FiveThirtyEight's July 24, 2020 Riddler

Emma Knight

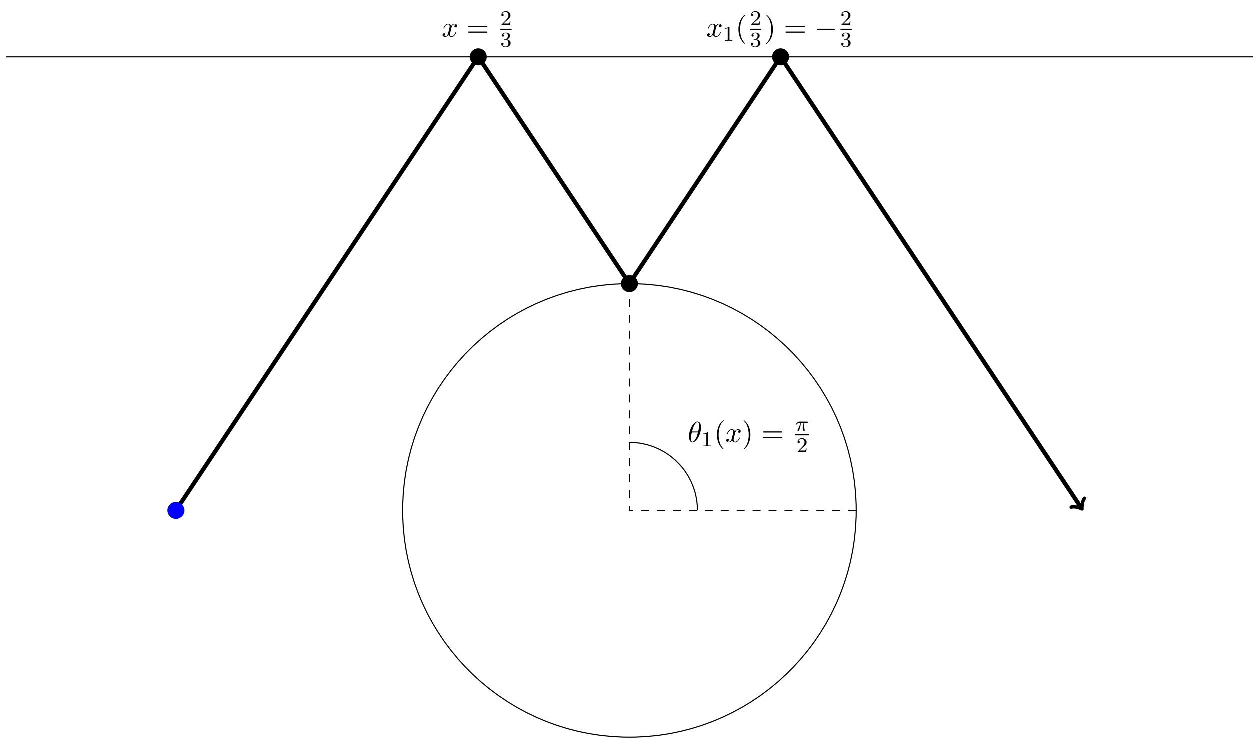
July 25, 2020

This week's riddler is a neat little geometry problem:

Question 1. *Riddler Pinball is a game with an infinitely long wall and a circle whose radius is 1 inch and whose center is* 2 *inches from the wall. The wall and the circle are both fixed and never move. A single pinball starts* 2 *inches from the wall and* 2 *inches from the center of the circle.*

To play, you flick the pinball toward a spot of your choosing along the wall, specified by its distance x from the point on the wall that's closest to the circle. What's the greatest number of bounces you can achieve? And, more importantly, what value of x gets you the most bounces?

Let's start by thinking about what happens when $x = \frac{2}{3}$. One can draw the picture below:

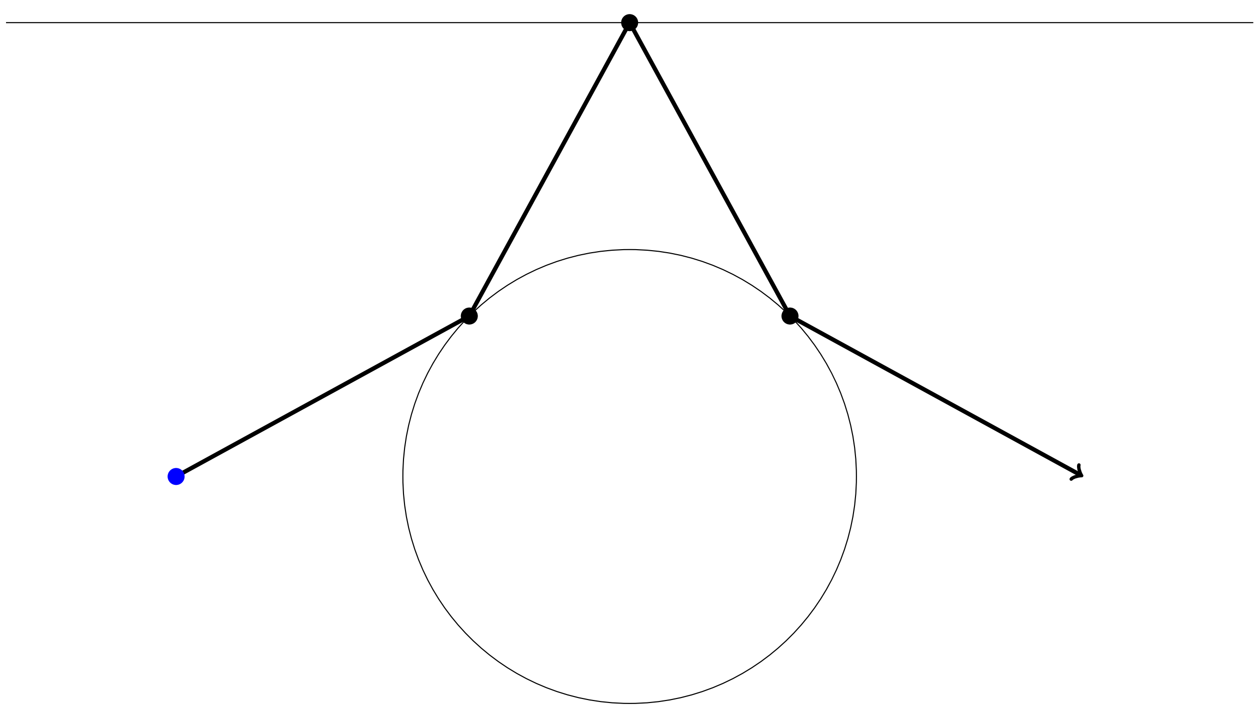


For x near $\frac{2}{3}$, the ball must meet the bumper at a point determined by an angle which I will denote $\theta_1(x)$. Additionally, for x near $\frac{2}{3}$, there is a second point of intersection along the wall which I will denote $x_1(x)$. θ_1 and x_1 are both clearly an increasing continuous functions of x, and on the interval of xs where x_1 is defined, its image is all of $\mathbb{R}$. Thus, there must exist some $x_{1,1}$ such that $x_1(x_{1,1}) = 0$. Near $x_{1,1}$, there is a second point of intersection on the bumper, which I will denote by $\theta_2(x)$. Again, θ_2 is clearly an increasing function of x, and it's not too hard to see that there is a value $x_{0,2}$ such that $\theta_2(x_{0,2}) = \frac{\pi}{2}$.

This process repeats, giving functions x_k and θ_k, as well as x-values $x_{0,k}$ and $x_{1,k}$ with the properties that $\theta_k(x_{0,k}) = \frac{\pi}{2}$ and $x_k(x_{1,k}) = 0$. If one chooses to aim at $x_{0,k}$, one gets $4k-1$ bounces, while $x_{1,k}$ produces $4k+1$ bounces. But notice that $x_{1,1} < x_{0,2} < x_{1,2} < \cdots$ and all of the $x_{i,k}$s are less than 2 basically by definition. Thus, there exists a number $\tilde{x} = \lim x_{0,k} = \lim x_{1,k}$. What happens if you aim at $\tilde{x}$?

Define $b(x)$ to be the number of bounces that occur after shooting off in towards x. Adopt the convention that a tangency to the bumper counts as a bounce, as well as the convention that if the final ray is parellel to the x-axis, that counts as a bounce as well. Then $b(x)$ is upper semincontinuous, and $b(x_{1,k}) = 4k+1$, so $b(\tilde{x}) = +\infty$. Since you clearly can't beat $+\infty$, this must be optimal.

Notice that one could get a similar approach by starting at $x = 4 - 4\sqrt{2}$ (a negative number; this will actually hit the bumper first) and decreasing x to $-\infty$. The starting position is shown below:



Now, all that's left is to compute $\tilde{x}$. Many thanks to Brett Berger for insightful conversations about this half of the write-up. To that end, it is correct to think about what happens when the line the ball is moving along is near the line $x = 0$. Label the points the ball hits on the wall $x_0 = \tilde{x}, x_1, \ldots$, and write the equation for the line coming out the wall $x - x_n = m_n(y-2)$. If x_n

and m_n are small, then the bumper is just a straight line, so one collides with the bumper at the point $(x_n - m_n, 1)$. The angle of the line from the center of the bumper to the intersection point is roughly $x_n - m_n$ while the angle of the original line is roughly m_n. Thus, the angle of the line coming out of the bumper is $2x_n - 3m_n$, and so one gets $x_n + 1 = 3x_n - 4m_n$. Moreover, the new slope is just $m_{n+1} = -(2x_n - 3m_n) = -2x_n + 3m_n$. Thus, one has that $\left(\begin{smallmatrix} x_{n+1} \\ m_{n+1} \end{smallmatrix}\right) = \left(\begin{smallmatrix} 3 & -2 \\ -4 & 3 \end{smallmatrix}\right)\left(\begin{smallmatrix} x_n \\ m_n \end{smallmatrix}\right)$. A simple computation shows that the eigenvalues of this matrix are $3 \pm 2\sqrt{2}$, and the $3 - 2\sqrt{2}$ eigenvector is $\begin{pmatrix} \sqrt{2} \\ 1 \end{pmatrix}$. Thus, if one manages to get x_n and m_n to be small and in a ratio close to $\sqrt{2} : 1$ then the pinball should bounce for a long long time.

One can easily see that $m_0 = 1 - \frac{x}{2}$. Define $f_k(x) = x_k - m_k\sqrt{2}$. Near $\tilde{x}$, this is a well-defined function of x and $\tilde{x}$ is close to a zero of f_k. It is not too bad to write code to compute values of f_k and approximate zeroes. I found the following zeroes of f_k:

k	the zero
0	.8284271247461902
1	.8224910999491976
2	.8224863290728398
3	.8224863249469717
4	.8246473262941052

I strongly suspect that floating point errors are balooning and by the time that you get $n = 4$, the answer is dominated by such errors. Thus, I am happy saying that you should flick the ball at an x-value of approximately .8224632, but commiting to more decimal places seems foolish.

Finally, here is the code:

```
import math

## This computes the intersection point on the
##circle given that your initial point is at x
##and your slope is m.
def intersection(x, m):
    a = 1+m*m
    b = 2*m*(x-2*m)
    c = (x-2*m)**2-1
    ynew =(-b+math.sqrt(b*b-4*a*c))/(2*a)
    xnew = m*(ynew-2) + x
    return [xnew, ynew]

##This computes the next pair of x and m from
##your current pair.
def nextpoint(x, m):
    v = intersection(x, m)
    n = v[0]/v[1]
    mnew = ((2*n-m+m*n*n)/(1+2*m*n-n*n))
    dy = 2-v[1]
```

```
    xnew = v[0]+dy*mnew
    return [xnew, -1*mnew]

##This just iterates the previous function k times.
def iterate(x, m, k):
    if (k == 0):
        return[x, m]
    v = nextpoint(x, m)
    return iterate(v[0], v[1], k-1)

##This is the function that we are looking for a zero
##of.
def f(x, k):
    m = 1-(x/2)
    v = iterate(x, m, k)
    return v[0]-math.sqrt(2)*v[1]

##These two functions approximate a zero of f by doing
##midpoint division.
def nextpair(a, b, n):
    y1 = f(a, n)
    y2 = f((a+b)/2, n)
    y3 = f(b, n)
    if (y1/y2 > 0):
        return([(a+b)/2, b])
    if (y2/y3 > 0):
        return([a, (a+b)/2])

def approximate(a, b, n, k):
    if (k == 0):
        return (a+b)/2
    v = nextpair(a, b, n)
    return approximate(v[0], v[1], n, k-1)

print(approximate(.82, .83, 0, 40))
print(approximate(.82, .83, 1, 40))
print(approximate(.82, .83, 2, 40))
print(approximate(.82, .83, 3, 40))
print(approximate(.82, .825, 4, 40))
```