HW 5 Part 2

Jackson Wills

April 19, 2020

Files to include in Appendix: BackwardEuler.jl Adams*Bashforth2.jl GL*RK.jl MyImplicit-Midpoint.jl MyForwardEuler.jl Adaptive*time*step*Runge*Kutta.jl

Simple Pendulum

First I need to solve the ODE and put it into first order form

```
using Plots
using SymPy
using DifferentialEquations
@vars p q theta t

Ham = p^2/2 +cos(q)
```

$$\frac{p^2}{2} + \cos(q)$$

I need find the nondimentional momentum of the angle

```
thetadot = diff(Ham,p)
pdot = -diff(Ham,q) |> subs(q=>theta)
```

$$\sin(\theta)$$

therefore $thetaddot = sin(theta)$

we need to get it in first order form thetaddot = x1dot thetadot = x2dot

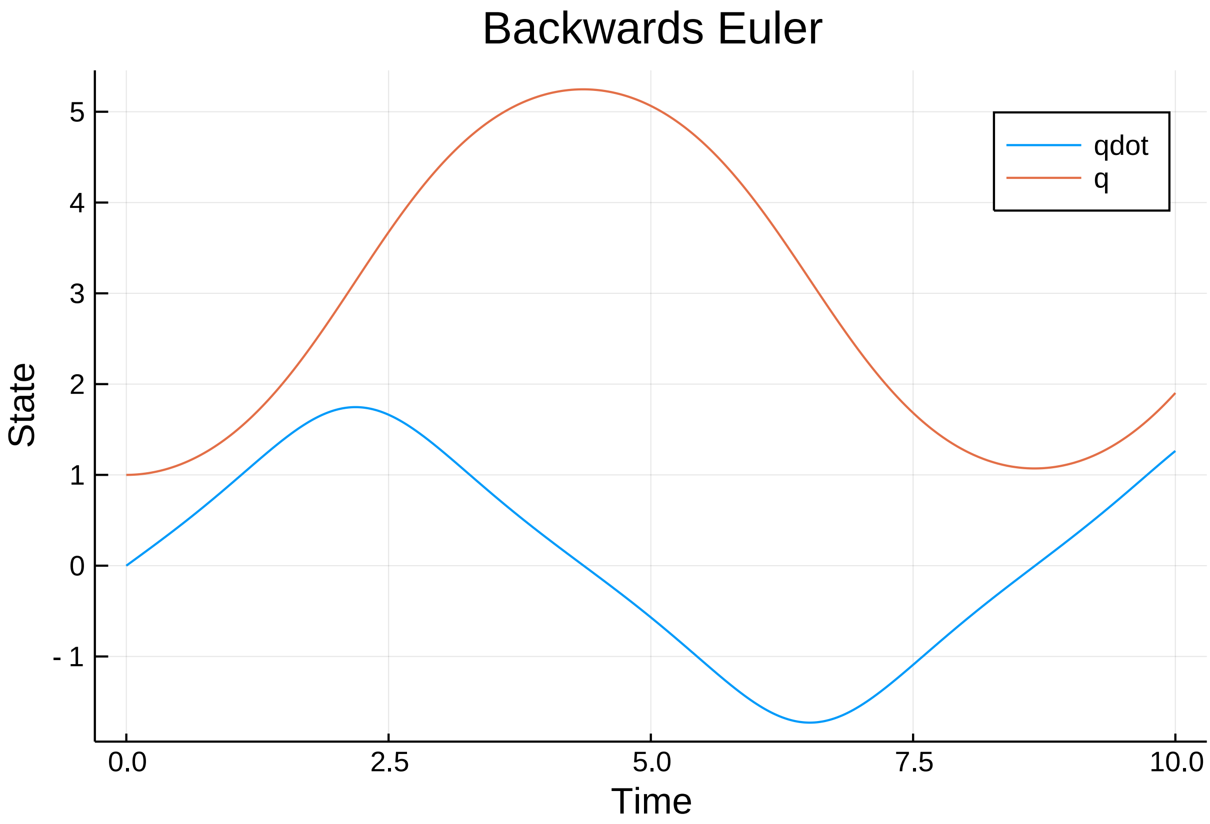
Now, there are two first order ODEs to solve.

1. $x1dot = sin(x2)$
2. $x2dot = x1$

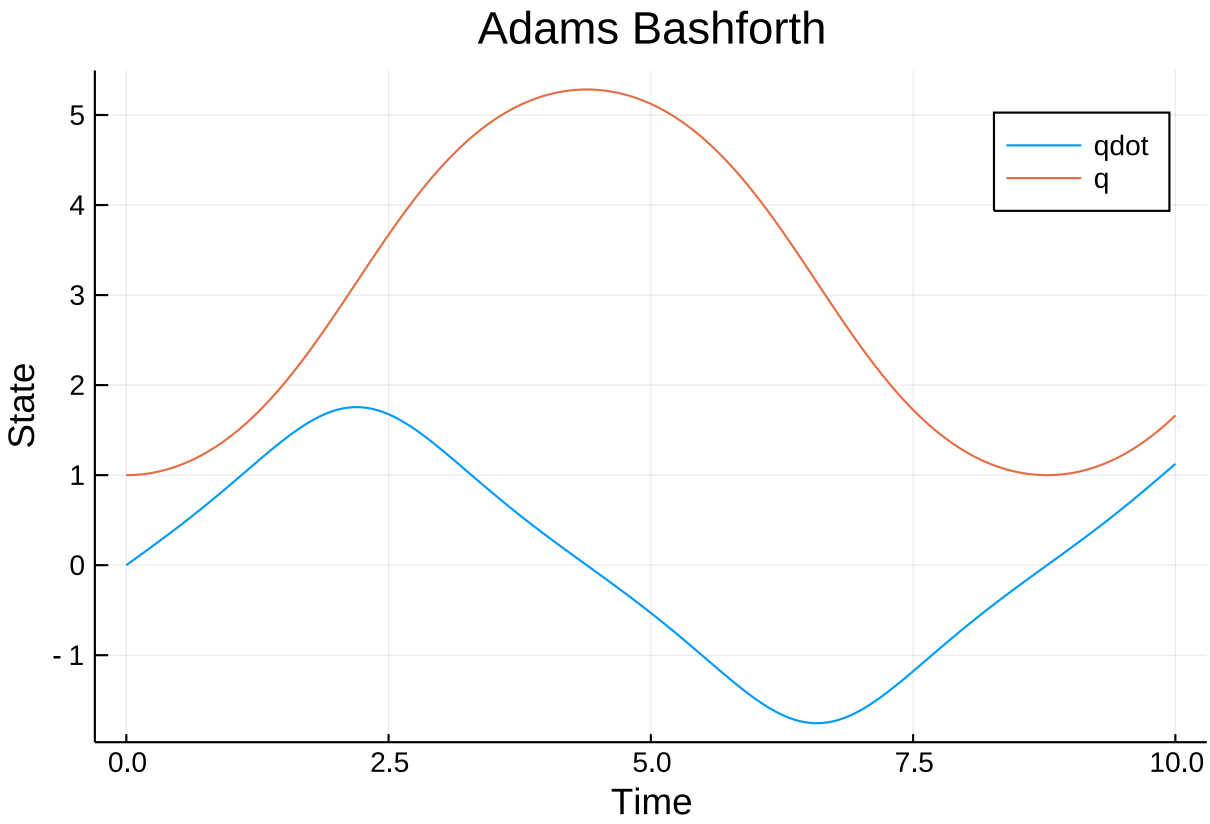
Now to solve them.

```
# Define givens
f(x) = [sin(x[2]) x[1]]
h = .01
x0 = [0 1.]
tf = 10;
```

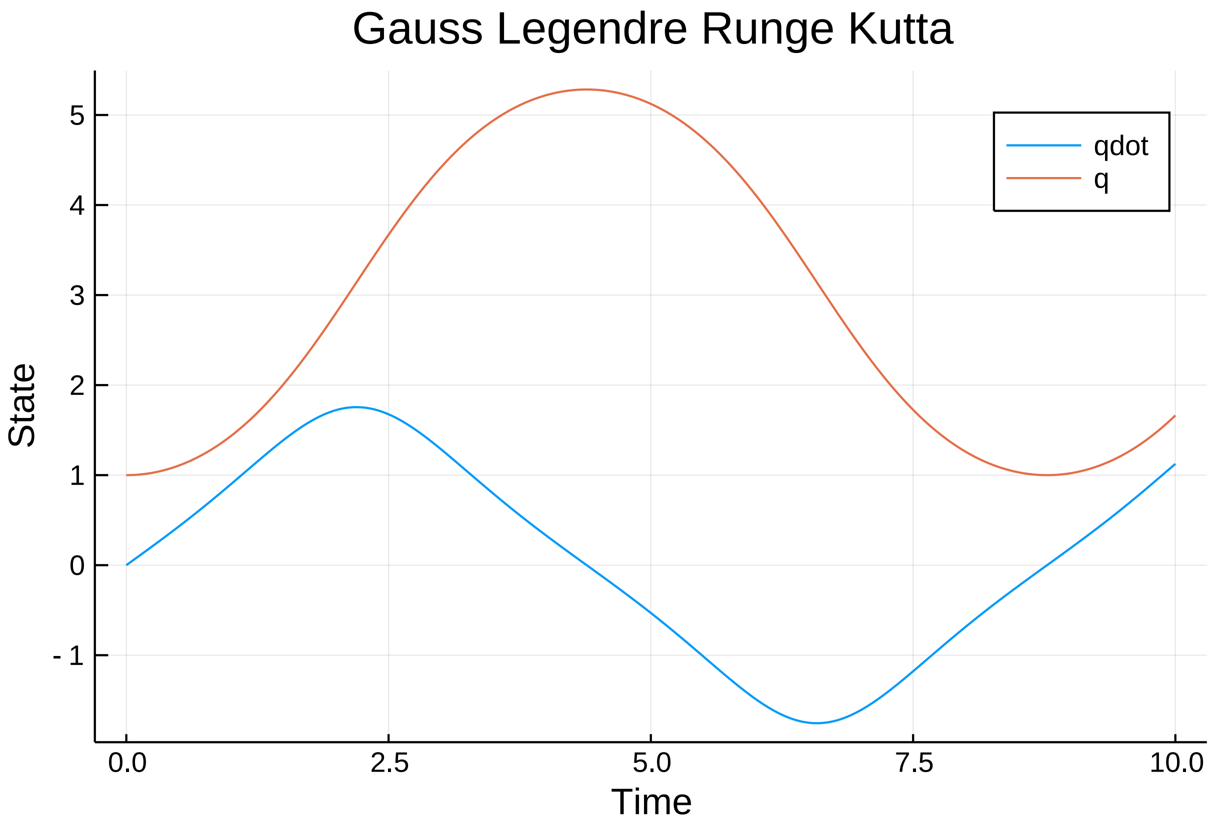
```
include("BackwardEuler.jl")
x_BE,t = BackwardEuler.beuler(f,tf,h,x0)
plot(t,x_BE[:,1],label = "qdot")
    plot!(t,x_BE[:,2],label = "q")
    xlabel!("Time")
    ylabel!("State")
    title!("Backwards Euler")
```



```
include("Adams_Bashforth2.jl")
x_AB,t = Adams_Bashforth2.ab2(f,tf,h,x0)
plot(t,x_AB[:,1],label = "qdot")
    plot!(t,x_AB[:,2],label = "q")
    xlabel!("Time")
    ylabel!("State")
    title!("Adams Bashforth")
```

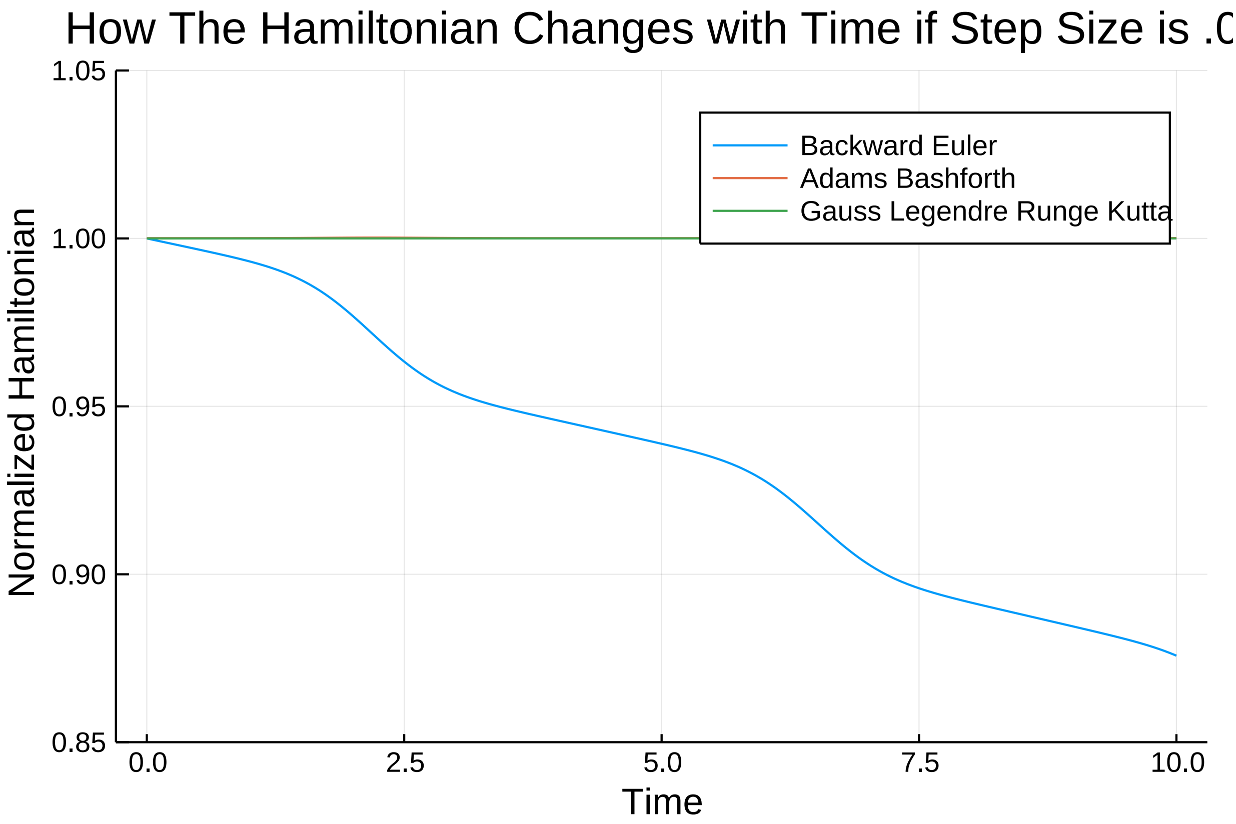



```
include("GL_RK.jl")
x_GL,t = GL_RK.gl_rk(f,tf,h,x0)
plot(t,x_GL[:,1],label = "qdot")
    plot!(t,x_GL[:,2],label = "q")
    xlabel!("Time")
    ylabel!("State")
    title!("Gauss Legendre Runge Kutta")
```



Now, I'll see what happens to the Hamiltonian as the solutions integrate in time.

```
H_BE = x_BE[:,1].^2/2 + cos.(x_BE[:,2])
H0 = x_BE[1,1].^2/2 + cos.(x_BE[1,2])

H_AB = x_AB[:,1].^2/2 +cos.(x_AB[:,2])

H_GL = x_GL[:,1].^2/2 +cos.(x_GL[:,2])

plot(t,H_BE/H0,label = "Backward Euler",yticks = .85:0.05:1.05,)
    plot!(t,H_AB/H0,label = "Adams Bashforth")
    plot!(t,H_GL/H0,label = "Gauss Legendre Runge Kutta")
    xlabel!("Time")
    ylabel!("Normalized Hamiltonian")
    title!("How The Hamiltonian Changes with Time if Step Size is .01")
    ylims!((.85,1.05))
```



AB2 was the easiest to implement for me becasue there were no implicit steps

Now, I'll compare the computation times

For Backwards Euler:

```
@elapsed BackwardEuler.beuler(f,tf,h,x0)
```

```
0.037092266
```

For Adams Bashforth 2

```
@elapsed Adams_Bashforth2.ab2(f,tf,h,x0)
```

```
0.024613975
```

For Gauss Legendre Runge Kutta

```
@elapsed GL_RK.gl_rk(f,tf,h,x0)
```

```
0.085601626
```

Gauss Legendre Runge Kutta took the longest

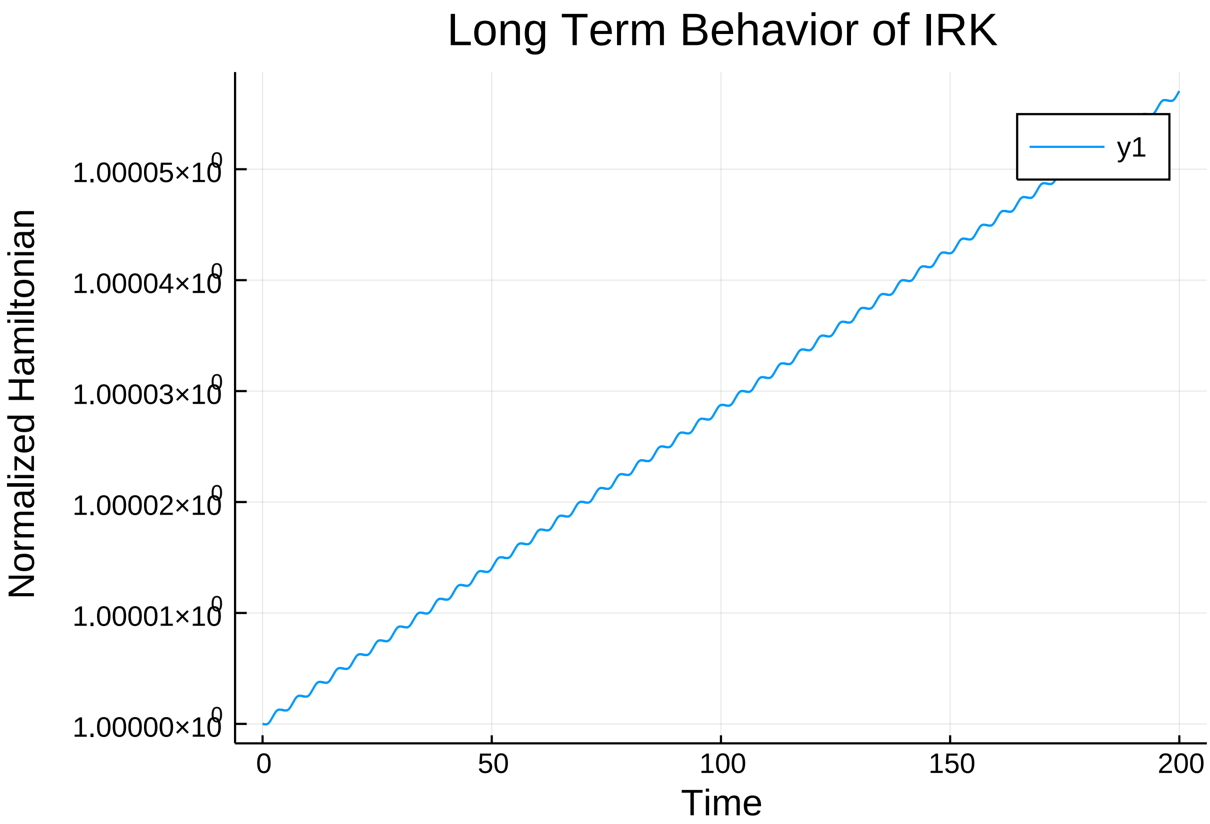
You asked us: Does the long term behavior of IRK method of (5.10) remain well behaved?

And I think the answer is that it depends. It probably depends on the step size and the eigenvalues of the problem.

So here, I integrated the above problem with IRK over a long time and with a fairly small step size.

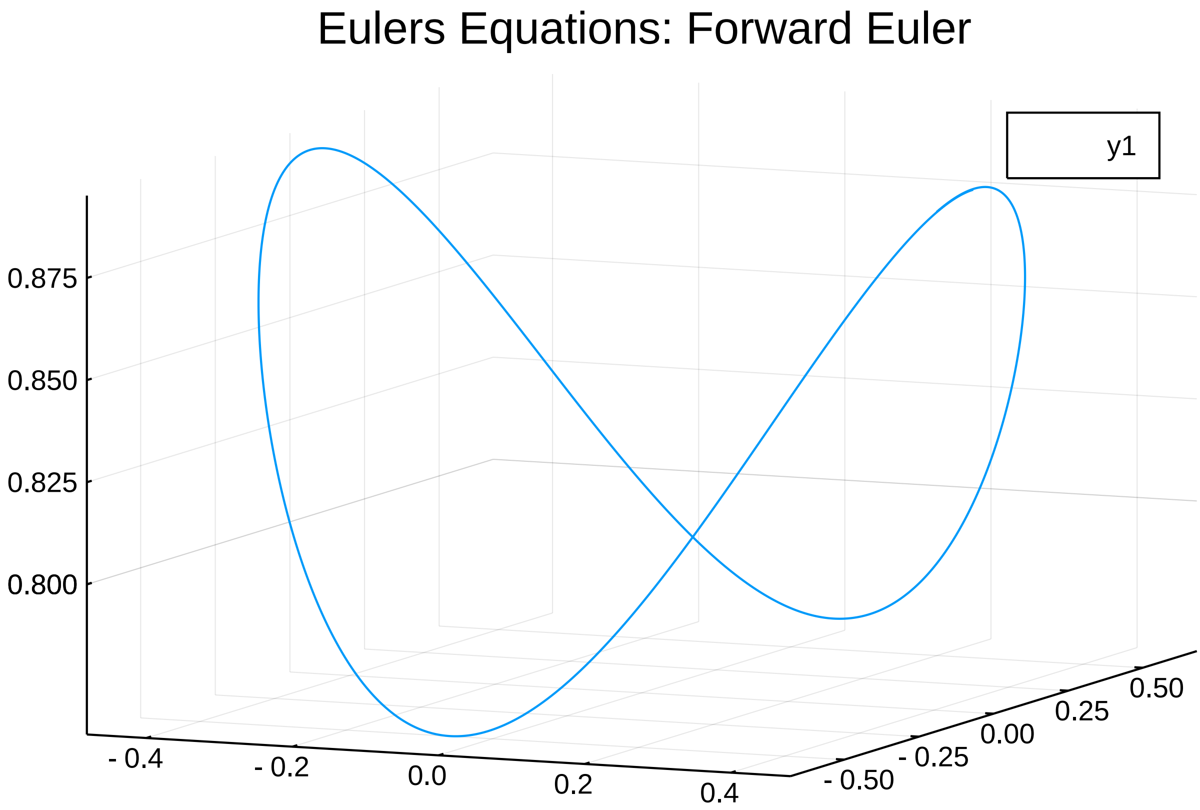
And the I'd say the long term behavior is pretty good!

```
tf_long = 200
h_long = .01
x_long,t = GL_RK.gl_rk(f,tf_long,h_long,x0)
H_GL_long = x_long[:,1].^2/2 +cos.(x_long[:,2])
plot(t,H_GL_long/H0)
    xlabel!("Time")
    ylabel!("Normalized Hamiltonian")
    title!("Long Term Behavior of IRK")
```

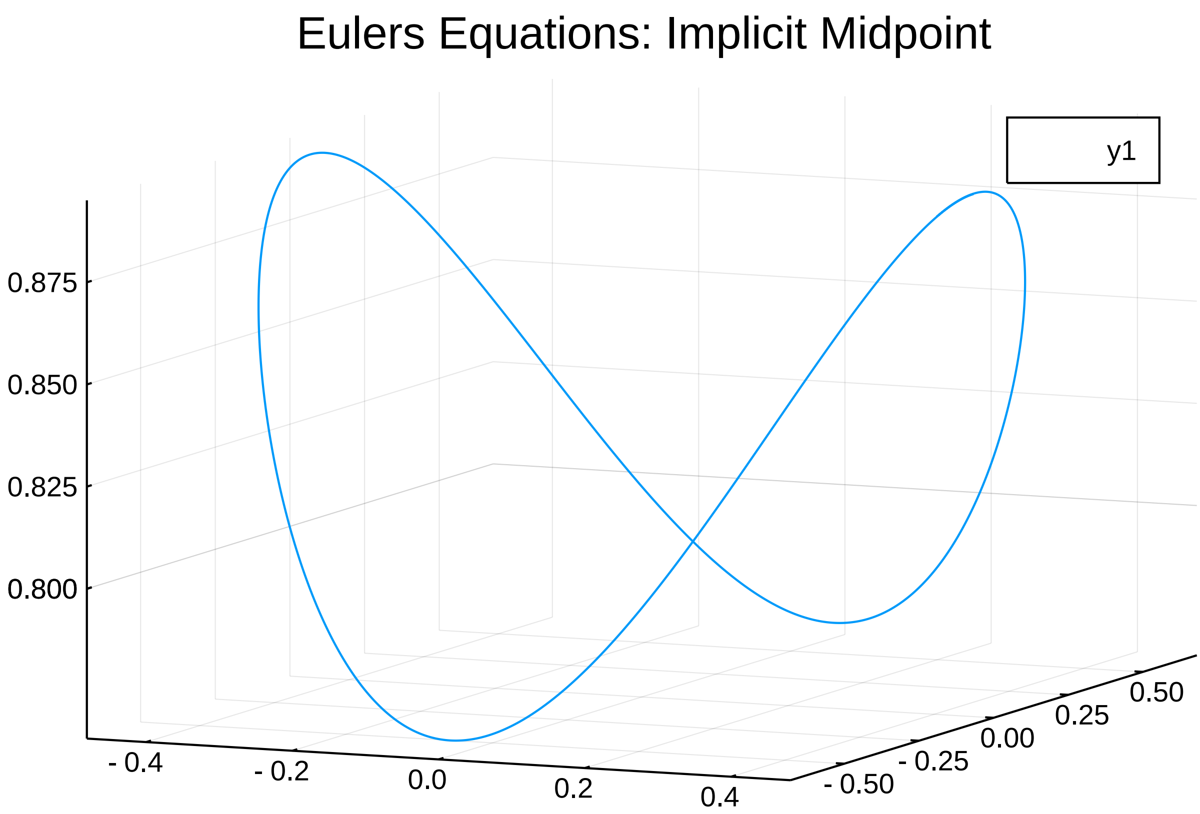


Now for The Euler Equations

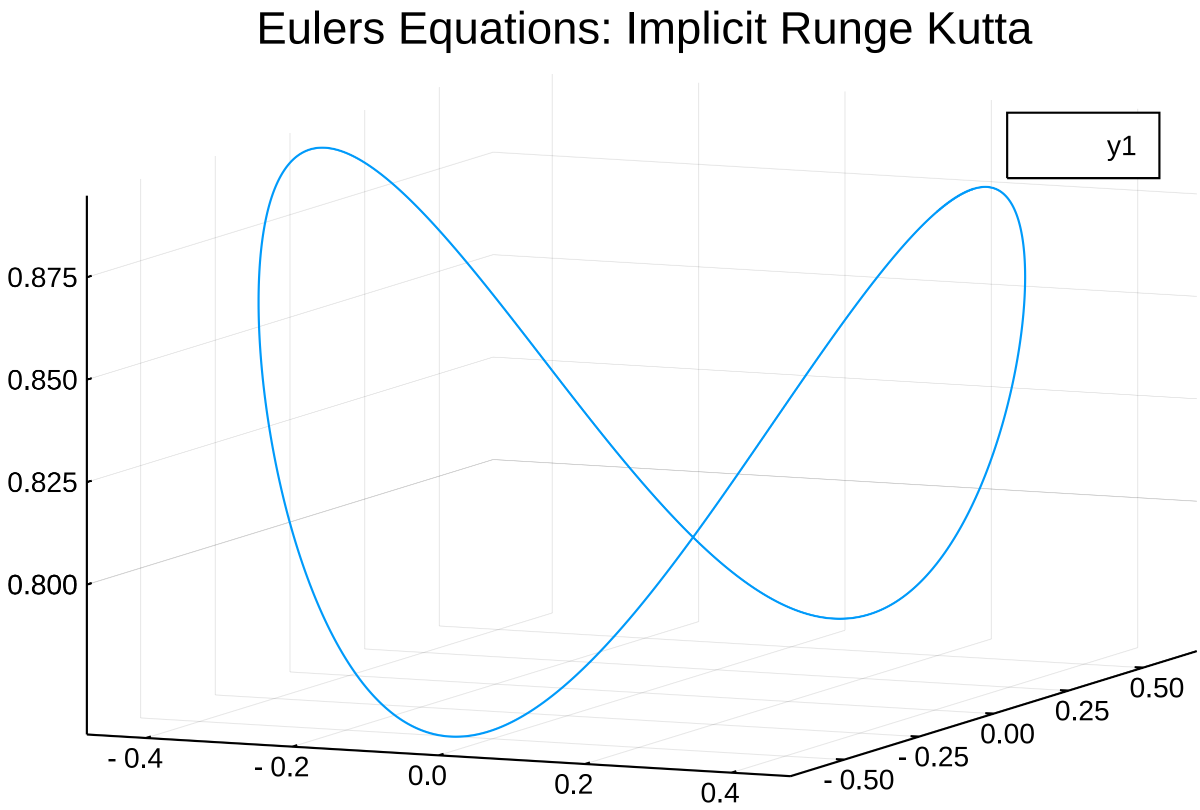
```
y0 = [cos(1.1) 0 sin(1.1)]

I1 = 2
I2 = 1
I3 = 2/3


a1 = (I2-I3)/I2/I3
a2 = (I3-I1)/I3/I1
a3 = (I1-I2)/I1/I2
f(y) = [a1*y[2]*y[3] a2*y[3]*y[1] a3*y[1]*y[2]]
h = .001
tf = 11
H0 = 1/2*(y0[1]^2/I1+y0[2]^2/I2+y0[3]^2/I3);

include("MyForwardEuler.jl")
y_FE,t = MyForwardEuler.feuler(f,tf,h,y0)
H_FE = 1/2*(y_FE[:,1].^2/I1+y_FE[:,2].^2/I2+y_FE[:,3].^2/I3)
plot3d(y_FE[:,1],y_FE[:,2],y_FE[:,3])
    title!("Eulers Equations: Forward Euler")
```




```
include("MyImplicitMidpoint.jl")
y_IM, t = MyImplicitMidpoint.my_implicit_mid(f,tf,h,y0)
H_IM = 1/2*(y_IM[:,1].^2/I1+y_IM[:,2].^2/I2+y_IM[:,3].^2/I3)
plot3d(y_IM[:,1],y_IM[:,2],y_IM[:,3])
    title!("Eulers Equations: Implicit Midpoint")
```



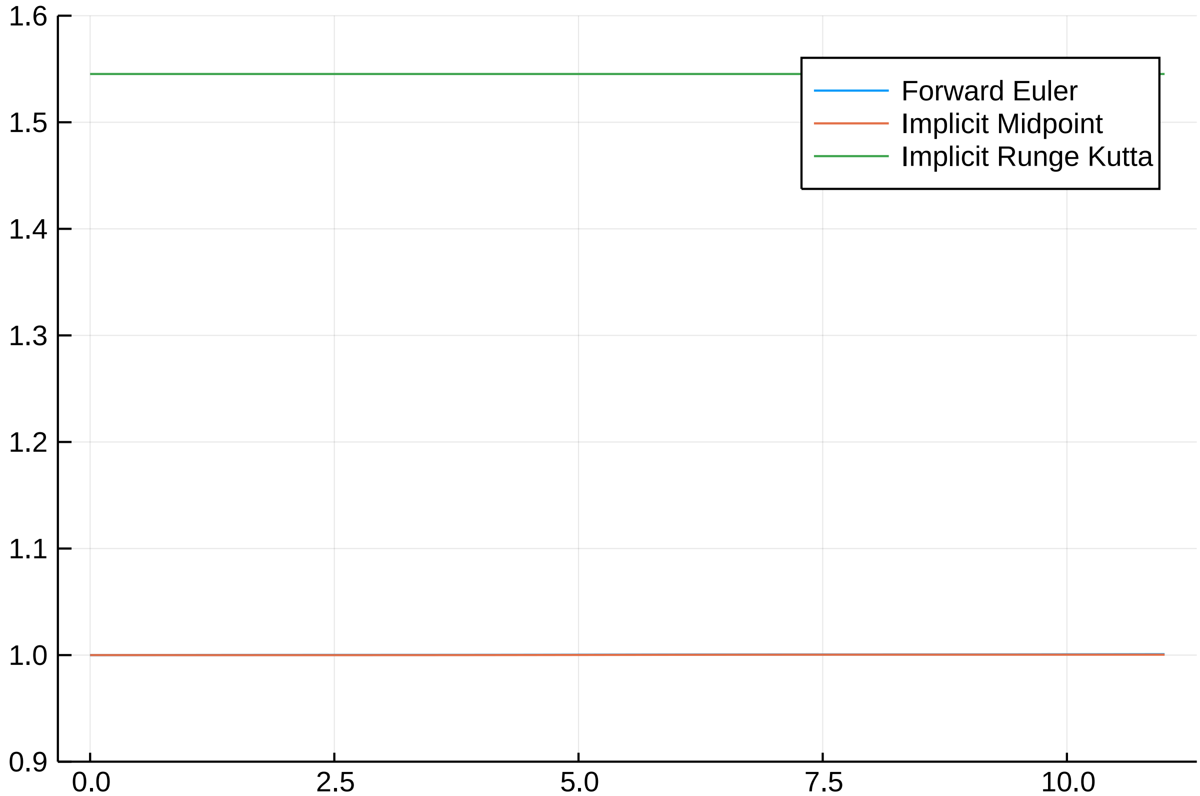

```
include("GL_RK.jl")
y_GL,ti = GL_RK.gl_rk(f,tf,h,y0)
H_GL = 1/2*(y_GL[:,1].^2/I1+y_GL[:,2].^2/I2+y_GL[:,3].^2/I3)/H0
plot3d(y_GL[:,1],y_GL[:,2],y_GL[:,3])
    title!("Eulers Equations: Implicit Runge Kutta")
```



The next plot, shows whether or not the Hamiltonian changes as the integrator moves forward. Only the IRK does not conserve the Hamiltonian.

Which, admittedly, is strange, because for the simple pendulum I said that IRK conserved the hamiltonian in its long term behavior....

```
plot(t,H_FE/H0,label= "Forward Euler")
    plot!(t,H_IM/H0,label = "Implicit Midpoint")
    plot!(t,H_GL/H0,label = "Implicit Runge Kutta")
    ylims!((.9,1.6))
```

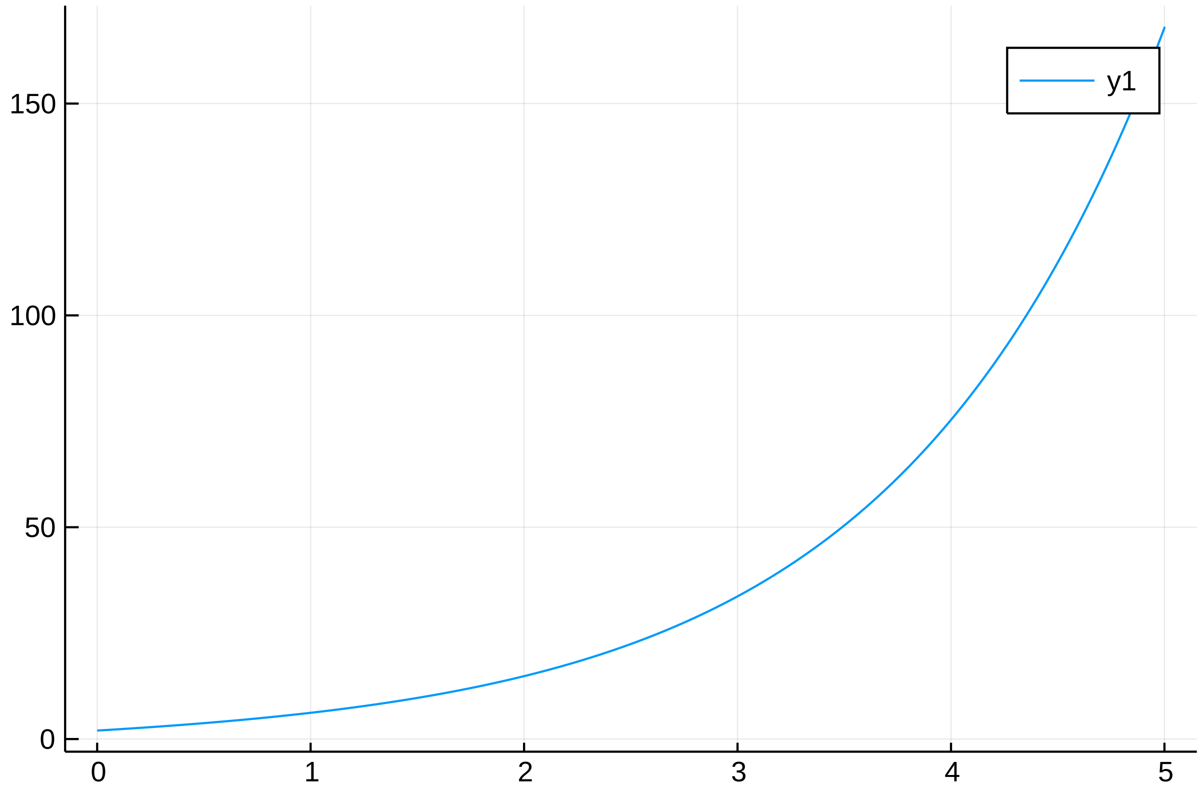


But... life goes on

Adaptive step size Runge-Kutta

First, I tried to do the example problem you gave.

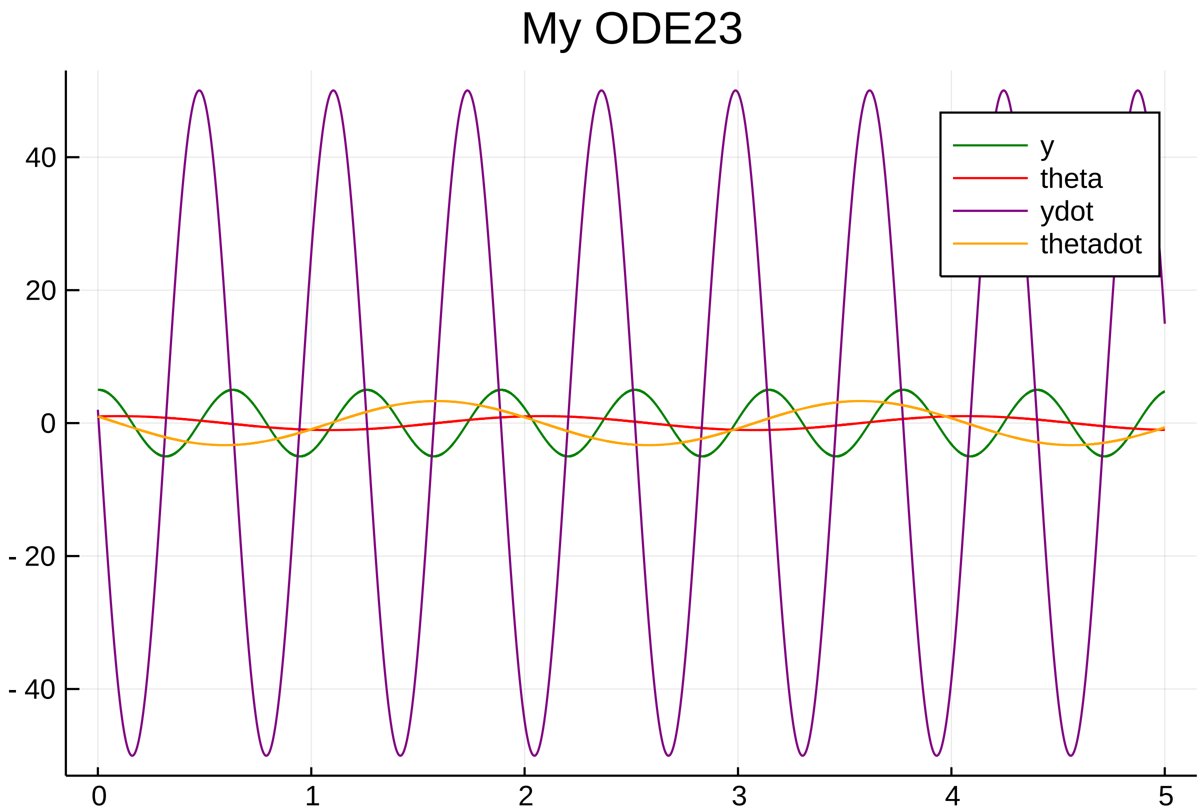
```
f(t,x) = [4*exp.(.8*t)] - .5*x
x0 = [2]
tf = 5.


include("Adaptive_time_step_Runge_Kutta.jl")

t,x = Adaptive_time_step_Runge_Kutta.vRK(f,tf,x0)
plot(t,x[:,1])
```



Which matched the solution given by Wolfram Alpha

Next, I used my function to solve the Airfoil problem and compared it to Julias built in solver

```
m = 1 #kg
J = m*.5^2
k = 100
kt = 2.5
f_af(t,x) = [x[3] x[4] -k*x[1]/m -kt*x[2]/J]
x0 = [5 1 2 1.]

t,x = Adaptive_time_step_Runge_Kutta.vRK(f_af,tf,x0)
plot(t,x[:,1],label = "y",color = "green")
    plot!(t,x[:,2],label = "theta",color = "red")
    plot!(t,x[:,3],label = "ydot",color = "purple")
    plot!(t,x[:,4],label = "thetadot",color = "orange")
    title!("My ODE23")
```




```
function builtin!(dx,x,p,t)
    dx[1] = x[3]
    dx[2] = x[4]
    dx[3] = -100*x[1]/1
    dx[4] = -2.5*x[2]/.5^2
end

tf=5.
x0 = [5 1 2 1.]
tspan = (0,tf)
prob = ODEProblem(builtin!,x0',tspan)
sol = solve(prob,BS3())
```

```
Error: UndefVarError: solve not defined
```

```
plot(sol,vars=(0,1),label = "y",color = "green")
```

```
Error: UndefVarError: sol not defined
```

```
plot!(sol,vars=(0,2),label = "theta",color = "red")
```

```
Error: UndefVarError: sol not defined
```

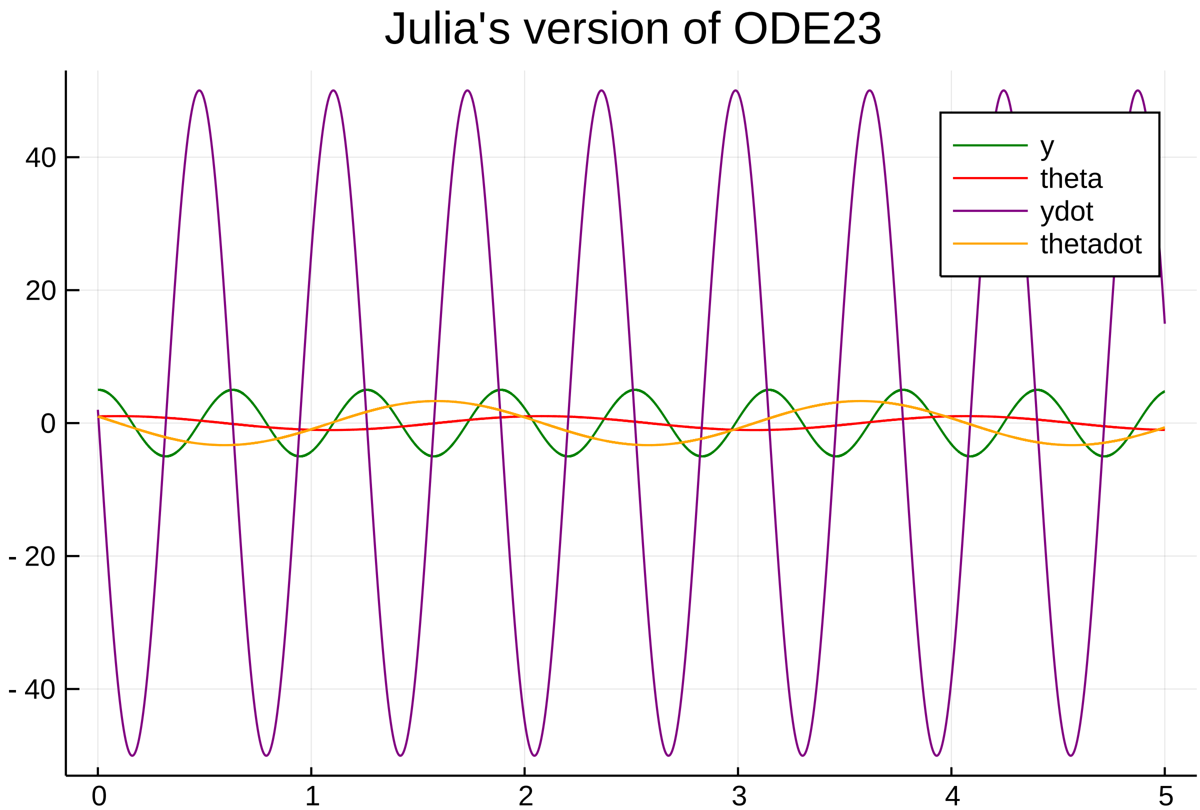
```
plot!(sol,vars=(0,3),label = "ydot",color = "purple")
```

```
Error: UndefVarError: sol not defined
```

```
plot!(sol,vars=(0,4),label = "thetadot",color = "orange")
```

```
Error: UndefVarError: sol not defined
```

```
title!("Julia's version of ODE23")
```



APPENDIX

```
module BackwardEuler

using LinearAlgebra

export beuler

function beuler(f, tf, h, x0; tol = 1e-4, iterMax = 500 )

    # x(i+1) = x(i) + h*f(i+1)

    time = 0:h:tf
    n = length(time)

    x = x0

    for i = 1:n-1
        # x(i+1) = x(i) + h*f( x(i+1) )

        # Predict via forward Euler x[i+1]
        y = x[i,:]' + h*f( x[i,:] )

        flag = 0
        iter = 0
        while flag == 0
            iter += 1
            y = x[i,:]' + h*f(y)

            residual = norm( y - x[i,:]' - h*f(y))

            if residual <= tol
```

```
                flag = 1
            elseif iter >= iterMax
                flag = -1
                error("Error: failed to converge")
            end
        end

        x = vcat(x,y)

    end

    return x, time

end

end;
```

```
module Adams_Bashforth2

using LinearAlgebra

export ab2

function ab2(f, tf, h, x0)
    time = 0:h:tf
    n = length(time)


    # Initial Condition
    x = x0

    # 1 Euler Step (this is cheap and possibly bad)
    fn = f(x)
    x = vcat(x0, x0 + h*fn)

    # AB2 the rest
    for i = 2:n-1
        fn_m1 = fn # f(x[i-1,:][1], x[i-1,:][2])
        fn = f(x[i,:])
        #x[i+1,:] = x[i,:] + h/2*( 3*f(x[i,:], time[i]) - f(x[i-1,:], time[i-1])  )
        xnext = x[i,:]' + h/2*( 3*fn - fn_m1 )
        x = vcat(x,xnext)
    end

    return x, time
end



end;
```

```
module GL_RK

using LinearAlgebra

export gl_rk
```

```
function gl_rk(f,tf,h,x0)
  time = 0:h:tf
  n = length(time)

  x = x0


  c1 = 1/2 - sqrt(3)/6
  c2 = 1/2 + sqrt(3)/6
  a11 = 1/4
  a12 = 1/4 - sqrt(3)/6
  a21 = 1/4 + sqrt(3)/6
  a22 = 1/4
  b1 = 1/2
  b2 = 1/2

  k1_guess = zeros(1,length(x0))
  k2_guess = zeros(1,length(x0))
  for i=1:n-1
    for j=1:500
      k1 = f(x[i,:]' + h*a11*k1_guess + h*a12*k2_guess)
      k2 = f(x[i,:]' + h*a21*k1 + h*a22*k2_guess)
      if norm(k1_guess - k1) <= .0001
        if norm(k2_guess - k2) <= .0001
          break
        end

      end
      k1_guess = k1
      k2_guess = k2

    end


    xnext = x[i,:]' + h*(b1*k1_guess + b2*k2_guess)
    x = vcat(x,xnext)
  end



 return x,time

end # function gl_rk

end;  # module GL_RK
```

```
module MyImplicitMidpoint

using LinearAlgebra

function my_implicit_mid(f,tf,h,x0)
    x = x0
    time = 0:h:tf
    n = length(time)

    for i=1:n-1
    xnext_guess = x[1,:]'
```

```
        for j = 1:500
            xnext = x[i,:]' + h*f(1/2*(x[i,:]' + xnext_guess))
            if norm(xnext-xnext_guess) <= .001
                x = vcat(x,xnext)
                break
            end
            xnext_guess = xnext
        end # j for loop
    end #i for loop

    return x,time


end # end my_implicit_mid

end;  # moduleMyImplicitMidpoint
```

```
module MyForwardEuler

export feuler

function feuler(f,tf,h,x0)
    x = x0
    time = 0:h:tf
    n = length(time)
    for i=1:n-1
        xnext = x[i,:]' + h*f(x[i,:])
        x = vcat(x,xnext)
    end
    return x,time


end # function feuler


end; # modul MyForwardEuler
```

```
module Adaptive_time_step_Runge_Kutta

using LinearAlgebra

export vRk

function vRK(f,tf,x0;h0 = 1.,tol = 1e-8)

    c1 = 1/2
    c2 = 3/4
    c3 = 1

    a1 = 1/2
    a21 = 0
    a22 = 3/4
    a31 = 2/9
    a32 = 1/3
    a33 = 4/9
```

```
b11 = 2/9
b12 = 1/3
b13 = 4/9
b14 = 0
b21 = 7/24
b22 = 1/4
b23 = 1/3
b24 = 1/8

t = [0]
x = x0
h = h0
flag = 0
i = 0
error_m1 = 100
while flag == 0
    i += 1
    for j= 1:500
        k1 = f(t[i],x[i,:]')
        k2 = f(t[i]+c1*h,x[i,:]'+h*k1*a1)
        k3 = f(t[i] + c2*h, x[i,:]' + h*k1*a21 + h*k2*a22)

        xnext = x[i,:]' + h*(b11*k1 + b12*k2 + b13*k3)

        k4 = f(t[i]+c3*h,xnext)

        znext = x[i,:]' + h*(b21*k1 + b22*k2 + b23*k3 + b24*k4)

        error = norm(xnext-znext)


        if error <= tol
            t = vcat(t,t[i].+h)
            h = .9*h*min(max(error_m1/error,.3),2)
            x = vcat(x,xnext)
            break
        else
            h = h/2
        end #if error


        error_m1 = error
    end # j for loop

    if t[i] >= tf
        flag = 1
    end
    if i >= 50000
        flag = -1
        error("Too many points")
    end



end # while loop

return t, x
```

```
end #function vRK

end;  # moduleAdaptive_time_step_Runge_Kutta
```