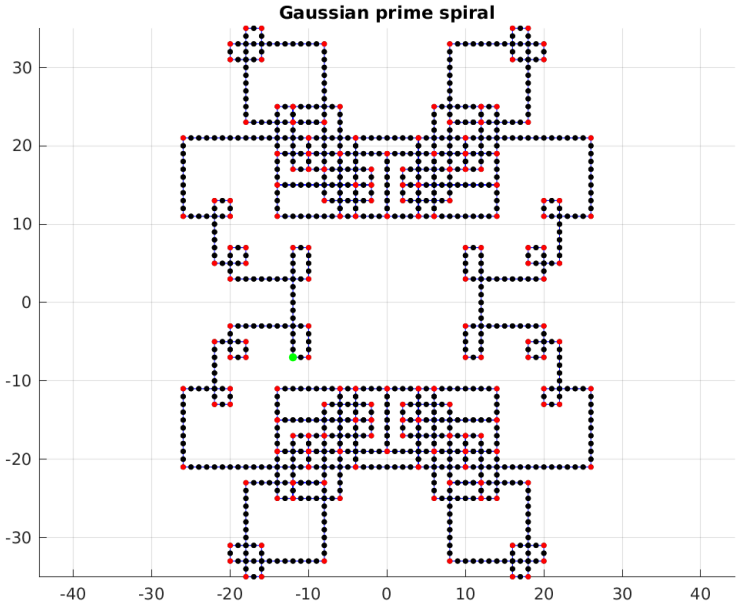
Gaussian Primes
Mathematical Programming with Python

https://people.sc.fsu.edu/~jburkardt/classes/math1800_2023/gaussian_primes/gaussian_primes.pdf



Gaussian Primes

- *Gaussian integers: $c = a + bi$, a and b are integers*
- *Gaussian primes: similar to integer primes.*
- *Consider the Gaussian primes as obstacles in the complex plane.*
- *Start at any Gaussian integer, move one unit at a time*
- *If you reach a Gaussiam prime, turn left, and continue.*
- *Unsolved conjecture: you always return home(?)*

1 Overview

A *Gaussian integer* is any complex number of the form $c = a + bi$ where a and b are integers. The set of Gaussian integers is closed under multiplication. Multiplying $c_1 * c_2$ yields another Gaussian integer c_3. Some Gaussian integers c can never be written as such a product, except for the uninteresting case where one factor is a complex unit $1, -1, i, -i$ and the other is c times the same complex unit. These special Gaussian integers are known as *Gaussian primes*. Every Gaussian integer has a factorization in terms of Gaussian primes and the complex units. Thus, the Gaussian primes play the same role for Gaussian integers that primes play for integers.

2 Review Complex Multiplication

A complex number c can be written $c = x + yi$, where the real numbers x and y are the real and imaginary parts of c, and i is the imaginary unit, such that $i^2 = -1$. This representation is sometimes called the Cartesian representation of c.

There is a corresponding polar representation $c = r\,e^{i\theta}$, where $r = \sqrt{x^2 + y^2}$ and $\theta = \tan^-1(\frac{y}{x})$.

To multiply $c_3 = c_1 * c_2$ we compute $c_3 = x_3 + y_3 i$ as:

$$x_3 = x_1 * x_2 - y_1 * y_2$$
$$y_3 = x_1 * y_2 + y_1 * x_2$$

or $c_3 = r_3\,e^{i\theta_3}$:

$$r_3 = r_1 * r_2$$
$$\theta_3 = \theta_1 + \theta_2 \mod(2\pi)$$

In Python, we can form a complex number as

```
c = complex ( x, y )
```

or

```
c = x + yj
```

3 Distribution of Gaussian Primes and Composites

The Gaussian primes (red dots in the graph) are dense near the origin, but become sparser as we move away.

4 Unsolved problems

1. There is a mathematical conjecture that asserts that, along any horizontal or vertical grid line, there must be infinitely many Gaussian primes in either direction.

2. One cannot walk to infinity on the real line if one uses steps of bounded length and steps only on the prime numbers. This is simply a restatement of the classic result that there are arbitrarily large gaps in the primes. The proof is simple: a gap of size k is given by $(k+1)!+2, (k+1)!+3, ...+(k+1)!+(k+1)$.

   But the same problem is unsolved for a walk to infinity over the Gaussian primes. The question is much more complex because of the additional dimension. There are arbitrarily large disks in the complex plane that contain no Gaussian primes at all, but presumably we could try to walk around such obstacles.

   As one effort to understand this problem, Nobuyuki Tsuchimura showed in 2004 that it is not possible to reach the distance 80015782 from the origin, stepping only on Gaussian primes, if all the steps must have length 6 or smaller.

3. We are about to describe an iteration called the Gaussian Prime Spiral. So far, no matter what starting point people have used, the iteration forms a closed loop, returning to its starting point. But no one has been able to prove that this always happens. And so a cautious programmer (such as Christian Hill) will put in a maximum number of steps and then stop the iteration if necessary!

5 When is c a Gaussian prime?

There is a test to determine whether a Gaussian integer $c = a + bi$ is a Gaussian prime.

Theorem 1 *The Gaussian integer $c = a + bi$ is a Gaussian prime if one of the following three conditions holds:*

1. *a is 0 and $|b|$ is a prime with $|b| \mod 4 = 3$;*
2. *b is 0 and $|a|$ is a prime with $|a| \mod 4 = 3$;*
3. *a and b are nonzero, and $a^2 + b^2$ is prime.*

Consider some simple examples:

```
 0       no   b is 0, but 0 is not a prime
 1       no:  b is 0, but 1 is not a prime
-2       no:  b is 0, 2 is a prime, but 2 mod 4 is 2
 2.5     no,  a is not an integer!
 3       yes: b is 0, 3 is a prime, and 3 mod 4 is 3
13       no:  b is 0, 13 is a prime, but 13 mod 4 is 1
      i  no:  a is 0, but 1 is not a prime
-5 + 2i  yes: 25 + 4 = 29 is prime.
 6 - 3i  no:  36 + 9 = 45 is not prime.
```

6 Implementing the Primality Test

I suggest writing a Python function `is_gaussian_prime(c)` which accepts a complex number `c` and returns `True` or `False` result according to whether `c` is a Gaussian prime.
Here are things we need to check:

```
a <= real part of c
b <= imaginary part of c

if a or b is not an integer, return False

if a is 0, return value of (|b| is prime and |b| mod 4 = 3 )
if b is 0, return value of (|a| is prime and |a| mod 4 = 3 )

if a and b are not zero, return value of (a^2+b^2 is prime)
```

Let's try to create this code before looking at three implementations.

First, here is our textbook author, Christian Hill. Note that he wrote his own `is_prime()` function as well:

```
def is_gaussian_prime ( c ):
  """ Return True if c = x + iy is a Gaussian prime (otherwise False). """

  x, y = c
  if x==0 or y==0:
    a = abs(x) or abs(y)          # This works, but maybe too clever for us!
    return is_prime(a) and a % 4 == 3
  a = x**2 + y**2
  return is_prime(a)
```

Secondly, we have John D Cook, who uses `isprime()` from `sympy`:

```
def isgaussprime ( z: complex ):
  from sympy import isprime
  a, b = int ( z.real ), int ( z.imag )
  if a*b != 0:
    return isprime ( a**2 + b**2 )
  else:
    c = abs ( a + b )
    return isprime ( c ) and c % 4 == 3
```

Finally, your instructor:

```
def is_gaussian_prime ( c )

  from sympy import isprime

  a = int ( abs ( c.real ) )
  b = int ( abs ( c.imag ) )

  if ( c.real != round ( c.real ) ):
    value = False
  elif ( c.imag != round ( c.imag ) ):
    value = False
  elif ( a == 0 ):
    value = ( isprime ( b ) and ( ( b % 4 ) == 3 ) )
  elif ( b == 0 ):
    value = ( isprime ( a ) and ( ( a % 4 ) == 3 ) )
  else:
    value = isprime ( a * a + b * b )

  return value
```

I wanted to include a check that a and b are integers. This was surprisingly difficult to do correctly!

7 Testing the Primality Test

Let's test our implementations of the primality test on our sample cases:

```
        Correct  HILL?    COOK?    JVB?
   0      F      -----    -----    -----
   1      F      -----    -----    -----
  -2      F      -----    -----    -----
   2.5    F      -----    -----    -----
   3      T      -----    -----    -----
  13      F      -----    -----    -----
       i  F      -----    -----    -----
  -5 + 2i T      -----    -----    -----
   6 - 3i F      -----    -----    -----
```

8 A Gaussian Prime Spiral

Instead of worrying about the pure mathematical problem of Gaussian primes, we will be interested in using the primality test, to display the distribution of Gaussian primes, and then take a ride on a Gaussian prime spiral. This task is described on page 99 of our textbook:

Problem P3.2.2: Consider the sequence of Gaussian integers traced out by an imaginary particle, initially at c_0, moving in the complex plane according to the following rule: it takes integer steps in its current direction

(± 1 in either the real or imaginary direction) but turns left after it encounters a Gaussian prime. Its initial direction is the positive real direction. The path traced out by the particle is called a Gaussian prime spiral.

Write a program to plot the Gaussian prime spiral starting at $c_0 = 5 + 23i$.

Christian Hill also provides a Python solution to his problem at:

`https://scipython.com/book2/chapter-3-simple-plotting/problems/p32/the-gaussian-prime-spiral/`

John D Cook also wrote a short online article, called *At the next prime, turn left*:

`https://www.johndcook.com/blog/2020/09/24/gaussian_integer_walk/`

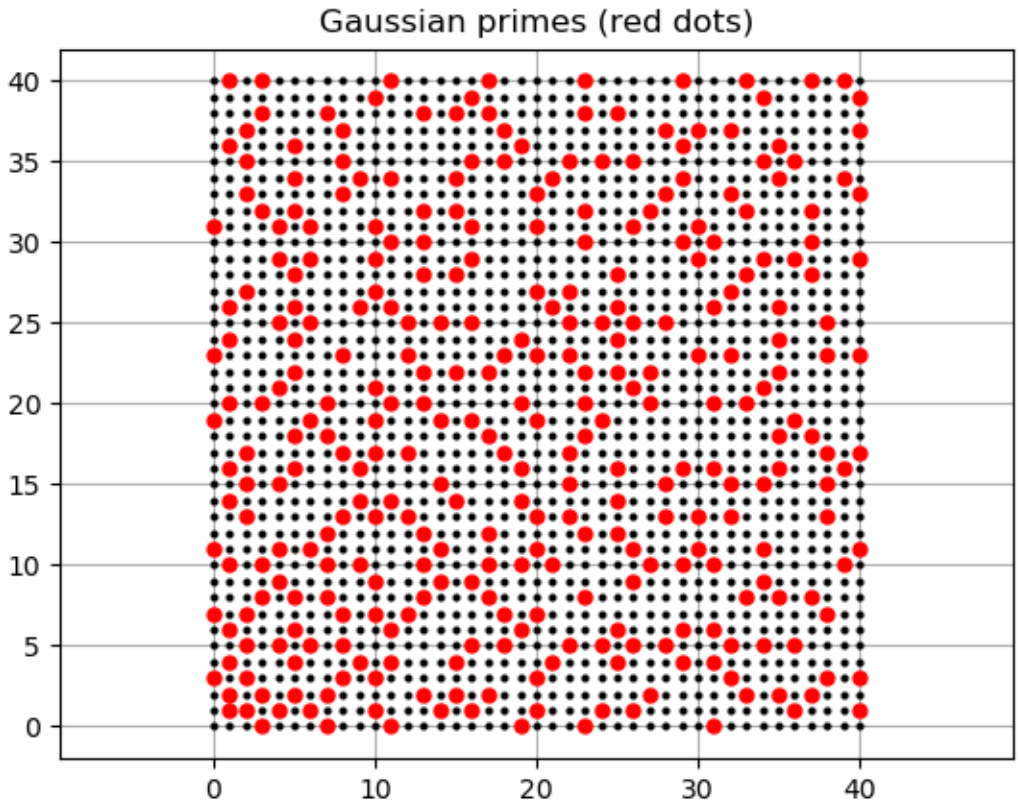
9 Plotting Gaussian Primes

Before we tackle the spiral, we can try our our primality test, and get a feeling for the distribution of the Gaussian primes, by picking a rectangular region of the complex plane, marking the Gaussian integers with black dots, and the Gaussian primes with red. If c is a Gaussian prime, then so are $-c$, $c * i$ and $-c * i$. That means that a picture centered at the origin will be left-right and up-down symmetric. To get an idea of what's going on, then, we will only plot a portion of the the upper right quarter of the complex plane.

We will use a function `gaussian_prime_display(clo,chi)`:

```
def gaussian_prime_display ( clo, chi ):

  alo = int ( clo.real )
  blo = int ( clo.imag )
  ahi = int ( chi.real )
  bhi = int ( chi.imag )

  plt.clf ( )

  for a in range ( alo, ahi + 1 ):
    for b in range ( blo, bhi + 1 ):
      if ( is_gaussian_prime ( a+b*1j ) ):
        plt.plot ( a, b, 'r.', markersize = 10 )
      else:
        plt.plot ( a, b, 'ko', markersize = 3 )

  plt.show ( )
  plt.close ( )

  return
```

Here's what we see with the command `gaussian_prime_display ( 0, 40+40j)`:



This plot suggests that there is one more degree of symmetry, across the diagonal, and that the Gaussian primes seem to thin out as we move away from the origin.

We will now take a walk across this landscape, regarding the red dots as obstacles. If we run into an obstacle, we turn to our left.

10 Taking a Spin on the Spiral

To describe our trip, we will want to specify an initial position c and an initial direction d (which we will usually take to be 1). We need a code that repeatedly computes the next step, changing direction after hitting any Gaussian prime, and stopping if it returns to its starting point. For safety, we might also limit the total number of steps, in case something goes wrong.

Notice, for a moment, that each time our direction d takes a left turn, it cycles through the values $1, i, -1, -i$ and then back to 1. In fact, a left turn can be accomplished automatically by multiplying the current direction by i, or, in Python:

```
if ( we just hit a Gaussian prime ):
  d = d * 1j
```

For plotting, we will need to think of our Gaussian integers in terms of their real and imaginary parts, a and b, while at other times we might want to think of the single complex number c.

Logically, the pseudocode looks like this

```
gaussian_prime_spiral ( cstart, d )

  step = 0
  loop forever

    if step == 0
      c = cstart
    else
      c = c + d
      if c is gaussian_prime
        d = d * 1j

    alist = alist + real c
    blist = blist + imaginary c

    if ( 0 < step and c == cstart )
      break

    step = step + 1

  plot ( alist, blist )
```

One of your suggested homework exercises will be to implement this code in Python.

11 Suggested starting points

The trajectory depends on the initial point and direction. As far as we know, every initial set of data makes a loop and returns home. The length of the loop seems to vary in an unpredictable way. Here are some possible starting points, all of which use the initial direction `d=1`. The last items on the list can take several minutes or more to compute:

```
  3 +   5j
  5 +  23j
  8 +  13j
-12 -   7j
 27 +  30j
127 + 130j
```