Primitive Pythagorean Triples

Introduction

Primitive Pythagorean Triples, PPTs, are the solutions to $a^2 + b^2 = c^2$ for which a, b and c are relatively prime. Relative prime means that the greatest common divisor between a, b and c is 1. Our task for this lab is to generate the first PPTs in the range of a,b from zero to five thousand. Next, we plot the data points and make out any discernible structures.

Section 1: Algorithm and Data

To complete this task we created two functions, one being the Euclidean algorithm and the other being a perfect square test.

In [2]:
```
def GCD(a,b):
    "Find the greatest common divisor of two numbers"
    while a != b:
        if a > b:
            a = a - b
        else::
            b = b - a
    return a
def is_square(x):
    "Determine if a^2 + b^2 is a square"
    if x**(1/2) == int(x**(1/2)):
        return True
    else:
        return False
```

The purpose of the **Euclidean algorithm** is to reduce the pair of numbers a and b to their greatest common divisor. The way this works is by subtracting the smaller value from the larger until they are equal. Once this condition is met, the value of their greatest common divisor, **GCD**, will be returned. As you will see, when we generate the PPTs, we require that the GCD between a and b must be 1, if not then they are not primitive.

In order for the three numbers to be Pythagorean triples, $a^2 + b^2 = c^2$ must be a perfect square. To test for that, we compared the actual value of c to its integer representation, if they are equivalent that means that c is a perfect square.

The combination of the two algorithms will allow us to generate the PPTs.

In [3]:
```
PPTa = []
PPTb = []
PPTc = []
a = 1
for a in range(a, 5001):
    for b in range(a, 5001):
        if GCD(a,b) == 1:
            if is_square(a**2 + b**2) == True:
                PPTa.append(a)
                PPTb.append(b)
                c = (a**2 + b**2)**(1/2)
                PPTc.append(int(c))
```

While that's running, let's explain what it's doing. First, in order to get rid of duplicate answers, we required that $a \leq b$ at any given value of a. Then we require that the GCD of a and b is equal to one so that they are primative. Additionally, we require that the $a^2 + b^2$ is a perfect square. Finally, we append the values of a, b and the calculated value of c to lists.

In [4]:
```
print(PPTa[0:100])
print(PPTb[0:100])
print(PPTc[0:100])
print(len(PPTa))
```

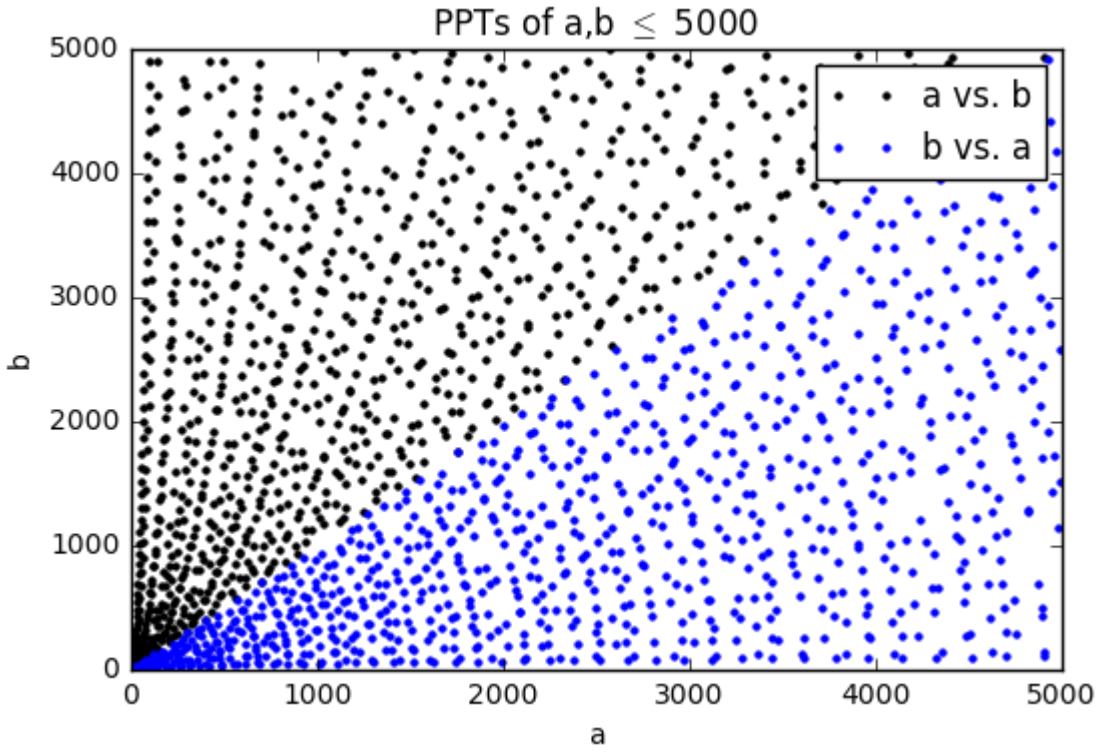
```
[3, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 20, 21, 23, 24, 25, 27, 28, 28,
29, 31, 32, 33, 33, 35, 36, 36, 37, 39, 39, 40, 41, 43, 44, 44, 45, 47, 48, 48,
49, 51, 51, 52, 52, 53, 55, 56, 57, 57, 59, 60, 60, 60, 61, 63, 64, 65, 65, 67,
68, 68, 69, 69, 71, 72, 73, 75, 75, 76, 76, 77, 79, 80, 81, 83, 84, 84, 84, 85,
85, 87, 87, 88, 88, 89, 91, 92, 92, 93, 93, 95, 95, 96, 96, 97, 99, 100, 100]
[4, 12, 24, 15, 40, 60, 35, 84, 112, 63, 144, 180, 21, 99, 220, 264, 143, 312,
364, 45, 195, 420, 480, 255, 56, 544, 612, 77, 323, 684, 80, 760, 399, 840, 92
4, 117, 483, 1012, 1104, 55, 575, 1200, 140, 1300, 165, 675, 1404, 1512, 783, 1
76, 1624, 1740, 91, 221, 899, 1860, 1984, 1023, 72, 2112, 2244, 285, 1155, 260,
2380, 2520, 1295, 2664, 308, 2812, 357, 1443, 2964, 3120, 1599, 3280, 3444, 18
7, 437, 1763, 132, 3612, 416, 3784, 105, 1935, 3960, 4140, 525, 2115, 476, 432
4, 168, 4512, 247, 2303, 4704, 4900, 621, 2499]
[5, 13, 25, 17, 41, 61, 37, 85, 113, 65, 145, 181, 29, 101, 221, 265, 145, 313,
365, 53, 197, 421, 481, 257, 65, 545, 613, 85, 325, 685, 89, 761, 401, 841, 92
5, 125, 485, 1013, 1105, 73, 577, 1201, 149, 1301, 173, 677, 1405, 1513, 785, 1
85, 1625, 1741, 109, 229, 901, 1861, 1985, 1025, 97, 2113, 2245, 293, 1157, 26
9, 2381, 2521, 1297, 2665, 317, 2813, 365, 1445, 2965, 3121, 1601, 3281, 3445,
205, 445, 1765, 157, 3613, 425, 3785, 137, 1937, 3961, 4141, 533, 2117, 485, 43
25, 193, 4513, 265, 2305, 4705, 4901, 629, 2501]
890
```

By printing the first 10 PPTs, we can see that the algorithm works and we confirm their accuracy by comparing them to a known list of PPTs. Furthermore, there are 890 PPTs in the range of zero to five thousand. The next thing to do is to plot a and b to see if we can find any patterns or structures.

Section 2: Analysis

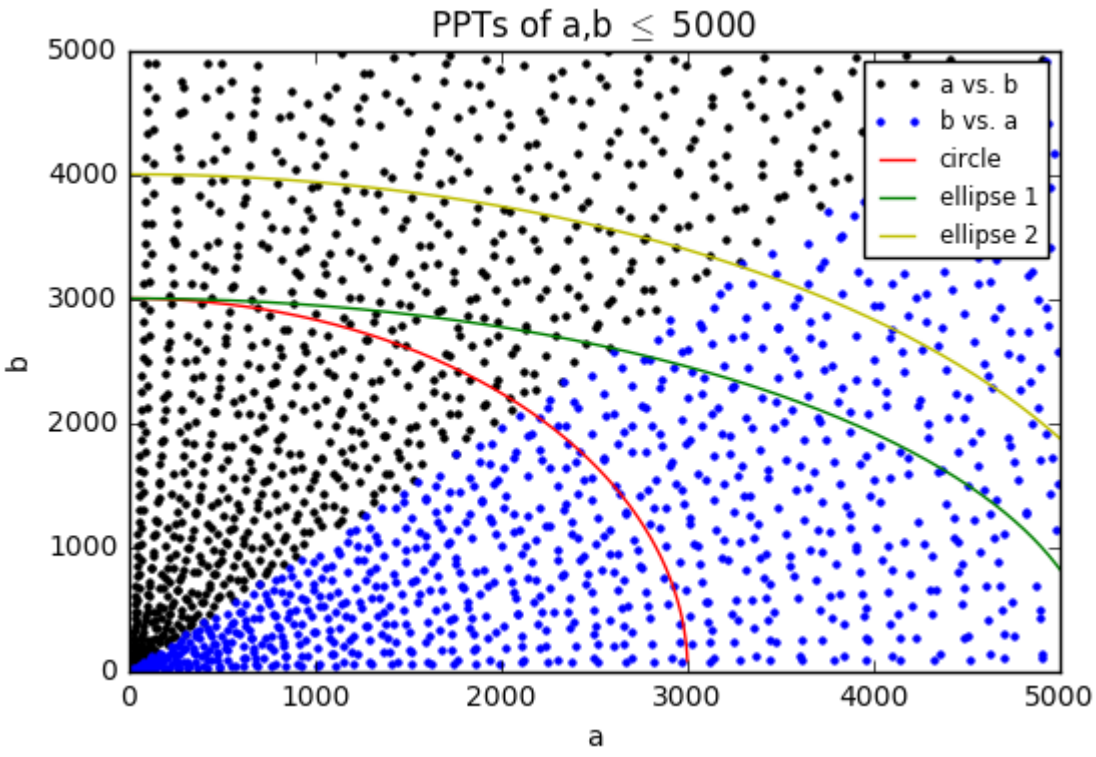
In [194]:
```
%matplotlib inline
```

In [195]:
```
import matplotlib.pyplot as plt
plt.plot(PPTa, PPTb, 'k.', ms = 5,label = 'a vs. b')
plt.plot(PPTb,PPTa, 'b.', ms = 5, label = 'b vs. a')
plt.title('PPTs of a,b $\leq$ 5000')
plt.xlabel('a')
plt.ylabel('b')
plt.legend()
plt.show();
```



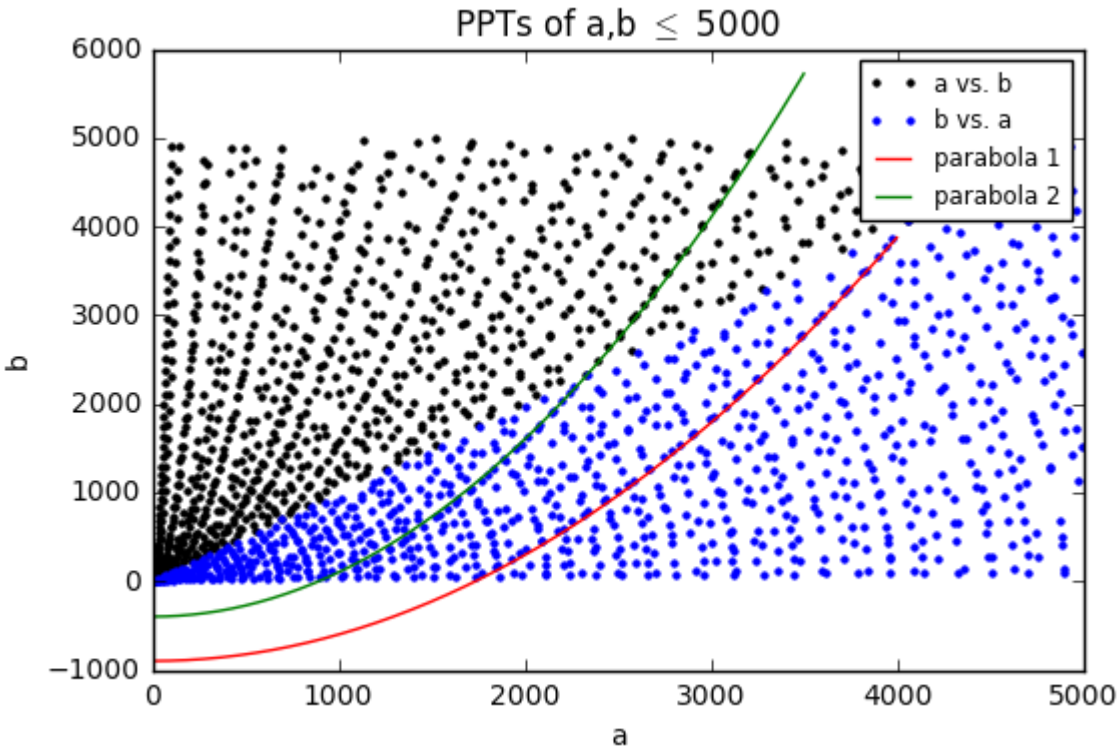
From this graph we can see quadratic structures from multiple directions. We plotted the reflection of a vs. b over $y = x$ to emphasize the structures a little more clearly. The structures seem to generate periodically and get more spaced out as a, b approach infinity. They also seem to be in different orientations and translations, even scaled differently. A possible reason for the quadratic curvature is due to the equation to solve for the PPTs. It follows the same form as a circle ($x^2 + y^2 = r^2$) and a similar form to an ellipse $\left(\frac{x^2}{a^2} + \frac{y^2}{b^2} = r^2\right)$. We'll overlay an ellipse and circle to show the resemblence.

In [207]:
```
import matplotlib.pyplot as plt
y = []    #list declarations for our data
x = []
d = []
e = []
f = []
g = []
for i in range(0,3000):
    y.append(((3000**2 - i**2))**(1/2))
    x.append(i)
for j in range(0,5000):
    e.append(((3000**2 - (1/3)*j**2))**(1/2))
    d.append(j)
for k in range(0,5000):
    f.append(((4000**2 - (1/2)*k**2))**(1/2))
    g.append(k)
plt.plot(PPTa, PPTb, 'k.', ms = 5,label = 'a vs. b')
plt.plot(PPTb,PPTa, 'b.', ms = 5, label = 'b vs. a')
plt.plot(x,y,'r', label = 'circle')
plt.plot(d,e,'g', label = 'ellipse 1')
plt.plot(g,f,'y', label = 'ellipse 2')
plt.title('PPTs of a,b $\leq$ 5000')
plt.xlabel('a')
plt.ylabel('b')
plt.legend(prop={'size':8.5})
plt.show();
```



After fitting both a circle and two ellipses on the graph it doesn't seem to fit the data that well. Now we are going to try to fit a parabala to the graph because a few of the structures seem to be purely parabolic.

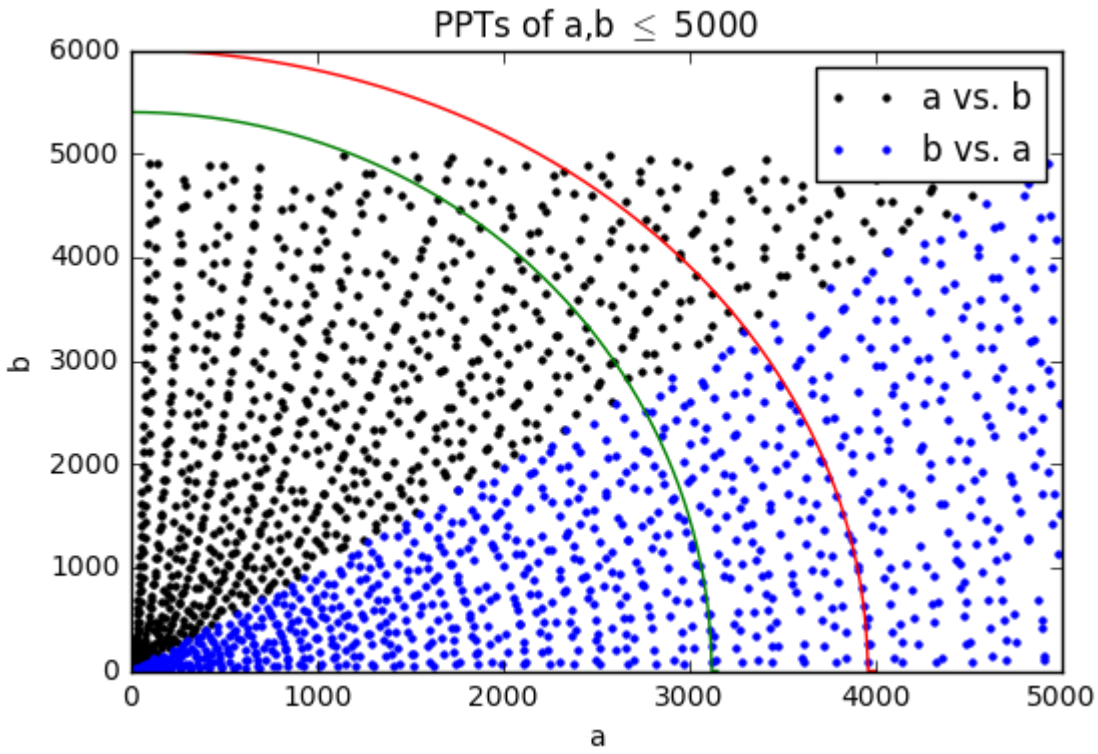
In [209]:
```
import matplotlib.pyplot as plt
pary1 = []    #list declarations for our data
parx1 = []
pary2 = []
parx2 = []
for i in range(0,4000):
    pary1.append((-900 + (1/3350)*i**2))
    parx1.append(i)
for i in range(0,3500):
    pary2.append((-400 + (1/2000)*i**2))
    parx2.append(i)
plt.plot(PPTa, PPTb, 'k.', ms = 5,label = 'a vs. b')
plt.plot(PPTb,PPTa, 'b.', ms = 5, label = 'b vs. a')
plt.plot(parx1,pary1,'r', label = 'parabola 1')
plt.plot(parx2,pary2,'g', label = 'parabola 2')
plt.title('PPTs of a,b $\leq$ 5000')
plt.xlabel('a')
plt.ylabel('b')
plt.legend(prop={'size':8.5})
plt.show();
```



After fitting two parabolas to the figure, we can see that the data seems to follow a parabolic curve very well. We are trying to draw a relation to the equation to find the PPTs in the first place but it doesn't have a $y = x^2$ type relation. The large fraction in front of the independent variable i is there because that value widens and narrows the parabola and we need it to be narrowed to fit our data.

In [198]:
```
import matplotlib.pyplot as plt
hypy1 = []    #list declarations for our data
hypx1 = []
hypy2 = []
hypx2 = []
for i in range(0,4000):
    hypx1.append(((6000**2) - (2.3)*i**2)**(1/2))
    hypy1.append(i)
for i in range(0,3150):
    hypx2.append(((5400**2) - 3.*i**2)**(1/2))
    hypy2.append(i)
plt.plot(PPTa, PPTb, 'k.', ms = 5,label = 'a vs. b')
plt.plot(PPTb,PPTa, 'b.', ms = 5, label = 'b vs. a')
plt.plot(hypy1,hypx1,'r')
plt.plot(hypy2,hypx2,'g')
plt.title('PPTs of a,b $\leq$ 5000')
plt.xlabel('a')
plt.ylabel('b')
plt.legend()
plt.show();
```

```
C:\Users\ProBook\Anaconda3\lib\site-packages\numpy\core\numeric.py:482: Complex
Warning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



After trying to fit the hyperbolas to the figure, we can't get a reliable fit; they do not seem to follow the same path. As for the warning that printed out in our code, that comes from the fact that we have the potential for a complex number for y since we have an increasing x subtracting from a constant under a square root. For the sake of fitting though, this isn't a huge issue.

Section 3: Conclusion

We were able to generate the first 890 PPTs using our algorithm and plotted them to identify any structures formed by them. After our analysis of the data, we think the parabola is the most accurate of all the functions we tried to fit. For certain PPTs, the function for determining them could be represented as $y = c + x^2$ where c is the y intercept. As you can see from the graph, this only happens for the PPTs that lie on the parabolic shape. As for the other structures, they seem to follow shapes similar to ellipses and hyperbolas, but we couldn't quite get an accurate fit.