**HOMEWORK 1**
Elena Gómez
January 18, 2023

In [1]:

```
#ENV["PYTHON"]=""
#Pkg.build("PyCall")
```

**PROBLEM 0:** List your collaborators
I worked alone

**PROBLEM 1:** Some quick, simple theory

**1.What is $1/\sqrt{(x)}$ when $x = 10^9$? What is the limit of the sequence $1/\sqrt{(x)}$ as $x \to \infty$? What type of convergence is this?**

ANSWER
When $x = 10^9$, $1/\sqrt{(x)} = 3.16227766 \cdot 10^-5$

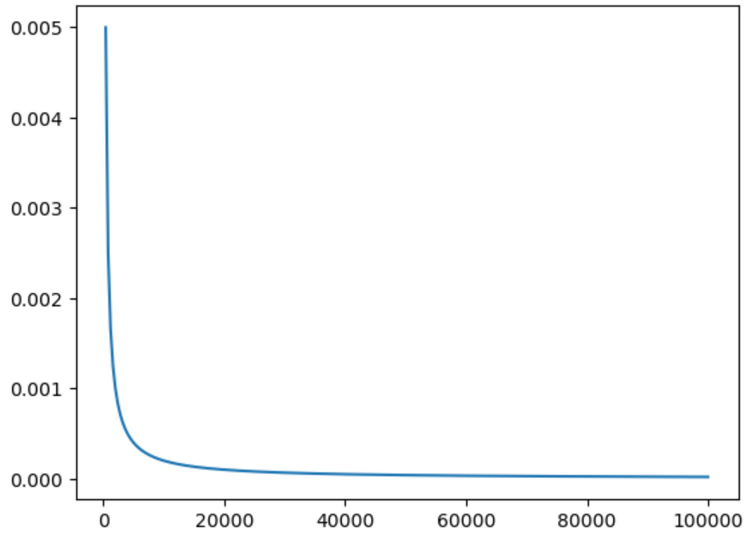
The limit of $1/\sqrt{(x)}$ when $x \to \infty$ is 0.

Because when $(x) \to \infty$ , $1\sqrt{(x)} goesto0$

The limit of the sequence $1/\sqrt{(x)}$ can be easily seen in the following plot

In [2]:

```
using PyPlot
f(x) = 1/(x^1/2)
xs = range(-0, 100000, length=251)
ys = f.(xs)
ys[xs .== 0.0] .= NaN
plot(xs, ys)
```




Out[2]:

```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x000001AE72E7CF10>
```

As we can see in the plot as $x \to \infty$ , $1/\sqrt{(x)}$ goes to 0.

The convergence of this sequence is arithmetic because $||x - x_t|| \leq \frac{1}{K^\alpha}$ where $\alpha = 1/2$

**2. Show, using the definition, that the sequence $1 + k^{-k}$ converges superlinearly to 1.**

ANSWER

A sequence converges superlinearly if $lim_{n \to \infty} \frac{||x_{n+1} - x'||}{||x_n - x'||} = 0$

and it converges superlinearly to 1 if $lim_{n \to \infty} \frac{||x_{n+1} - 1||}{||x_n - 1||} = 0$

$x_n = 1 + k^{-k}$ and $x_{n+1} = 1 + (k+1)^{-k+1}$

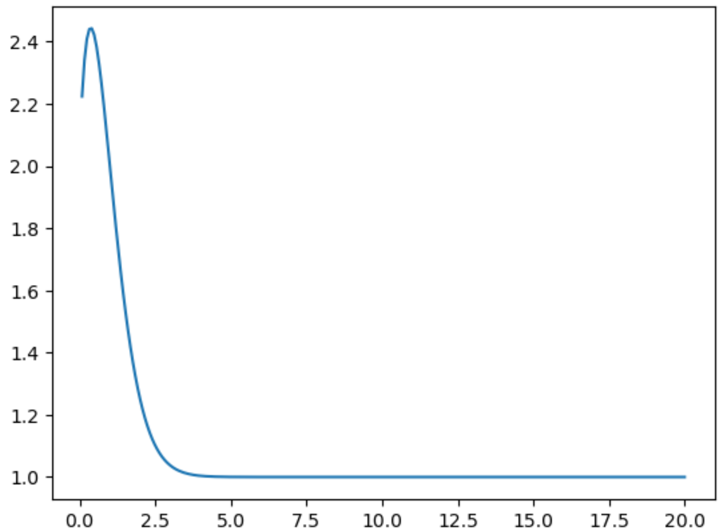
So $lim_{n\to\infty}\frac{||1+(k+1)^{-(k+1)}-1||}{||1+k^{-k}-1||} = lim_{n\to\infty}\frac{||(k+1)^{-(k+1)}||}{||k^{-k}||} = lim_{n\to\infty}\frac{||k^k||}{||(k+1)^{(k+1)}||}$

As $||(k+1)^{(k+1)}|| \gg ||k^k||$, $lim_{n\to\infty}\frac{||1+(k+1)^{-(k+1)}-1||}{||1+k^{-k}-1||} = 0$

The convergence can be seen in the following plot

In [3]:

```
f(x) = 1+(x^(-x))
xs = range(-0, 20, length=251)
ys = f.(xs)
ys[xs .== 0.0] .= NaN
plot(xs, ys)
```




Out[3]:

```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x000001AE7472E680>
```

**3.Suppose we have a sequence $x_{k+1} = \sqrt{x_k}$.Show that this sequence convergesfor all positive inputs $x_0$. What is the the rate of convergence.**

ANSWER FOR THE POSITIVE SOLUTIONS OF THE ROOT

Let's show that the sequene is monotonically decreasing.
Let $x_0 = \sqrt{2}$, and $x_1 = \sqrt[4]{2}$
$x_1 - x_0 = \sqrt[4]{2} - \sqrt{2} \approx -0.22 < 0$
So, we assume that the sequence is monotonically decreasing
$x_{k+1} - x_k = \sqrt{x_k} - \sqrt{x_{k-1}} = \sqrt[4]{x_{k-1}} - \sqrt{x_{k-1}} < 0$
By the induction hypotheses $x_{k+1} - x_k < 0$ .

The sequence is bounded below by 1.
Let's suppose $x_k > 1$ .
$xk + 1 = \sqrt{x_k} = \sqrt{1} = 1$
$If x_k > 1$, then also $x_{k+1} > 1$

Let's compute the rate of convergence
$lim_{k\to\infty}\frac{||x_{k+1}-x'||}{||x_k-x'||}$, where $x' = 1$
$lim_{k\to\infty}\frac{||x_{k+1}-1||}{||x_k-1||} = lim_{k\to\infty}\frac{||\sqrt{x_k}-1||}{||\sqrt{x_{k-1}}-1||} = 1$

**4. Let z = 9. Consider the sequence**
$x_{k+1} = (1/2)x_k(3 - zx_k^2), x_0 = 1/2.$
**Show that this converges and give the convergence type (algebra, linear, quadratic, etc.)**

$x_1 - x_0 = [\frac{1}{2}\cdot\frac{1}{2}\cdot(3 - 9\cdot(\frac{1}{2})^2))] - \frac{1}{2} = [\frac{1}{2}\cdot\frac{1}{2}\cdot(3 - \frac{9}{4})] - \frac{1}{2} = [\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{3}{4}] - \frac{1}{2} = \frac{3}{16} - \frac{1}{2} = -\frac{5}{16} < 0$
So, we assume that the sequence is monotonically decreasing.

The sequence is bounded below by 0.
Let's suppose $x_k > 0$ .
$xk + 1 = (\frac{1}{2}) \cdot 0 \cdot (3 - z \cdot 0) = 0$
If $x_k > 0$, then also $x_{k+1} > 0$

The type of convergence is superlinear

Because
$lim_{k \to \infty} \frac{||x_{k+1} - x'||}{||x_k - x'||} = 0$, where $x' = 0$
$lim_{k \to \infty} \frac{||x_{k+1} - x'||}{||x_k - x'||} = lim_{k \to \infty} \frac{||x_{k+1}||}{||x_k||} = 0$, Because $||x_k|| > ||x_{k+1}||$

**5. (A little more interesting, note that I haven't yet worked out the answer to this one; it may be well-known in other areas, etc. Don't feel obliged to compute a full solution, but for full points you must show your investigation. ) Suppose we have a sequence $x_{k+1} = |log x_k|$. Does this converge for any input? For all inputs? For some? What else can you say about it? Are there limit points?**

The sequence is monotonically decreasing until $x_k < 1$

$x_{k+1} - x_k = |log x_k| - x_k < 1$
Because $|log x_k| < x_k$

Once $x_k < 1$,let's see what happens numerically.

In [5]:

```
x_0=0.99
x_1=log(0.99)
print("x_1=",x_1)
print(" ")
x_2=log(abs(x_1))
print("x_2=",x_2)
print(" ")
x_3=log(abs(x_2))
print("x_3=",(x_3))
```

```
x_1=-0.01005033585350145 x_2=-4.600149226776579 x_3=1.5260887435724684
```

Seeing this results we can say that the sequence is not strictly monotonically increasing or decreasing

If $_k = 0$ is achieved, the sequence is convergent because the sequence cannnot continue. Because $log 0$ does not exist. This could happend, for example if x_0=1. And the limit would be 0

**PROBLEM 2:** Angled raptors

**1. Modify the the Raptor chase example function to compute the survival time of a human in a raptor problem where you switch direction at 0.25 seconds and 0.75 seconds. Show your modified function, and show the survival time when running directly at the slow raptor (up to time 0.25) and then reversing your direction and running away from it.**

We add 3 conditions to determine which angle the human has to take depending on the time.
Later we compute the algorithm with the angles [0,pi,pi] because the human has to run directly at the slow raptor and then reverse the sirection and run away from it.

In [6]:

```
using Printf, LinearAlgebra
```

In [7]:

```

using Printf, LinearAlgebra
vhuman=6.0
vraptor0=10.0 # the slow raptor velocity in m/s
vraptor=15.0 #

raptor_distance = 20.0

raptor_min_distance = 0.2 # a raptor within 20 cm can attack
tmax=10.0 # the maximum time in seconds
nsteps=1000

"""
This function will compute the derivatives of the
positions of the human and the raptors
"""
function compute_derivatives(angle,h,r0,r1,r2)
    dh = [cos(angle),sin(angle)]*vhuman
    dr0 = (h-r0)/norm(h-r0)*vraptor0
    dr1 = (h-r1)/norm(h-r1)*vraptor
    dr2 = (h-r2)/norm(h-r2)*vraptor
    return dh, dr0, dr1, dr2
end

"""
This function will use forward Euler to simulate the Raptors
"""
function simulate_raptors(angle1,angle2,angle3; output::Bool = true)
    # initial positions
    h = [0.0,0.0]
    r0 = [1.0,0.0]*raptor_distance
    r1 = [-0.5,sqrt(3.)/2.]*raptor_distance
    r2 = [-0.5,-sqrt(3.)/2.]*raptor_distance

    # how much time el
    dt = tmax/nsteps
    t = 0.0

    hhist = zeros(2,nsteps+1)
    r0hist = zeros(2,nsteps+1)
    r1hist = zeros(2,nsteps+2)
    r2hist = zeros(2,nsteps+2)

    hhist[:,1] = h
    r0hist[:,1] = r0
    r1hist[:,1] = r1
    r2hist[:,1] = r2

    for i=1:nsteps
        if 0<=t<0.25
            angle=angle1
        end
        if 0.25<=t <0.75
            angle=angle2
        end
        if 0.75<=t
            angle=angle3
        end

        dh, dr0, dr1, dr2 = compute_derivatives(angle,h,r0,r1,r2)
        h += dh*dt
        r0 += dr0*dt
        r1 += dr1*dt
        r2 += dr2*dt
        t += dt

        hhist[:,i+1] = h
        r0hist[:,i+1] = r0
        r1hist[:,i+1] = r1
        r2hist[:,i+1] = r2

        if norm(r0-h) <= raptor_min_distance ||
            norm(r1-h) <= raptor_min_distance ||
            norm(r2-h) <= raptor_min_distance
            if output
                @printf("The raptors caught the human in %f seconds\n", t)
            end

            # truncate the history
            hhist = hhist[:,1:i+1]
            r0hist = r0hist[:,1:i+1]
            r1hist = r1hist[:,1:i+1]
            r2hist = r2hist[:,1:i+1]

            break
        end
    end
    return hhist, r0hist, r1hist, r2hist
end

```

```
simulate_raptors(0,pi,pi)

The raptors caught the human in 1.290000 seconds
```

Out[7]:

```
([0.0 0.06 … -4.679999999999998 -4.7399999999999975; 0.0 0.0 … 0.0 0.0], [20.0 19.9 … 7.199999999999976 7.0999999999999766; 0.0 0.0 … 0.0 0.0], [-10.0 -9.925 … -4.477758259386442 -4.614899150934627; 17.32050807568877 17.190604265121106 … 0.0896100 6198804163 0.028845139003075712], [-10.0 -9.925 … -4.477758259386442 -4.614899150934627; -17.32050807568877 -17.19060426512 1106 … -0.08961006198804163 -0.028845139003075712])
```

**2. Utilize a grid-search strategy to determine the best angles for the human to run to maximize the survival time. Show the angles.**

I will use the angles 0, pi/4, pi/2 and pi/3. As it is my first time programming in Julia I did a list of the combination of this angles in groups of 3. I know that this is a very manual way, but I tried to do in a better way and I was not able, because this lenaguage is new for me.

In [8]:

```
#list of angles

l5=[[0,0,0],[0,0,(pi/4)],[0,0,(pi/2)],[0,0,(pi/3)],
    [0,(pi/4),0],[0,(pi/4),(pi/4)],[0,pi/4,pi/2],[0,pi/4,pi/3],
    [0,pi/2,0],[0,pi/2,pi/2],[0,pi/2,pi/4],[0,pi/2,pi/3],
    [0,pi/3,0],[0,pi/3,pi/2],[0,pi/3,pi/4],[0,pi/3,pi/3],


    [pi/4,0,0],[pi/4,0,pi/4],[pi/4,0,pi/2],[pi/4,0,pi/3],
    [pi/4,pi/4,0],[pi/4,pi/4,pi/4],[pi/4,pi/4,pi/2],[pi/4,pi/4,pi/3],
    [pi/4,pi/2,0],[pi/4,pi/2,pi/2],[pi/4,pi/2,pi/4],[pi/4,pi/2,pi/3],
    [pi/4,pi/3,0],[pi/4,pi/3,pi/2],[pi/4,pi/3,pi/4],[pi/4,pi/3,pi/3],

    [pi/2,0,0],[pi/2,0,pi/4],[pi/2,0,pi/2],[pi/2,0,pi/3],
    [pi/2,pi/4,0],[pi/2,pi/4,pi/4],[pi/2,pi/4,pi/2],[pi/2,pi/4,pi/3],
    [pi/2,pi/2,0],[pi/2,pi/2,pi/2],[pi/2,pi/2,pi/4],[pi/2,pi/2,pi/3],
    [pi/2,pi/3,0],[pi/2,pi/3,pi/2],[pi/2,pi/3,pi/4],[pi/2,pi/3,pi/3],

    [pi/3,0,0],[pi/3,0,pi/4],[pi/3,0,pi/2],[pi/3,0,pi/3],
    [pi/3,pi/4,0],[pi/3,pi/4,pi/4],[pi/3,pi/4,pi/2],[pi/3,pi/4,pi/3],
    [pi/3,pi/2,0],[pi/3,pi/2,pi/2],[pi/3,pi/2,pi/4],[pi/3,pi/2,pi/3],
    [pi/3,pi/3,0],[pi/3,pi/3,pi/2],[pi/3,pi/3,pi/4],[pi/3,pi/3,pi/3]



    ]
```

Out[8]:

```
64-element Vector{Vector{Float64}}:
 [0.0, 0.0, 0.0]
 [0.0, 0.0, 0.7853981633974483]
 [0.0, 0.0, 1.5707963267948966]
 [0.0, 0.0, 1.0471975511965976]
 [0.0, 0.7853981633974483, 0.0]
 [0.0, 0.7853981633974483, 0.7853981633974483]
 [0.0, 0.7853981633974483, 1.5707963267948966]
 [0.0, 0.7853981633974483, 1.0471975511965976]
 [0.0, 1.5707963267948966, 0.0]
 [0.0, 1.5707963267948966, 1.5707963267948966]
 [0.0, 1.5707963267948966, 0.7853981633974483]
 [0.0, 1.5707963267948966, 1.0471975511965976]
 [0.0, 1.0471975511965976, 0.0]
 ⋮
 [1.0471975511965976, 0.7853981633974483, 0.0]
 [1.0471975511965976, 0.7853981633974483, 0.7853981633974483]
 [1.0471975511965976, 0.7853981633974483, 1.5707963267948966]
 [1.0471975511965976, 0.7853981633974483, 1.0471975511965976]
 [1.0471975511965976, 1.5707963267948966, 0.0]
 [1.0471975511965976, 1.5707963267948966, 1.5707963267948966]
 [1.0471975511965976, 1.5707963267948966, 0.7853981633974483]
 [1.0471975511965976, 1.5707963267948966, 1.0471975511965976]
 [1.0471975511965976, 1.0471975511965976, 0.0]
 [1.0471975511965976, 1.0471975511965976, 1.5707963267948966]
 [1.0471975511965976, 1.0471975511965976, 0.7853981633974483]
 [1.0471975511965976, 1.0471975511965976, 1.0471975511965976]
```

In [9]:

```
using Printf, LinearAlgebra
```

Then in implement a for lop in order to compute the function using all the possible combination in order to obtain the time depending on the combination of angles we use

In [10]:

```

vhuman=6.0
vraptor0=10.0 # the slow raptor velocity in m/s
vraptor=15.0 #

raptor_distance = 20.0

raptor_min_distance = 0.2 # a raptor within 20 cm can attack
tmax=10.0 # the maximum time in seconds
nsteps=1000

"""
This function will compute the derivatives of the
positions of the human and the raptors
"""
function compute_derivatives(angle,h,r0,r1,r2)
    dh = [cos(angle),sin(angle)]*vhuman
    dr0 = (h-r0)/norm(h-r0)*vraptor0
    dr1 = (h-r1)/norm(h-r1)*vraptor
    dr2 = (h-r2)/norm(h-r2)*vraptor
    return dh, dr0, dr1, dr2
end

"""
This function will use forward Euler to simulate the Raptors
"""
function simulate_raptors2( output::Bool = true)
    # initial positions
    h = [0.0,0.0]
    r0 = [1.0,0.0]*raptor_distance
    r1 = [-0.5,sqrt(3.)/2.]*raptor_distance
    r2 = [-0.5,-sqrt(3.)/2.]*raptor_distance

    # how much time el
    dt = tmax/nsteps
    t = 0.0

    hhist = zeros(2,nsteps+1)
    r0hist = zeros(2,nsteps+1)
    r1hist = zeros(2,nsteps+2)
    r2hist = zeros(2,nsteps+2)

    hhist[:,1] = h
    r0hist[:,1] = r0
    r1hist[:,1] = r1
    r2hist[:,1] = r2



    l5=[[0,0,0],[0,0,(pi/4)],[0,0,(pi/2)],[0,0,(pi/3)],
    [0,(pi/4),0],[0,(pi/4),(pi/4)],[0,pi/4,pi/2],[0,pi/4,pi/3],
    [0,pi/2,0],[0,pi/2,pi/2],[0,pi/2,pi/4],[0,pi/2,pi/3],
    [0,pi/3,0],[0,pi/3,pi/2],[0,pi/3,pi/4],[0,pi/3,pi/3],


    [pi/4,0,0],[pi/4,0,pi/4],[pi/4,0,pi/2],[pi/4,0,pi/3],
    [pi/4,pi/4,0],[pi/4,pi/4,pi/4],[pi/4,pi/4,pi/2],[pi/4,pi/4,pi/3],
    [pi/4,pi/2,0],[pi/4,pi/2,pi/2],[pi/4,pi/2,pi/4],[pi/4,pi/2,pi/3],
    [pi/4,pi/3,0],[pi/4,pi/3,pi/2],[pi/4,pi/3,pi/4],[pi/4,pi/3,pi/3],

    [pi/2,0,0],[pi/2,0,pi/4],[pi/2,0,pi/2],[pi/2,0,pi/3],
    [pi/2,pi/4,0],[pi/2,pi/4,pi/4],[pi/2,pi/4,pi/2],[pi/2,pi/4,pi/3],
    [pi/2,pi/2,0],[pi/2,pi/2,pi/2],[pi/2,pi/2,pi/4],[pi/2,pi/2,pi/3],
    [pi/2,pi/3,0],[pi/2,pi/3,pi/2],[pi/2,pi/3,pi/4],[pi/2,pi/3,pi/3],

    [pi/3,0,0],[pi/3,0,pi/4],[pi/3,0,pi/2],[pi/3,0,pi/3],
    [pi/3,pi/4,0],[pi/3,pi/4,pi/4],[pi/3,pi/4,pi/2],[pi/3,pi/4,pi/3],
    [pi/3,pi/2,0],[pi/3,pi/2,pi/2],[pi/3,pi/2,pi/4],[pi/3,pi/2,pi/3],
    [pi/3,pi/3,0],[pi/3,pi/3,pi/2],[pi/3,pi/3,pi/4],[pi/3,pi/3,pi/3]



    ]

    results=[]

    for j in l5
        h = [0.0,0.0]
        r0 = [1.0,0.0]*raptor_distance
        r1 = [-0.5,sqrt(3.)/2.]*raptor_distance
        r2 = [-0.5,-sqrt(3.)/2.]*raptor_distance

        # how much time el
        dt = tmax/nsteps
        t = 0.0

        hhist = zeros(2,nsteps+1)
        r0hist = zeros(2,nsteps+1)
        r1hist = zeros(2,nsteps+2)
        r2hist = zeros(2,nsteps+2)
```

```

        hhist[:,1] = h
        r0hist[:,1] = r0
        r1hist[:,1] = r1
        r2hist[:,1] = r2
        for i=1:nsteps
            if 0<=t<0.25
                angle=j[1]
            end
            if 0.25<=t <0.75
                angle=j[2]
            end
            if 0.75<=t
                angle=j[3]
            end

            dh, dr0, dr1, dr2 = compute_derivatives(angle,h,r0,r1,r2)
            h += dh*dt
            r0 += dr0*dt
            r1 += dr1*dt
            r2 += dr2*dt
            t += dt

            hhist[:,i+1] = h
            r0hist[:,i+1] = r0
            r1hist[:,i+1] = r1
            r2hist[:,i+1] = r2

            if norm(r0-h) <= raptor_min_distance ||
                norm(r1-h) <= raptor_min_distance ||
                norm(r2-h) <= raptor_min_distance
                if output
                  push!(results,t)
                  @printf("The raptors caught the human in %f seconds\n", t)
                end

                # truncate the history
                hhist = hhist[:,1:i+1]
                r0hist = r0hist[:,1:i+1]
                r1hist = r1hist[:,1:i+1]
                r2hist = r2hist[:,1:i+1]

                break


            end
        end
    end



    #return hhist, r0hist, r1hist, r2hist,result
    return results
end

```

Out[10]:

```
simulate_raptors2
```

In [11]:

```

results=simulate_raptors2()
```

```
1.480000000000001
1.280000000000001
1.410000000000001
1.460000000000001
1.1400000000000008
1.300000000000001
1.2400000000000009
1.370000000000001
⋮
1.390000000000001
1.380000000000001
1.2000000000000008
1.310000000000001
1.320000000000001
1.0700000000000007
1.1900000000000008
1.1400000000000008
1.440000000000001
1.1500000000000008
1.310000000000001
```

From the function we obtain a list of all the results, the seconds that the human survives depending on the angles we use.

So we find the maximum time in the list of results

In [12]:

```
#we find the maximum survival time
maximum=0
for i in results
    if i>maximum

        maximum=i
    end

end
print(maximum)
```

```
1.5400000000000011
```

Then we find the position of the maximum time in seconds in the list

In [13]:

```
#we find which position it has in the list of results in order to find later which angles are
function myCondition(y)
    return maximum == y
end

println( findfirst(myCondition, results) )

```

```
4
```

Finally, we find the angles in the list of angles . The position of the angles in the angles list it's the same position that the maximum time in seconds in the results list

In [14]:

```
#we find the angles
l5[4]
```

Out[14]:

```
3-element Vector{Float64}:
 0.0
 0.0
 1.0471975511965976
```

So, the angles that maximizes the time in seconds are:
angle1= 0
angle2=0
angle3=pi/3

**3. Discuss the major challenge for solving this problem with the current strategy if we added a fourth angle at 0.5 seconds. (Or do so!) And a fifth angle at 0.75 seconds. (Or if you are feeling ambitious, solve these and see where you start running into trouble and discuss why that is!**

The challenge would be that the complexity of the methos would be high, so if we add a fourth or even a fifth angle, the function will need to much time to be computed

In [ ]:

```

```