Python Project 4

Nicholas Thompson (15019385) and Dinesh Kalamegam(16003714)

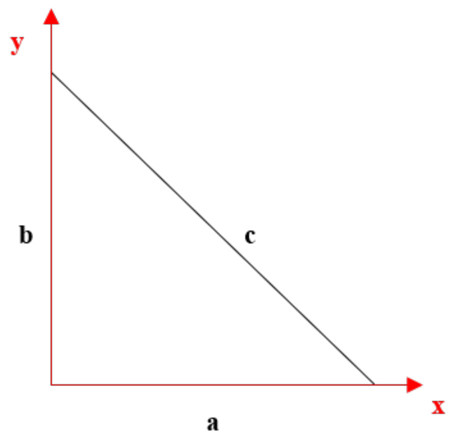
March 16, 2018

1 Introduction

This document provides information about the design decisions made in Python Project 4 , mainly delving into the non-trivial parts of the program which required mathematical thought in detail here so we do not clutter the code with comments.

2 Setting up the coordinate system

We imagine that the right angled triangle is placed into a 2D coordinate axes. The data file provides the triples a, b, c where $a < b < c$ this meant that a reasonable assumption for out coordinate grid is that the side a is along the x axis and the side y is along the y axis as the diagram below shows



3 Generating Uniformly Distributed Random Points and angle

3.1 Points

We initially tried generating the random points in a triangle just by finding random points in the x-y plane

```
x = random.uniform(0,a)
y = random.uniform(0,b)
```

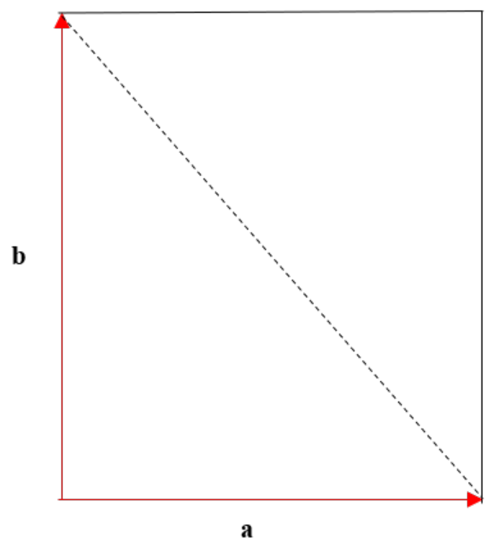
However we see that this does not work because the point may lie outside the triangle. So we use basic trignometery to see that $maxY = \frac{b(a-x)}{a}$ so we generate x the same as before but now limit y to this value like so

```
x = random.uniform(0,a)
maxY = (b(a-x))/a
y = random.uniform(0,maxY)
```

So we have the points inside the triangle but now there is another problem. The points in this triangle are not truly uniformly distributed which causes anamalous behaviour. This is because our y-value is dependent on the value of x which meant that a higher percentage of our points ended up in the bottom half of the triangle.

To fix the issue we thought about the triangle as a rectangle with sides a and b and diagonal c. To generate uniformly distributed points for a rectangle, we can generate a random number between 0 and 1 and then multiply by a to get the x-coordinate. Then to get the y-coordinate we generate another random number between 0 and 1 and then multiply by b. This doesn't work for our triangle as roughly half of the points would end up lying outside our triangle. To fix this issue, we reject any points that lie outside of our triangle; in other words we only accept the two random numbers generated if their sum is less than or equal to 1. This then gives us uniformly distributed points within our triangle.

To understand why this works we can consider vectors. Let **a** be a vector in the i direction and let **b** be a vector in the j direction. The magnitude of the vectors are determined by two different independent random numbers, r_1 and r_2 respectively (using rand.uniform). Both take values between 0 and 1. Thus the magnitude of a is between 0 and a and the magnitude of b is between 0 and b. All possible combinations of these vectors (since they are perpendicular and form a right angle) gives us a rectangle. The four vertices of our rectangle are formed by vectors $0i + 0j$, $\mathbf{a}i + 0j$, $0i + \mathbf{b}j$, $\mathbf{a}i + \mathbf{b}j$. Our triangle is formed by the three vertices: $0i + 0j$, $\mathbf{a}i + 0j$, $0i + \mathbf{b}j$.



To achieve points only within our triangle, the sum of our vectors must lie within the triangle. It is best to see this by considering the geometry/symmetry of the rectangle defined by the vectors. The dashed-line cuts our rectangle in half exactly and this line can be precisely reached from the origin by the vectors $r_1\mathbf{a} + r_2\mathbf{b}$ where $r_1 + r_2 = 1$.

Therefore, for the vector sum to lie in our triangle we require $r_1+r_2 \leq 1$ i.e. this guarantees that our points are uniformly distributed only inside our triangle.

3.2 Angle

Generating a random angle is *trivial.* We simply choose a random value between 0 and 2π using rand.uniform

4 Finding the probability

4.1 Equation of ant

To find the probability we consider the path of of the ant in a straight line equation given a random point (x_0, y_0) and angle θ . We recall the equation

$$y = mx + c$$

We first determine the gradient m. consider the *tangent* of θ, we see that for the ant

$$tan(\theta) = \frac{\Delta y}{\Delta x}$$

So $tan(\theta) = m$ Now we need to determine c

$$\begin{aligned} y &= mx + c \\ &\implies c = y - mx \\ &\implies c = y_0 - tan(\theta) \cdot x_0 \end{aligned}$$

So the path of the ant can be modelled by the straight line equation

$$y = tan(\theta) \cdot x + (y_0 - tan(\theta) \cdot x_0)$$

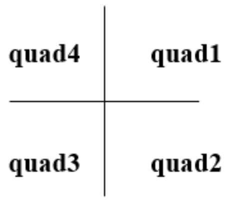
in the code m is the antGradient and c is the bIntercept. We also find the intercept with the x axis as it may prove to be of use. So to to this we solve for $y = 0$ i.e.

$$\begin{aligned} 0 &= tan(\theta) \cdot x + (y_0 - tan(\theta) \cdot x_0) \\ &= antGradient \cdot x + bIntercept \\ &\implies x = -\frac{bIntercept}{antGradient} \end{aligned}$$

This value of x will be called the aIntersect

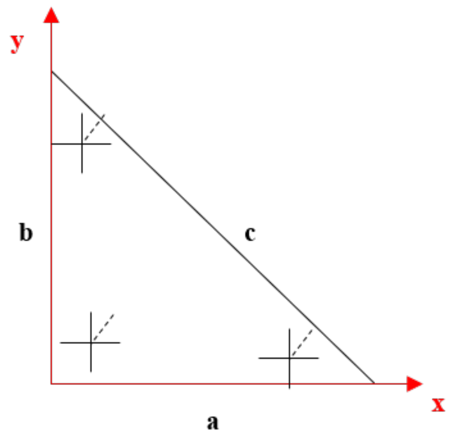
4.2 Quads

Ok now that we have done this we have to decide which side the ant goes to. We imagine the ant's direction to be split into four quadrants i.e. "quad" in the code. Then for each side we follow the instructions given and set the aCounter, bCounter and cCounter to 0



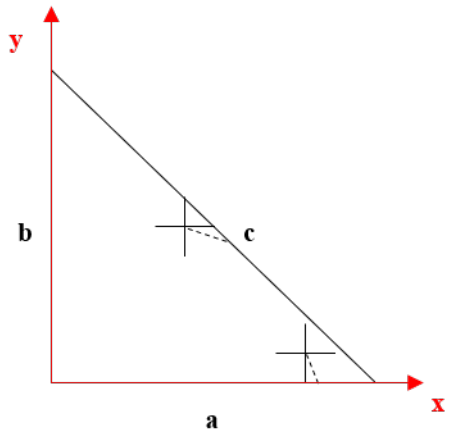
Now examine each each of the 4 cases invdividually

4.2.1 Quad1



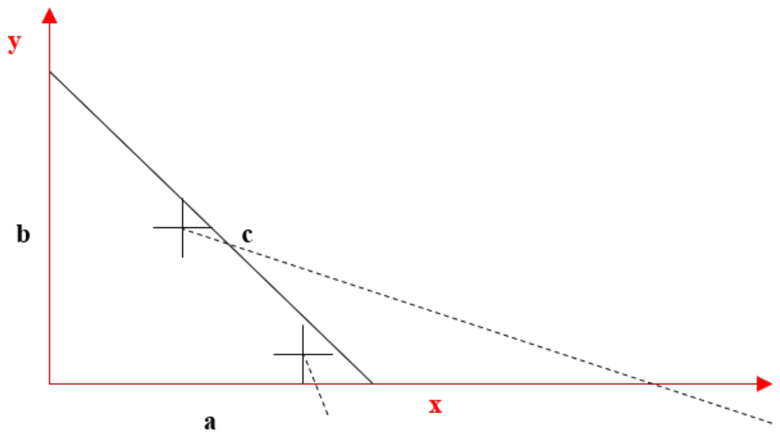
Quad1 is defined for the values $0 \leq \theta \leq \frac{\pi}{2}$. When we have the ant pointing in this direction we can trivially see that no matter where the ant is it would always go to the side c. We do not need to check anything else and may increase the cCounter by 1

4.2.2 Quad2



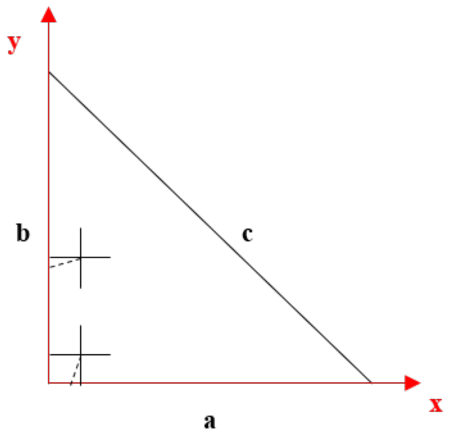
Quad2 is defined for the values $\frac{\pi}{2} \leq \theta \leq \pi$. Now here we see that we need to do more work. There are two possible intersections either with c or with a as the diagram above shows that are possible.

In this Quad we are particularly interested in aIntersect. That is when we solve for given the equation of the ants path in this direction we can have aIntersect $< a$ or aIntersect $> a$.

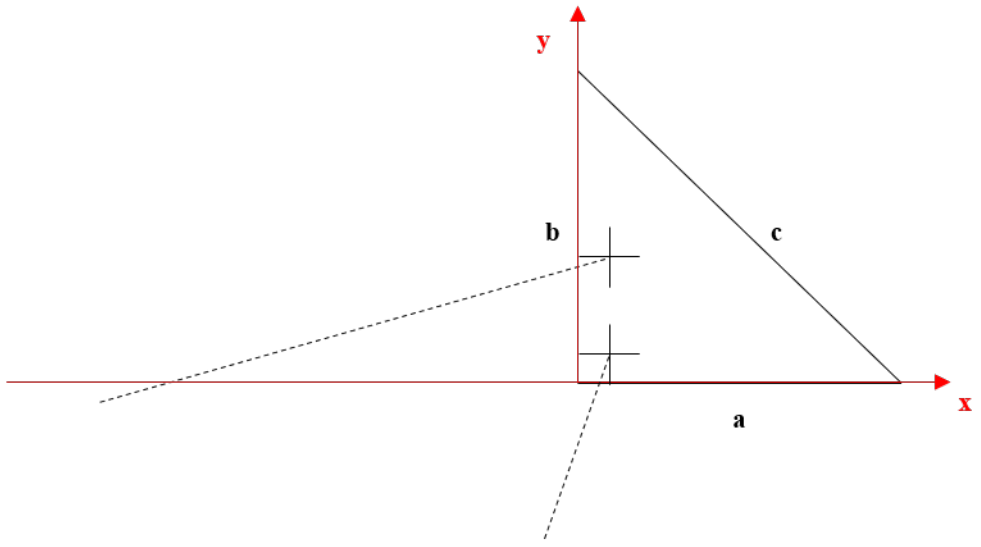


In the diagram above the upper marker shows when aIntersect $> a$ and the lower marker shows when aIntersect $< a$. We conclude that in the case aIntersect $< a$ intersection is with side a and we increase aCounter otherwise we increase cCounter.

4.2.3 Quad3

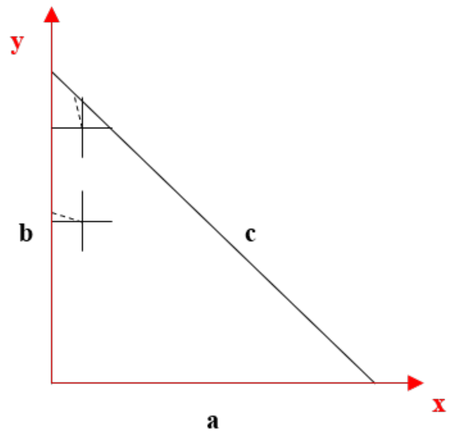


Quad3 is defined for the values $\pi \leq \theta \leq \frac{3\pi}{2}$. There are two possible intersections either with a or with b as the diagram above shows that are possible. In this Quad we are again interested in aIntersect. Now we observe the following diagram

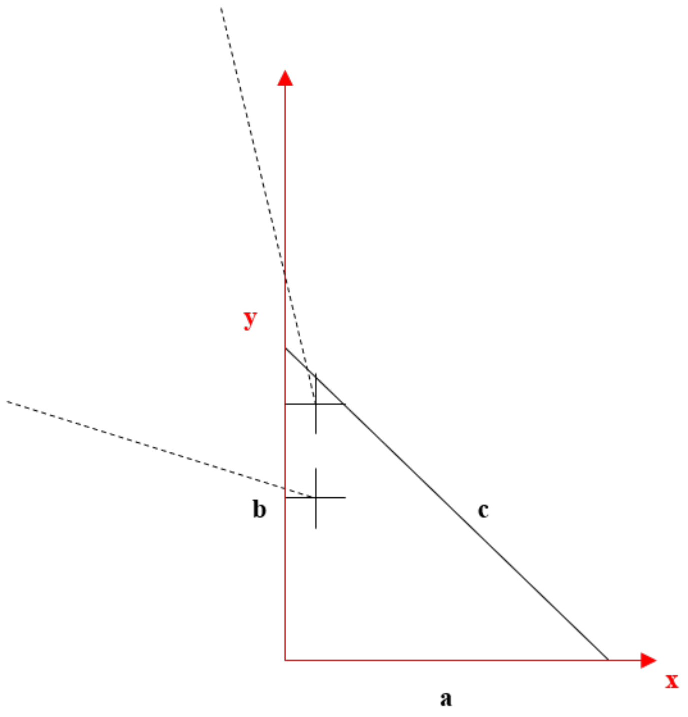


In the diagram above the upper marker shows when aIntersect < 0 and the lower marker shows when aIntersect > 0. We conclude that in the case aIntersect < 0 intersection is with side b and we increase bCounter otherwise we increase aCounter.

4.2.4 Quad4



Quad4 is defined for the values $\frac{3\pi}{2} \leq \theta \leq 2\pi$. There are two possible intersections either with side b or with c as the diagram above shows that are possible. In this Quad we are now interested in the bIntercept. Now observe the following diagram



In the diagram above the upper marker shows when bIntercept $> b$ and the lower marker shows when bIntercept $< b$. We conclude that in the case bIntercept $< b$ intersection is with side b and we increase bCounter otherwise we increase cCounter.

4.3 Using the results

Now that we have values for aCounter, bCounter and cCounter we simply divide each one by N this gives us the exit probabilities as required.

5 Lowest and Highest probabilities

As we calculate the exit probabilities for each triangle we store the probability and array into a dictionary that takes the format

```
{probability : array}
```

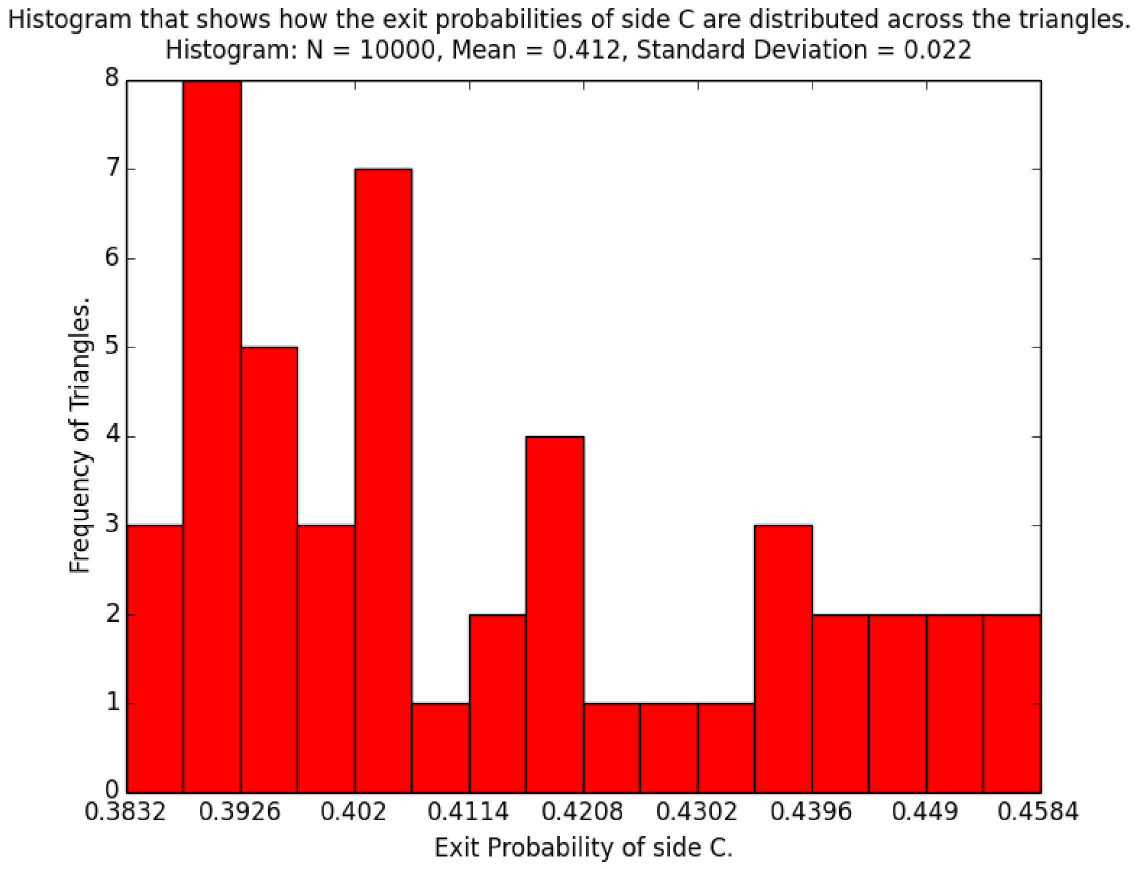
The reason for this is that we can then sort the dictionary by key value and place the probabilities in another array. The first value in this sorted probability array will always contain the smallest probability and the last value will always contain the largest probability. By doing it like this we have a one to one mapping of probability $\rightarrow$ array so we can simply get the array by looking up the dictionary value that had the key of the probability.

6 Results and Histogram

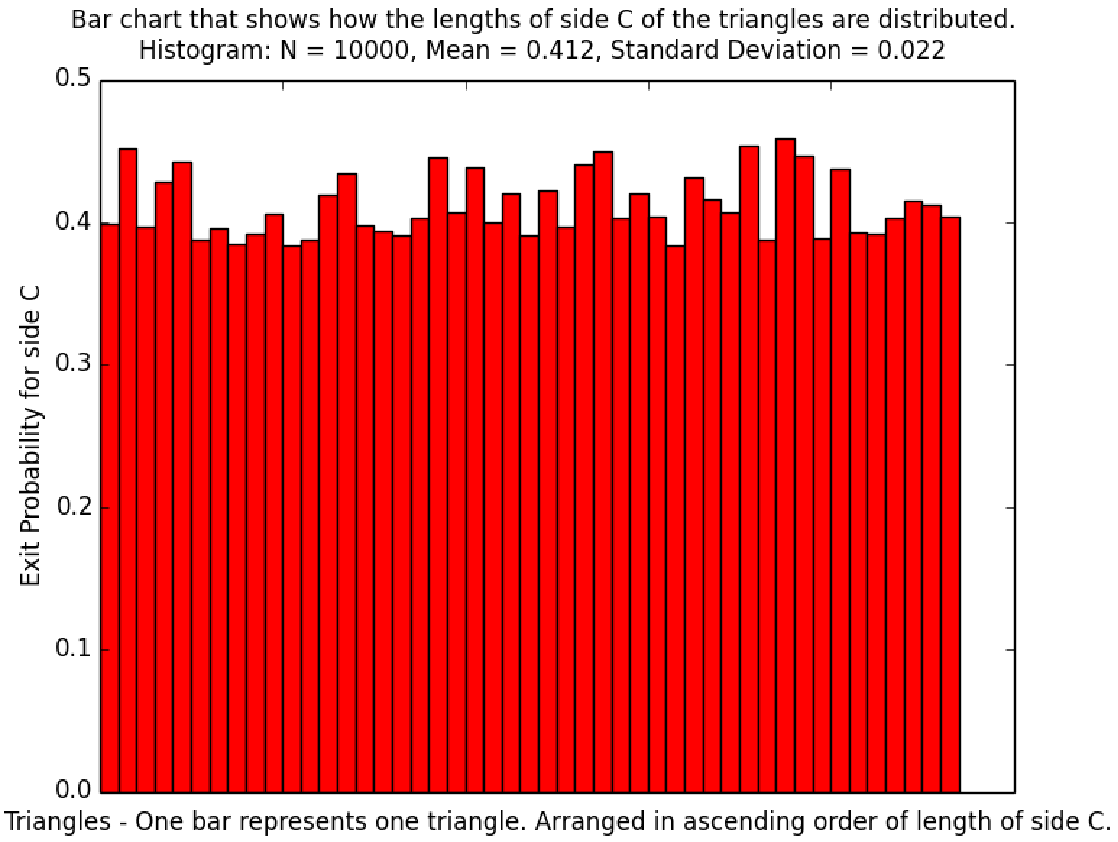
As we increase the size of N we notice several things. The smallest and largest probability values change with increasing N and so do the triangles that produce these values. For smaller N there is a greater difference between the smallest and largest probability but as we increase N this difference gets smaller. This is also shown by the fact that the standard deviation of the data decreases as we increase N. At large N the probability values begin to stabilize and thus there is less variation in the triangles that produce the largest and smallest values.

For our histogram (See figure1.png), we interpret the specification to show how the frequency of Pythagorean triangles are distributed over different exit probabilities for the longest side c. This seems the most useful because **a**) our project is mainly concerned with the exit probabilities of the longest side and **b**) in the data file the triangles are ordered in ascending order according to their longest side. Originally we supplied **plt**.**hist** function with the array of probabilities of all triangles for the longest side c.

However, using just the built in function we were unable to achieve the histogram we wanted. Therefore, we had to calculate the intervals and frequencies ourselves and then plot these using **plt**.**bar** function. To calculate evenly spaced intervals we used a built in function **numpy**.**linspace**. Then using these intervals, we calculated frequency for each interval by adding up all the probabilities that fell within the intervals. When N is changed in our program this then changes, our array of probabilities which in turn changes our histogram. It allows us to see the spread of exit probabilities for the Pythagorean triples and how this changes with different N. Below is an example of our histogram that was generated with $N = 10,0000$.



For our own additional interest we also decided to plot a bar graph (see figure2.png) so we could see if there was any trend between increasing length of side C of the triangles and exit probability for side C. In the bar graph each bar represents a triangle. They are arranged left to right in ascending order of length of side C. Therefore, the first bar represents our $3, 4, 5$ triangle with side C length 5. An example is presented below.



From our example it seems that there is no correlation between increasing length of side C and the exit probability for side C. This was confirmed by running the program several times with increasing N. It seems that a more complex relationship is needed to describe the trends of probabilities. We believe that this could potentially be that the exit probability is determined by the ratio of the lengths of the sides. The greater the length of side C relative to sides A, B would mean our exit probability is higher. Of course this is just our theory and more testing would need to be done to confirm this.