MATH 356 Number Theory

Python Worksheet: Elliptic Curves, the Sieve of Eratosthenes, and Fermat Numbers

1. We want to find non-trivial rational solutions to $y^2 = x^3 - 2$. In this case, "non-trivial" means "other than $(3, \pm 5)$."

   (a) Begin by graphing this curve:

   ```
   from sympy import *
   x=symbols('x')
   plot(sqrt(x**3-2),-sqrt(x**3-2))
   ```

   (I'm plotting the two halves separately because I'm working around another problem...)

   I had to execute that block twice in order for the graph to appear. Perhaps our class experts can explain why...

   (b) Implicitly differentiate $y^2 = x^3 - 2$ with respect to x. (Remember that y is a function of x and use the Chain Rule.) Do this by hand.

   (c) Find an equation of the tangent line to this curve (by hand).

   (d) Add your tangent line to the graph by making the function you found the third item in the list of curves.

   (e) Now solve for the second point of intersection of the line with the elliptic curve:

   ```
   solve(y**2-x**3+2,x)
   ```

   with y replaced by what you found for the tangent line.

   (f) With that value of x, find the corresponding value of y (again using your tangent line). Is (x, y) a rational point?

   (g) Now repeat this process starting with your new point: find an equation of the tangent line and determine where it meets the elliptic curve again. Is it a rational point also?

2. Let's implement the Sieve of Eratosthenes in Python. This requires us to find divisors of n, but we only need to search up to the square root of n.

   (a) Create the prime-testing routine.

   `import math` (We need the floor function and the square root function.)

   `def isprime(n):` (We are naming our function "isprime.")

   `factors=[]` (This sets an empty list named "factors.")

   `for i in range(1,math.floor(math.sqrt(n))+1):` (Loop: $1 \ldots \sqrt{n}$.)

   `if n%i==0:` (Checks to see whether $i|n$.)

   `factors.append(i)` (If so, it adds i to the list of factors.)

   `if len(factors)==1:` (Checks to see if there was more than 1 factor.)

   `return print(n, "is prime",factors)` (If not, n is prime!)

   `else:`

   `return print(n, "is not prime",factors)` (Otherwise, n is not prime.)

(b) Now use Python to find all the primes from 1 to 1000. The simplest way is to write a loop that tells you in each case. But you could instead modify isprime or write a separate bit that just keeps track of the ones that are prime.

3. We now look into Fermat numbers, which are numbers of the form $F_n = 2^{2^n} + 1$.

   (a) Begin by defining a function that calculates Fermat numbers:

   ```
   def F(n):     return 2**(2**n)+1
   ```

   (b) Find F_1, F_2, F_3, and F_4.

   (c) Use isprime to explore whether the Fermat numbers are always prime.