Project: Primality of Fermat and Mersenne Numbers
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/primality/primality.pdf

---

Fermat and Mersenne identified two classes of integers that were likely to include many primes.

---

1 Overview

Finding new prime numbers is useful, because they are part of many encryption systems. Fermat and Mersenne both considered special classes of numbers that seemed to contain many large primes. Checking the primality of a very large number is a difficult task. One tool is the Lucas-Lehmer test. In this project, you will look at using implementing this tool in Python and applying it to several candidates that might be prime.

Note that recently, a new largest Mersenne prime was discovered $p = 2^{136279841}$

Finally, note the interesting pattern of the binary digits of the Fermat and Mersenne numbers:

Fermat	$2^{2^n}+1$	Mersenne	2^p-1
Decimal	Binary	Decimal	Binary
$2^0+1=1$	11	$2^2-1=3$	11
$2^2+1=5$	101	$2^3-1=7$	111
$2^4+1=17$	10001	$2^5-1=31$	11111
$2^8+1=257$	100000001	$2^7-1=127$	1111111
$2^{16}+1=65537$	10000000000000001	$2^{13}-1=8191$	1111111111111

2 Fermat primes

While searching for ways to generate primes, Pierre Fermat considered *Fermat numbers*, with the simple formula

$$F_n = 2^{2^n} + 1$$

The first values in the sequence are

$$F_0 = 2^{2^0} + 1 = 2$$
$$F_1 = 2^{2^1} + 1 = 5$$
$$F_2 = 2^{2^2} + 1 = 17$$
$$F_3 = 2^{2^3} + 1 = 257$$
$$F_4 = 2^{2^4} + 1 = 65,537$$
$$F_5 = 2^{2^5} + 1 = 4,294,967,297$$

Fermat was able to verify that the F_0 through F_4 were prime, and thought he might have found a way to produce an endless prime sequence. Euler, however, found a factorization of F_5, and it is generally believed that F_4 is the last Fermat prime.

This does mean that F_5 is a good test case for codes that check whether a number is prime, or that look for the factors of a composite number.

3 Fermat's primality test

Fermat's little theorem states that, if p is prime, and a is not a multiple of p, then

$$a^{p-1} = -1 (\text{mod } p)$$

Because computationally, the `mod()` function returns positive remainders, we might prefer to write this as:

$$a^{p-1} = p - 1 (\text{mod } p)$$

Now suppose we are given a number which might or might not be prime. If we find a value a with $2 \leq a \leq p-2$ such that the condition fails, then n cannot be a prime. (Why don't we have to check $a = p - 1$?). For a large n, there are many values of a to consider. It would be too expensive to check them all. What we are hoping is for a quick negative result after checking 3 or 4 values of a chosen at random. Thus, Fermat's primality test is generally probabilistic. If it fails, then n is definitely not a prime. But if it is passed, we really only can say that n was "lucky" so far; so far, it only looks like a possible prime.

4 Mersenne primes

Marin Mersenne looked at Mersenne numbers of the form

$$M_n = 2^n - 1$$

It is possible to show that, if n is not a prime, then M_n cannot be prime. In fact, if k divides n, then $2^k - 1$ divides $2^n - 1$. (This is also true for bases other than 2). Turning this around, we ask, if n is prime, will M_n be a Mersenne prime?

The sequence starts out promising:

$$M_2 = 2^2 - 1 = 3$$
$$M_3 = 2^3 - 1 = 7$$
$$M_5 = 2^5 - 1 = 31$$
$$M_7 = 2^7 - 1 = 127$$
$$M_11 = 2^{11} - 1 = 2047$$
$$M_13 = 2^{13} - 1 = 8191$$
$$M_17 = 2^{17} - 1 = 131071$$

All but one of these value is prime. (Can you spot the nonprime?) However, the sequence soon begins to return more nonprimes than primes. Nonetheless, researchers have continued to find ever higher indices for which M_n is prime. The 51st Mersenne prime, $2^{82,589,933} - 1$, is the largest prime number known.

The GIMPS (Great Internet Mersenne Prime Search) at `https://www.mersenne.org/primes/` records all the known Mersenne primes, and coordinates an ongoing search for new values.

5 The Lucas-Lehmer Test

Given how large the Mersenne numbers become, we have to be extremely clever if we want to be able to determine whether a large Mersenne number is prime. Luckily, because of the special structure that these numbers have, the Lucas-Lehmer test provides an exact answer efficiently. Let M_n be a given Mersenne number. Then the Lucas-Lehmer test defines a sequence s_i which starts at $s_0 = 4$ with subsequent values defined recursively

$$s_i = s_{i-1}^2 - 2$$

Then M_n is prime if and only if $s_{n-2} = 0 \mod M_n$ Here is pseudocode from Wikipedia for this calculation:

```
Lucas{Lehmer(p)
    var s = 4
    var M = 2^p - 1
    repeat p - 2 times:
        s = ((s x s) - 2 ) mod M
    if s == 0 return PRIME else return COMPOSITE
```

Notice that when we update the value of s, we use modular arithmetic every time, rather than simply for the check at the end. It turns out that this is legal, and keeps our s values from increasing without limit.

It's not hard to turn this into a Python code:

```
def lucas_lehmer ( n ):

  if ( n == 2 ):
    return True

  Mn = 2**n - 1
  s = 4
  for _ in range ( n - 2 ):
    s = ( ( s * s - 2 ) % Mn )

  return ( s == 0 )
```

Listing 1: lucas_lehmer.py

The Mersenne numbers quickily become gigantic. It is amazing that we can still determine the primality of these monster numbers within a reasonable amount of computer time. As an exercise, we can try to tabulate the size of some of these numbers versus the time it takes to run the Lucas-Lehmer test.

Note the underscore used in the `for()` loop. This is programming convention to suggest that the loop index variable is of no interest, so we are giving it the stupidest name we can think of. Surprisingly, you can give any variable the underscore name, and it will hold a value and can be used in arithmetic like any other variable. I don't really think it's useful, but you will see other people doing it, and now you don't have to be so puzzled.