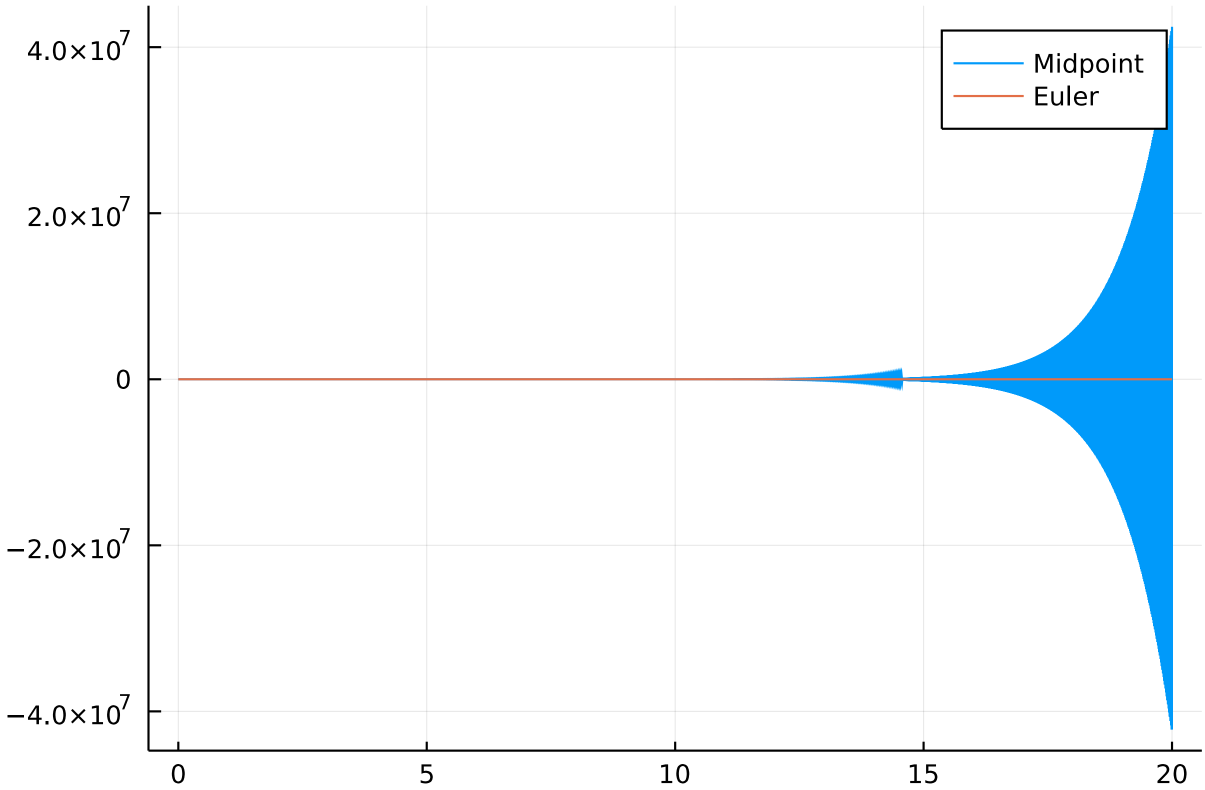


The midpoint method is not well behaved.

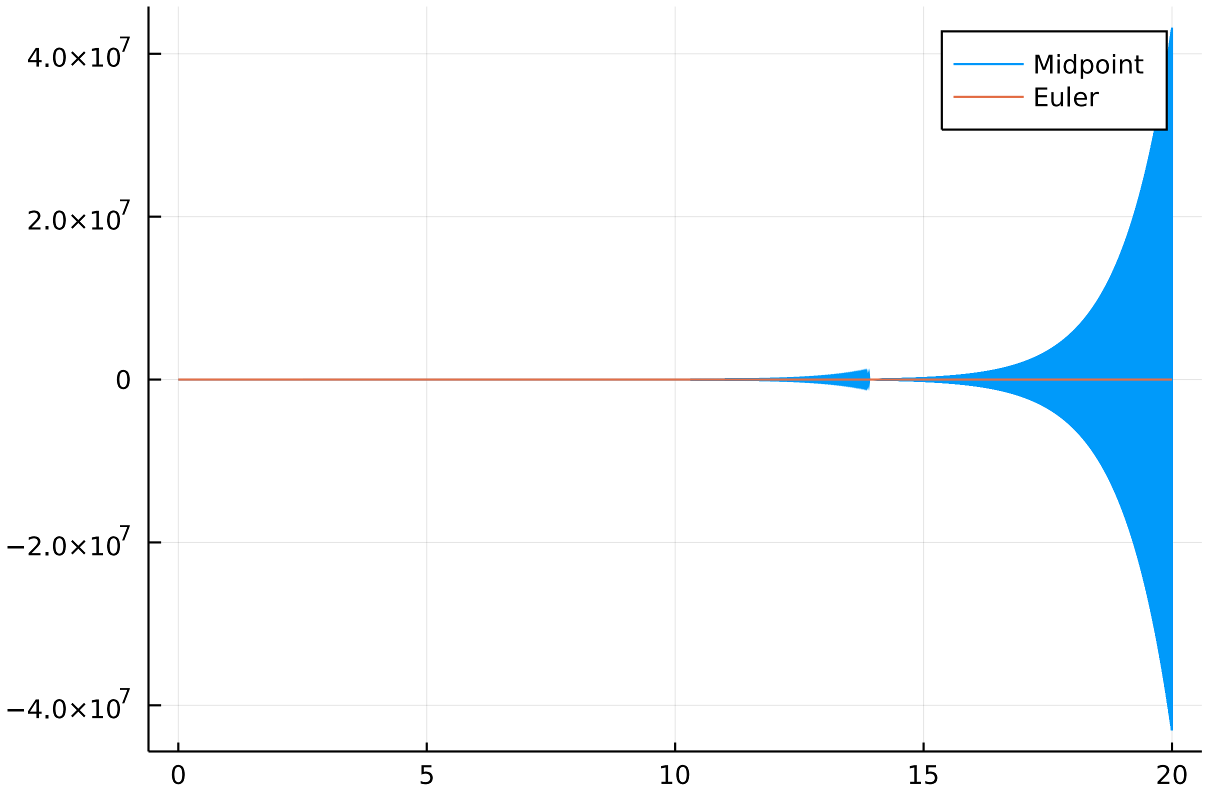
6.3 Part 3

Reducing now the timestep for the same time period $T = 20$

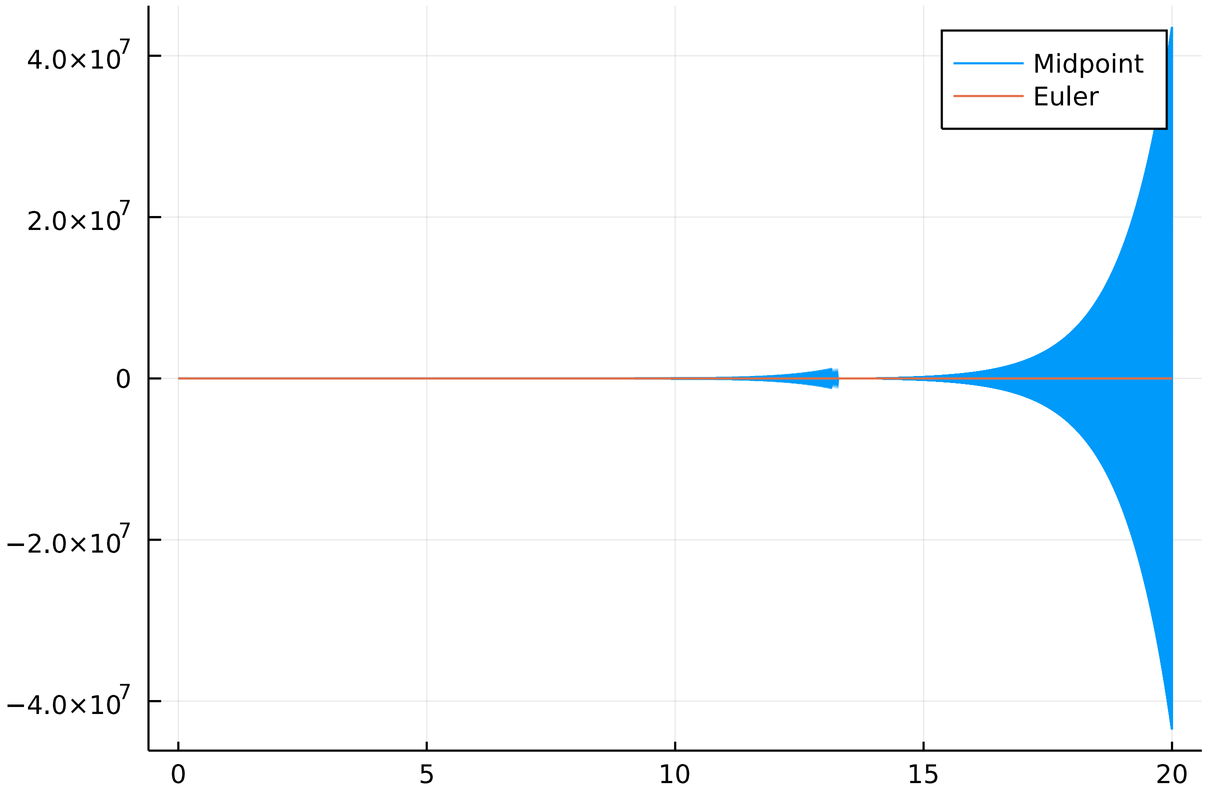
```
for i = 5:7
    local h = 0.2/(2^i)
    local N = Int(T/h)
    local umid = Midpoint2Step(func, N, T, t0, u0, u1)
    local ueul = Euler(func, N, T, t0, u0)

    local tList = collect(0:N)*(T/N)
    local p1 = plot(tList,umid, label = "Midpoint")
    local p2 = plot!(tList,ueul, label = "Euler")
    display(p2)
end
```



Midpoint
Euler
4.0×10^7
2.0×10^7
0
−2.0×10^7
−4.0×10^7
0
5
10
15
20



Midpoint
Euler
4.0×10^7
2.0×10^7
0
−2.0×10^7
−4.0×10^7
0
5
10
15
20



The midpoint method does not improve.