Homework 8, CSE 232

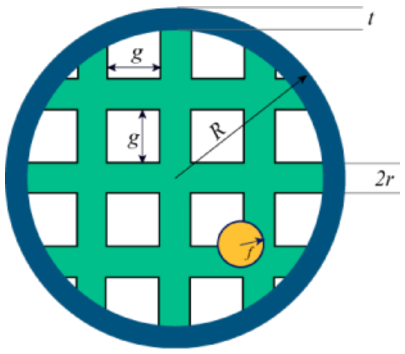
Due October 29 Note: On most of the problem sets through the semester, I'll put a horizontal line with "Optional" under it. Any problems below this section are encouraged - I think they're interesting and will help you learn the subject - but not necessary to complete in order to get credit for the homework.

Problem 1

This week's problem is Google Code Jam Qualification Round 2008 - Fly Swatter. Here's the problem statement:

> What are your chances of hitting a fly with a tennis racquet?
>
> To start with, ignore the racquet's handle. Assume the racquet is a perfect ring, of outer radius R and thickness t (so the inner radius of the ring is R−t).
>
> The ring is covered with horizontal and vertical strings. Each string is a cylinder of radius r. Each string is a chord of the ring (a straight line connecting two points of the circle). There is a gap of length g between neighbouring strings. The strings are symmetric with respect to the center of the racquet i.e. there is a pair of strings whose centers meet at the center of the ring.
>
> The fly is a sphere of radius f. Assume that the racquet is moving in a straight line perpendicular to the plane of the ring. Assume also that the fly's center is inside the outer radius of the racquet and is equally likely to be anywhere within that radius. Any overlap between the fly and the racquet (the ring or a string) counts as a hit.

```
In [1]: from IPython.display import Image, display
        display(Image(filename='./week8_hwimg.png'))
```



Input

> One line containing an integer N, the number of test cases in the input file.
>
> The next N lines will each contain the numbers f, R, t, r and g separated by exactly one space. Also the numbers will have at most 6 digits after the decimal point.

Output

> N lines, each of the form "Case #k: P", where k is the number of the test case and P is the probability of hitting the fly with a piece of the racquet.
>
> Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

f, R, t, r and g will be positive and smaller or equal to 10000.

t < R

f < R

r < R

Small dataset

1 ≤ N ≤ 30

The total number of strings will be at most 60 (so at most 30 in each direction).

Large dataset

1 ≤ N ≤ 100

The total number of strings will be at most 2000 (so at most 1000 in each direction).

a) The problem as stated requires us to work with a large fly of radius f. Suppose we had a problem where $f = 0$. How could we answer the probability of the zero-radius fly being hit by the racquet in terms of areas? We can make simultaneous adjustments to multiple parameters without changing the final probability. How can we do this so that $f' = 0$ after the change? What are the adjusted values of all the other parameters?

If the fly is a point (with radius 0), then the probability it is hit is the ratio of area taken up by the racquet ring and strings to the total (circular) area of the racquet. Equivalently, it is 1 minus the probability it is not hit, which is the ratio of area unoccluded by the racquet within the outer radius to the total racquet area. We can adjust the other parameters as follows: $f' = 0$, $t' = t + f$, $r' = r + f$, and $g' = g - 2f$. That is, we reduce the size of the gaps in the racquets and increase the size of the strings and outer ring while reducing the fly radius.

b) We can solve this problem by adding together the areas of the gaps between the strings. There is a symmetry in the problem that will simplify our computation. Describe how to use this symmetry to restrict the number of sections to examine.

The racquet is symmetric with respect to the x and y axes, if we center the origin at the center of the racquet. We want to add up the areas of all the empty squares in the racquet and divide that area by the total area of the racquet. This ratio is identical if we only consider the upper right quadrant of the racquet - both the numerator and denominator of our ratio will be divided by four.

c) For the sections of area to be added together, there are two trivial cases to consider and four non-trivial cases. In the trivial cases, either the full square of the gap is inside the inner rim (for area g'^2) or it is fully outside the inner rim (for area 0). Enumerate the four non-trivial cases and provide a formula to calculate their areas. You may find it useful to draw a diagram, and a formula to calculate the areas of circular segments will be of interest.

In the four non-trivial cases, the circle of the inner ring will intersect the square gap between strings, and we wish to find only the area of the section to the lower left of the circle. The four cases will be the sides of the square that the inner circle intersects, which will always be two different ones. The cases are:

1) Intersection on the upper and right sides

2) Intersection on the left and right sides

3) Intersection on the upper and lower sides

4) Intersection on the left and right sides.

In each case, there will be a circular segment, the area of which is $S = \frac{1}{2}R^2(\theta - sin(\theta))$, where R is the *inner* radius of the racquet and θ is the angle between the intersection points. Here are the area formulas for each case, where $x1$ and $x2$ are the left and right positions of the square and $y1$ and $y2$ the lower and upper positions, and $g = x2 - x1 = y2 - y1$ the length of the side of the square.

1) $A = g^2 - (dx * dy)/2 + S$, where dx and dy are the distances from the intersection points to the upper right corner.

2) $A = g * (dy1 + dy2)/2 + S$, where $dy1$ and $dy2$ are the distances from the bottom of the square to the intersection points on each side.

3) $A = g * (dx1 + dx2)/2 + S$, where $dx1$ and $dx2$ are the distances from the left of the square to the intersection points on each side.

4) $A = (dx * dy)/2 + S$, where dx and dy are the distances from the lower right corner to the intersection points.

d) Using the formulas you came up with in part c, write an $O(N^2)$ solution to the problem by iterating through all squares within your region of interest. Test it on the Google Code Jam website and attach your code.

```
In []: from __future__ import division
       import sys
       import numpy as np

       def circular_segment_area(rad, theta):
           return pow(rad, 2) * (theta - np.sin(theta)) / 2.0

       def get_area_exact(x1, y1, g, t):
           #Given the position of the lower left corner of a square
           #in the upper right quadrant of the racquet, find the
           #area of the unobstructed region (which will be a square
           #potentially occluded by a circular arc)

           #The occluding circle has radius 1 - t (we've normalized):
           R_adj = 1.0 - t

           #Check if the lower left corner is occluded (and we return 0)
           if pow(x1, 2) + pow(y1, 2) >= pow(R_adj, 2):
               return 0.0

           #Check if the upper right corner isn't occluded (and we
           #return the full area, g^2)
           x2 = x1 + g
           y2 = y1 + g
           if pow(x2, 2) + pow(y2, 2) <= pow(R_adj, 2):
               return pow(g, 2)

           #Otherwise we have some occlusion to worry about, and there
           #are four different options:
           # 1) only the upper right (UR) corner is occluded
           # 2) both the UR and UL corners are occluded
```

```
    # 3) both the UR and LR corners are occluded
    # 4) all of the UR, UL, and LR corners are occluded
    # Any other options are excluded by the checks we've already done.
    ul_occ = pow(x1, 2) + pow(y2, 2) >= pow(R_adj, 2)
    lr_occ = pow(x2, 2) + pow(y1, 2) >= pow(R_adj, 2)

    #Now the unoccluded area in the square is the sum of a single
    #circular segment and rectangular or right triangular regions.
    if not lr_occ:
        if not ul_occ:
            #Case 1, intersecting the upper and right edges
            segment = circular_segment_area(R_adj,
                np.arcsin(y2 / R_adj) - np.arccos(x2 / R_adj))
            #print segment, pow(g, 2)
            dx = x2 - np.sqrt(pow(R_adj, 2) - pow(y2, 2))
            dy = y2 - np.sqrt(pow(R_adj, 2) - pow(x2, 2))
            num = pow(g, 2) - dx * dy / 2 + segment
        else:
            #Case 2, intersecting the left and right edges
            segment = circular_segment_area(R_adj,
                np.arccos(x1 / R_adj) - np.arccos(x2 / R_adj))
            #print segment, pow(g, 2)
            y_left = np.sqrt(pow(R_adj, 2) - pow(x1, 2)) - y1
            y_right = np.sqrt(pow(R_adj, 2) - pow(x2, 2)) - y1
            num = g * (y_left + y_right) / 2 + segment
    else:
        if not ul_occ:
            #Case 3, intersecting the upper and lower edges
            segment = circular_segment_area(R_adj,
                np.arcsin(y2 / R_adj) - np.arcsin(y1 / R_adj))
            #print segment, pow(g, 2)
            x_upper = np.sqrt(pow(R_adj, 2) - pow(y2, 2)) - x1
            x_lower = np.sqrt(pow(R_adj, 2) - pow(y1, 2)) - x1
            num = g * (x_lower + x_upper) / 2 + segment
        else:
            #Case 4, intersecting the left and lower edges
            segment = circular_segment_area(R_adj,
                np.arccos(x1 / R_adj) - np.arcsin(y1 / R_adj))
            #print segment, pow(g, 2)
            dx = np.sqrt(pow(R_adj, 2) - pow(y1, 2)) - x1
            dy = np.sqrt(pow(R_adj, 2) - pow(x1, 2)) - y1
            num = dx * dy / 2 + segment
    return num

def solve_prob(f, R, t, r, g):
    #First adjust t (rim thickness), r (string radius),
    #and g (string gap) for the fly's radius f, so that we
    #can treat the fly as a point and then divide each value by R,
    #the outer radius of the raquet, so we're working on the
    #unit circle.
    #So, f=0 and R=1 after these adjustments.
    t = (t + f) / R
    r = (r + f) / R
    g = (g - 2 * f) / R
```

```
        #The gap might be negative now, and if it is, the fly will
        #always be hit!
        if g <= 0:
            return 1.0

        #Now we need to add up the areas of all the squares.  We'll
        #loop through the squares in the upper right quadrant (the
        #system is symmetric), indexed by their lower left corners.
        #
        #These values will range from r (the smallest) up to the
        #inner radius (1-t) in steps of 2r+g
        free_area = 0.0
        for x1 in np.arange(r, 1-t, g + 2 * r):
            for y1 in np.arange(r, 1-t, g + 2 * r):
                free_area += get_area_exact(x1, y1, g, t)

        #Calculate the area of the full quadrant of the racquet
        #and return 1 minus the fraction of free area.
        return 1.0 - 4 * free_area / np.pi

    infile = open("%s" % sys.argv[1], 'r')
    outfile = open("%s.out" % sys.argv[1][:-3], 'w')
    cases = int(infile.readline().strip())
    for i in range(cases):
        #Read in the raw parameters
        f, R, t, r, g = [float(num) for num in infile.readline().split()]
        output = solve_prob(f, R, t, r, g)
        outfile.write("Case #%i: %.6f\n" % (i+1, output))
        print "Case #%i: %.6f" % (i+1, output)
    infile.close()
    outfile.close()
```

Optional

e) Suppose you weren't sure about the exact geometric formulas in part c for each possible case. Write an approximate recursive solution to finding the area of each gap by repeatedly dividing the square into four pieces, making sure that you are within the error margin required.

Simply replace the calls in the solution above to `get_area_exact` with calls to `get_area_recursive` below:

```
In [ ]: def get_area_recursive(x1, y1, g, t, epsilon=1e-9):
            #Here we'll use a recursive approach
            R_adj = 1 - t

            if pow(x1, 2) + pow(y1, 2) >= pow(R_adj, 2):
                return 0.0

            x2 = x1 + g
            y2 = y1 + g
            if pow(x2, 2) + pow(y2, 2) <= pow(R_adj, 2):
                return pow(g, 2)
```

```
if pow(g, 2) <= epsilon:
    return pow(g, 2) / 2.0

midx = x1 + g / 2.0
midy = y1 + g / 2.0

return get_area_recursive(x1, y1, g / 2.0, t) + \
       get_area_recursive(midx, y1, g / 2.0, t) + \
       get_area_recursive(x1, midy, g / 2.0, t) + \
       get_area_recursive(midx, midy, g / 2.0, t)
```