PURDUE UNIVERSITY · CS 31400
NUMERICAL ANALYSIS

HOMEWORK
Patrick Talley
November 19, 2016

Please answer the following questions in complete sentences in submit the solution on Blackboard November 22, 2016 by 5pm.

Currently incomplete, there will be other problems added.

Homework 6

Problem 1: Derive a method (20 points)

1. Chapter 10, Exercise 2 (10 points)
2. Chapter 10, Exercise 3 (10 points)

Problem 1 Solution

1.

$$\int_0^1 ae^x + bcos(\frac{\pi}{2}x)$$

$$ae^x + \frac{2b}{\pi}sin(\frac{\pi}{2}x)\Big|_0^1$$

$$2(e-1) + \frac{2b}{\pi} = A_of(0) + A_1f(1)$$

$$A_0 + A_1e = e - 1$$

$$A_o = \frac{2}{\pi}$$

$$A_1 = \frac{(e-1)\pi - 2}{e\pi}$$

$$\int_0^1 ae^x + bcos(\frac{\pi}{2}x) = \frac{2}{\pi}(a+b) + \frac{(e-1)\pi-2}{e\pi}ae$$

2. (a)

$$f(x) = a + bx$$

$$\int_{-1}^1 f(x) = 2a$$

$$f(\alpha) + f(-\alpha) = 2a \implies \forall\ \alpha \text{ this is exact}$$

(b)

$$f(x) = a + bx + cx^2 + dx^3$$

$$\int_{-1}^1 f(x)dz = 2a + \frac{2c}{3}$$

$$f(\alpha) + f(-\alpha) = a + b\alpha + c\alpha^2 + d\alpha^3 + a - b\alpha + c\alpha^2 - d\alpha^3 = 2a + 2c\alpha^2$$

$$\therefore\ \alpha = \frac{1}{\pm\sqrt{3}} \text{ are the only value such that the poly is exact}$$

(c)

$$f(x) = a + bx + cx^3 + dx^4$$

$$\int_{-1}^{1} f(x)dx = 2a + \frac{2d}{5}$$

$$f(\alpha) + f(-\alpha) = a + b\alpha + c\alpha^3 + d\alpha^4 + a - b\alpha - c\alpha^3 + d\alpha^4 = 2a + 2d\alpha^4$$

$$\therefore\ \alpha = ^{+}_{-} 5^{\frac{-1}{4}}$$

Problem 2: An error study (10 points)

Complete chapter 10, Problem 6c.

Problem 2 Solution

Let $k = \frac{x_i + x_{i+1}}{2}$

$$\int_{x_i}^{x_i+h} f(x)dx = \int f(k) + \int (x-k)f'(k) + \int \frac{(x-k)^2}{2} f''(\xi)$$

$$= xf(k) + \frac{(x-k)^2}{2} f'(k) + \frac{(x-k)^3}{6} f''(\xi) \Big|_{x_i}^{x_i+h}$$

$$\approx O(h)f(k) + O(h^2)f'(k) + O(h^3)f''(\xi)$$

$\implies$ the error term is $O(h^3)$. Then, let $n = \frac{b-a}{h}$
So, $h \sum_{i=1}^{n} = O(h^2)$

Problem 3: A new hire (15 points)

You have just been hired at Aperature Science Labs, one of the premier institutes of numerical computing. One of their specialities is evaluating numerical representations of complex phenomenon. You've been brought on in a top-secret project that is going to study the basics of numerical integration that underlie Aperature's advanced simulations. Your goal: produce a report on the accuracy and computational cost of evaluating

$$\int_0^1 \cos(x^2)\, dx$$

via numeric integration. Your report should contain a summary recommendation on your findings with state-of-the-art methods to evaluate this routine. It should include at least three different methods; and some form of chart that your boss can present to the weekly meeting.

Problem 3 Solution

Problem3

November 17, 2016

```
In [8]: f = x -> cos(x^2)

Out[8]: (anonymous function)

In [4]: function midInt(n,f)
            h = (1.0-0)/n
            xsteps = collect(0:h:1)
            mid_int = 0
            for i=2:n
                mid_int += f((xsteps[i]+xsteps[i-1])/2)
            end
            mid_int *= h
            return mid_int
        end

Out[4]: midInt (generic function with 2 methods)

In [5]: function trapInt(n,f)
            h = (1.0-0)/n
            xsteps = collect(0:h:1)
            trap_int = 0
            for i=2:n
                trap_int += (f(xsteps[i])+f(xsteps[i-1]))
            end
            trap_int *= h/2
            return trap_int
        end

Out[5]: trapInt (generic function with 2 methods)

In [6]: function simpInt(n,f)
            h = (1.0-0)/n
            xsteps = collect(0:h:1)
            simp_int = 0
            for i=2:n
                simp_int +=  ( f(xsteps[i-1]) + 4*f((xsteps[i-1]+xsteps[i])/2) + f(xsteps[i]) )
            end
            simp_int *= h/6
            return simp_int
        end

Out[6]: simpInt (generic function with 2 methods)

In [10]: actual = quadgk(f,0,1)[1]

Out[10]: 0.904524237900272
```

```
In [69]: error_mid = abs(mid_int - actual)
         error_trap = abs(trap_int - actual)
         error_simp = abs(simp_int - actual)

         @printf("Method \t\t\t Abs error \t Computation\n")
         @printf("Composite midpoint \t %.8f \t 4n \n", error_mid)
         @printf("Composite trapezoid \t %.8f \t 4n \n", error_trap)
         @printf("Composite Simpson's \t %.8f \t 7n", error_simp)
```

```
Method                          Abs error         Computation
Composite midpoint          0.00054107          4n
Composite trapezoid          0.00054128          4n
Composite Simpson's          0.00054114          7n
```

```
In [ ]: lastE = 7
        mError = zeros(lastE,1)
        tError = zeros(lastE,1)
        sError = zeros(lastE,1)

        @printf("10^x \t Simpsons \t\t\t Trapezoid \t\t\t Midpoint\n")
        for i=1:(lastE)
            mError[i] = abs(midInt(10^(i), f) -actual)
            tError[i] = abs(trapInt(10^(i), f) -actual)
            sError[i] = abs(simpInt(10^(i), f) -actual)

            @printf("%d \t %.16f \t\t %.16f \t\t %.16f \n", i, mError[i],tError[i], sError[i])
        end
```

```
10^x          Simpsons                          Trapezoid                          Midpoint
1          0.0612637202364131                0.0628925177178588                0.0618066527302285
2          0.0054796773363898                0.0055002335204161                0.0054865293977314
3          0.0005410731737462                0.0005412830609605                0.0005411431361522
4          0.0000540379435920                0.0000540400467866                0.0000540386446575
5          0.0000054031002166                0.0000054031212253                0.0000054031072072
6
```

```
In [ ]:
```

Problem 4: Sleepless nights (20 points)

1. Chapter 11, Problem 3 (10 points)
2. Chapter 11, Problem 8 (10 points)

Problem 4 Solution

1.
$$\frac{d}{dt}t^{3/2} = \frac{3}{2}t^{1/2}$$

Sub in $y = t^{3/2} \implies t = y^{2/3}$

$$y' = \frac{3}{2}(y^{2/3})^(1/2) = \frac{3}{2}y^{1/3}$$

The reason this will not work is that it requires the value of y to change, and since it will be 0 Euler's method will not be able to properly evaluate the derivative.

In [6]:
```
function euler(t,h)
    y = 3/2 * t^(1/3)
    return y + h*y
end
```

Out[6]: euler (generic function with 1 method)

In [7]:
```
h=0.1
ts = 0:h:3
n = length(ts)
y = zeros(n,1)
actual = zeros(n,1)
@printf("Time \t Approx \t Exact \t\t Error Percent \n")
@printf("%d \t %.8f \t %.8f \t %.4f\n", ts[1], y[1], actual[1], abs(actual[
1]-y[1]))
for i=2:n
    y[i] = y[i-1] + euler(y[i-1],h)
    actual[i]=ts[i]^(3/2)
    @printf("%.2f \t %.8f \t %.8f \t %.4f\n", ts[i], y[i], actual[i], abs(a
ctual[i]-y[i])/actual[i]*100)
end
```

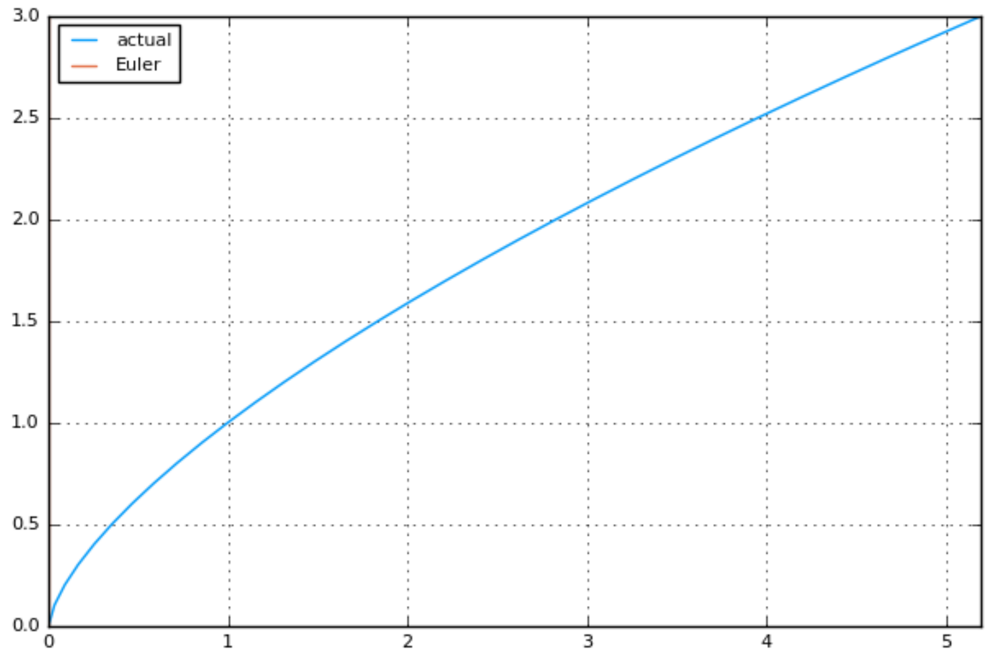
```
Time     Approx          Exact           Error Percent
0        0.00000000      0.00000000      0.0000
0.10     0.00000000      0.03162278      100.0000
0.20     0.00000000      0.08944272      100.0000
0.30     0.00000000      0.16431677      100.0000
0.40     0.00000000      0.25298221      100.0000
0.50     0.00000000      0.35355339      100.0000
0.60     0.00000000      0.46475800      100.0000
0.70     0.00000000      0.58566202      100.0000
0.80     0.00000000      0.71554175      100.0000
0.90     0.00000000      0.85381497      100.0000
1.00     0.00000000      1.00000000      100.0000
1.10     0.00000000      1.15368973      100.0000
1.20     0.00000000      1.31453414      100.0000
1.30     0.00000000      1.48222805      100.0000
1.40     0.00000000      1.65650234      100.0000
1.50     0.00000000      1.83711731      100.0000
1.60     0.00000000      2.02385770      100.0000
1.70     0.00000000      2.21652882      100.0000
1.80     0.00000000      2.41495342      100.0000
1.90     0.00000000      2.61896926      100.0000
2.00     0.00000000      2.82842712      100.0000
2.10     0.00000000      3.04318912      100.0000
2.20     0.00000000      3.26312733      100.0000
2.30     0.00000000      3.48812270      100.0000
2.40     0.00000000      3.71806401      100.0000
2.50     0.00000000      3.95284708      100.0000
2.60     0.00000000      4.19237403      100.0000
2.70     0.00000000      4.43655272      100.0000
2.80     0.00000000      4.68529615      100.0000
2.90     0.00000000      4.93852205      100.0000
3.00     0.00000000      5.19615242      100.0000
```

In [3]:
```
using Plots
plot(actual,ts,label="actual")
plot!(y,ts,label="Euler")
```

```
[Plots.jl] Initializing backend: pyplot
```

```
WARNING: using Plots.h in module Main conflicts with an existing identifier
.
```

Out[3]:

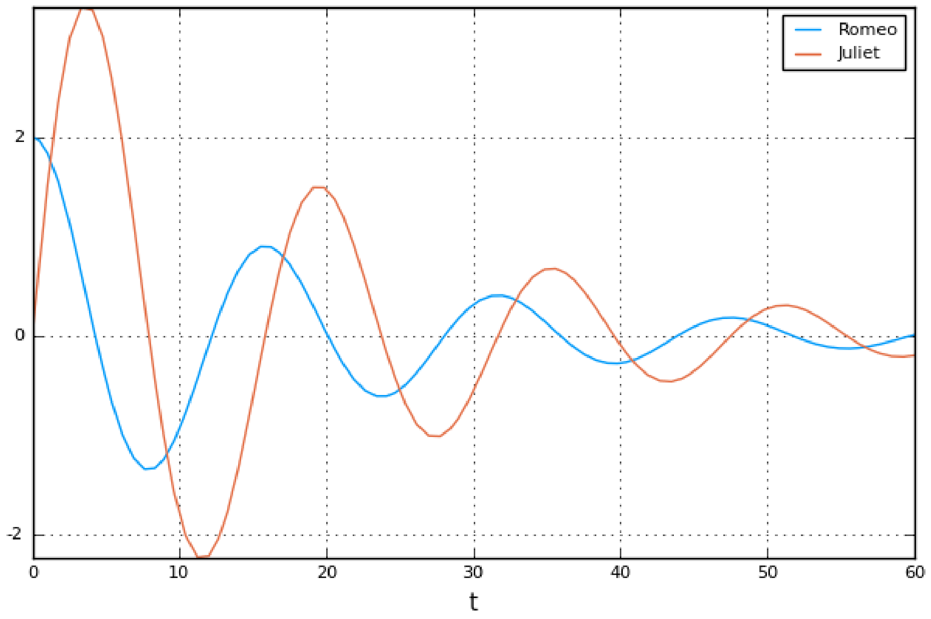


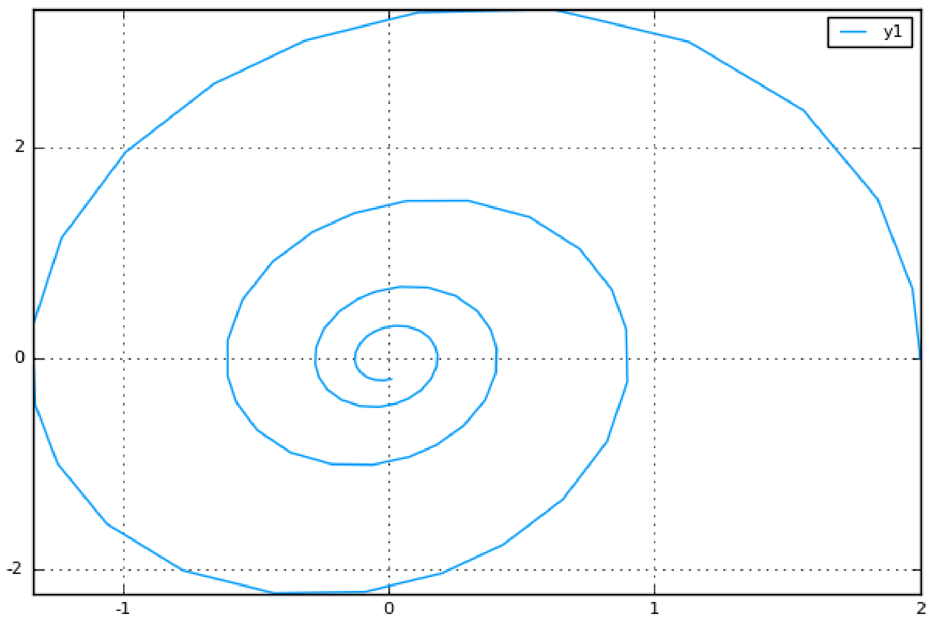
In [22]:
```
savefig("problem4A.png")
```

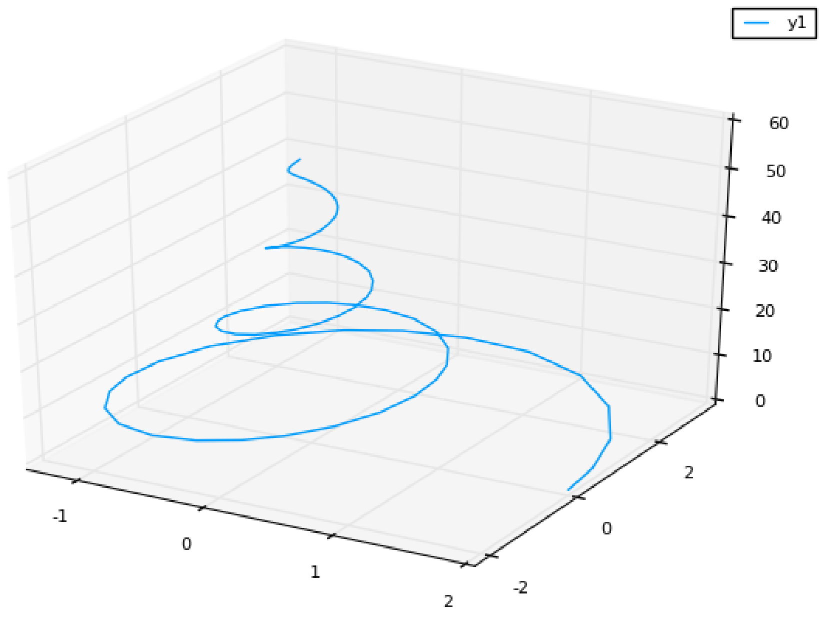
In [ ]:

2. (a) The equations in 11.23 have Juliet's emotions brought down solely by Romeo's whose will rise as a result. This makes an elliptical orbit as observed.

(b)







(c) Because Juliet's emotions are brought down by her own state, the emotions will converge towards indiference.

Problem4B

November 17, 2016

```
In [3]: using Plots, ODE;

In [1]: function love(t,f)
            (x, y) = f
            dx_dt = -0.2*y
            dy_dt = 0.8*x -0.1*y

            [dx_dt; dy_dt]
        end;

In [5]: start = [2.0; 0.0]
        ts = [0.0; 60.0];

In [6]: t, res = ode45(love, start, ts)

        r = map(x -> x[1], res)
        j = map(x -> x[2], res);

In [9]: plot(t, r, label="Romeo")
        plot!(t, j, label="Juliet")
        xlabel!("t")

In [10]: plot(r,j)

In [11]: plot(r,j,t)

In [ ]:
```

Problem 5: A choice! (15 points)

Please choose your own adventure! You only need to do one of the following problems.

Theory Chapter 11, Problem 11 (You will have to read in the book about how to show multi-step methods are convergent.)

Experiment Chapter 11, Problem 14. Here, you should use the version of 4th order Runge-Kutta in Section 11.2.5. Make sure you show at least one illustration of the numerical problem described with small $w(0)$.

Problem 5 Solution

1.
$$\begin{aligned} x'(t) &= z(t) \\ y'(t) &= w(t) \\ z'(t) &= \frac{-x}{(\sqrt{x^2+y^2})^3} \\ w'(t) &= \frac{-y}{(\sqrt{x^2+y^2})^3} \end{aligned}$$

2.

3. My method requires using the correct value of h which must be calculated based on the question. ODE45 already handles that, and is more accurate in portraying the real system.

In [83]:
```
using Plots
```

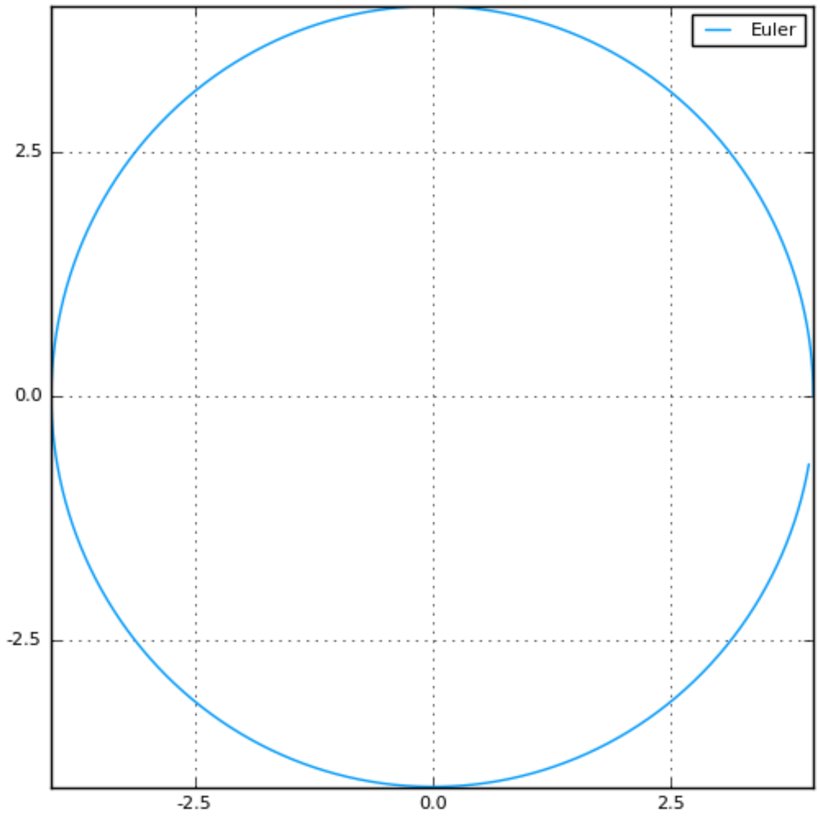
In [66]:
```
function runge_kutta(t,y,f,h)
    q1 = f(t, h*y)
    q2 = f(t + (h/2), y + (h/2) * q1)
    q3 = f(t + (h/2), y + (h/2) * q2)
    q4 = f(t + h, y + h*q3)
    return y + h/6 * (q1+ 2*q2 + 2*q3 + q4)
end
```

Out[66]:
```
runge_kutta (generic function with 1 method)
```

In [77]:
```
# Euler Method
h = 0.0025
t_max = 50
n = length(1:h:t_max)

x_e = zeros(n,1)
y_e = zeros(n,1)
w_e = zeros(n,1)
z_e = zeros(n,1)

dz_dt = (x,y) -> (-x)/(sqrt(x^2 + y^2)^3)
dw_dt = (x,y) -> (-y)/(sqrt(x^2 + y^2)^3)

x_e[1] = 4
w_e[1] = 0.5

for t=2:n
    z_e[t] = z_e[t-1] + h*dz_dt(x_e[t-1],y_e[t-1])
    w_e[t] = w_e[t-1] + h*dw_dt(x_e[t-1],y_e[t-1])
    x_e[t] = x_e[t-1] + h*z_e[t-1]
    y_e[t] = y_e[t-1] + h*w_e[t-1]
end
```

```
In [107]: plot(x_e,y_e,size=(500,500),label="Euler")
```

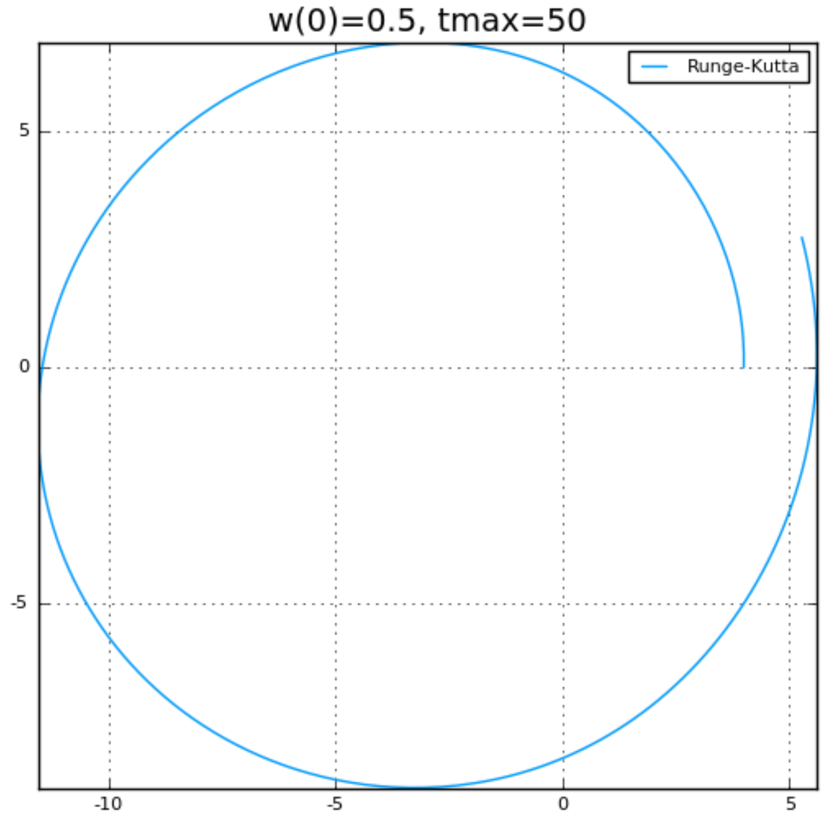
```
Out[107]:
```



In [85]:
```
# Runge Kutta
h = 0.25
n = length(1:h:t_max)

x_r = zeros(n,1)
y_r = zeros(n,1)
w_r = zeros(n,1)
z_r = zeros(n,1)

x_r[1] = 4
w_r[1] = 0.5

dz_dt = (x,y) -> (-x)/(sqrt(x^2 + y^2)^3)
dw_dt = (x,y) -> (-y)/(sqrt(x^2 + y^2)^3)

for t=2:n
    q1_z = dz_dt(x_r[t-1], h * y_r[t-1])
    q2_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_z)
    q3_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_z)
    q4_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_z)

    q1_w = dw_dt(x_r[t-1], h * y_r[t-1])
    q2_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_w)
    q3_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_w)
    q4_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_w)

    z_r[t] = z_r[t-1] + (h/6)*(q1_z + 2*q2_z + 2*q3_z + q4_z)#runge_kutta(x
_r[t-1], y_r[t-1], dz_dt, h)
    w_r[t] = w_r[t-1] + (h/6)*(q1_w + 2*q2_w + 2*q3_w + q4_w)#runge_kutta(x
_r[t-1], y_r[t-1], dw_dt, h)
    x_r[t] = x_r[t-1] + h*z_r[t-1]
    y_r[t] = y_r[t-1] + h*w_r[t-1]
end
```

```
In [89]: plot(x_r,y_r,size=(500,500),label="Runge-Kutta",title="w(0)=0.5, tmax=50")
```

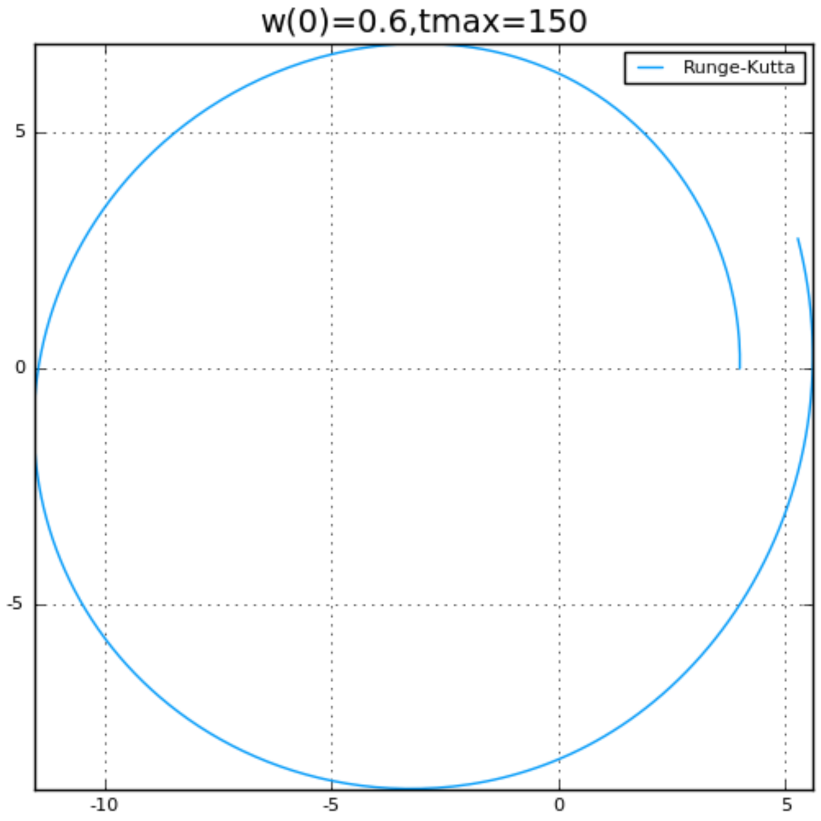
Out[89]:



In [87]:
```
# Runge Kutta
h = 0.5
t_max=150
n = length(1:h:t_max)

x_r = zeros(n,1)
y_r = zeros(n,1)
w_r = zeros(n,1)
z_r = zeros(n,1)

x_r[1] = 4
w_r[1] = 0.6

dz_dt = (x,y) -> (-x)/(sqrt(x^2 + y^2)^3)
dw_dt = (x,y) -> (-y)/(sqrt(x^2 + y^2)^3)

for t=2:n
    q1_z = dz_dt(x_r[t-1], h * y_r[t-1])
    q2_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_z)
    q3_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_z)
    q4_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_z)

    q1_w = dw_dt(x_r[t-1], h * y_r[t-1])
    q2_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_w)
    q3_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_w)
    q4_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_w)

    z_r[t] = z_r[t-1] + (h/6)*(q1_z + 2*q2_z + 2*q3_z + q4_z)#runge_kutta(x
_r[t-1], y_r[t-1], dz_dt, h)
    w_r[t] = w_r[t-1] + (h/6)*(q1_w + 2*q2_w + 2*q3_w + q4_w)#runge_kutta(x
_r[t-1], y_r[t-1], dw_dt, h)
    x_r[t] = x_r[t-1] + h*z_r[t-1]
    y_r[t] = y_r[t-1] + h*w_r[t-1]
end
```

In [88]:
```
plot(x_r,y_r,size=(500,500),label="Runge-Kutta",title="w(0)=0.6,tmax=150")
```

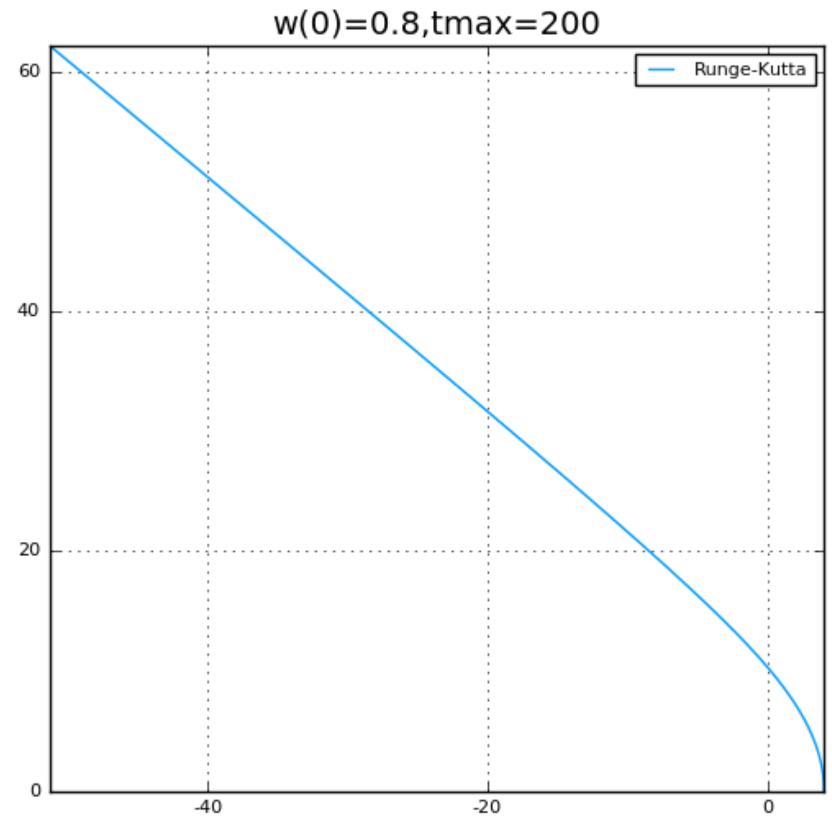
Out[88]:



In [93]:
```
# Runge Kutta
h = 0.5
t_max = 200
n = length(1:h:t_max)

x_r = zeros(n,1)
y_r = zeros(n,1)
w_r = zeros(n,1)
z_r = zeros(n,1)

x_r[1] = 4
w_r[1] = 0.8

dz_dt = (x,y) -> (-x)/(sqrt(x^2 + y^2)^3)
dw_dt = (x,y) -> (-y)/(sqrt(x^2 + y^2)^3)

for t=2:n
    q1_z = dz_dt(x_r[t-1], h * y_r[t-1])
    q2_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_z)
    q3_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_z)
    q4_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_z)

    q1_w = dw_dt(x_r[t-1], h * y_r[t-1])
    q2_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_w)
    q3_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_w)
    q4_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_w)

    z_r[t] = z_r[t-1] + (h/6)*(q1_z + 2*q2_z + 2*q3_z + q4_z)#runge_kutta(x
_r[t-1], y_r[t-1], dz_dt, h)
    w_r[t] = w_r[t-1] + (h/6)*(q1_w + 2*q2_w + 2*q3_w + q4_w)#runge_kutta(x
_r[t-1], y_r[t-1], dw_dt, h)
    x_r[t] = x_r[t-1] + h*z_r[t-1]
    y_r[t] = y_r[t-1] + h*w_r[t-1]
end
```

```
In [94]: plot(x_r,y_r,size=(500,500),label="Runge-Kutta",title="w(0)=0.8,tmax=200")
```

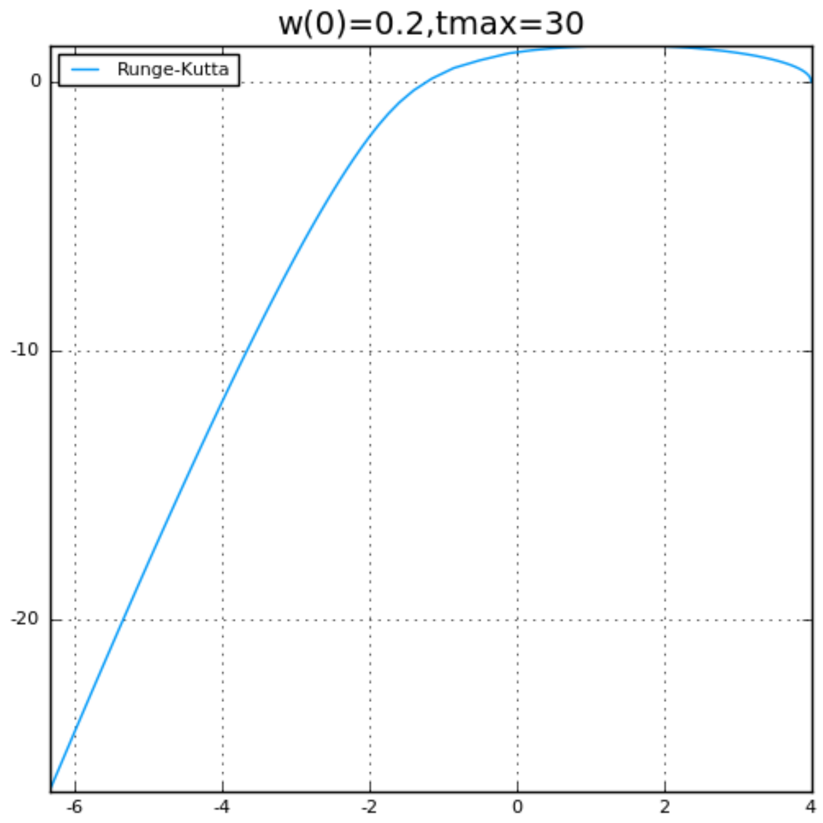
Out[94]:



In [97]:
```
# Runge Kutta
h = 0.25
t_max = 30
n = length(1:h:t_max)

x_r = zeros(n,1)
y_r = zeros(n,1)
w_r = zeros(n,1)
z_r = zeros(n,1)

x_r[1] = 4
w_r[1] = 0.2

dz_dt = (x,y) -> (-x)/(sqrt(x^2 + y^2)^3)
dw_dt = (x,y) -> (-y)/(sqrt(x^2 + y^2)^3)

for t=2:n
    q1_z = dz_dt(x_r[t-1], h * y_r[t-1])
    q2_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_z)
    q3_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_z)
    q4_z = dz_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_z)

    q1_w = dw_dt(x_r[t-1], h * y_r[t-1])
    q2_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q1_w)
    q3_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + (h/2) * q2_w)
    q4_w = dw_dt(x_r[t-1]+ (h/2), y_r[t-1] + h * q3_w)

    z_r[t] = z_r[t-1] + (h/6)*(q1_z + 2*q2_z + 2*q3_z + q4_z)#runge_kutta(x
_r[t-1], y_r[t-1], dz_dt, h)
    w_r[t] = w_r[t-1] + (h/6)*(q1_w + 2*q2_w + 2*q3_w + q4_w)#runge_kutta(x
_r[t-1], y_r[t-1], dw_dt, h)
    x_r[t] = x_r[t-1] + h*z_r[t-1]
    y_r[t] = y_r[t-1] + h*w_r[t-1]
end
```

In [98]:
```
plot(x_r,y_r,size=(500,500),label="Runge-Kutta",title="w(0)=0.2,tmax=30")
```

Out[98]:



In [99]:
```
using ODE
```

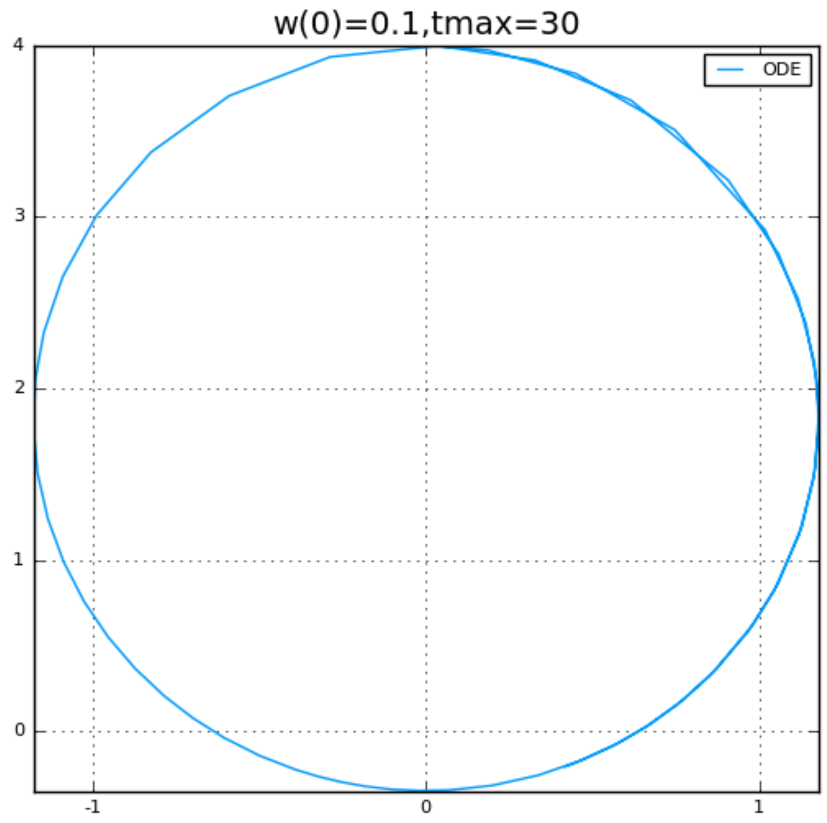
In [103]:
```
function gm(t, f)
    (x, y, z, w) = f
    dy_dt = z
    dx_dt = w
    dz_dt = -x/((sqrt(x^2+y^2)^3))
    dw_dt = -y/((sqrt(x^2+y^2)^3))

    [dy_dt; dx_dt; dz_dt; dw_dt]
end;
```

In [104]:
```
start = [4.0; 0.0; 0.0; 0.2]

ts = [0.0; 30.0];
```

In [105]:
```
t, res = ode45(gm, start, ts)

y = map(h -> h[1], res)
x = map(h -> h[2], res);
```

```
In [106]: plot(x,y,size=(500,500),label="ODE",title="w(0)=0.1,tmax=30")
```

Out[106]:



```
In [ ]:
```

Problem 6: Look and explain! (10 points)

Take a look at the raptorchase code from the start of class. Go through it an explain what each segment does! You may use as much or as little granularity as you want.

Problem 6

At the start sets constants.
Set max number of steps equal to 1000. For this that would mean each step takes place as 1/100th of a second
Compute derivatives function

1. The change of a human is given by the speed of the human and the x-direction and y-direction from angles
2. From raptor 0 starting location, determines where the raptor's direction will be in order to face where the human is at that time step, then send the raptor in that direction by a unit of its velocity

Simulate Raptors This section will take initial positions, then will run forward Euler method on the raptop positions based on the behavior of the raptors and humans. Since the human will always move in a single direction, and the derivative is know it will be a straight line. The raptor's behavior depends on the position of the human, so this constantly shifting derivative and direction will look like a curve. The paths described are created by using an $h = 0.01$ and the derivative function, and the position at time=t