Twin Prime / Goldbach

Introduction

In this document, we shall be investigating two famous conjectures related to prime numbers. The Goldbach conjecture was established by its author, Christian Goldbach, during a letter to the famous mathematic Leonhard Euler in the 1700s. The Twin Prime conjecture was made in the 1800s by several mathematicians, and is the central problem behind mathematician Yitang Zhang's recent breakthrough discoveries in the early 2010s. Both conjectures remain unsolved today, although they are still being monitored and investigated by many scholars around the globe.

Twin Prime Conjecture

"A twin prime is a prime number that is either 2 less or 2 more than another prime number—for example, either member of the twin prime pair (41, 43). In other words, a twin prime is a prime that has a prime gap of two. Sometimes the term twin prime is used for a pair of twin primes; an alternative name for this is prime twin or prime pair." - https://en.wikipedia.org/wiki/Twin_prime

The Twin Prime conjecture states that there are infinitely many twin prime pairs. Examples of Twin Primes are shown in the following code:

```
def twin_primes(N):
    pairs = []
    i = 1
    while (i < N):
        i, j = next_prime(i), i
        if i-j == 2:
            pairs.append((i,j))
    return pairs
```

```
twin_primes(50)
```

```
    [(5, 3), (7, 5), (13, 11), (19, 17), (31, 29), (43, 41)]
```

```
lst = twin_primes(10^8)
len(lst)
```

```
    440312
```

As you can see, there are only a considerable few sets of primes that are twins.

Goldbach Conjecture

The Goldbach conjecture states that every even whole number greater than two can be written as the sum of two prime numbers.

For both conjectures, we shall work to develop techniques that help to understand the relative properties, patterns, and principles of primes. For the Goldbach conjecture, we will explore how the representations of sums of primes grow, and how to correct for the over representation of such sums. This will be developed into a heuristic that supports our investigations. For Twin Primes, we shall see how restricting the goldbach conjecture to being the sum of twin cores, which we shall define later, affects our results.

Body

Investigating Goldbach

We will first create a list of the prime-sum representations of numbers up to 100000.

```
N = 100000
```

```
P = []
for i in srange(3,N):
    if is_prime(i):
        P.append(i)

R = [0 for i in srange(2*N)]
for p in P:
    for q in P:
        R[p+q]+=1
```

To test our conjecture, we want to make sure that each even number greater than two has at least one representation.

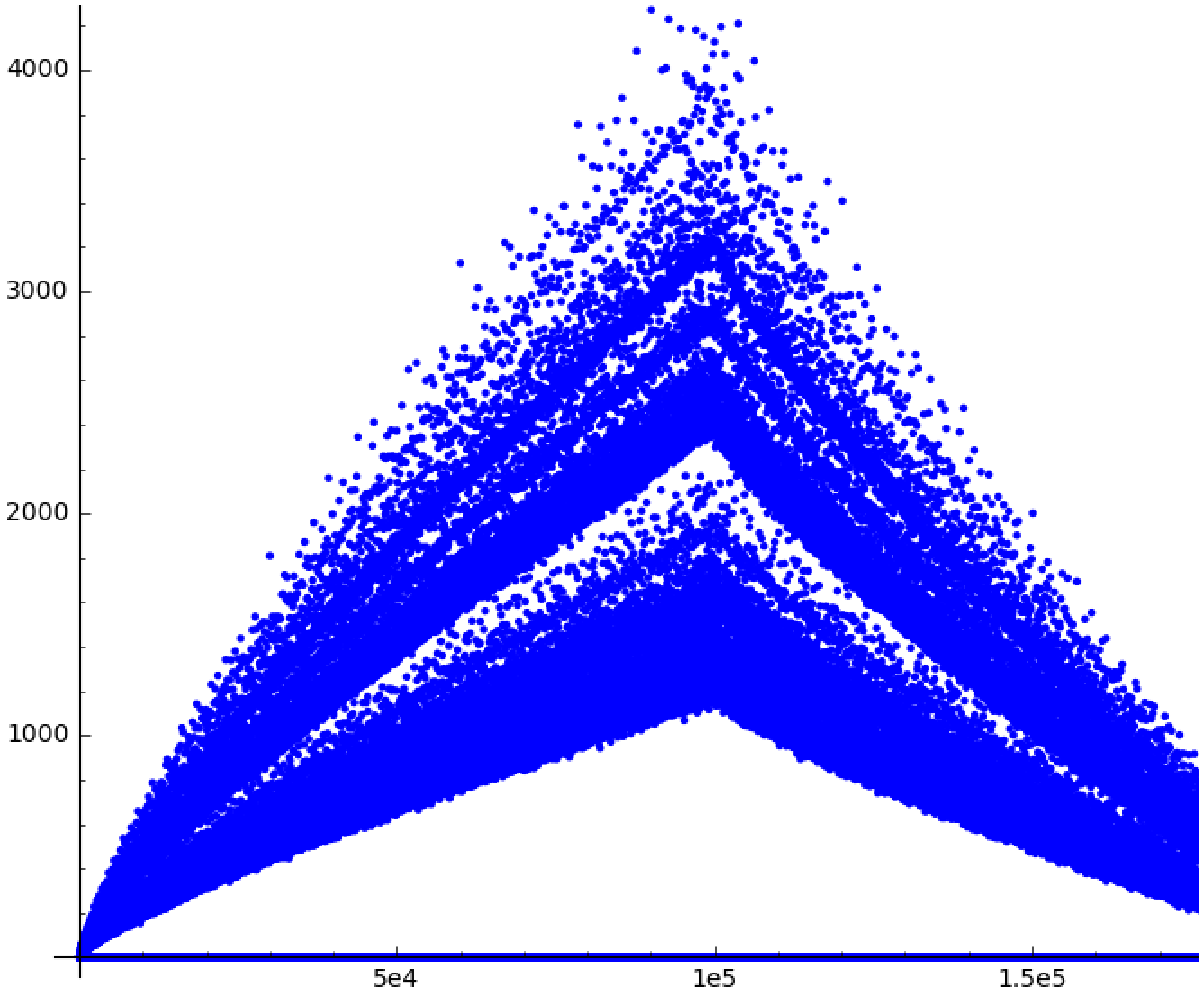
```
def test_goldbach(R):
    for i in srange(2,N/2):
        if R[2*i] == 0:
            return("goldbach false, " + str(2*i))
    return("goldbach supported")
```

```
test_goldbach(R)
```

```
    'goldbach supported'
```

Now that our conjecture is supported for even numbers up to 100000, let's take a look at the number of representations for each of these numbers.

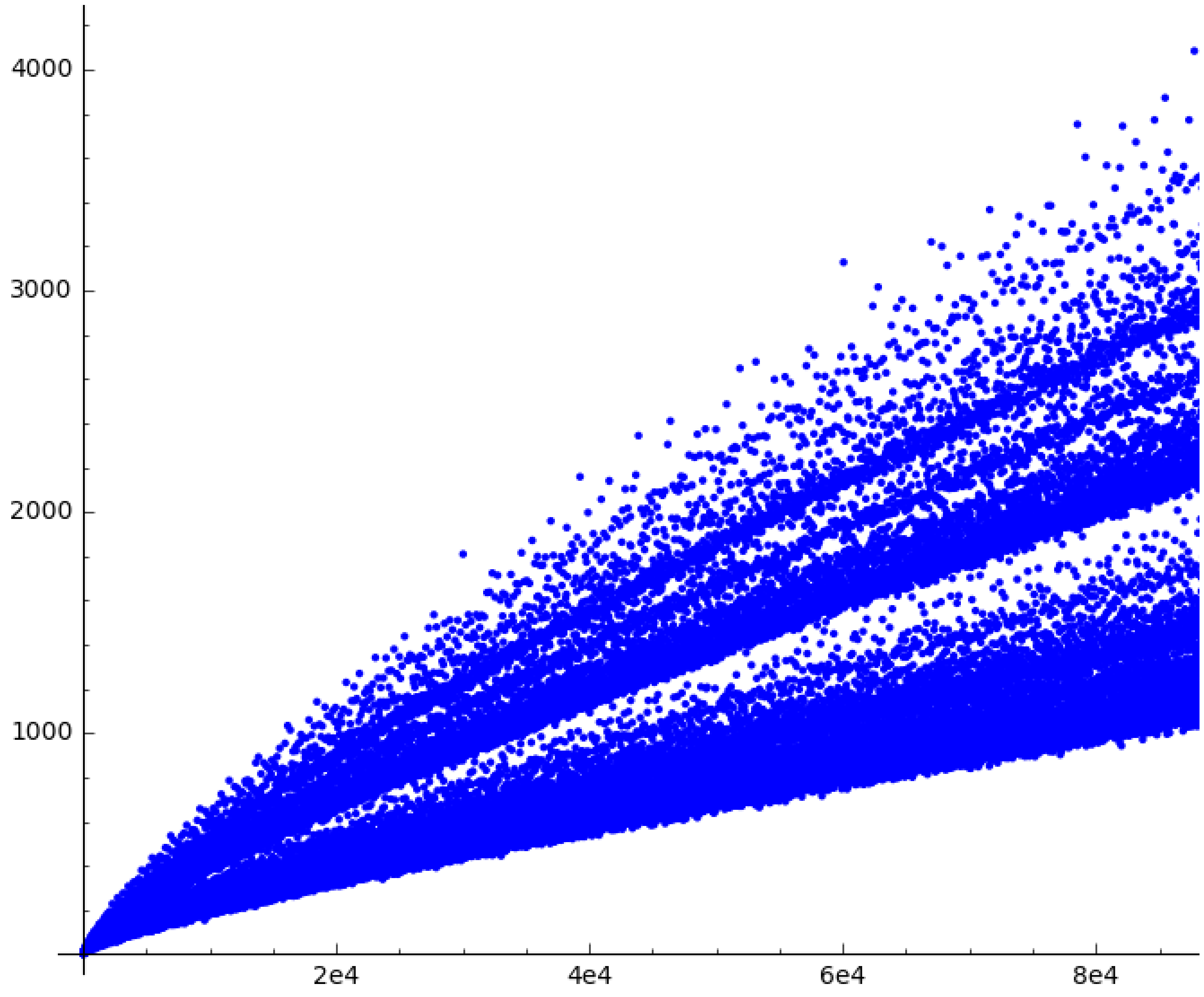
```
list_plot(R)
```



To develop a better idea for the growth of representations, lets eliminate all the zeros (which come from the odd numbers) and only look at the prime-sums up to 100000.

```
R = [[i,R[i]] for i in srange(N) if R[i]>0]
```

```
list_plot(R)
```



Now that we have a good visual for the growth of representations, we will begin investigating the clear striations that branch throughout the plot.

```
2*3*5*7*11*13*15
```

```
450450
```

Let's start by looking at how the quantity of prime factors dividing N reflects the number of its representations in our data.

At most, a number less than $N = 100000$ will have 6 prime factors:

$$2 * 3 * 5 * 7 * 11 * 13 * 15 = 450450$$

Therefore, we can separate all numbers up to N into 6 groups, one for each of the quantities of prime factors.

```
f1=[]
f2=[]
f3=[]
f4=[]
f5=[]
```

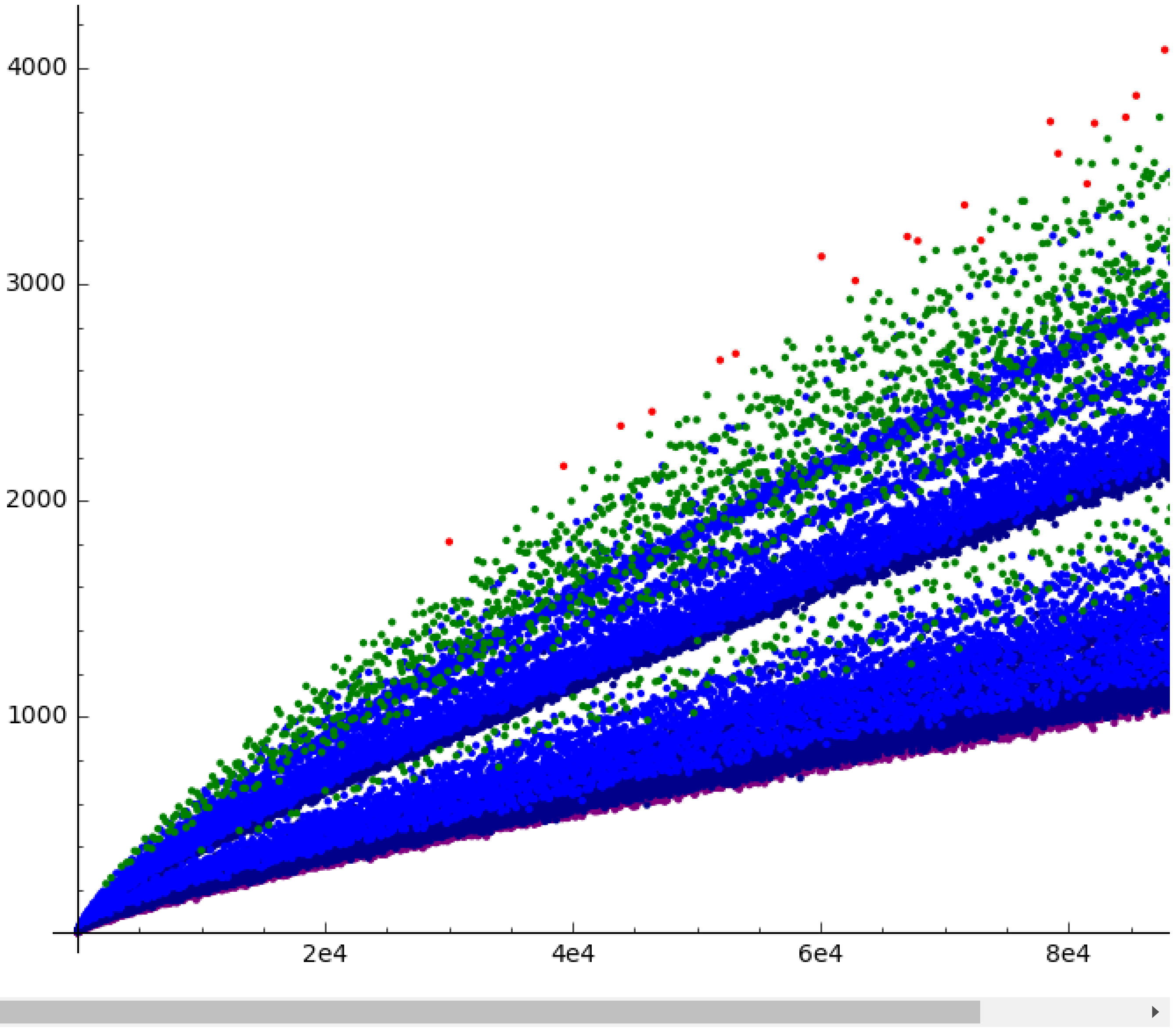
```
f6=[]
for s in R:
    if len(factor(s[0])) == 1:
        f1.append(s)
    if len(factor(s[0])) == 2:
        f2.append(s)
    if len(factor(s[0])) == 3:
        f3.append(s)
    if len(factor(s[0])) == 4:
        f4.append(s)
    if len(factor(s[0])) == 5:
        f5.append(s)
    if len(factor(s[0])) == 6:
        f6.append(s)
```

Let's label our groups dark for when we expect low representation, and warm when we expect high representation.

For our data, we would expect more representations from numbers with more prime factors; therefore group 1 will be black, which is the darkest, and group 6 will be red, the warmest.

```
A=list_plot(f1,color='black')
B=list_plot(f2,color='purple')
C=list_plot(f3,color='darkblue')
D=list_plot(f4,color='blue')
E=list_plot(f5,color='green')
F=list_plot(f6,color='red')
```

```
show(A+B+C+D+E+F)
```



It's now easy to see that the striations are heuristically defined by some correlation to a number's prime factors.

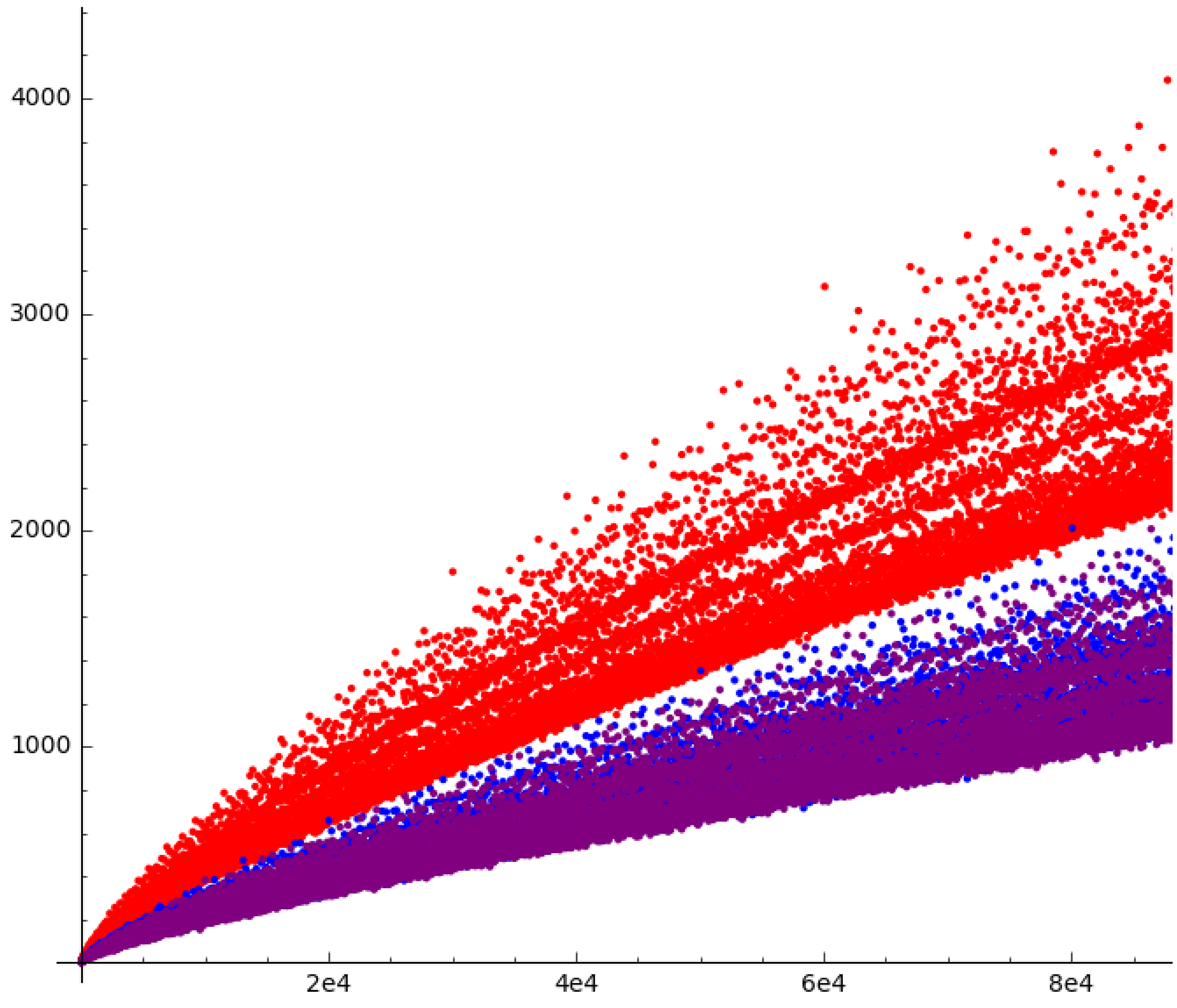
To explore this more, let's take only one prime factor: 3. Our numbers fall into three categories; there are those with a remainder of 0 modulo 3, indicating that the number has 3 has a prime factor, and there are also those with remainders of 1 and 2.

We will now separate our numbers into these three categories. We will follow the same convention as above and label our plots according to what we expect its representations to be, either dark or warm.

```
r0=[]
r1=[]
r2=[]
for s in R:
    if s[0]%3==0:
        r0.append(s)
    if s[0]%3==1:
        r1.append(s)
    if s[0]%3==2:
        r2.append(s)
```

```
A=list_plot(r0,color='red')
B=list_plot(r1,color='blue')
C=list_plot(r2,color='purple')
```

```
show(A+B+C)
```



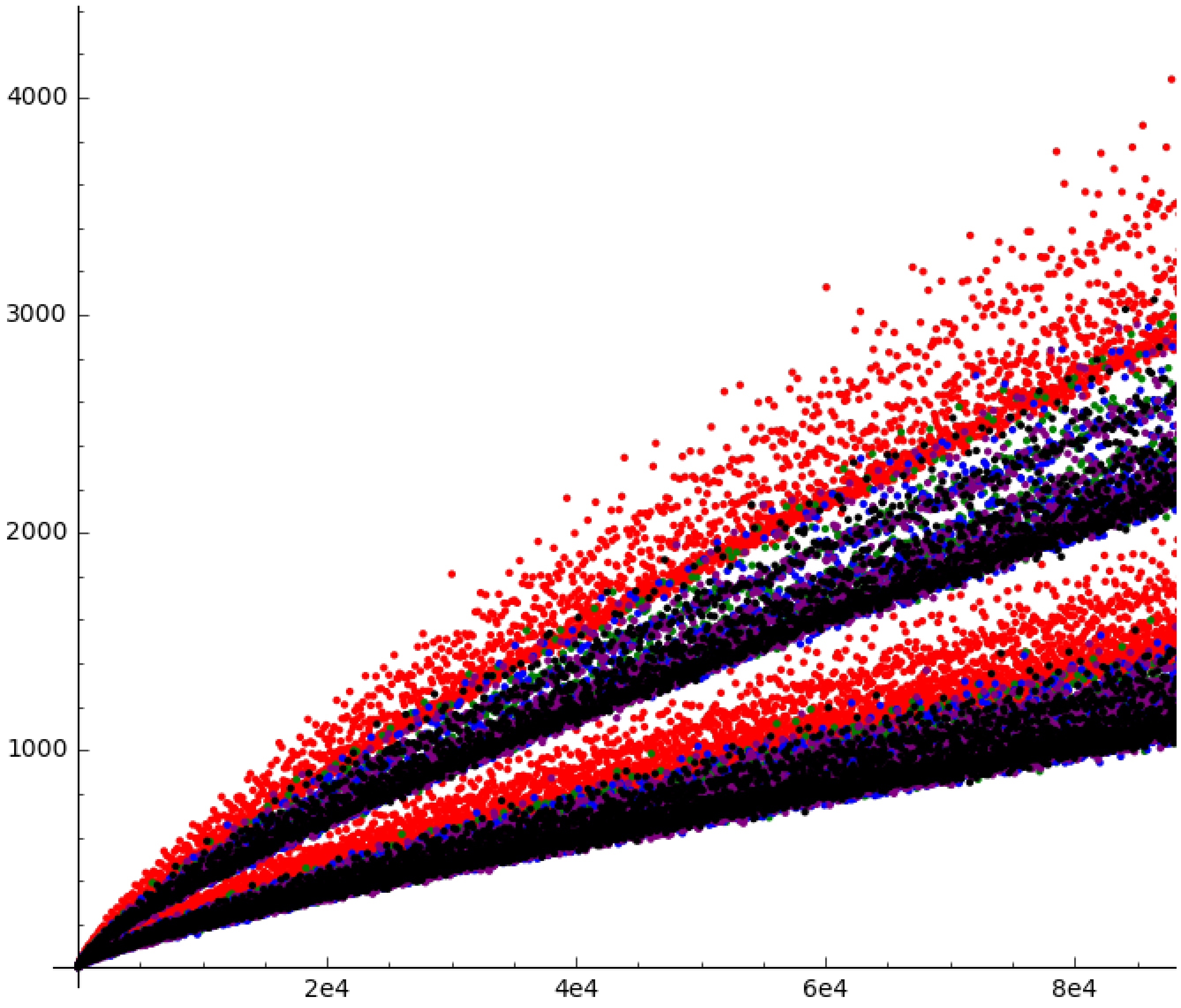
This result appears even more unaminous than the previous one. Clearly, numbers which approach a divisibility by 3 are being more represented within our data.

Let's try the same thing with 5.

```
t0=[]
t1=[]
t2=[]
t3=[]
t4=[]
for s in R:
    if s[0]%5==0:
        t0.append(s)
    if s[0]%5==1:
        t1.append(s)
    if s[0]%5==2:
        t2.append(s)
    if s[0]%5==3:
        t3.append(s)
    if s[0]%5==4:
        t4.append(s)
```

```
A=list_plot(t0,color='red')
B=list_plot(t1,color='green')
C=list_plot(t2,color='blue')
D=list_plot(t3,color='purple')
E=list_plot(t4,color='black')
```

```
show(A+B+C+D+E)
```



The results for a number's divisibility by 5 appears special, in that there are two branches within the striations of representations. Why so?

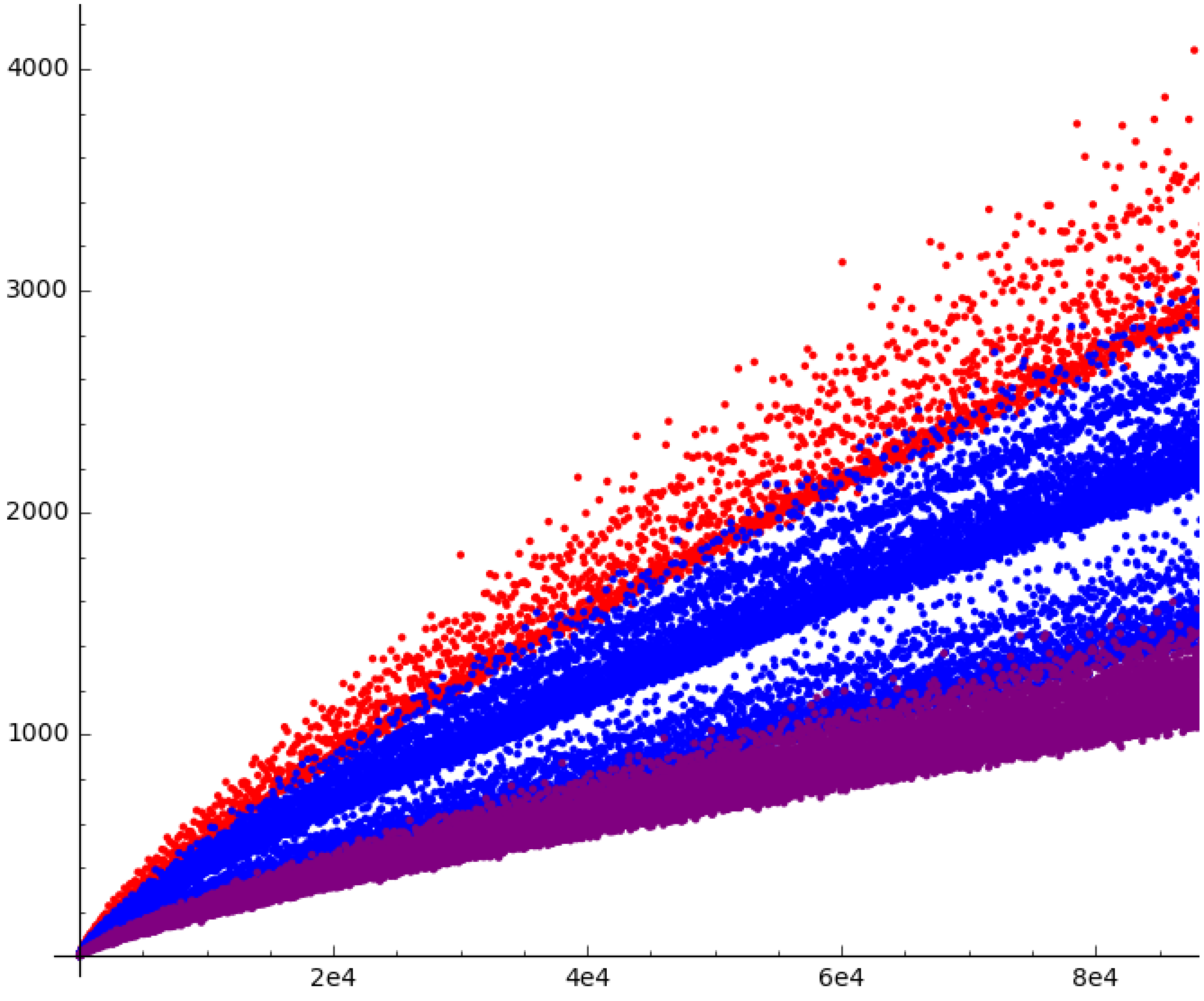
One way of thinking about these results is that, within the striation where numbers are divisible by 3, there seems to be another hierarchy of preference where divisibility by 5 is better represented.

This implies that the metric which determines the striations might be some product of the correlations for prime factors to representations.

```
u0=[]
u1=[]
u2=[]
for s in R:
    if s[0]%5==0 and s[0]%3==0:
        u0.append(s)
    elif s[0]%5==0 or s[0]%3==0:
        u1.append(s)
    else:
        u2.append(s)
```

```
A=list_plot(u0,color='red')
B=list_plot(u1,color='blue')
C=list_plot(u2,color='purple')
```

```
show(A+B+C)
```



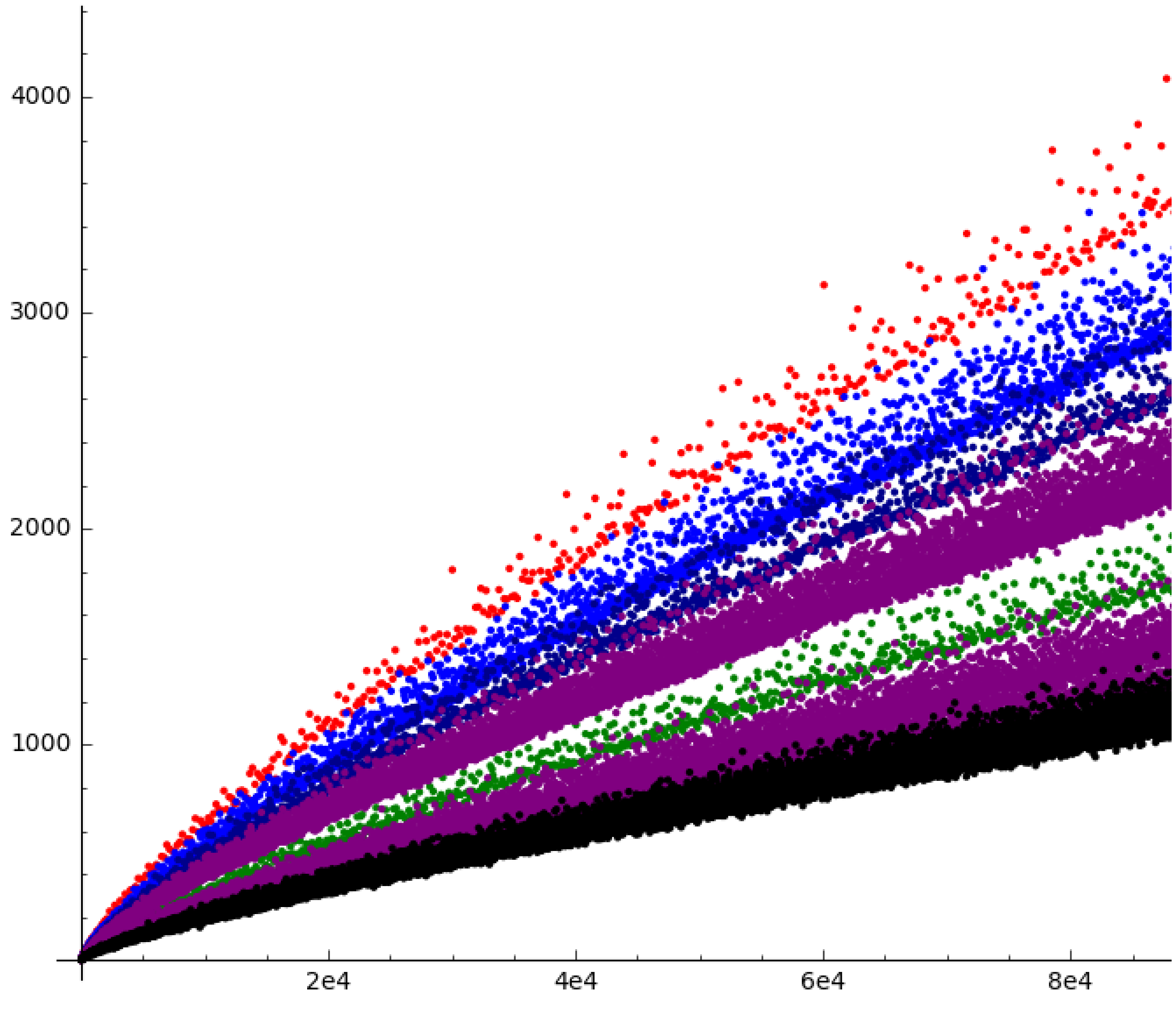
When we consider the divisibilities alone of these two $numbers$, the striations "straighten out" and become more like the results for remainders of 3.

Let's consider now whether or not numbers are divisible by the three prime factors 3, 5 and 7. Similar to our methods above, we will explore the results by looking at what striations are built from the filtering.

```
u0=[]
u1=[]
u2=[]
u3=[]
u4=[]
u5=[]
for s in R:
    if s[0]%7==0 and s[0]%5==0 and s[0]%3==0:
        u0.append(s)
    elif s[0]%7==0 and s[0]%5==0:
        u1.append(s)
    elif s[0]%5==0 and s[0]%3==0:
        u2.append(s)
    elif s[0]%7==0 and s[0]%3==0:
        u3.append(s)
    elif s[0]%7==0 or s[0]%5==0 or s[0]%3==0:
        u4.append(s)
    else:
        u5.append(s)
```

```
A=list_plot(u0,color='red')
B=list_plot(u1,color='green')
C=list_plot(u2,color='blue')
D=list_plot(u3,color='darkblue')
E=list_plot(u4,color='purple')
F=list_plot(u5,color='black')
```

```
show(A+B+C+D+E+F)
```



Like before, the striations come out somewhat normalized, and there is a clear preference for numbers being divisible by all three primes. However, unlike the other results, we have falsely projected a number's divisibility by 7| and 5| (excluding 3|) to be represented more than numbers that are divisible by 7| and 3| (excluding 5|). In fact, the numbers belonging to the first set of divisibilities actually only show in the bottom group of striations; clearly something more involved is at work.

Hardy-Littlewood Findings

Suppose you added two prime numbers, $p1$ and $p2$, such that $p1 \equiv 1 \pmod 3$ and $p2 \equiv 2 \pmod 3$. Then the principles of modular arithmetic show that $(p1+p2) \equiv 0 \pmod 3$, meaning 3 divides the sum of primes $(p1+p2)$. But how many different ways can we add arbitrary primes such that their sums are similarly divisible?

Assume that all primes are equally split between their viable residue classes under some modulo m. For now, let $m=3$; this means that the prime numbers are equally split between the residue classes 1 and 2, as a residue class of 0 is not viable for a prime number greater than 3.

Now consider the sums of these prime numbers:

$p1$ and $p2 \equiv 1 \pmod 3$, then $(p1+p2) \equiv 2 \pmod 3$;

$p1 \equiv 1 \pmod 3$ and $p2 \equiv 2 \pmod 3$, then $(p1+p2) \equiv 0 \pmod 3$;

$p1 \equiv 2 \pmod 3$ and $p2 \equiv 1 \pmod 3$, then $(p1+p2) \equiv 0 \pmod 3$;

$p1$ and $p2 \equiv 2 \pmod 3$, then $(p1+p2) \equiv 1 \pmod 3$;

It appears that sums under the residue class of 0 are double-represented in our data. This explains why there were more representations for numbers divisible by 3, and provides support toward a metric for our heuristical analysis.

Let's now consider the sums of primes with the residue classes of 5:

Suppose $p1 \equiv 1 \pmod 5$, then $p2 \equiv 1/2/3/4 \pmod 5$ yields $2/3/4/0 \pmod 5$

Suppose $p1 \equiv 2 \pmod 5$, then $p2 \equiv 1/2/3/4 \pmod 5$ yields $3/4/0/1 \pmod 5$

Suppose $p1 \equiv 3 \pmod 5$, then $p2 \equiv 1/2/3/4 \pmod 5$ yields $4/0/1/2 \pmod 5$

Suppose $p1 \equiv 4 \pmod 5$, then $p2 \equiv 1/2/3/4 \pmod 5$ yields $0/1/2/3 \pmod 5$

It appears that sums under the residue class of 0 are represented by 4 combinations, while all of the other residue classes are represented by 3. Therefore, numbers divisible by 5 in our data should be over-represented by a factor of $4/3$. One very crucial point needs to now be made. What should we expect out of numbers that are both divisible by 3 *and* 5? Will there be 2 or $4/3$ as many representations in our data? Think back to the observation we made while considering the two large striations when investigating the divisibility by five; it appears heuristically that the metric involved operates under some multiplicative principle.

Let's now define our heuristic in code. The Hardy-Littlewood correction factor is shown below:

```
def HL(n):
    m=2*n
    factor_list=list(m.factor())  # this list of factors/powers *will* include
2, so we can know to ignore it!
    hl=1
    for i in factor_list[1:]:
        p=i[0]
        hl=hl*(p-1)/(p-2)
    return(hl)
```

The Hardy-Littlewood correction factor will allow us to remove the striations from our data, as it "corrects" for the over-representations of numbers highlighted above. The factor can be expressed as follows:

$$HL(n) = \prod_{p|n}(p-1)/(p-2)$$

Another way of expressing the factor is as the product of the proportions whereby a number is over-represented by the sum of prime numbers.

```
# factors are 3
HL(3)
```

```
    2
```

```
# factors are 5
HL(5)
```

```
    4/3
```

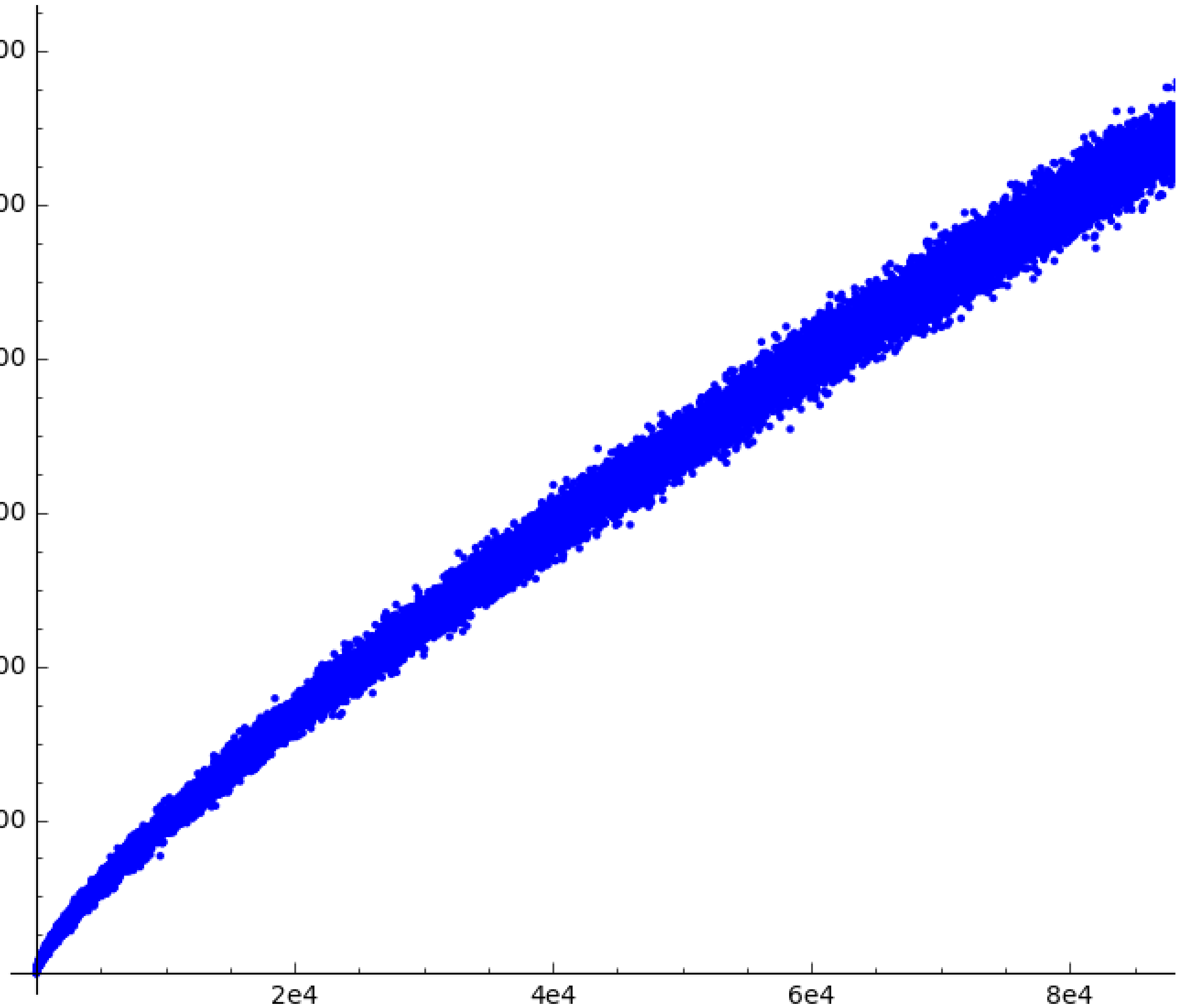
```
# factors are 3 and 5
HL(15)
```

```
    8/3
```

Using our metric, let's now remove the striations from the data.

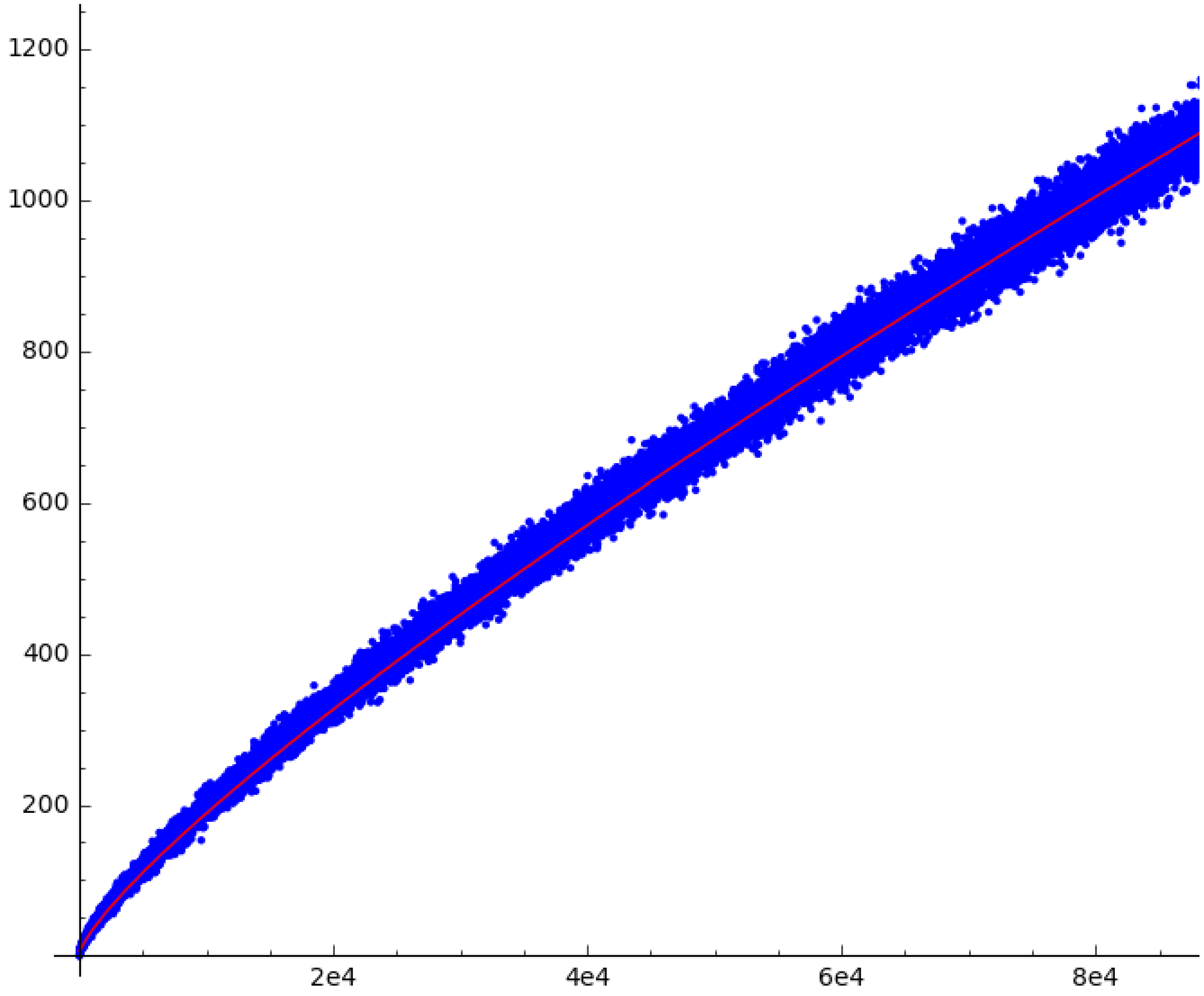
```
hl_adjusted=[0]
for s in R:
    hl_adjusted.append([s[0],s[1]/HL(s[0])])
```

```
list_plot(hl_adjusted)
```



```
P = plot(2*4/5*x/log(x)^2,x,2,N,color='red')
```

```
show(list_plot(hl_adjusted) + P)
```



As you can see, the growth of the representations are clearly defined by some function close to $x/log(x)$.

Investigating Twin Prime Goldbach

The Twin Prime Goldbach conjecture can be given as follows: every even whole number greater than some value C can be written as the sum of twin prime cores, where the cores are defined as k for twin primes of the form $p = 6k \pm 1$.

Let's start by going through our list of primes and finding the cores of each twin prime pair.

```
N=100000
P=[]
for i in srange(3,N):
    if is_prime(i):
        P.append(i)
```

```
TP_cores=[]
for i in srange(2,len(P)):
    if P[i]-P[i-1]==2:
        TP_cores.append((P[i]-1)/6)
```

```
TP_cores[:10]
```

```
    [1, 2, 3, 5, 7, 10, 12, 17, 18, 23]
```

As you can see from above, the first few cores are $1, 2, 3, 5, \ldots$.

Let's take a moment to calculate what these twin prime pairings are:

For $k = 1$, the twin primes are $p_1 = 6 * 1 - 1 = 5$ and $p_2 = 6 * 1 + 1 = 7$;

For $k = 1$, the twin primes are $p_1 = 6 * 2 - 1 = 11$ and $p_2 = 6 * 2 + 1 = 13$;

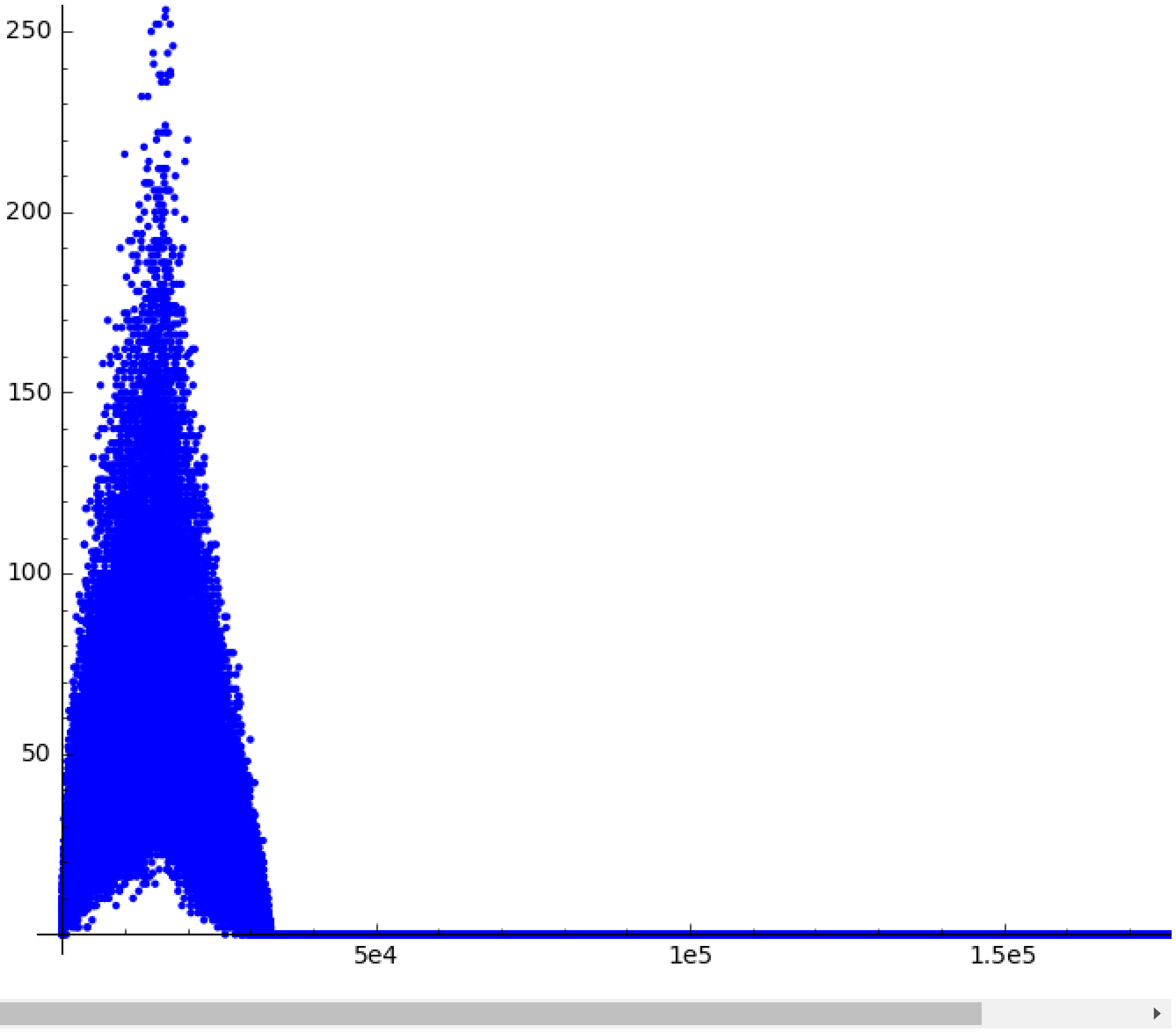
For $k = 1$, the twin primes are $p_1 = 6 * 3 - 1 = 17$ and $p_2 = 6 * 3 + 1 = 19$;

For $k = 1$, the twin primes are $p_1 = 6 * 5 - 1 = 29$ and $p_2 = 6 * 5 + 1 = 31$;

It looks like our code sufficiently finds the cores as expressed above. Now let's plot the representations of the sums of our twin prime cores.

```
Q = [0 for i in srange(2*N)]
for c in TP_cores:
    for d in TP_cores:
        Q[int(c)+int(d)] += 1
```

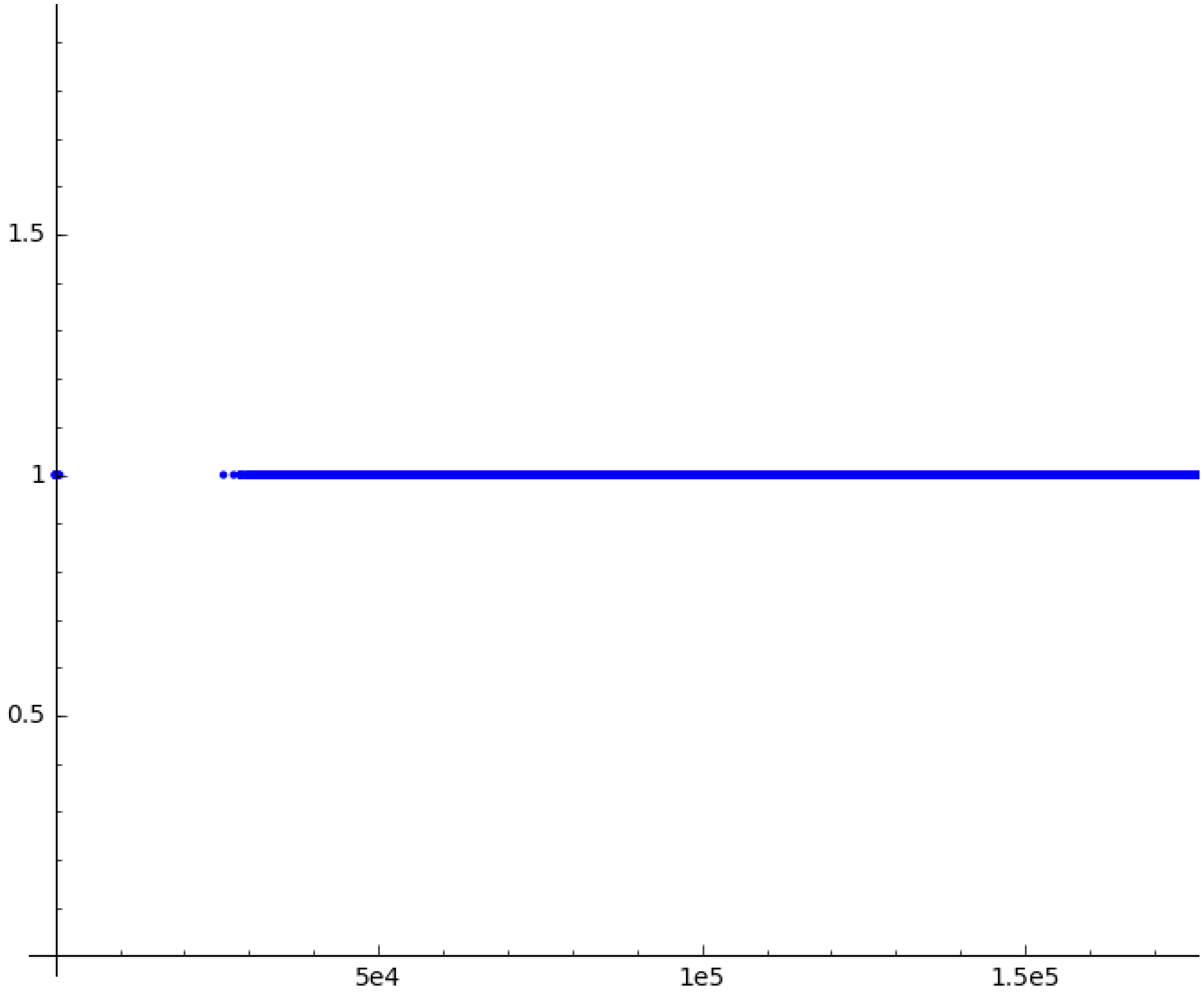
```
list_plot(Q)
```



Like our investigations for the Goldbach conjecture, we will want to remove the zeros and only consider the representations up to N. However, let's first look at the zeros in our plot to try and determine after what value our cores can start sufficiently representing numbers.

```
zeros = [[i,1] for i in srange(2*N) if Q[i] == 0]
```

```
list_plot(zeros)
```



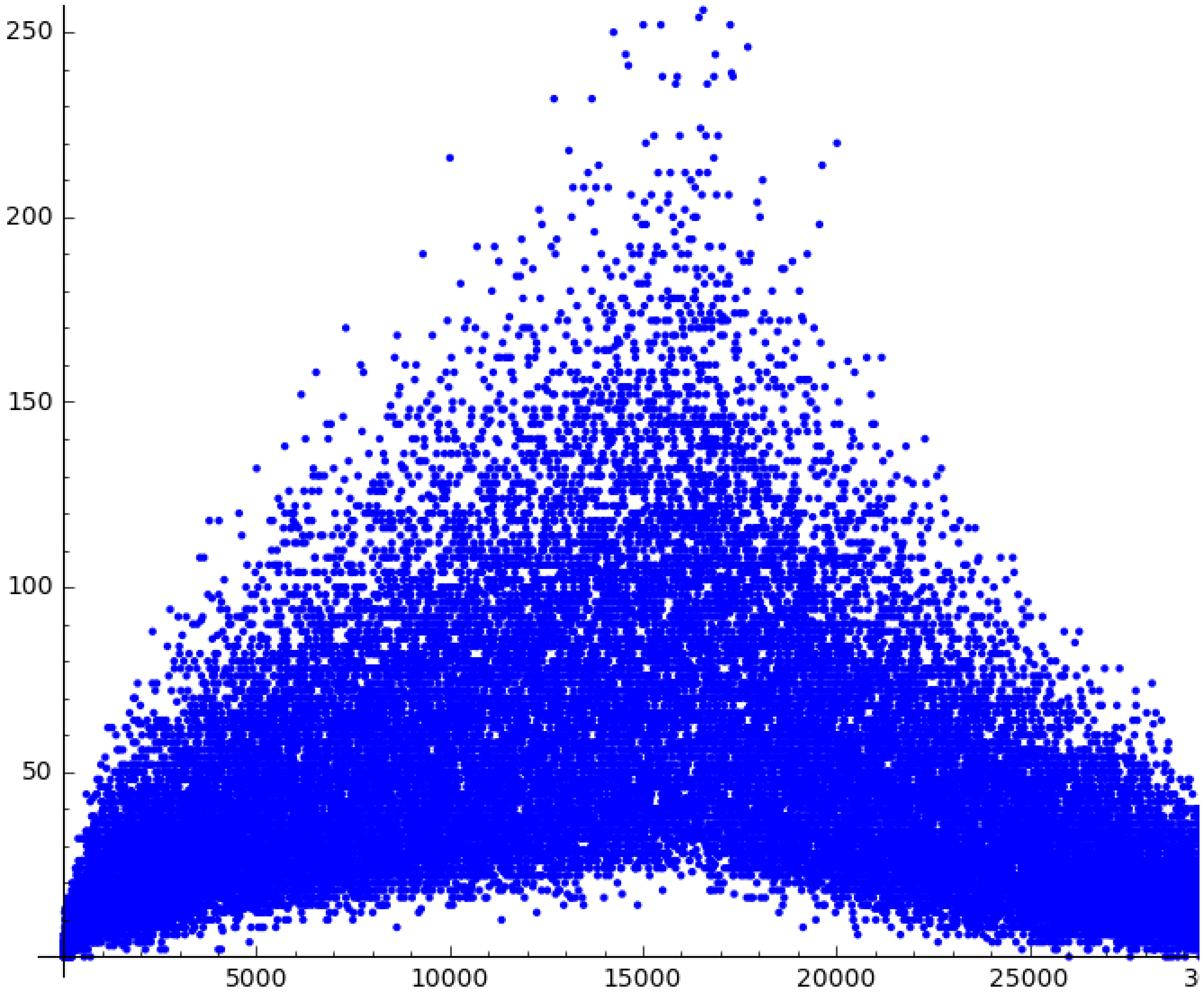
Keep in mind that our cores are only calculated for up to $N = 100000$, thus the sums of our cores should not be expected to to surpase $N/6$, or roughly 16000.

Right now our representations are including too many numbers that surpass the expectations for our data. Let's go back through and fix this.

```
M = TP_cores[-1]
```

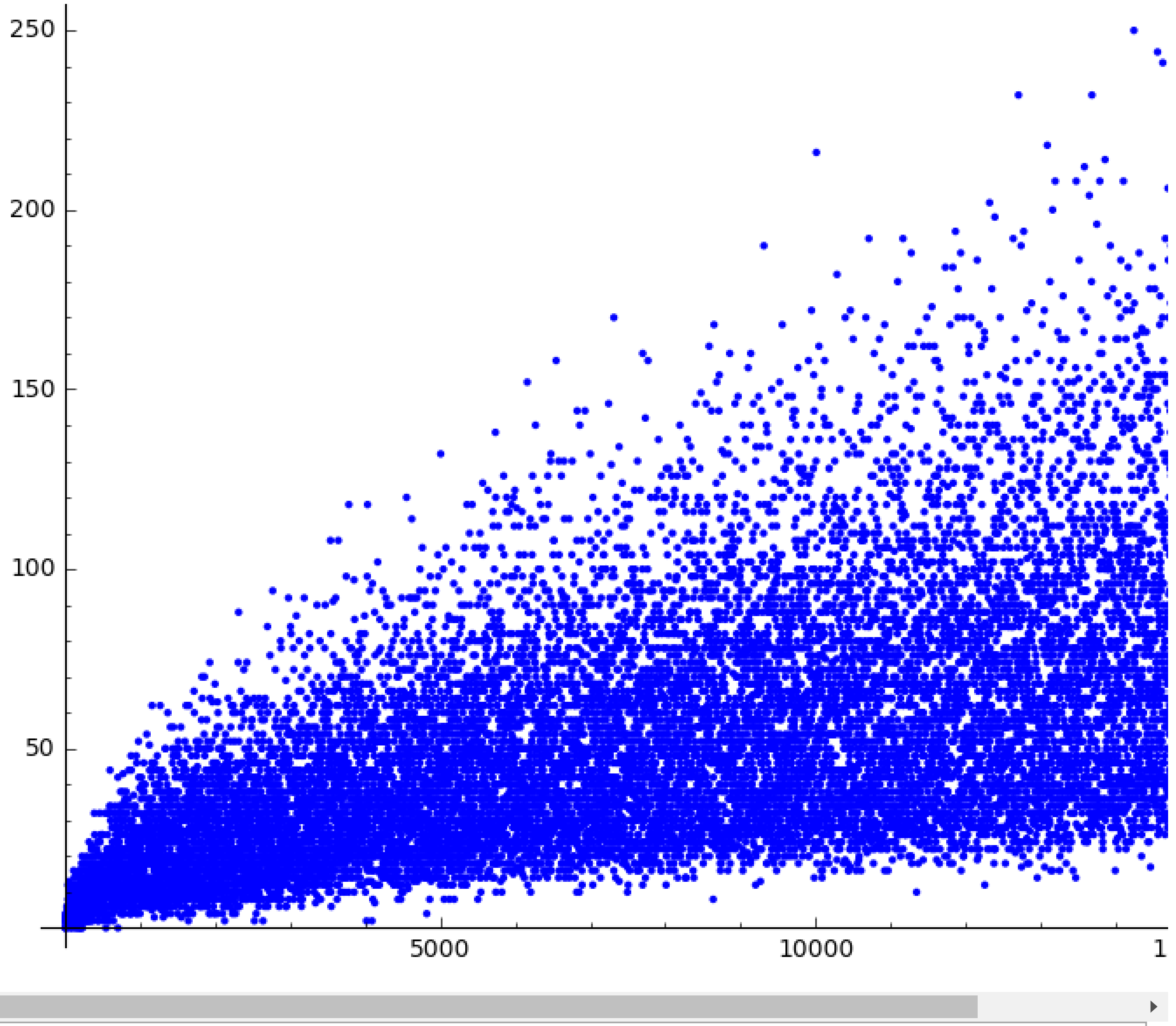
```
Q = Q[:2*M]
```

```
list_plot(Q)
```



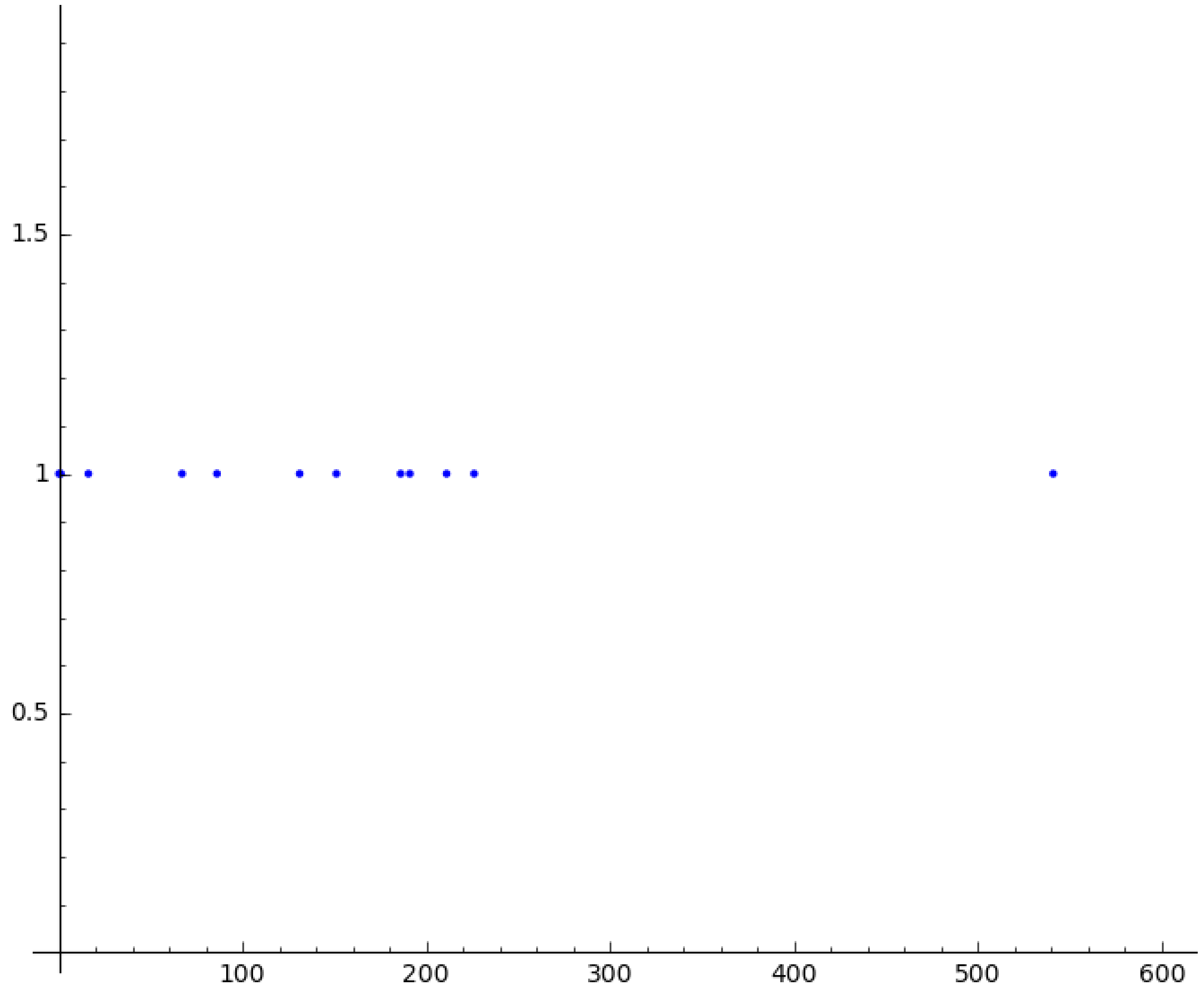
```
Q = Q[:M]
```

```
list_plot(Q)
```



```
zeros = [[i,1] for i in srange(M) if Q[i] == 0]
```

```
list_plot(zeros)
```



Now when we calculate the zeros, we correctly capture the beginning portion from the previous graph. It looks like for $N = 10000$, the last number that our cores cannot represent is $C = 700$.

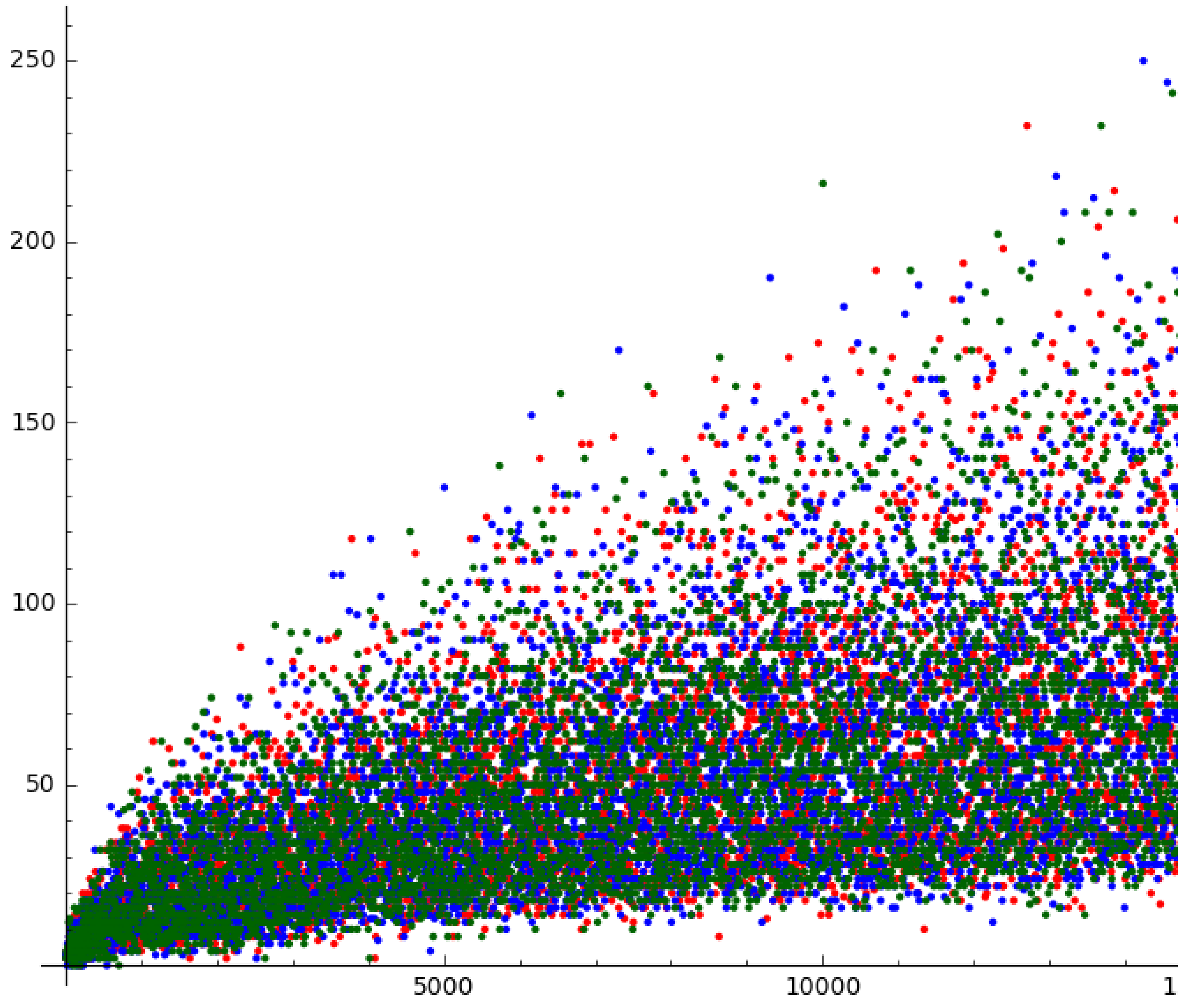
The question follows that if our Twin Prime Goldbach is similar in representations to our Goldbach conjecture, then if similar trends exist within the striations of the core representatinos.

We will see that this is not necessarily the case. Take for example, divisibility by 3. In our investigations into the Goldbach, we saw clear striations formed when we highlighted the different residues of 3. Let's see what happens for our core sums.

```
u0=[]
u1=[]
u2=[]
for i in srange(3,M):
    if i%3==0:
        u0.append([i,Q[i]])
    elif i%3==1:
        u1.append([i,Q[i]])
    else:
        u2.append([i,Q[i]])
```

```
A=list_plot(u0,color='red')
B=list_plot(u1,color='blue')
C=list_plot(u2,color='darkgreen')
```

```
show(A+B+C)
```



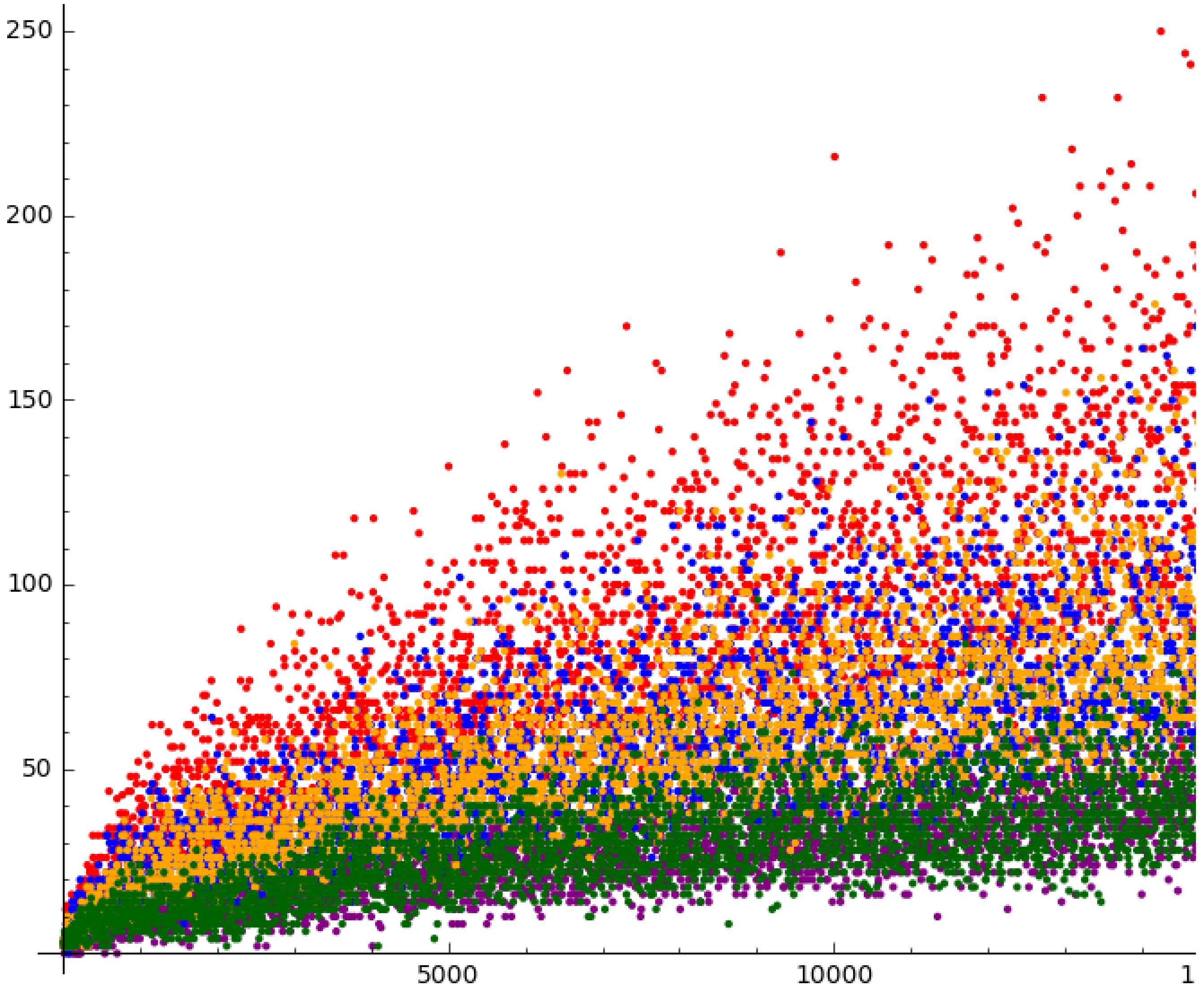
It would seem that there is no correlation between a number's divisibility by 3 and its number of representations by core sums.

Let's now consider divisibility by 5.

```
u0=[]
u1=[]
u2=[]
u3=[]
u4=[]
for i in srange(3,M):
    if i%5==0:
        u0.append([i,Q[i]])
    elif i%5==1:
        u1.append([i,Q[i]])
    elif i%5==2:
        u2.append([i,Q[i]])
    elif i%5==3:
        u3.append([i,Q[i]])
    else:
        u4.append([i,Q[i]])
```

```
A=list_plot(u0,color='red')
B=list_plot(u1,color='purple')
C=list_plot(u2,color='blue')
D=list_plot(u3,color='orange')
E=list_plot(u4,color='darkgreen')
```

```
show(A+B+C+D+E)
```



Interesting. It looks like divisibility by 5 is favored by our core sums, whereas divisibility by 3 is not. Could there be a reason for this?

One thing that I found during my research into this trend was the following property of twin prime cores:

"Except for a(1)=1, all numbers in this sequence are congruent to (0, 2 or 3) mod 5." - https://oeis.org/A002822

In otherwords, none of our prime cores are congruent to 1 or 4 mod 5. Consider now the sum of any two prime cores. Just like in our reasoning above, we can clearly see how sums with a residue close to 0 mod 5 are favored:

Suppose $p1 \equiv 0 \pmod{5}$, then $p2 \equiv 0/2/3 \pmod{5}$ yields $0/2/3 \pmod{5}$

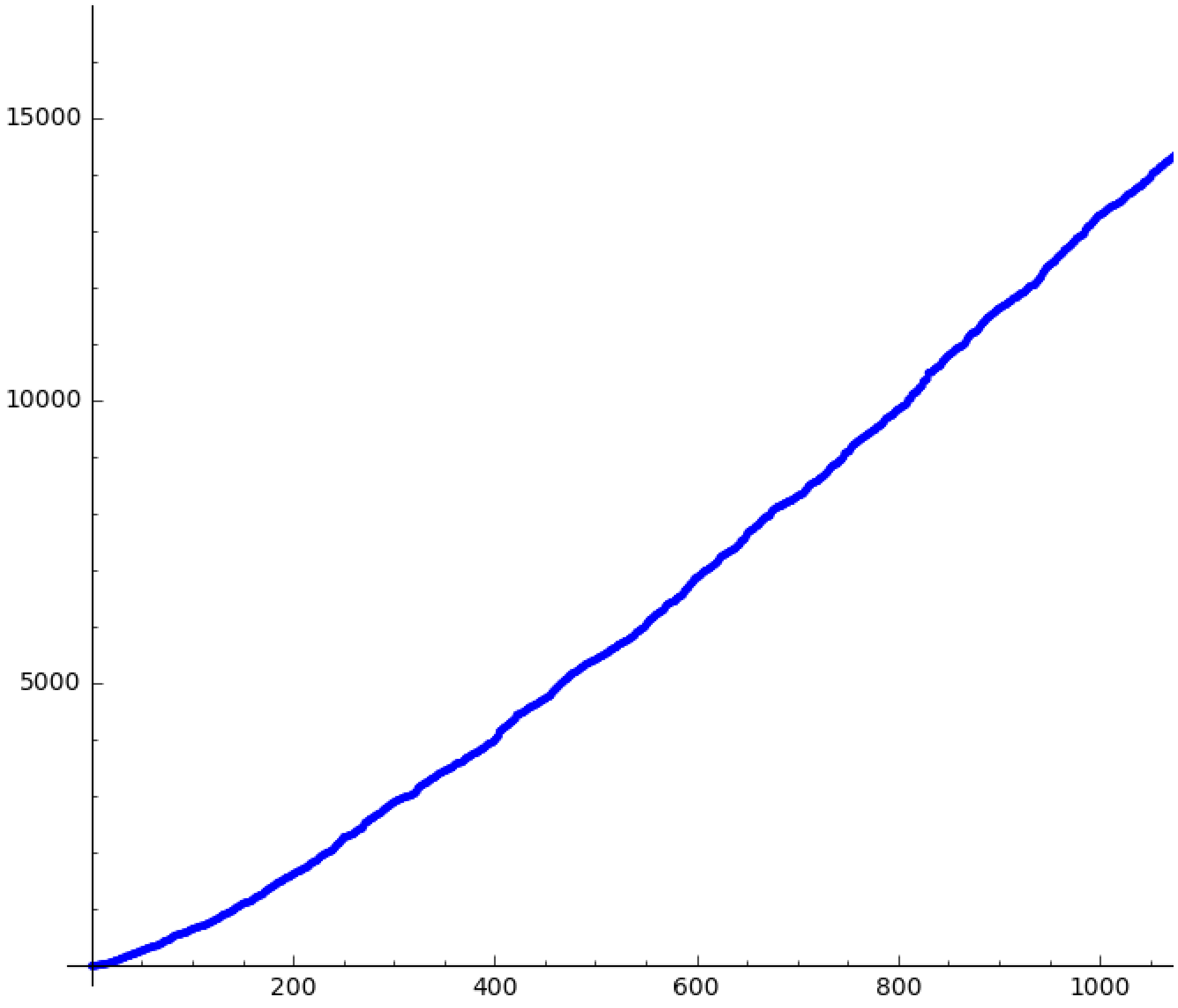
Suppose $p1 \equiv 2 \pmod{5}$, then $p2 \equiv 0/2/3 \pmod{5}$ yields $2/4/0 \pmod{5}$

Suppose $p1 \equiv 3 \pmod{5}$, then $p2 \equiv 0/2/3 \pmod{5}$ yields $3/0/1 \pmod{5}$

Therefore, sums with residues 0 are favored three times more than residues 1 and 4, and $3/2$ as much as 2 and 3.

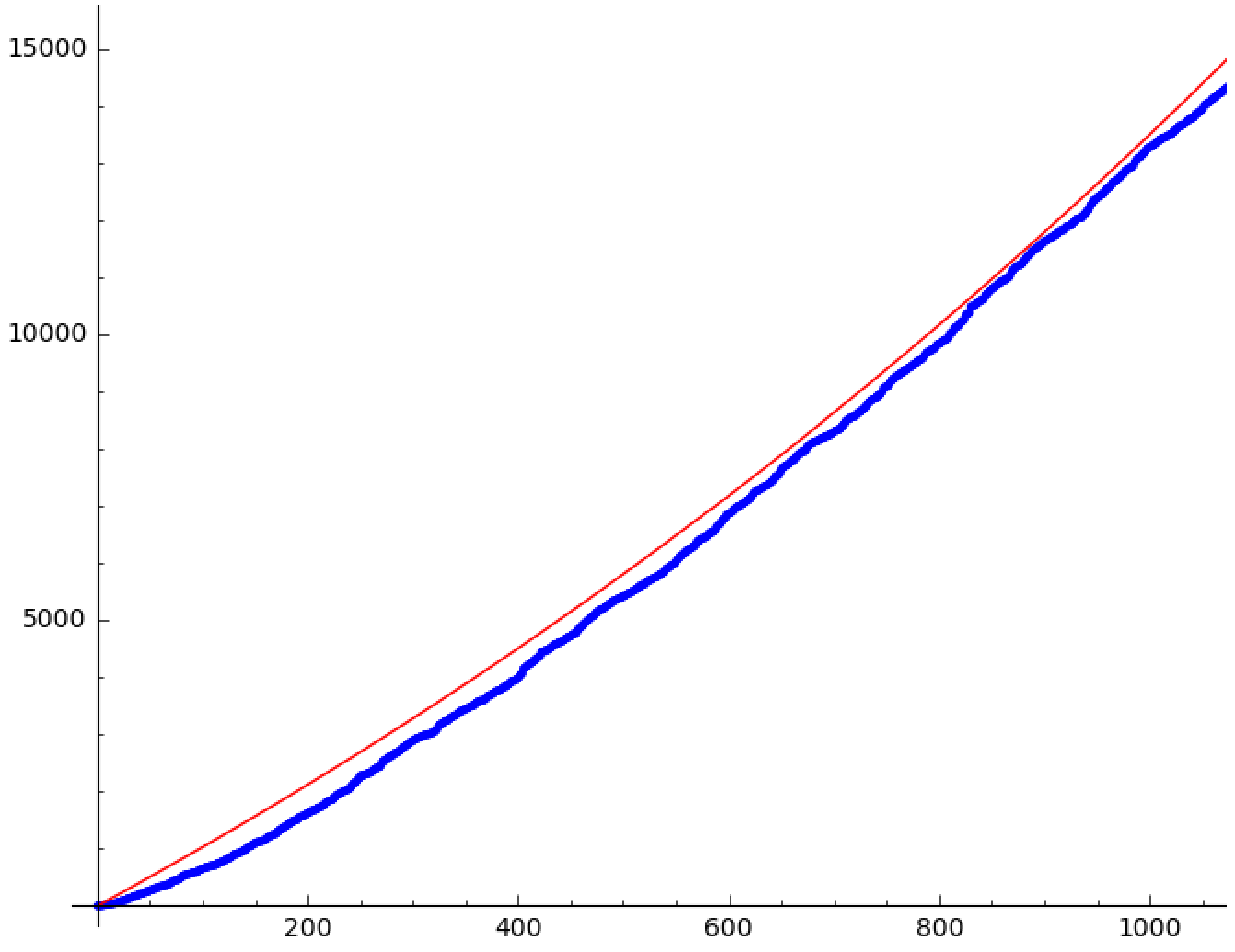
Now let's see if we can graph the growth of our cores, as we did in the Goldbach.

```
list_plot(TP_cores)
```



It might seem like our cores grow linearly, but keep in mind that the vertical growth here is much more than the horizontal growth. This means that our function is actually exponential:

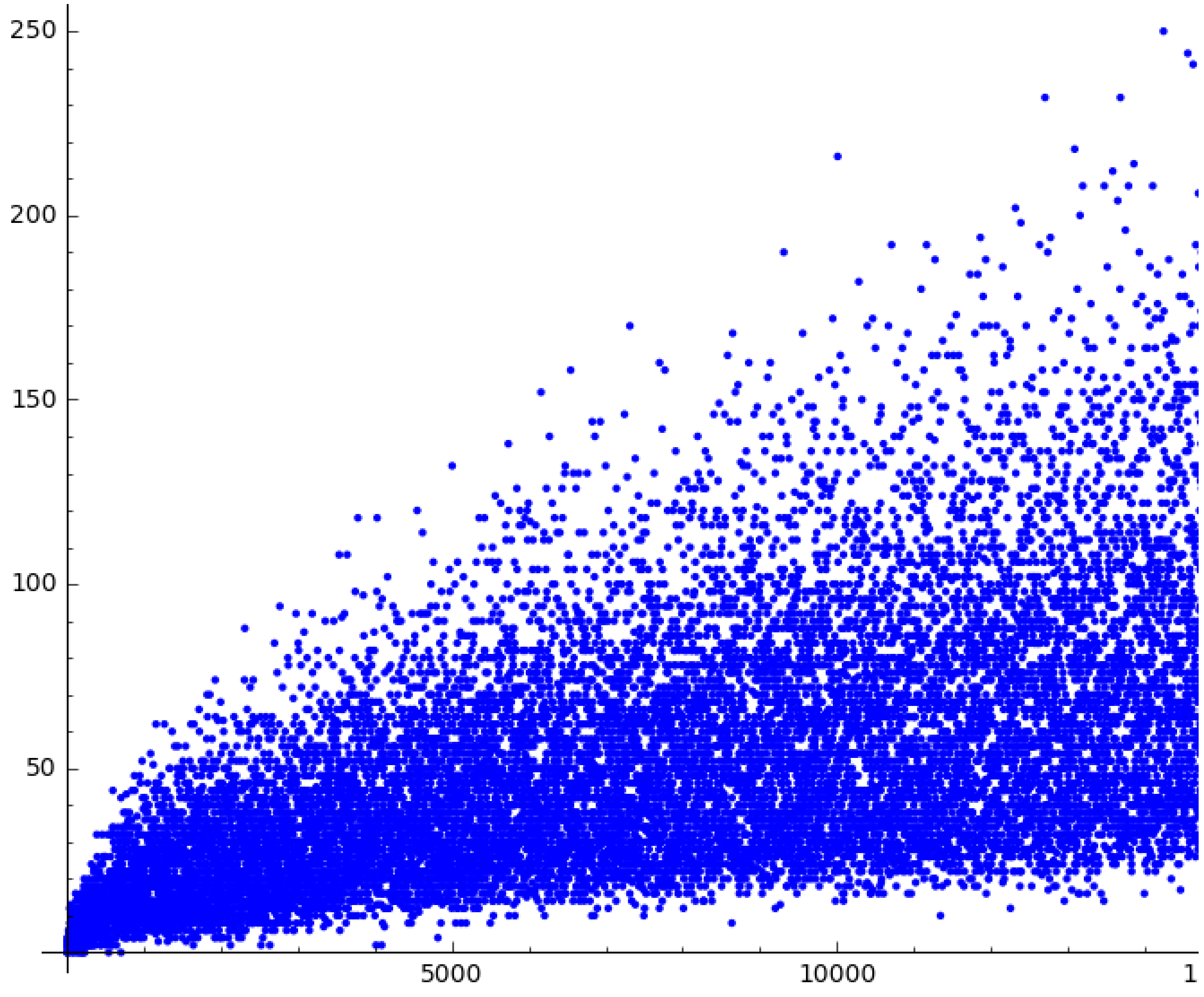
```
A = list_plot(TP_cores)
B = plot(10*x*1.0003^x,x,2,1200,color='red')
show(A+B)
```



There's one more thing that we can consider within our twin prime core sums. This is whether or not the cores themselves are represented in our data.

If k_n is some twin prime core as defined above, then k_n is a super core if the number of representations of k_n given by $k_n = k_i + k_j$ for $i, j < n$ is itself some twin prime core k_m.

```
list_plot(Q)
```



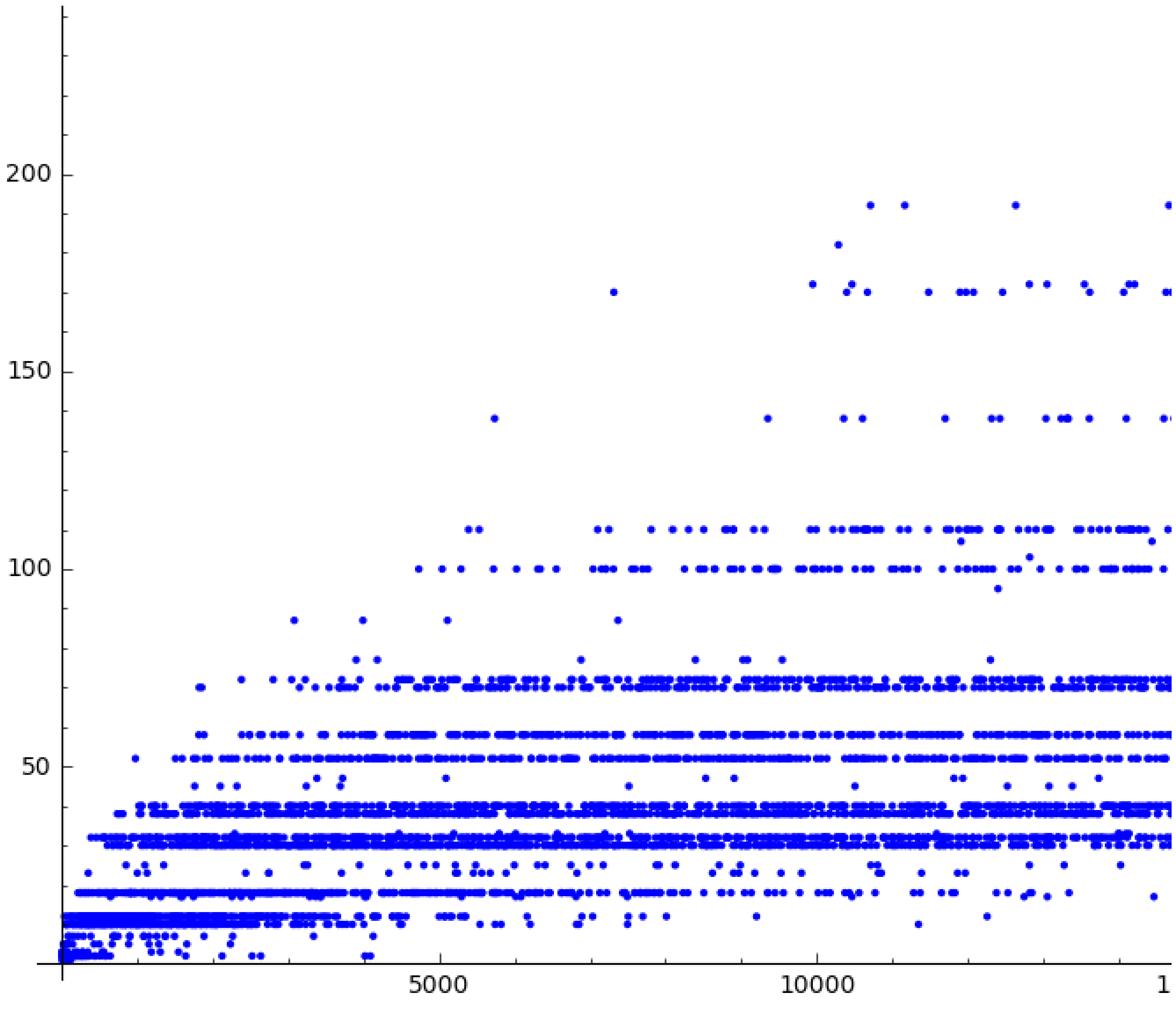
```
WQ = [[i,Q[i]] for i in srange(0,len(Q)) if is_prime(6*Q[i]-1) and
is_prime(6*Q[i]+1)]
```

```
QQ = [[i,Q[i]] for i in srange(0,len(Q)) if is_prime(6*Q[i]-1) and
is_prime(6*Q[i]+1) and is_prime(6*i-1) and is_prime(6*i+1)]
```

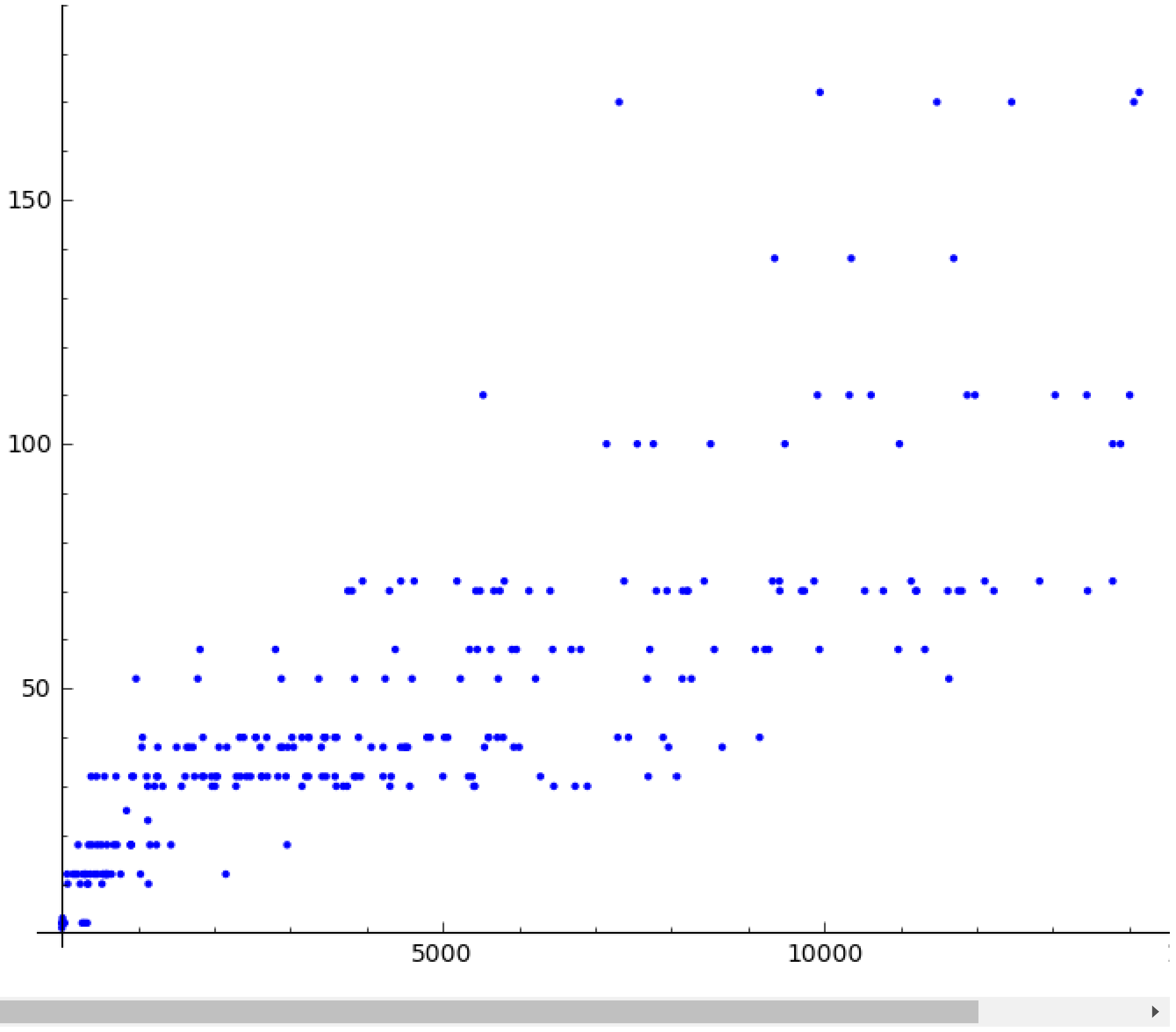
```
WQ[:12]
```

```
    [[2, 1],
     [3, 2],
     [4, 3],
     [5, 2],
     [6, 3],
     [7, 2],
     [9, 2],
     [10, 3],
     [11, 2],
     [14, 3],
     [18, 2],
     [20, 5]]
```

```
list_plot(WQ)
```



```
list_plot(QQ)
```



Conclusion