This is to solve the Sage portion of Chapter 4 of

"A journey through the realm of numbers:

from quadratic equations to quadratic reciprocity."

In [1]:
```
# Testing the validity of the Fermat-Euler theorem
```

In [2]:
```
def fermat_test(n):
    return all([p==2 or p%4==1 or e%2==0 for p, e in factor(n)])
# tests whether all p congruent to 3 mod 4 appear with even power
```

In [3]:
```
fermat_test(5)
```

Out[3]:
```
True
```

In [4]:
```
fermat_test(15)
```

Out[4]:
```
False
```

In [5]:
```
def sum2sq(t=100):   # list of sum of two squares
    N = floor(sqrt(t)) + 1
    return [m^2+n^2 for m in range(1,N)
                    for n in range(N)
                    if m^2+n^2 <= t]
```

In [6]:
```
def only_if_part(t): # first direction of Fermat-Euler
    return all(fermat_test(k) for k in sum2sq(t))
```

In [7]:
```
only_if_part(10^5)   # testing only_if up to 10^5
```

Out[7]:
```
True
```

In [8]:
```
def if_part(t):      # second direction of Fermat-Euler
    sumlist = sum2sq(t)
    return all(not(fermat_test(k)) or (k in sumlist)
               for k in range(1,t))
```

In [9]:
```
if_part(10^5)        # testing if up to 10^5
```

Out[9]:
```
True
```

In [10]:
```
# Division with remainder and greatest common divisor
```

In [11]:
```
def div_rem(m, d):              # assumes d!=0
    z = m / d
    a = round(z.real())         # round does what it avertises
    b = round(z.imag())         # real and imaginary parts
    return m - (a+i*b)*d        # returns only the remainder
```

In [12]:
```
div_rem(3+2*i, 2+i)
```

Out[12]:
```
-1
```

In [13]:
```
def Euclid(m1, m2):             # we assume m1!=0
    if m2 == 0:
        return m1               # vanishing case
    else:
        r = div_rem(m1, m2)
        return Euclid(m2, r)    # ascent property
```

In [14]:
```
m1 = (1+i)*(2+i)^3*7
m2 = 2*5*(3+2*i)^2          # recall 5=(2+i)(2-1) and 2=-i(1+i)^2
                            # so the answer should be (1+i)(2+i)
print("The gcd of ",m1," and ", m2,"is", Euclid(m1,m2), ".")
```

```
The gcd of  91*I - 63  and  120*I + 50 is I - 3 .
```

In [15]:
```
i*(1+i)*(2+i)           # checking by hand using the unit i
```

Out[15]:
```
I - 3
```

In [16]:
```
def div_rem2(m, d):                     # assumes d!=0
    z = m / d
    a = round(z.real())
    b = round(z.imag())
    return (a+i*b, m - (a+i*b)*d)       # returns quotient first
```

In [17]:
```
div_rem2(3+2*i, 2+i)
```

Out[17]:
```
(2, -1)
```

In [18]:
```
def Euclid2(m1, m2):                        # we assume m1!=0
    if m2 == 0:
        return (m1, 1, 0)                   # vanishing case
    else:
        (q, r) = div_rem2(m1, m2)           # m1=q*m2+r, r=m1-q*m2
        (d, old1, old2) = Euclid2(m2, r)    # d=old1*m2+old2*r
        return (d, old2, old1-q*old2)
```

In [19]:
```
m1 = (1+i)*(2+i)^3*7
m2 = 2*5*(3+2*i)^2
print("The gcd of", m1," and ", m2, "is")
d, c1, c2 = Euclid2(m1, m2)
print(d,"=(",c1,")*(",m1,")+(",c2,")*(",m2,").")
```

```
The gcd of 91*I - 63  and  120*I + 50 is
I - 3 =( -20*I - 29 )*( 91*I - 63 )+( 30*I - 1 )*( 120*I + 50 ).
```

In [20]:
```
c1*m1 + c2*m2                          # just checking if true
```

Out[20]:
```
I - 3
```

In [21]:
```
# Testing irreducibilty (crudely) in the Gaussian integers
```

In [22]:
```
def divides(d, m):                          # assumes d!=0
    z = m / d                               # checks for divisibility
    return (z.real() in Integers()
            and z.imag() in Integers())
```

In [23]:
```
divides(2+i, 5)
```

Out[23]:
```
True
```

In [24]:
```
def Gaussian_prime(p):
    N = p.real()^2+p.imag()^2
    r = floor(sqrt(N))
    candidates = [a+i*b for a in range(r) # looking for divisors
                       for b in range(r) # in first quadrant
                       if 1 < a^2+b^2 and a^2+b^2 < N]
    return N > 1 and all(not(divides(q, p)) for q in candidates)
```

In [25]:
```
for b in range(6):              # looking for primes in
    for a in range(b,10):       # first quadrant
        if Gaussian_prime(a + i*b):
            print(a+i*b, " is a Gaussian prime.")
```

```
3  is a Gaussian prime.
7  is a Gaussian prime.
I + 1  is a Gaussian prime.
I + 2  is a Gaussian prime.
I + 4  is a Gaussian prime.
I + 6  is a Gaussian prime.
2*I + 3  is a Gaussian prime.
2*I + 5  is a Gaussian prime.
2*I + 7  is a Gaussian prime.
3*I + 8  is a Gaussian prime.
4*I + 5  is a Gaussian prime.
4*I + 9  is a Gaussian prime.
5*I + 6  is a Gaussian prime.
5*I + 8  is a Gaussian prime.
```

In [26]:
```
# Working with modular arithmetic within Z/(d) in sage
```

In [27]:
```
a = mod(2,10)        # defines the residue class of 2 modulo 10
a^10                 # takes the 10th power
```

Out[27]:
```
4
```

In [28]:
```
type(a)              # remembers its nature
```

Out[28]:
```
<class 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>
```

In [29]:
```
a.modulus()          # and its modulus
```

Out[29]:
```
10
```

In [30]:
```
Zd = Integers(10)    # the quotient ring modulo (10)
table([[m*n for m in Zd] for n in Zd])
```

Out[30]:

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	0	2	4	6	8
0	3	6	9	2	5	8	1	4	7
0	4	8	2	6	0	4	8	2	6
0	5	0	5	0	5	0	5	0	5
0	6	2	8	4	0	6	2	8	4
0	7	4	1	8	5	2	9	6	3
0	8	6	4	2	0	8	6	4	2
0	9	8	7	6	5	4	3	2	1

In [31]:
```
def mult_table(p):   # general mutltiplication table
    Fp = Integers(p)
    return table([[m*n for m in Fp] for n in Fp])
```

In [32]:
```
mult_table(5)
```

Out[32]:

0	0	0	0	0
0	1	2	3	4
0	2	4	1	3
0	3	1	4	2
0	4	3	2	1

In [33]:
```
mult_table(7)
```

Out[33]:

0	0	0	0	0	0	0
0	1	2	3	4	5	6
0	2	4	6	1	3	5
0	3	6	2	5	1	4
0	4	1	5	2	6	3
0	5	3	1	6	4	2
0	6	5	4	3	2	1

In [34]:
```
def inv_table(p):   # awkward definition using our command
    return table([[m, mod((Euclid2(m,p)[1]/Euclid2(m,p)[0]),p)]
                  for m in range(1,p)])
    # we divide by the gcd in case it equals -1 instead of 1
```

In [35]:
```
inv_table(5)
```

Out[35]:

1	1
2	3
3	2
4	4

In [36]:
```
inv_table(7)
```

Out[36]:

1	1
2	4
3	5
4	2
5	3
6	6

In [37]:
```
def inv_table(p):   # using the build in modular division
    return table([[m, m^(-1)] for m in Integers(p) if m!=0])
```

In [38]:
```
inv_table(5)
```

Out[38]:

1	1
2	3
3	2
4	4

In [39]:
```
inv_table(7)
```

Out[39]:

1	1
2	4
3	5
4	2
5	3
6	6

In [40]:
```
# Analysing squares within Fp
```

In [41]:
```
def is_square(a=-1, p=13):    # looks for a square root of a
    found = false             # modulo p
    for b in Integers(p):
        if b^2 == a:
            found = true
            break
    return (found, b)
```

In [42]:
```
is_square()
```

Out[42]:
```
(True, 5)
```

In [43]:
```
is_square(-1, 11)
```

Out[43]:
```
(False, 10)
```

In [44]:
```
def when_square(a=-1, t=100):
    for p in range(1, t):
        if is_prime(p):
            (found,b) = is_square(a,p)
            if found:
                print(a, " is a square modulo", p,
                      " (because of ", b,").")
            else:
                print(a, " is not a square modulo", p, ".")
```

In [45]:
```
when_square(-1, 40)
# checks for a square root of -1 modulo all primes below 40
```

```
-1  is a square modulo 2  (because of  1 ).
-1  is not a square modulo 3 .
-1  is a square modulo 5  (because of  2 ).
-1  is not a square modulo 7 .
-1  is not a square modulo 11 .
-1  is a square modulo 13  (because of  5 ).
-1  is a square modulo 17  (because of  4 ).
-1  is not a square modulo 19 .
-1  is not a square modulo 23 .
-1  is a square modulo 29  (because of  12 ).
-1  is not a square modulo 31 .
-1  is a square modulo 37  (because of  6 ).
```

In [46]:
```
when_square(2, 40)  # same but for square root of 2
```

```
2  is a square modulo 2  (because of  0 ).
2  is not a square modulo 3 .
2  is not a square modulo 5 .
2  is a square modulo 7  (because of  3 ).
2  is not a square modulo 11 .
2  is not a square modulo 13 .
2  is a square modulo 17  (because of  6 ).
2  is not a square modulo 19 .
2  is a square modulo 23  (because of  5 ).
2  is not a square modulo 29 .
2  is a square modulo 31  (because of  8 ).
2  is not a square modulo 37 .
```

In [47]:
```
# finding square roots in Z(n)
```

In [48]:
```
def count(d=15):
    Zd = Integers(d)
    return len([a for a in Zd if a^2==1])
```

In [49]:
```
count()  # how many square roots of 1 exist in Z/(15)
```

Out[49]:
```
4
```

In [50]:
```
for d in range(2,20):
    print(d, count(d))
```

```
2 1
3 2
4 2
5 2
6 2
7 2
8 4
9 2
10 2
11 2
12 4
13 2
14 2
15 4
16 4
17 2
18 2
19 2
```

In [51]:
```
[d for d in range(2, 50) if count(d) == 2]
```

Out[51]:
```
[3,
 4,
 5,
 6,
 7,
 9,
 10,
 11,
 13,
 14,
 17,
 18,
 19,
 22,
 23,
 25,
 26,
 27,
 29,
 31,
 34,
 37,
 38,
 41,
 43,
 46,
 47,
 49]
```

In [52]:
```
# This might lead one to conjecture the following criteria
# for precisely 2 square roots of 1 in Z/(d):
def prec2(d):
    if d == 4:                # d=4 is a special power of 2
        return True           # since it alone gives two roots
    factors = factor(d)
    if len(factors) == 1:     # odd prime powers are ok
        return factors[0][0]>2
    elif len(factors) == 2:   # twice an odd prime power is ok
        return factors[0] == (2,1)
    else:                     # in all other cases there are
        return False          # more square roots of 1
```

In [53]:
```
# This tests our condition for d up to 1000.
all((count(d)==2) == prec2(d) for d in range(2,1000))
```

Out[53]:
```
True
```

In [54]:
```
# Representing primes in the form a^2+d*b^2
```

In [55]:
```
def sumsqd(d=1 ,t=100):
    N = floor(sqrt(t)+1)
    list = [a^2+d*b^2 for a in range(N)
                      for b in range(N)
                      if is_prime(a^2+d*b^2) and a^2+d*b^2<t]
    return sorted(Set(list))
```

In [56]:
```
sumsqd()
```

Out[56]:
```
[2, 5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97]
```

In [57]:
```
sumsqd(2)
```

Out[57]:
```
[2, 3, 11, 17, 19, 41, 43, 59, 67, 73, 83, 89, 97]
```

In [58]:
```
list2 = sumsqd(2,10^4)
for d in range(2,11):
    print(d, Set([mod(n,d) for n in list2]))
```

```
2 {0, 1}
3 {0, 1, 2}
4 {1, 2, 3}
5 {1, 2, 3, 4}
6 {1, 2, 3, 5}
7 {1, 2, 3, 4, 5, 6}
8 {1, 2, 3}
9 {1, 2, 3, 4, 5, 7, 8}
10 {1, 2, 3, 7, 9}
```

In [59]:
```
# Hence congruences modulo 2,3,4,5,7,9 seem quite useless
# let us study a possible congruence condition modulo 6
all(p in list2 for p in range(10^4)
                if is_prime(p) and mod(p,6) in [1,2,3,5])
```

Out[59]: False

In [60]:
```
# Ok, maybe we should use congruence modulo 8.
all(p in list2 for p in range(10^4)
                if is_prime(p) and mod(p,8) in [1,2,3])
```

Out[60]: True

In [61]:
```
# This suggests (correctly) that a prime p is representable
# in the form a^2+2*b^2 if and only if p is congruent to
# 1,2, or 3 modulo 8.
```

In [62]:
```
# Let us repeat this with d=3 and replace our above guess work
# concerning the right congruence condition by checking all
# possible conjecture for d considered in some range.
list3 = sumsqd(3, 10^4) # all represented numbers below 10^4
conglist = [(d, Set([mod(n, d) for n in list3]))
            for d in range(2,11)]
            # all congruences of these numbers for d<11
for (d,cong) in conglist:
    if all(p in list3 for p in range(10^4)
                      if is_prime(p) and (mod(p, d) in cong)):
        print(d, cong)
```

```
3 {0, 1}
6 {1, 3}
9 {1, 3, 4, 7}
```

In [63]:
```
# This suggests for primes p that p is representable in the form
# a^2+3*b^2 if and only if p=3 or p is congruent to 1 modulo 3.
```

In [64]:
```
def test3(n):
    return all(p==3 or p%3==1 or e%2==0 for p, e in factor(n))
```

In [65]:
```
test3(3*5)
```

Out[65]: False

In [66]:
```
test3(3*25)
```

Out[66]: True

In [67]:
```
sorted(Set(m^2+3*n^2 for m in range(100)
                     for n in range(100)
                     if m^2+3*n^2<10^4))\
== [n for n in range(10^4) if n==0 or test3(n)]
```

Out[67]: True

In [68]:
```
# Hence the conjectured congruence modulo 3 and the generalization
# of the Fermat-Euler theorem also seem to work very well.
```

In [69]:
```
# Working with quadratic fields and its rings of integers directly.
```

In [70]:
```
# To define the ring of Gaussian integers we define the 'field'
K.<i> = QuadraticField(-1)
# and then use "maximal_order" to define its ring of integers.
R = K.maximal_order()
```

In [71]:
```
factor(R(60))
```

Out[71]:
```
(-i) * (-i - 2) * (i + 1)^4 * (2*i + 1) * 3
```

In [72]:
```
# this shows that 60 factors as -i*(-i-2)*(i+1)^4(2i+1)*3
# we may instead have written -(1+i)^4*3*(2+i)*(2-i)
# but these are equivalent factorizations (i.e. the same
# after permuting the factors and changing the primes by units)
```

In [73]:
```
# here is another example that we can set up just as quickly
K2.<r> = QuadraticField(-2)  # here r=sqrt(2)*i
R2 = K2.maximal_order()
factor(R2(60))
```

Out[73]:
```
(-r - 1) * (r - 1) * r^4 * 5
```

In [74]:
```
# this shows that 5 remains a prime in Z[r], but 2 and 3 factor
print(is_prime(R2(2)), factor(R2(2)))
print(is_prime(R2(3)), factor(R2(3)))
print(is_prime(R2(5)), factor(R2(5)))
```

```
False (-1) * r^2
False (-r - 1) * (r - 1)
True 5
```

In [75]:
```
for p in range(40):
    if is_prime(p):
        if is_prime(R2(p)):
            print(p,"stays a prime")
        else:
            print(p,"factors as", factor(R2(p)))
```

```
2 factors as (-1) * r^2
3 factors as (-r - 1) * (r - 1)
5 stays a prime
7 stays a prime
11 factors as (-1) * (r - 3) * (r + 3)
13 stays a prime
17 factors as (-2*r + 3) * (2*r + 3)
19 factors as (-3*r + 1) * (3*r + 1)
23 stays a prime
29 stays a prime
31 stays a prime
37 stays a prime
```

In [76]:
```
# From the above we might notice a pattern modulo 8.
print(Set(mod(p,8) for p in range(10^4)
                   if is_prime(p) and is_prime(R2(p))))
```

```
{5, 7}
```

In [77]:
```
# This suggests (correctly) that p is inert for R2 (i.e. stays a
# prime in R2) if and only if p is congruent 5 or 7 modulo 8.
# Equivalently p is ramified or split (becomes a power or product
# of primes) if and only if p is 2 or congruent 1 or 3 modulo 8.
# this matches an ealier answer above ....
```

In [78]:
```
K3.<s> = QuadraticField(-3)      # here s=sqrt(3)*i
R3 = K3.maximal_order()          # R3 is the ring of integers
print(factor(R3(60)))
```

```
(-1) * (-s)^2 * 2^2 * 5
```

In [79]:
```
(1+s) / 2                        # the sixth root of unity
```

Out[79]:
```
1/2*s + 1/2
```

In [80]:
```
(1+s) / 2 in R3    # belongs to the ring of integers defined above
```

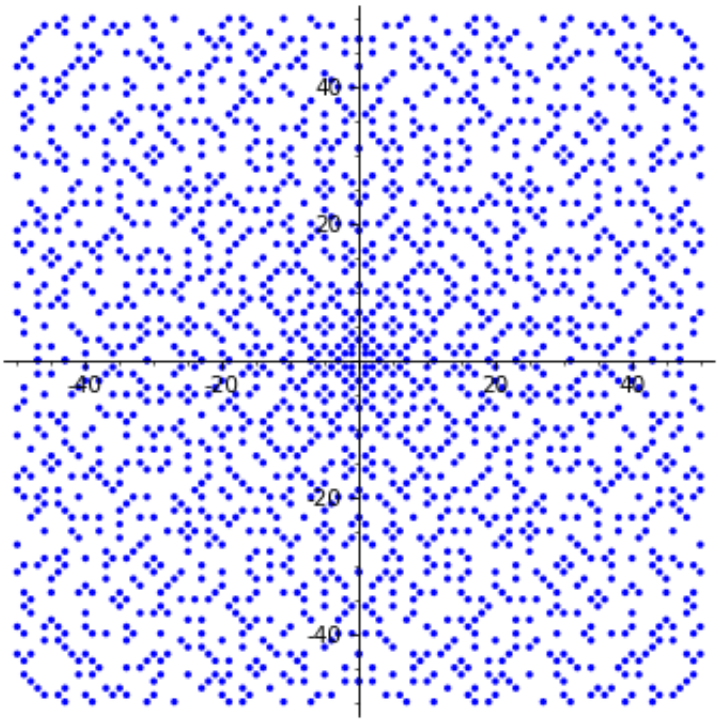
Out[80]:
```
True
```

In [81]:
```
K5.<t> = QuadraticField(-5)      # here t=sqrt(5)*i
R5 = K5.maximal_order()          # again the ring of integers
```

In [ ]:
```
factor(R5(60))    # this creates an error since the ring R5
                  # is not Euclidean and unique factorization
                  # doesn't work there
```

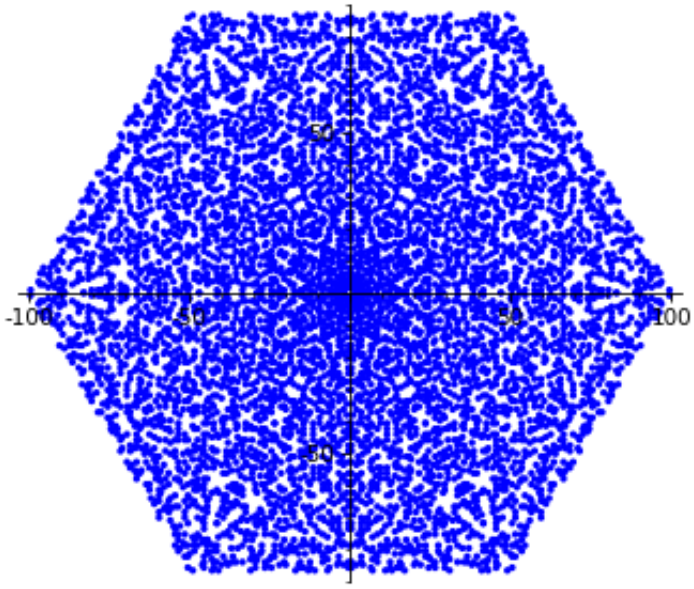
In [83]:
```
# visualizing primes in Gaussian integers
prime_list = []
for a in range(0,51):
    for b in range(0,51):
        if is_prime(a+b*i): # even 1+0*i belongs to R
            prime_list.append((a, b))
            prime_list.append((a, -b))
            prime_list.append((-a, b))
            prime_list.append((-a, -b))
points(prime_list, dpi=75, aspect_ratio=1)
```

Out[83]:



In [84]:
```
# visualizing primes in the Eisenstein integers
r3 = sqrt(3).n()
prime3_list = []
for a in range(101):
    for b in range(101):
        if a+b < 101 and is_prime(a+b*(1+s)/2):
            prime3_list.append((a+b/2, b/2*r3))
            prime3_list.append((a+b/2, -b/2*r3))
            prime3_list.append((-a-b/2, b/2*r3))
            prime3_list.append((-a-b/2, -b/2*r3))
            prime3_list.append((a/2-b/2, (a+b)/2*r3))
            prime3_list.append((a/2-b/2, -(a+b)/2*r3))
points(prime3_list, dpi=75, aspect_ratio=1)
```

Out[84]:



In [ ]: