

Pathways from Fermat's Last Theorem

Brittany Wagoner

May 5, 2009

Theorem 1. *From Fermat's Last Theorem the equation:*

$$a^n + b^n = c^n$$

for integers a, and b, has no integer solutions for $n > 2$.

Abstract

Every mathematician wants to either prove something or prove someone else wrong. My initial instinct, working with Fermat's Last Theorem, was of course to find a triple where equation (1) works when $n > 2$ where a, b , and c are integers. But, there is a 150 page proof that says a triple like this is not possible. So I played with some other ideas. My work involved three paths. I first played with an equation of the form:

$$a^n + b^{n+1} = c^{n+2}$$

for different values of n . On the next path I worked with the following altered formula for Fermat's theorem, with different n and m values.

$$a^n + b^n + m = c^n$$

The last exploration of Fermat's theorem involved finding close triples. For example, on the website, MathWorld.com, I stumbled upon two close triples:

$$1782^{12} + 1841^{12} = 1922^{12}$$

$$3987^{12} + 4365^{12} = 4472^{12}$$

The first equation is close to nine digits and the second is close to ten digits. Then the combination of the second and third paths led me to a different way of finding close triples.

1 The Essentials

To start I needed some basic programs. The first program I made was called *Solve* which required an input of two integers a and b and degree n . This solved the following equation for c :

$$a^n + b^n = c^n \tag{1}$$

When $n = 1$ or $n = 2$ this returns integers. However, when $n > 2$ some more interesting solutions appeared. Generally, if n was odd then $Solve(a,b,n)$ returned one noncomplex, positive solution and the rest were complex. If n was even then the program return one noncomplex positive, one noncomplex negative, and remaining complex solutions. The solutions resulting from this function were returned in a list. I was not concerned with any solution that was complex so to return the solutions I wanted I wrote a function called *TestSolution*. This function took a list (from $Solve(a,b,n)$) and returned only the positive integer solution. Obviously when $n > 2$ this would not return an integer solution to the above equation. If there was no integer solution the program would return (0).

I put these programs together to create *Find* which involves a loop and is intended to return successful triples of Fermat's equation for degree n over a certain range. This program takes two values, x and y that represent a range from which to pick a and b and a desired value for the degree of the equation. Obviously this program will not return anything when $n > 2$ so it provided a check but it was used a model for other programs that were built later.

I put *Solve* and *TestSolution* together again to create a simple program called *CInteger* that would tell me quickly if two integers a and b and with degree n had an integer solution. This simply called the *Solve* program then the *TestSolution* and returned the value obtained from the *TestSolution* program. I used this in later programs to make sure that a triple did not exist by asking if *CInteger* returned (0). This program was simply used for convenience rather than calling two programs.

2 FindN

$$a^n + b^{n+1} = c^{n+2} \tag{2}$$

The purpose of working with the above was simply because it was an altered form of Fermat's equation. The results, however, showed nothing surprising. I wrote two programs to explore this section: *SolveN* and *FindN*. The *SolveN* functions similarly to *Solve* except it returns solutions to the above formula when two integers a,b and degree n are given. *FindN* lets us use *SolveN* for a range of numbers x to y . When $n = 0$ then our above equation becomes:

$$1 + b = c^2$$

Obviously this is a quadratic equation so we surely will receive solutions. When $n = 1$ we get the following equation:

$$a + b^2 = c^3$$

which returned plenty of triples. When $n = 2$, equation (2) becomes:

$$a^2 + b^3 = c^4$$

This returned few triples but when $n = 3$ or $n > 3$ the value of n has now made the equation too complicated and I did not return any successful triples. This makes sense because we are now getting into a more complicated equation similar to Fermat's equations when n gets larger than 2. The results that once n reached a certain value the equation (2) would not return any triples was no surprise.

3 FindM

The next path involved work with the altered equation:

$$a^n + b^n + m = c^n \tag{3}$$

The goal for this path was to see if solutions exist for this equation with $n > 2$ when m is an integer. The most exciting value to work with here for m is $m = 1$. Obviously if there is a triple to equation (3) when $m = 1$, for $n > 2$, then it seems as if this triple would not be too far off if used in the equation (1).

The work for this path is similar to the work done for path one however it has much more exciting results. I did some slight editing to the program *Solve* and made *SolveM* which uses the equation (3) but requires an additional input of a value for m . I then edited *FindN* to make *FindM* which uses *SolveM* and *TestSolution* (similar to the original *Find*). It requires a range of values for a and b , a value of the degree for the equation and the additional value of m . This program returns more than just a triple however. To keep computations clear, the program will return a sentence *For degree n (value of n) and when m equals (value of m), the following triples work* and it will list any successful triples, if any. The triples will be returned as lists within a set. If there are no triples that work then an empty set will be returned.

This program was hard-working and it did not run quickly. To start exploring with *FindM* I worked with an interval from 1 to 100 and started with degree, $n = 3$. First I ran the program with $m = 1$; I was very surprised and excited to see that the program did return one triple on this interval:

- [6,8,9]

I then performed several trials where I increased the value of m but kept my interval of 1 to 100 at degree $n = 3$. I ran the program with following values of m and had the following results:

- $m = 2$, [5,6,7],[24,47,49]
- $m = 3$, empty set
- $m = 4$, empty set
- $m = 5$, empty set

At this point I began to wonder if there are only solutions (at least on this interval) for $m < 3$ but then I tried more values of m :

- $m = 6$, [1,1,2],[43,58,65]
- $m = 7$, empty set

After this I had a thought that maybe it only worked for values of $m < 3$ or values of m that were multiples of 3. Then I computed *FindM* for $m = 8$:

- $m = 8$, [12,16,18],[17,40,41]

From these computations I do not think that there is a relationship for the value of m and whether or not there are triples that exist for equation (3) at that value of m . Granted this is based on a small interval so there could be triples for the other values of m but this interval also showed me that existing triples do not depend on m . In addition, based on the above results there does not seem to be a relationship between m and the number of triples that did exist on the interval either. To see that there were triples beyond 100 I ran (a very long program) *FindM* on an interval of 1 to 300 with $n = 3$ and $m = 1$. Maple returned the following triples:

- [6,8,9],[71,138,144],[135,138,172]

Obviously I wanted to test this program for degrees greater than $n = 3$. I did similar tests with a range from 1 to 100 and similar m values but with degree $n = 4$. The results for these tests, however, were slightly disappointing. After running several tests with several different intervals and several different values of m , I did not return any successful triples for equation (3) degree, $n = 4$. I used the small interval first so that it was easy to check several different values of m . I ran individual tests for m from 1 to 4. After returning empty sets I tried a loop that ran for values of m from 4 to 8 (using the same interval) and *FindM* returned more empty sets. Slightly disappointed I tried one big run with an interval from 1 to 500 and $m = 1$ and an empty set was returned. Sadly, I do not think equation (3) works for $n > 4$. I would like to run this program with other, higher values of n to confirm this conjecture.

4 Close

As I mentioned earlier I found two, supposedly correct, triples for $n = 12$ on MathWorld.com. The website said these were only correct to a certain number of digits. I thought this was an interesting way of looking at and measuring triples. Using the triples from MathWorld, I build some extensive programs.

First I looked at the triple

$$1782^{12} + 1841^{12} = 1922^{12} \tag{4}$$

and I computed the left and right side to see that they were really close to nine digits. They are extremely large numbers but their first nine digits do match and they are:

- [2, 5, 4, 1, 2, 1, 0, 2, 5]

I used this information to help construct and check a program called *Close1*. This program takes two integers, a and b , and degree n and returns an integer value to equation (1) for c as well as the number of matching digits and the digits that match.

To construct this program I had to go through a few steps. I knew I needed an integer for the value of c to find matching digit results but where did that integer come from? First I discovered where the number 1922 in equation (4) came from. If I plug 1782 and 1841 into *Solve* I will get back the solutions for c for equation (1) when $n = 12$. Obviously, my results are going to be mostly complex values because $n > 2$. There is, however, one nonnegative, noncomplex solution and if I truncate that one solution I get the value 1922. I checked this with the other triple that MathWorld.com demonstrated:

$$3987^{12} + 4365^{12} = 4472^{12} \tag{5}$$

Completing the same steps I also returned 4472 as a truncated solution of 3987 and 4365 when $n = 12$. So I figured that the c value was the truncated nonnegative, noncomplex solution. I then wrote another program *ReturnC* that would easily compute and return the solution to be truncated when I would input two integers and a degree. This program simply calls the *Solve* program with the given inputs. It evaluates the solutions and then looks for and returns the nonnegative, noncomplex solution in decimal (*evalf*) form. Then I realized that in order to compare numbers (left side of equation (1) to right side of) I would need to turn these numbers into lists so that I could see to what digits they are the same. The next program *ToList2* takes a number and returns that number as a list of its digits. From here, I was able to go on and write *Close1*. This program goes through many steps.

First it takes the two numbers and runs *CInteger* to first make sure that the given numbers do not actually result in a successful triple. The program continues if there is no integer solution or it will return the sentence, *The triple [a,b,c] works when n is* and it will give the value of the degree as well. Otherwise, the program will run *ReturnC* and give the nonnegative, noncomplex solution and truncate it, I will call this s to match it's name in the program. The program then computes two equations:

$$\begin{aligned} a^n + b^n \\ s^n \end{aligned}$$

The numerical values of each are thrown into *ToList2*. The lists are then thrown into a loop that compares corresponding values of each list until it reaches a position with non matching digits. The matching digits are then put into another list. The program returns *The triple [a,b,s] is close to* and gives the number of digits and the list of the digits. I checked this solution with equations (4) and (5) and returned the correct nine and ten digits respectively.

My initial plan with *Close1* was to run a similar program to the *Find* programs that would return several triples and matching digits based on a given range. I was able to construct a program called *CloseTriples*. One inputs a range from x to y from which a and b are chosen and a degree n . *CloseTriples* runs a loop based on this range and does a lot of computations of *Close1*. The program then returns each output of *Close1* that was computed. This program returns a lot of interesting information but it is currently very difficult to work with. My future work on this Fermat project would revolve around this Program. My initial intent was to use this program to return triples that are significantly close; In other words were close to several digits. I do not care about the triples that are close to 1 or 2 digits. I am currently stuck here, however, because *Close1* returns a lot of information: it returns the triple and the matching digits. I would like *CloseTriples* to return the values from *Close1* with long lists but I have to be careful when programming this not to lose the values of the triple that resulted in the long lists. In the mean time, it is still interesting to input any range and see all the *close* triples to come out of that range.

I wanted to see if, given an appropriate range, *CloseTriples* would return the triple [1782,1841,1922] and check that it was correct to nine digits. I ran the program *CloseTriples*(1781,1842,12). I actually had to interrupt the computation because it was taking so long and returning A LOT of information. However, I did find the result *The triple [1782,1841,1922] is close to 9 digits [2,5,4,1,2,1,0,2,5]!* I also discovered another triple that is close to five digits, *The triple [1784,1804,1901] is close to 5 digits [2,2,2,7,3]* which is the second largest number of matching digits after the previous triple (and before I interrupted the computation).

5 Secret Path

In the midst of *FindM* and *Close1* I stumbled upon some interesting observations. Since I did get some triples that worked for $m = 1$ I kept thinking "Those are really close if adding 1 makes such a difference". Then I thought to see how close they were based on matching digits. Since *FindM* returns triples, I thought I would write another version of *Close1* to see how close these were. I wrote *Close2* that requires three given integers (a, b , and c) and degree n . This does similar computations to *Close1* and returns *these are close to number of digits out of total digits in number* and then returns

the list of matching digits. I noticed that the number of matching digits were significant (a majority of the total number of digits). Then I wondered if the triple that *FindM* returned was a closer triple than what I would get if I put only the values of a and b into *Close1*. Recall that *Close1* uses a solution of the given integers as c . The resulting lists of *Close1* were smaller than those of *Close2*. So, solutions to equation (3) can result in *close triples*. Referring to the computations and results listed in the *FindM* section above, I discovered the following (I will show the higher valued triples because the significance of matching digits is easier to see):

- $m = 1$, *Close1*(71,138,3): Close to 2 digits
- $m = 1$, *Close2*(71,138,144,3): Close to 6 digits
- $m = 2$, *Close1*(24,47,3): Close to 2 digits
- $m = 2$, *Close2*(24,47,49,3): Close to 5 digits
- $m = 3$, *Close1*(43,58,3): Close to 1 digits
- $m = 3$, *Close2*(43,58,65,3): Close to 4 digits

The other thing to notice about the values of c for the above values that *FindM* returned, is that they are very close to the truncated solution that *ReturnC* returns for the given values of a and b .

- *ReturnC*(71,138,3)=143, compared to 144 above.
- *ReturnC*(24,47,3)=48, compared to 49 above.
- *ReturnC*(43,58,3)=64, compared to 65 above.

Recall that truncating the solution came from my discovery of how the value for (c) for the equations that came from MathWorld was found.

As I stated above I would like to do more work with *CloseTriples* so that I can accomplish my initial goal but I would also like to find out more about equation (3) and if there are solutions when $m > 3$.

References

- [1] Eric W. "Fermat's Last Theorem." From MathWorld, A Wolfram Web Resource. <http://mathworld.wolfram.com/FermatsLastTheorem.html>.