Entiers_Gauss2

January 27, 2021

1 Entiers de Gauss (II)

2 Primalité et factorisation

Marc Lorenzi

25 janvier 2021

[1]:
```
import matplotlib.pyplot as plt
import math
import random
import arith
import time
from gauss import Gauss, I, random_gauss, pgcd
```

On suppose dans ce notebook que le lecteur a lu le premier document sur les entiers de Gauss. Le module contenu dans le fichier `gauss.py` contient la classe `Gauss`.

On rappelle que l'anneau des entiers de Gauss est le sous-anneau de $\mathbb{C}$

$$\mathbb{Z}[i] = \{x + iy, x, y \in \mathbb{Z}\}$$

L'ensemble des éléments inversibles de $\mathbb{Z}[i]$ est

$$\mathcal{U} = \{1, -1, i, -i\}$$

Pour tout $z \in \mathbb{Z}[i]$, $\overline{z} \in \mathbb{Z}[i]$. On pose $N(z) = |z|^2 = z\overline{z} \in \mathbb{Z}$.

$\mathbb{Z}[i]$ est un anneau euclidien. En conséquence, ses idéaux sont tous principaux, c'est à dire de la forme $a\mathbb{Z}[i]$ où $a \in \mathbb{Z}[i]$. On dispose de la notion de pgcd. Celui-ci est défini à inversible près et peut être calculé via l'algorithme d'Euclide. Le lemme de Gauss et le théorème de Bézout sont valides dans l'anneau $\mathbb{Z}[i]$.

2.1 1. Primalité

2.1.1 1.1 Éléments irréductibles, éléments premiers

Définition. Soit $z \in \mathbb{Z}[i]$.

- On dit que z est *irréductible* lorsque z n'est pas inversible, et

$$\forall a, b \in \mathbb{Z}[i], z = ab \implies a \in \mathcal{U}_{\mathbb{G}} \text{ ou } b \in \mathcal{U}_{\mathbb{G}}$$

- On dit que z est *premier* lorsque z n'est pas inversible et n'est pas nul, et

$$\forall a, b \in \mathbb{Z}[i], z \mid ab \implies z \mid a \text{ ou } z \mid b$$

La primalité et l'irréductibilité sont deux notions équivalentes dans $\mathbb{Z}[i]$.

Proposition. Soit z un élément premier de $\mathbb{Z}[i]$. Alors z est irréductible.

Démonstration. Supposons $z = ab$, où $a, b \in \mathbb{Z}[i]$. Comme z est premier, z divise a ou z divise b. Par exemple, z divise a. Il existe donc $w \in \mathbb{Z}[i]$ tel que $a = wz$. Reportons, il vient $z = wzb$. En simplifiant par z (z est non nul), il vient $wb = 1$. Ainsi, $b \in \mathcal{U}$.

Passons au sens « moins évident ». La proposition ci-dessous s'appelle le *lemme d'Euclide*.

Proposition. Soit z un élément irréductible de $\mathbb{Z}[i]$. Alors z est premier.

Démonstration. z est clairement non inversible et non nul. Supposons que z divise ab, où $a, b \in \mathbb{Z}[i]$. Écrivons $ab = wz$. Soit δ un pgcd de a et de w. Écrivons

$$w = \delta w' \text{ et } a = \delta a', \text{ où } w' \wedge a' = 1$$

Il vient $\delta a'b = \delta w'z$ d'où, puisque $\delta \neq 0$, $a'b = w'z$. Par le lemme de Gauss, w' divise b. Écrivons donc $b = w'b'$, d'où $a'w'b' = w'z$ et donc $a'b' = z$. Comme z est irréductible, on a par exemple $a' \in \mathcal{U}$, et donc z divise b', donc a fortiori z divise b.

Une question s'impose : quels sont les entiers de Gauss premiers ? Voici deux évidences.

Proposition. Soit $z \in \mathbb{Z}[i]$. Si $N(z)$ est premier dans $\mathbb{Z}$ alors z est premier dans $\mathbb{Z}[i]$.

Démonstration. Soit $p = N(z)$. Supposons p premier dans $\mathbb{Z}$. Écrivons $z = ab$. Alors $N(a)N(b) = N(z) = p$ et donc $N(a) = 1$ ou $N(b) = 1$. Dit autrement, l'un des deux entiers de Gauss a et b est inversible.

Proposition. Soit $n \in \mathbb{Z}$ un entier composé dans $\mathbb{Z}$. Alors n est aussi composé dans $\mathbb{Z}[i]$.

Démonstration. Évident. La décomposition de n dans $\mathbb{Z}$ en est aussi une dans $\mathbb{Z}[i]$.

Remarquons toutefois que la réciproque est fausse :

$$(1 + i)(1 - i) = 2$$

$$(2 + i)(2 - i) = 5$$

Ainsi, 2 et 5 sont premiers dans $\mathbb{Z}$, mais pas dans $\mathbb{Z}[i]$.

Remarquons aussi que par la première proposition, $1 \pm i$ et $2 \pm i$ sont premiers dans $\mathbb{Z}[i]$. Nous venons donc d'écrire 2 et 5 comme des produits d'entiers de Gauss premiers.

2.1.2 1.2 Entiers relatifs premiers dans $\mathbb{Z}[i]$

Commençons par regarder quels sont les éléments premiers de $\mathbb{Z}$ qui sont aussi premiers dans $\mathbb{Z}[i]$.

Proposition. Soit p un entier premier positif. Alors p est composé dans $\mathbb{Z}[i]$ si et seulement si p est la somme de deux carrés dans $\mathbb{Z}$.

Demonstration. Supposons que $p = a^2 + b^2$. Alors $p = (a + ib)(a - ib)$ est composé dans $\mathbb{Z}[i]$.

Inversement, supposons p composé dans $\mathbb{Z}[i]$. Écrivons $p = uv$, où $u, v \in \mathbb{G}$ ne sont pas inversibles. On a alors

$$N(p) = p^2 = N(u)N(v)$$

Ainsi, $N(u) = N(v) = p$ puisque ces entiers divisent p^2 et sont tous deux différents de 1. On a donc $u\overline{u} = p$. Posons $u = a + ib$, il vient $p = (a + ib)(a - ib) = a^2 + b^2$.

Proposition. Soit p un entier premier positif. Alors p est composé dans $\mathbb{Z}[i]$ si et seulement si $p \equiv 1 \bmod 4$.

Demonstration. Nous avons déjà vu que 2 n'est pas premier dans $\mathbb{Z}[i]$. Donc $p \equiv 1$ ou $3 \bmod 4$. Pour tout $a \in \mathbb{Z}$, on a $a^2 \equiv 0$ ou 1 modulo 4. Ainsi, étant donnés $a, b \in \mathbb{Z}$, $a^2 + b^2 \equiv 0$, 1 ou 2 modulo 4. Si $p \equiv 3 \bmod 4$, p n'est donc pas la somme de deux carrés et est donc premier dans $\mathbb{G}$ par la proposition précédente.

Supposons maintenant $p \equiv 1 \bmod 4$. Admettons **provisoirement** l'existence d'une racine carrée de -1 modulo p, c'est à dire d'un entier $r \in [|1, p-1|]$ tel que $r^2 \equiv -1 \bmod p$. On a donc $p \,|\, r^2 + 1 = (r + i)(r - i)$ dans l'anneau $\mathbb{G}$. Mais p ne divise pas $r + i$ et ne divise pas $r - i$ (exercice, ou alors aller voir un peu plus loin). Donc, p ne satisfait pas au lemme d'Euclide : p n'est pas premier dans $\mathbb{Z}[i]$.

Reste à montrer que si $p \equiv 1 \bmod 4$ il existe une racine carrée de -1 modulo p. Considérons la fonction `racine_m1` ci-dessous. Elle prend en paramètre un entier p congru à 1 modulo 4 et renvoie une racine carrée de -1 modulo p.

Remarque. Cette fonction est *très subtile*. **On ne voit pas pour l'instant pourquoi elle fonctionne**.

[2]:
```
def racine_m1(p):
    while True:
        a = random.randint(1, p - 1)
        r = arith.power_mod(a, (p - 1) // 4, p)
        if (r * r + 1) % p == 0: return r
```

Tentons sur des nombres premiers aléatoires. La fonction `random_prime14` renvoie un entier premier congru à 1 modulo 4 ayant N bits.

[3]:
```
def random_prime14(N):
    p = random.randint(2 ** (N - 1), 2 ** N)
    while not p % 4 == 1 or not arith.is_prime(p):
        p = arith.next_prime(p + 1)
    return p
```

[4]:
```
p = random_prime14(100)
r = racine_m1(p)
print('p =', p)
print('r =', r)
print('r^2 + 1 =', (r * r + 1) % p)
```

```
p = 117112979691177730126461702889 3
r = 112640440238843895822476323218 2
r^2 + 1 = 0
```

Pourquoi cette fonction marche-t-elle ? Notons $\mathbb{Z}_p$ le corps des entiers modulo p. Posons $p = 4m+1$. Pour tout $x \in \mathbb{Z}_p$, on a par le petit théorème de Fermat $x^{4m} = x^{p-1} = 1$. Ainsi, $\left(x^{2m}\right)^2 = 1$. Mais l'équation $X^2 = 1$ a au plus deux solutions dans le *corps* $\mathbb{Z}_p$. Comme -1 et 1 sont solutions,

$$x^{2m} = \pm 1$$

Maintenant, l'équation $x^{2m} = 1$ a au plus $2m = \frac{p-1}{2}$ racines dans le corps $\mathbb{Z}_p$. Il y a donc au moins $p-1-2m = 2m$ éléments x de $\mathbb{Z}_p$ tels que $x^{2m} = -1$. Si nous raisonnons sur l'équation $x^{2m} = -1$, elle a au plus $2m$ racines dans le corps $\mathbb{Z}_p$ et il y a donc au moins $p-1-2m = 2m$ éléments x de $\mathbb{Z}_p$ tels que $x^{2m} = 1$.

En résumé, il y a exactement $2m$ éléments $x \in \mathbb{Z}_p$ tels que $x^{2m} = 1$ et exactement $2m$ éléments $x \in \mathbb{Z}_p$ tels que $x^{2m} = -1$.

Prenons $x \in \mathbb{Z}_p$ tel que $x^{2m} = -1$. Soit $r = x^m$. On a alors $r^2 = -1$.

Chaque itération de la boucle `while` de la fonction `racine_m1` a ainsi une probabilité $\frac{1}{2}$ de réussite (i.e. de renvoyer une racine de -1). La probabilité que la boucle effectue au moins n itérations est donc $\frac{1}{2^n}$ et cette probabilité tend vers 0 lorsque n tend vers l'infini. La fonction termine donc presque sûrement et, si elle termine, renvoie bien une racine carrée de -1 modulo p.

2.1.3 1.3 Le théorème des deux carrés

En rassemblant les deux propositions vues dans le paragraphe précédent, nous obtenons un théorème dû à Fermat :

Proposition. Soit p un nombre premier positif congru à 1 modulo 4. Il existe alors $a, b \in \mathbb{N}$ tels que $p = a^2 + b^2$.

Faisons mieux que cela en décrivant **comment calculer** de tels entiers a et b.

Proposition.

- Soit p un nombre premier positif congru à 1 modulo 4.
- Soit $r \in [|0, p-1|]$ tel que $r^2 \equiv -1 \bmod p$.
- Soit enfin $\delta = a + ib = p \wedge (r + i)$.

On a alors $a^2 + b^2 = p$.

Démonstration. Tout d'abord, $\delta \neq 1$. En effet, par définition de r, p divise $(r+i)(r-i)$. En supposant un court instant que p est premier avec $r+i$, alors, par le lemme de Gauss, p divise $r-i$. Cela signifie que $\frac{r}{p} - \frac{1}{p}i$ est un entier de Gauss, contradiction.

Comme δ divise p, $N(\delta)$ divise $N(p)$. Ainsi, a^2+b^2 divise p^2. Comme $\delta \neq 1$ (à inversible près), $N(\delta) = p$ ou p^2. Mais δ divise $r+i$, donc $N(\delta)$ divise $N(r+i) = r^2+1$. Comme $(p-1)^2 \equiv 1 \bmod p$, on a $0 \leq r < p-1$. De là,

$$N(\delta) \mid r^2+1 < (p-1)^2+1 = p^2+2(1-p) < p^2$$

Donc $N(\delta) \neq p^2$.

Finalement, $N(\delta) = a^2+b^2 = p$.

Voici la fonction `somme_carres`. Elle prend en paramètre un entier p qui est supposé premier positif et congru à 1 modulo 4. Elle renvoie un couple (a,b) d'entiers naturels tels que $a^2+b^2=p$.

[5]:
```
def somme_carres(p):
    r = racine_m1(p)
    delta = pgcd(p + 0 * I, r + I)
    return (abs(delta.a), abs(delta.b))
```

[6]:
```
somme_carres(61)
```

[6]:
```
(6, 5)
```

Testons sur des nombres p aléatoires d'une centaine de chiffres en base 10.

[7]:
```
p = random_prime14(300)
print(p)
a, b = somme_carres(p)
print(a, b)
print(a ** 2 + b ** 2 == p)
```

```
1341808972133925418186336319051865453443374479243814465743392303522329404789364192571021477
799958950682933707417739240961027652044674414
837779594736100656425805654552660222181960241
True
```

Le délai qui peut survenir lors de l'évaluation de la cellule ci-dessus est dû à `random_prime14` et pas à `somme_carres`. Notre fonction `somme_carres` est *très* efficace. On peut montrer que le nombre d'opérations effectuées par celle ci est un $O(\log p)$.

> **Remarque 1.** Soit p un entier premier congru à 1 modulo 4. Il existe par ce que nous venons de voir deux entiers naturels a et b tels que $a^2+b^2=p$. A-t-on unicité de a et b ? Oui. En effet, $p = z\overline{z}$ où $z = a+ib$. Remarquons que $N(z) = p$ et donc que z est premier dans $\mathbb{Z}[i]$. Supposons maintenant que $p = c^2+d^2 = (c+id)(c-id)$, avec $c,d \geq 0$. Alors, z divise $(c+id)(c-id)$, mais comme z est premier, z divise l'un des facteurs de ce produit. Par exemple, z divise $c+id$, c'est à dire $c+id = zw$ où $w \in \mathbb{Z}[i]$.

Mais alors $N(c+id) = p = N(z)N(w) = pN(w)$, d'où $N(w) = 1$. Ainsi, w est inversible et $c + id = z, -z, iz$ ou $-iz$. De là, facilement, $(c, d) = (a, b)$ ou $(c, d) = (b, a)$.

Remarque 2. Quels sont les entiers naturels (pas nécessairement premiers) qui sont sommes de deux carrés ? Y a-t-il unicité de la décomposition en somme de carrés ? Nous reviendrons sur ce sujet un peu plus loin.

Notre fonction `somme_carrés` renvoie donc (à symétrie près) **LE** couple (a, b) d'entiers naturels tel que $a^2 + b^2 = p$.

2.1.4 1.4 Entiers de Gauss premiers

Venons en maintenant au cas général.

Proposition. Soit $z = a + ib \in \mathbb{Z}[i]$.

- Si $b = 0$, z est premier si et seulement si a est premier dans $\mathbb{Z}$ et $|a| = 3 \bmod 4$.
- Si $a = 0$, z est premier si et seulement si b est premier dans $\mathbb{Z}$ et $|b| = 3 \bmod 4$.
- Si a et b sont non nuls, z est premier si et seulement si $a^2 + b^2$ est premier dans $\mathbb{Z}$.

Démonstration.

- Nous avons déjà traité le cas où $b = 0$.
- Si $a = 0$, alors, en considérant iz, nous sommes ramenés au premier cas (rappelons que i est inversible dans $\mathbb{G}$, donc z est premier si et seulement si iz est premier).
- Passons au cas intéressant.
  - Supposons $z = ab$ non premier, où a, b ne sont pas inversibles. Alors $N(z) = N(a)N(b)$ et $N(a)$ et $N(b)$ sont des entiers différents de 1. Ainsi, $N(z)$ n'est pas un entier premier.
  - Supposons z premier. On a $z\overline{z} = N(z)$ et donc z divise $N(z)$. Par le lemme d'Euclide dans $\mathbb{Z}[i]$, si z divise un produit alors il divise l'un des facteurs : z divise donc dans $\mathbb{Z}[i]$ un diviseur entier premier de $N(z)$. Appelons p un tel diviseur premier. De là $N(z)$ divise $N(p) = p^2$ dans $\mathbb{Z}$. Supposons un court instant que $N(z) = p^2$. Écrivons $p = zw$ où $w \in \mathbb{Z}[i]$. On a $N(p) = N(z)N(w)$, ou encore $p^2 = p^2N(w)$. Ainsi, $N(w) = 1$ et donc w est inversible. De là, $z = \pm p$ ou $z = \pm ip$, ce qui contredit le fait que $z = a + ib$ avec $a, b \neq 0$. Ainsi, $N(z) = p$ est un entier premier.

Il est donc « facile » de savoir si un entier de Gauss est premier. Plus exactement, la proposition précédente nous montre qu'il suffit de savoir tester la primalité d'un entier relatif. La fonction `est_premier_gauss` fait le travail. Elle renvoie `True` si z est premier dans $\mathbb{Z}[i]$, et `False` sinon.

[8]:
```
def est_premier_gauss(z):
    a, b = abs(z.a), abs(z.b)
    if a != 0 and b != 0:
        p = z.norm()
        if p == 1: return False
        elif p == 2: return True
        elif p % 4 != 1: return False
        else: return arith.is_prime(z.norm())
    elif a == 0:
```

```
        return arith.is_prime(b) and b % 4 == 3
    else:
        return arith.is_prime(a) and a % 4 == 3
```

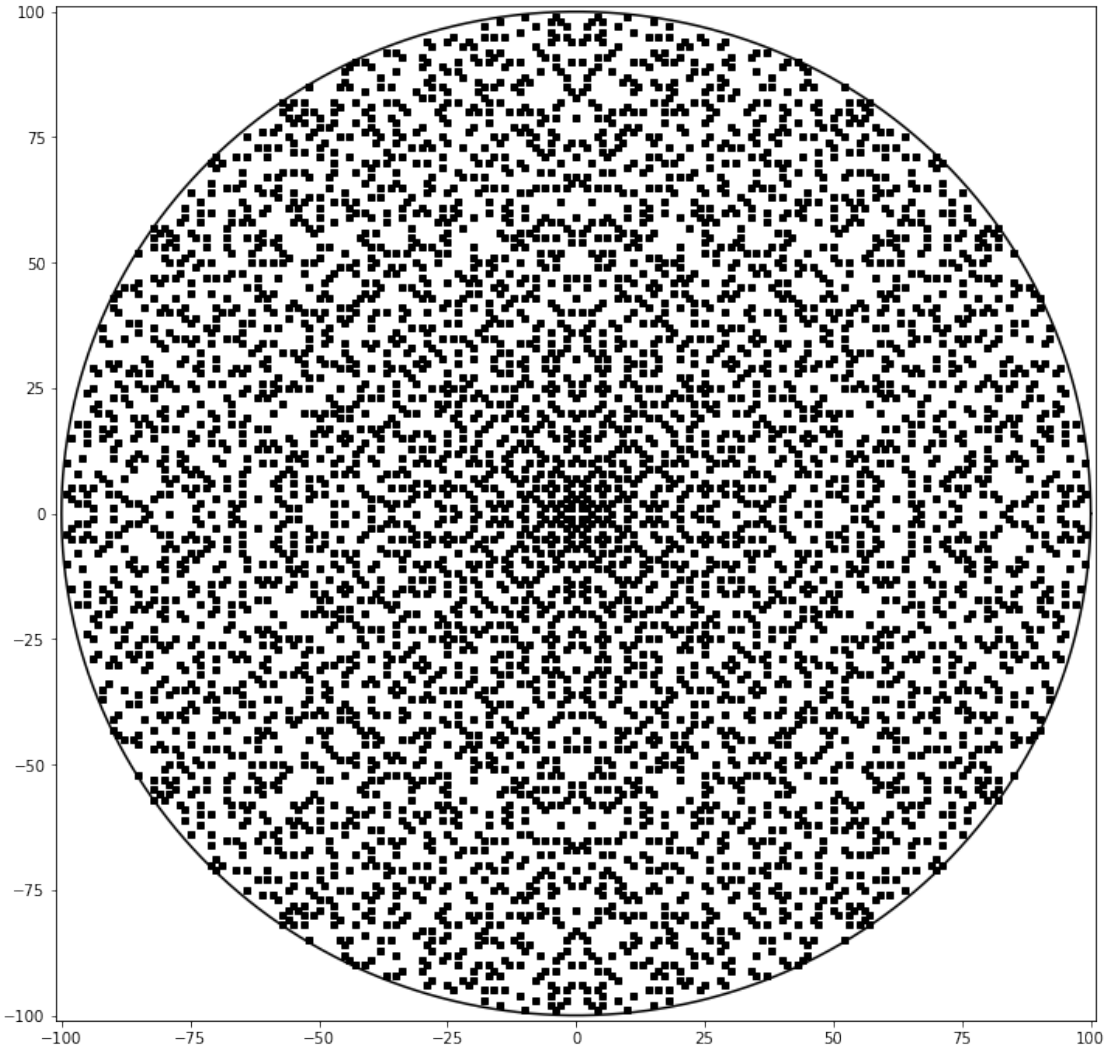
La fonction `plot_primes` prend en paramètre un entier $N > 0$. Elle affiche les entiers de Gauss premiers z tels que $|z| \leq N$.

[9]:
```
def plot_primes(N, ms=2, dbg=True):
    xs = []
    ys = []
    count = 0
    for a in range(-N, N + 1):
        for b in range(-N, N + 1):
            if a ** 2 + b ** 2 < N ** 2 and est_premier_gauss(a + I * b):
                xs.append(a)
                ys.append(b)
                count += 1
    plt.xlim(-N-1, N+1)
    plt.ylim(-N-1, N+1)
    plt.plot(xs, ys, 'sk', markersize=ms)
    ts = [2 * k * math.pi / 500 for k in range(501)]
    xs = [N * math.cos(t) for t in ts]
    ys = [N * math.sin(t) for t in ts]
    plt.plot(xs, ys, 'k')
    if dbg: print('Nombre d\'entiers premiers :', count)
```

[10]:
```
plt.rcParams['figure.figsize'] = (12, 12)
```

[11]:
```
plot_primes(100, ms=4)
```

```
Nombre d'entiers premiers : 4928
```



Si l'envie vous prenait d'explorer des zones ailleurs qu'autour de 0, voici une fonction `plot_primes2` qui prend 4 paramètres a', a'', b', b'' et affiche les entiers de Gauss premiers $a+ib$ tels que $a' \leq a \leq a''$ et $b' \leq b \leq b''$.

[12]:
```
def plot_primes2(amin, amax, bmin, bmax, ms=2):
    xs = []
    ys = []
    count = 0
    for a in range(amin, amax + 1):
        for b in range(bmin, bmax + 1):
            if est_premier_gauss(a + I * b):
                xs.append(a)
                ys.append(b)
                count += 1
    plt.xlim(amin, amax)
```

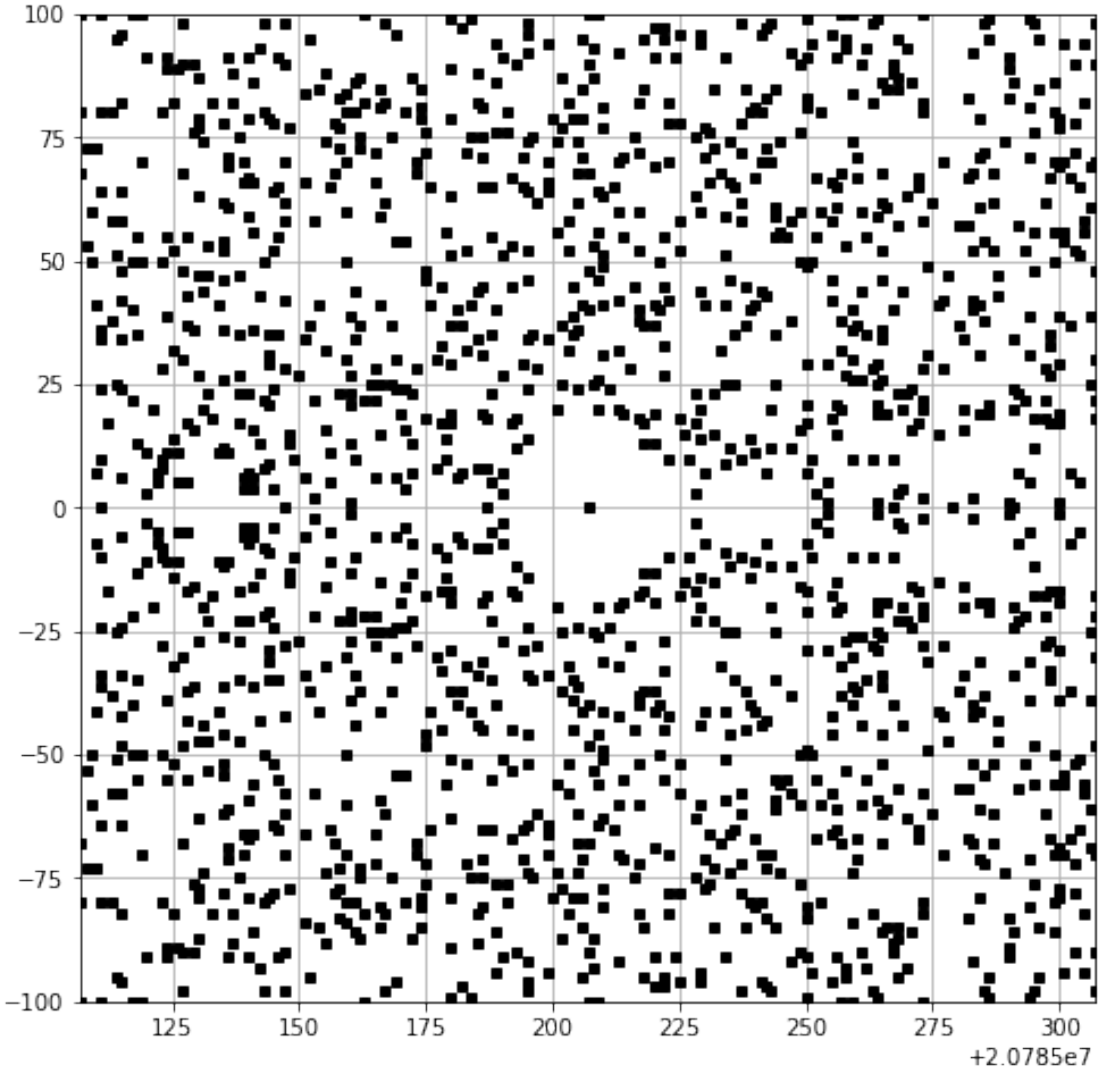
```
    plt.ylim(bmin, bmax)
    plt.plot(xs, ys, 'sk', markersize=ms)
    print('Nombre d\'entiers premiers :', count)
    plt.grid()
```

Voici une région intéressante. Nous y voyons un entier de Gauss isolé. Il s'agit en l'occurence de l'entier 20785207.

[13]:
```
plt.rcParams['figure.figsize'] = (8, 8)
```

[14]:
```
a = 20785207
d = 100
plot_primes2(a - d, a + d, -d, d, ms=4)
```

```
Nombre d'entiers premiers : 1539
```



2.2 2. Factorisation

2.2.1 2.1 Décomposition en produit de nombres premiers

L'anneau des entiers de Gauss étant un anneau principal, il est aussi *factoriel* : tout élément non nul de $\mathbb{Z}[i]$ s'écrit de façon « essentiellement unique » comme un produit de nombres premiers. Il y a deux raisons aux guillemets :

- L'ordre des facteurs, bien évidemment.
- Mais aussi la possibilité de remplacer n'importe quel nombre premier p dans la décomposition par up, où p est inversible, puis de multiplier un autre élément p' de la décomposition par u^{-1}.

La première source de non unicité n'est pas gênante. Pour supprimer la seconde source, on peut remarquer qui si p est premier dans $\mathbb{Z}[i]$, les autres nombres premiers associés à p sont $-p$, ip et $-ip$. Un et un seul de ces quatre nombres a une partie réelle strictement positive et une partie imaginaire positive ou nulle, et c'est celui-ci que nous choisirons. Munis de cette convention, notons

$$\mathcal{P}^+ = \{q \in \mathbb{Z}[i], q \text{ premier}, \ \mathrm{Re}\ q > 0, \ \mathrm{Im}\ q \geq 0\}$$

Nous avons alors le théorème suivant.

Proposition. Tout $z \in \mathbb{Z}[i] \setminus \{0\}$ s'écrit de façon unique

$$z = u \prod_{q \in \mathcal{P}^+} q^{\nu_q}$$

où $u \in \mathcal{U}$ et les $\nu_q \in \mathbb{N}$ sont des entiers naturels presque tous nuls. ν_q est la *valuation q-adique* de l'entier de Gauss z.

2.2.2 2.2 Facteurs premiers d'un entier de Gauss

Nous allons adopter une démarche naïve mais pas trop quand même, qui nous permettra de factoriser des entiers de Gauss de taille raisonnable. Tout d'abord, écrivons une fonction renvoyant la valuation q-adique de z.

[15]:
```
def valuation(z, q):
    mu = 0
    while z % q == 0:
        z = z // q
        mu = mu + 1
    return mu
```

[16]:
```
valuation((2 + I) ** 3 * (5 - 2 * I), 2 + I)
```

[16]: 3

La fonction `facteurs_premiers` renvoie la liste des couples (q, ν) où $q \in \mathcal{P}^+$ divise z et ν est la valuation q-adique de z.

On commence par factoriser la norme $N(z)$ de z dans $\mathbb{Z}$. En effet, si q divise z, alors $N(q)$ divise $N(z)$. Pour tous les diviseurs premiers p de $N(z)$ dans $\mathbb{Z}$, on examine un certain nombre de cas.

- Si $p = 2$, alors $1 + i$ divise z.
- Si $p \equiv 3 \bmod 4$, alors p est premier dans $\mathbb{Z}[i]$.
- Si $p \equiv 1 \bmod 4$, alors p n'est pas premier dans $\mathbb{Z}[i]$, et s'écrit de façon essentiellement unique $p = a^2 + b^2$, $a, b \in \mathbb{N}$. On a alors $p = -iq_1q_2$ où $q_1 = a + ib$ et $q_2 = b + ia$ sont premiers dans $\mathbb{Z}[i]$ (car de norme p). Il ne reste plus qu'à tester la divisibilité de z par q_1 et q_2.

[17]:
```
def facteurs_premiers(z):
    ds = []
    fs = arith.factor(z.norm())
    for p, nu in fs:
        if p == 2 and z % (1 + I) == 0:
            mu = valuation(z, 1 + I)
            z = z // (1 + I) ** mu
            ds.append((1 + I, mu))
        elif p % 4 == 3 and nu % 2 == 0 and z % p == 0:
            mu = valuation(z, p)
            z = z // p ** mu
            ds.append((Gauss(p), mu))
        elif p % 4 == 1:
            a, b = somme_carres(p)
            w1 = a + I * b
            w2 = b + I * a
            if z % w1 == 0:
                mu = valuation(z, w1)
                z = z // w1 ** mu
                ds.append((w1, mu))
            if  z % w2 == 0:
                mu = valuation(z, w2)
                z = z // w2 ** mu
                ds.append((w2, mu))
    return ds
```

La fonction `verif_factors` permet de tester que la liste renvoyée par `facteurs_premiers` convient.

[18]:
```
def verif_factors(fs, z):
    s = 1
    for p, mu in fs: s = s * p ** mu
    return ((z // s).inversible(), z % s == 0)
```

[19]:
```
z = random_gauss(100)
print(z)
fs = facteurs_premiers(z)
print(fs)
print(verif_factors(fs, z))
```

```
<-20, 2>
[(<1, 10>, 1), (<1, 1>, 2)]
(True, True)
```

Voici un test « exhaustif », qui factorise tous les entiers de Gauss $a + ib$ où $-N \leq a, b \leq N$ (sauf 0 !) et vérifie la correction de la factorisation. Chaque erreur (s'il y en a) affiche l'entier de Gauss qui pose problème. Notre souhait est évidemment que cette fonction n'affiche rien.

[20]:
```
def test_exhaustif(N):
    for a in range(-N, N +1):
        for b in range(-N, N + 1):
            if a != 0 or b != 0:
                z = a + I * b
                fs = facteurs_premiers(z)
                if verif_factors(fs, z) != (True, True):
                    print(z, fs)
    print('ok')
```

[21]:
```
test_exhaustif(50)
```

```
ok
```

On est très contents …

2.2.3 2.3 Diviseurs d'un entier de Gauss

Il est maintenant facile d'écrire une fonction qui renvoie tous les diviseurs d'un entier de Gauss non nul. La fonction `diviseurs_aux` prend en paramètre une liste $[(q_j, \nu_j), 1 \leq k \leq j]$ (créée par exemple par un appel à `facteurs_premiers`). Elle renvoie la liste des produits

$$\prod_{j=1}^{k} q_j^{\alpha_j}$$

où $0 \leq \alpha_j \leq \nu_j$.

[22]:
```
def diviseurs_aux(fs):
    if fs == []: return [1]
    else:
        ds0 = diviseurs_aux(fs[1:])
        q, nu = fs[0]
        ds1 = [q ** k * d for k in range(nu + 1) for d in ds0]
```

```
    return ds1
```

La fonction `diviseurs0` enchaîne un appel à `diviseurs_aux` et un appel à `facteurs_premiers`.

[23]:
```
def diviseurs0(z):
    ds1 = diviseurs_aux(facteurs_premiers(z))
    return ds1
```

[24]:
```
print(diviseurs0(123 + 456 * I))
```

```
[<1, 0>, <3, 0>, <1, 2>, <3, 6>, <69, 14>, <207, 42>, <41, 152>, <123, 456>]
```

Nous n'avons pour l'instant qu'un quart des diviseurs, car nous n'avons pris en considération que les nombres premiers de $\mathcal{P}^+$. Pour avoir tous les diviseurs, on multiplie les éléments de la liste renvoyée par `diviseurs0` par -1, i et $-i$ et on concatène le tout.

Il peut être aussi raisonnable de trier d'une façon ou d'une autre la liste de ces diviseurs. Choisissons l'ordre lexicographique.

[25]:
```
def diviseurs(z):
    ds1 = diviseurs0(z)
    ds2 = [-z for z in ds1]
    ds3 = [I * z for z in ds1]
    ds4 = [-I * z for z in ds1]
    ds = ds1 + ds2 + ds3 + ds4
    ds.sort(key=lambda z: z.b)
    ds.sort(key=lambda z: z.a)
    return ds
```

[26]:
```
print(diviseurs(123 + 456 * I))
```

```
[<-456, 123>, <-207, -42>, <-152, 41>, <-123, -456>, <-69, -14>, <-42, 207>,
<-41, -152>, <-14, 69>, <-6, 3>, <-3, -6>, <-3, 0>, <-2, 1>, <-1, -2>, <-1, 0>,
<0, -3>, <0, -1>, <0, 1>, <0, 3>, <1, 0>, <1, 2>, <2, -1>, <3, 0>, <3, 6>, <6,
-3>, <14, -69>, <41, 152>, <42, -207>, <69, 14>, <123, 456>, <152, -41>, <207,
42>, <456, -123>]
```

2.2.4 2.4 Dessiner les diviseurs

Terminons par des dessins. Écrivons une fonction qui prend un entier de Gauss en paramètre et affiche ses diviseurs.

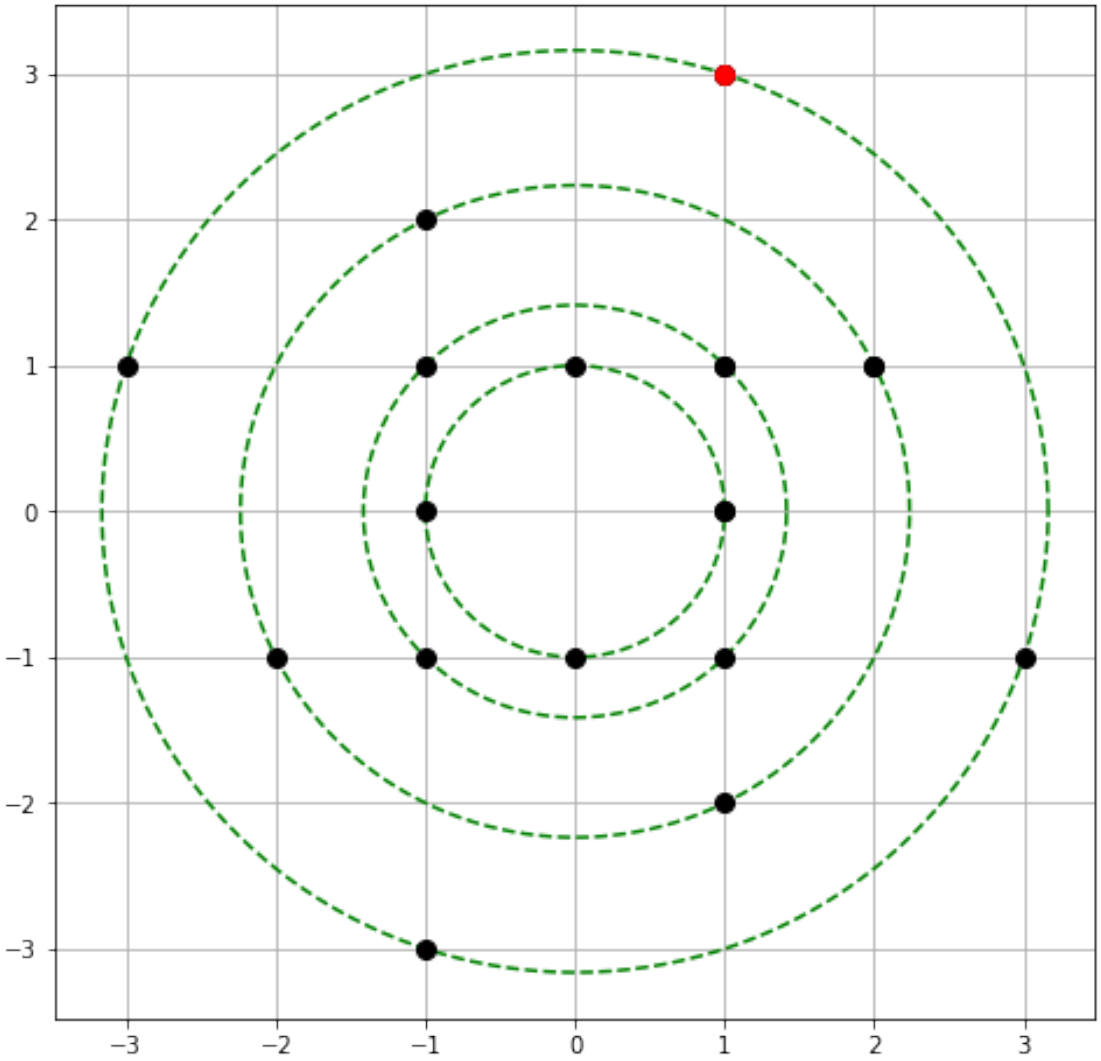
[27]:
```
def plot_circle(R):
    ts = [2 * k * math.pi / 500 for k in range(501)]
    xs = [R * math.cos(t) for t in ts]
    ys = [R * math.sin(t) for t in ts]
    plt.plot(xs, ys, '--g')
```

[28]:
```
def plot_diviseurs(z, circ=True, ms=4):
    zs = diviseurs0(z)
    for d in zs:
        if circ: plot_circle(math.sqrt(d.norm()))
        ws = [d, I * d, -d, -I * d, d]
        xs = [w.a for w in ws]
        ys = [w.b for w in ws]
        plt.plot(xs, ys, 'ok', markersize=ms)
    plt.plot([z.a], [z.b], 'or', markersize=ms)
    plt.grid()
```

[29]:
```
plt.rcParams['figure.figsize'] = (8, 8)
```

Voic les diviseurs de $1 + 3i$. Il y en a 16.

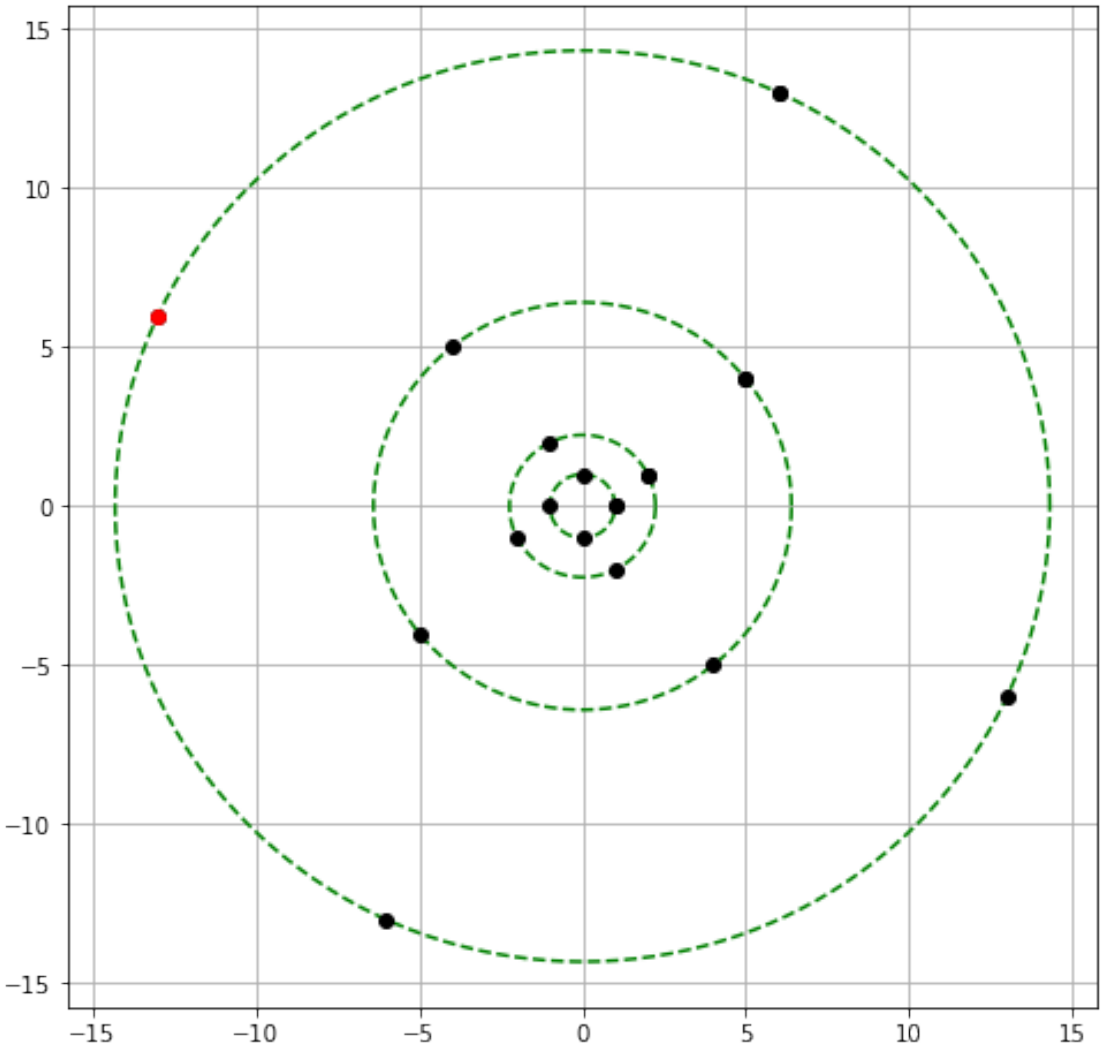
[30]:
```
plot_diviseurs(1 + 3 * I, circ=True, ms=8)
```



Voici une cellule pour s'amuser. On obtient une fois de temps en temps quelque chose d'intéressant …

[31]:
```
z = random_gauss(100)
print('z =', z)
print('Nombre de diviseurs :', len(diviseurs(z)))
plot_diviseurs(z, circ=True, ms=6)
```

```
z = <-13, 6>
Nombre de diviseurs : 16
```



2.3 3. Promenades dans les entiers de Gauss premiers

Ce qui va suivre est inspiré de l'article

A Stroll Through the Gaussian Primes (American Mathematical Monthly, Vol. 105, n° 4, April 1998) - Ellen Gethner, Stan Wagon, Brian Wick.

2.3.1 3.1 Le problème

Soit $\mathcal{P}$ l'ensemble des entiers de Gauss premiers. Soit d un réel strictement positif. Est-il possible de trouver dans $\mathcal{P}$ un chemin « allant de l'origine à l'infini » tel que deux sommets successifs de ce chemin soient à distance inférieure ou égale à d ?

Il est clair que ceci ne serait pas possible dans l'ensemble des **entiers relatifs** premiers, car il existe dans $\mathbb{Z}$ des intervalles de longueur aussi grande que l'on désire et ne contenant aucun nombre premier. Par exemple, si $n > d$, les entiers $(n+1)! + k$, $k \in [|2, n+1|]$, sont tous composés. Il est donc impossible de sauter d'un nombre premier inférieur à $(n+1)!$ à un nombre premier supérieur à $(n+1)! + n + 1$ par un saut de longueur inférieure à d.

Ce problème a été posé pour la première fois par Basil Gordon en 1962. *Il est non résolu à ce jour.*

Nous allons écrire une petite simulation permettant de voir ce qui se passe pour des petites valeurs de d.

2.3.2 3.2 Les voisins d'un entier de Gauss

Soit z un entier de Gauss premier. Soit $d > 0$. La fonction `voisins` renvoie tous les entiers de Gauss premiers différents de z dont la distance à z est inférieure ou égale à d.

[32]:
```
def voisins(z, d):
    a, b = z.a, z.b
    vs = []
    n = int(d) + 1
    for x in range(a - n, a + n + 1):
        for y in range(b - n, b + n + 1):
            w = x + I * y
            if (w - z).norm() <= d ** 2 and w != z and est_premier_gauss(w): vs.
↪append(w)
    return vs
```

2.3.3 3.3 La promenade

L'idée de notre promenade est simple. Nous partons d'un ensemble d'origine qui contient les 4 nombres premiers les plus proches de 0 : $\pm 1 \pm i$. On effectue ensuite n fois l'opération suivante :

- Prendre les voisins des nombres premiers obtenus à l'itération précédente (les éléments de `lev1` dans le code ci-dessous) et qui ne sont pas des sommets obtenus à l'itération précédant l'itération précédente (`lev0`).

- Joindre ces voisins aux sommets correspondants de `lev1`.
- En notant `lev2` la liste de tous ces voisins, remplacer `lev0` et `lev1` par `lev1` et `lev2`.

Il ne reste plus qu'à tracer les segments.

On sort automatiquement de la boucle si `lev2` est vide : cela signifie que la promenade est terminée.

La fonction `promenade` se contente de faire une boucle. Tout le travail est fait par la fonction `step`.

[33]:
```
def promenade(d, n, grd=False):
    t1 = time.time()
    lev0 = set([])
    lev1 = set([1 + I, 1 - I, -1 + I, -1 - I])
    segts = []
    max_nrm = 2
    for k in range(n):
        (lev2, s) = step(lev0, lev1, d)
        segts = segts + s
        lev0, lev1 = lev1, lev2
        if len(lev2) != 0:
            nrm = max([w.norm() for w in lev2])
            max_nrm = max(max_nrm, nrm)
        else:
            print('\nPromenade achevée en %d étapes' % k)
            break
        print(len(lev1), end=' ')
    t2 = time.time()
    plot_segments(segts)
    N = int(math.sqrt(max_nrm)) + 5
    plot_primes(N, ms=1, dbg=False)
    if grd: plt.grid()
    plt.savefig('walk.png')
    print()
    print('Temps de calcul : %5.2fs' % (t2 - t1))
```

La fonction `step` fait ce qui a été décrit ci-dessus.

[34]:
```
def step(lev0, lev1, d):
    s = []
    lev2 = set([])
    for z in lev1:
        ws = voisins(z, d)
        for w in ws:
            if w not in lev0:
                s.append((z, w))
                if w not in lev1: lev2.add(w)
    return (lev2, s)
```

Enfin, la fonction `plot_segments`, sans surprise, affiche … des segments contenus dans une liste s. La lsite s cntient des couples d'entiers de Gauss qui sont les extrémités des segments à tracer.

[35]:
```
def plot_segments(s):
    xs = []
    ys = []
    for z, w in s:
        a, b = z.a, z.b
        c, d = w.a, w.b
        xs.extend([a, c, None])
        ys.extend([b, d, None])
    plt.plot(xs, ys, 'k')
```

2.3.4 3.4 Allons-y

Avant de nous lancer, faisons une petite remarque. Soient $z = x + iy$ et $z' = x' + iy'$ deux entiers de Gauss premiers.

Supposons z et z' ni réels ni imaginaires purs. On a

$$|z' - z|^2 = (x' - x)^2 + (y' - y)^2 = (x^2 + y^2) + (x'^2 + y'^2) - 2(xx' + yy')$$

Les entiers $x^2 + y^2$ et $x'^2 + y'^2$ sont premiers, et donc impairs, sauf s'ils valent 2. Mais ceci ne peut survenir que si $z = \pm 1 \pm i$ ou $z' = \pm 1 \pm i$. Mis à part ces deux cas, $|z' - z|^2$ est donc pair (ce qui entraîne que $x' - x$ et $y' - y$ sont de même parité). Les valeurs intéressantes de d sont donc des racines carrées de nombres pairs.

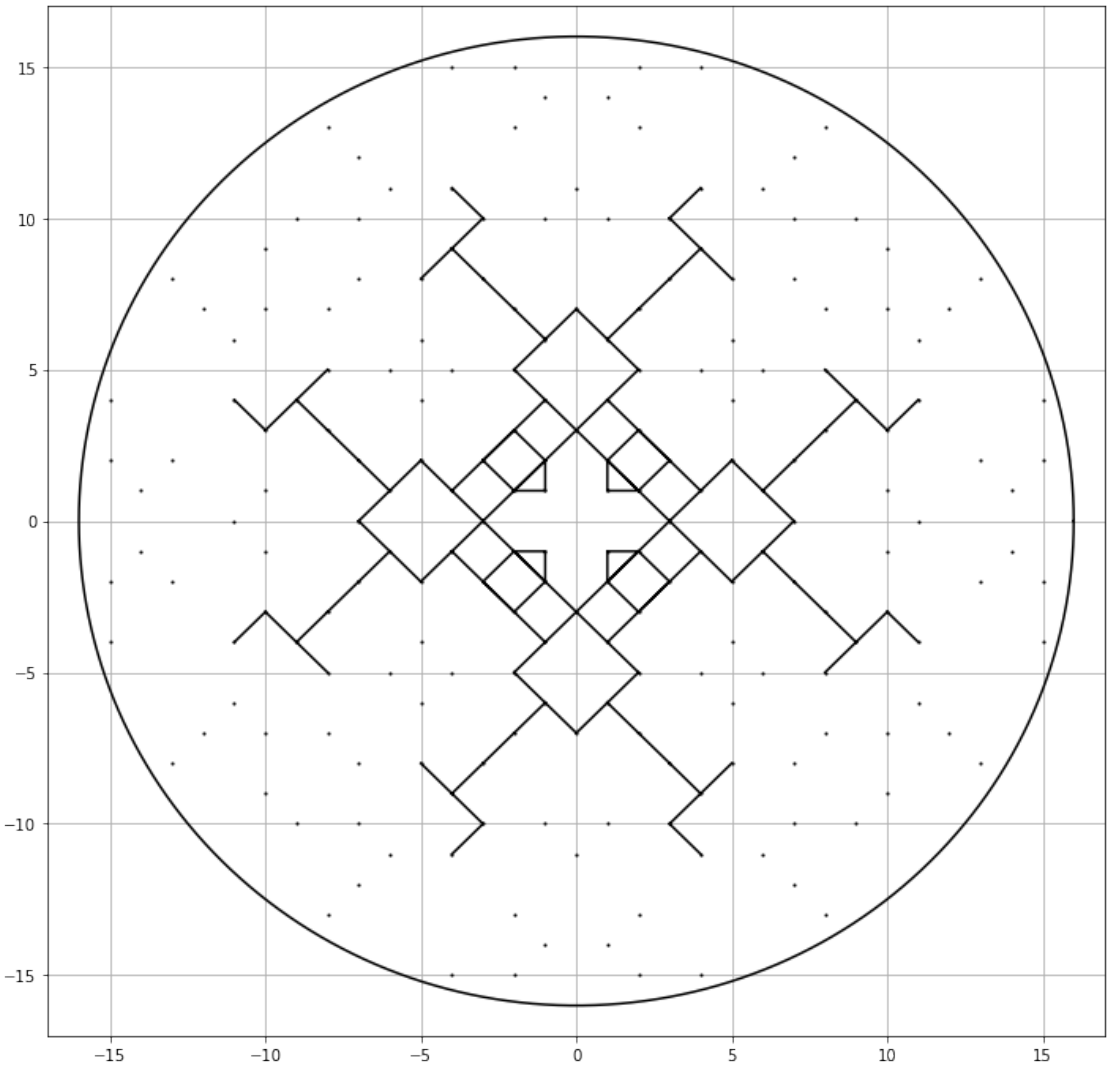
Lançons donc la promenade avec $d = \sqrt{2}$ (je vous laisse tenter $d = 1$).

[36]:
```
plt.rcParams['figure.figsize'] = (12, 12)
```

[37]:
```
promenade(math.sqrt(2), 1000, grd=True)
```

```
8 12 8 8 8 12 8 8 16 8
Promenade achevée en 10 étapes

Temps de calcul :  0.03s
```



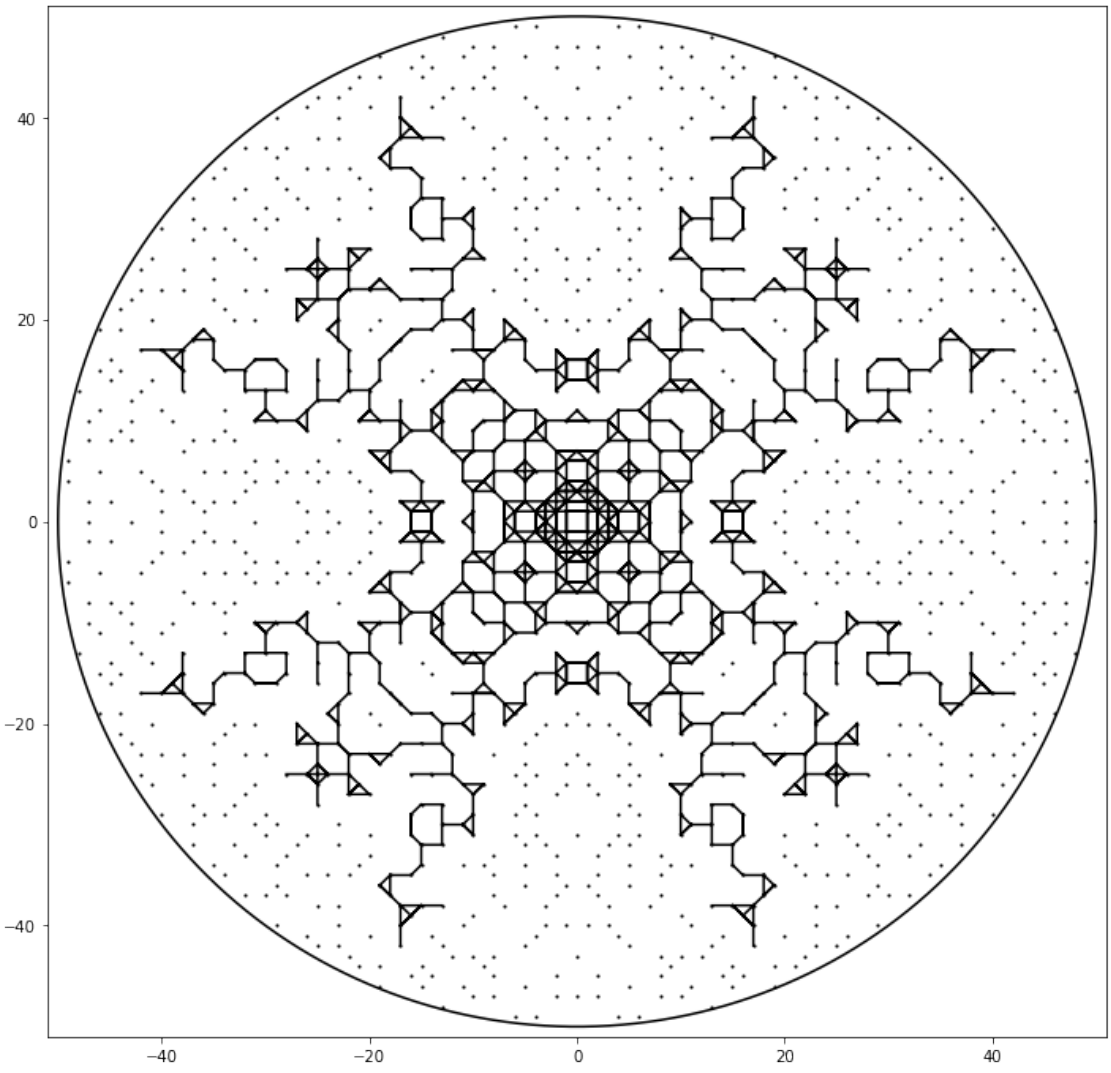
La promenade s'achève au bout d'une dizaine d'étapes.

Tentons $d = 2 = \sqrt{4}$, ce sera déjà beaucoup plus intéressant.

[38]:
```
promenade(2, 1000)
```

```
8 20 16 20 16 24 24 20 16 8 16 24 16 24 40 16 24 32 56 24 40 16 24 32 32 24 24 8
8 16 8 24 16
Promenade achevée en 33 étapes

Temps de calcul :  0.44s
```



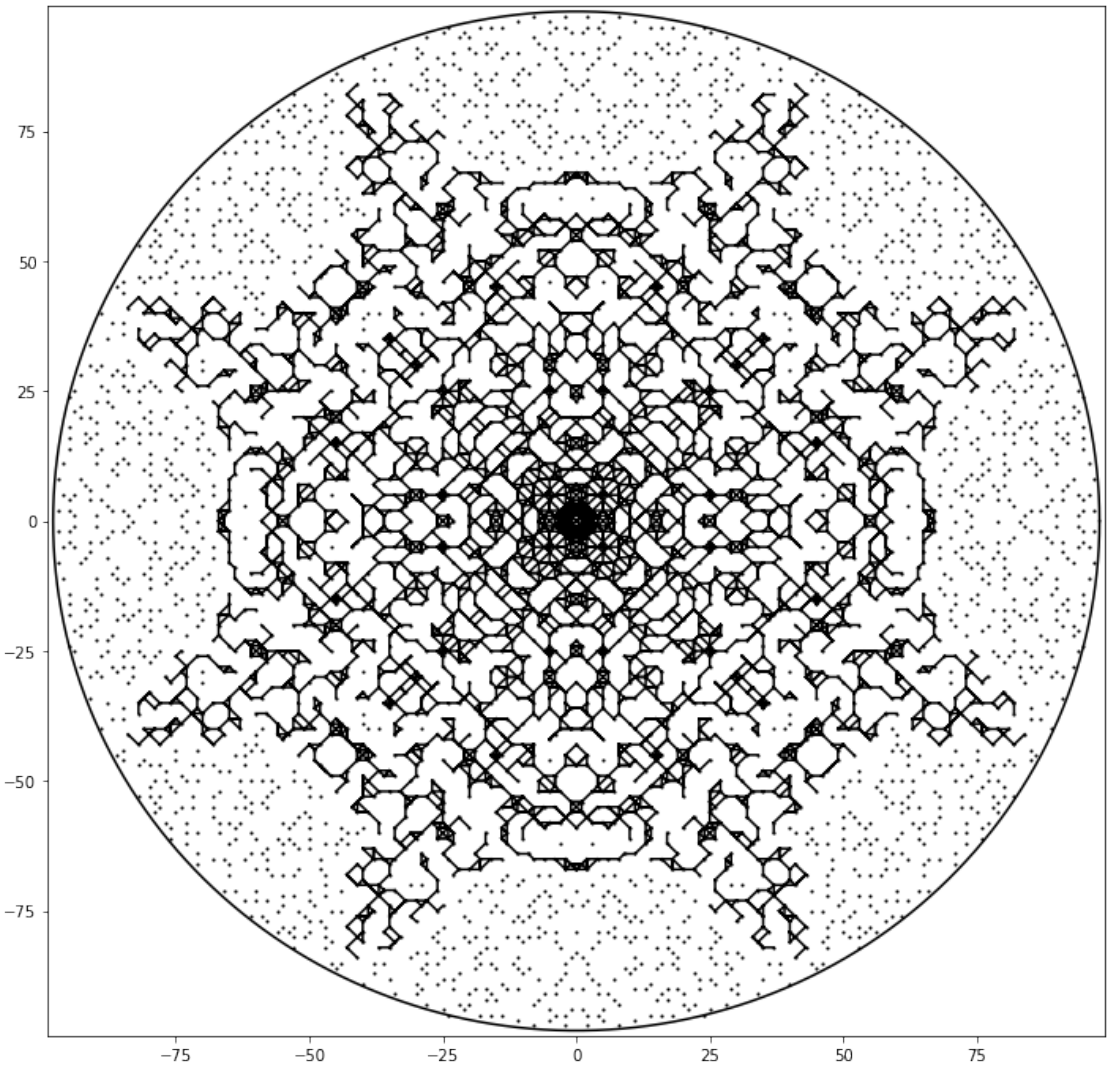
Une trentaine d'étapes permettent d'obtenir tous les nombre joignables à partir de nos nombres d'origine $\pm 1 \pm i$.

Je passe $\sqrt{6}$ qui n'apporte pas grand chose de nouveau (essayez). Lançons avec $\sqrt{8}$.

[39]:
```
promenade(math.sqrt(8), 1000)
```

```
20 24 28 32 48 28 40 48 44 48 40 48 56 56 64 72 92 68 48 48 64 88 32 48 80 144
96 80 96 68 68 80 36 16 16 32 32 40 72 92 88 88 72 56 40 48 56 64 48 64 64 40 8
16 8
Promenade achevée en 55 étapes

Temps de calcul :  2.34s
```



2.3.5 3.5 De plus grandes valeurs de d

Pour des valeurs de d au-delà de $\sqrt{10}$, on obtient *beaucoup plus* de points. Tracer des segments devient inutile, nous allons réécrire la fonction de tracé pour faire apparaître uniquement les points obtenus.

Pour gagner un facteur 8 dans le temps de calcul, on ne trace qu'un huitième du dessin : les points $x + iy$ tels que $0 \leq y \leq x$ (le reste du dessin s'en déduit par symétrie). Si vous voulez avoir la totalité du dessin, modifiez le code de `step2` en conséquence.

[40]:
```
def step2(lev0, lev1, d):
    lev2 = set([])
    for z in lev1:
        ws = voisins(z, d)
```

```
        for w in ws:
            if w.b >=0 and w.b <= w.a and w not in lev0 and w not in lev1:
                lev2.add(w)
    return lev2
```

[41]:
```
def promenade2(d, n, ms=0.1):
    t1 = time.time()
    lev0 = set([])
    lev1 = set([1 + I])
    xs = [1]
    ys = [1]
    max_nrm = 2
    for k in range(n):
        lev2 = step2(lev0, lev1, d)
        ajouter_points(xs, ys, lev2)
        lev0, lev1 = lev1, lev2
        if len(lev2) != 0:
            nrm = max([w.norm() for w in lev2])
            max_nrm = max(max_nrm, nrm)
        else: break
        if k % 10 == 0: print(k, ':', len(lev1), end=', ')
    print()
    print('Nombre de points :', len(xs))
    print('Norme max :', max_nrm)
    print('Module max : %.2f' % math.sqrt(max_nrm))
    t2 = time.time()
    print('Temps de calcul : %5.1fs' % (t2 - t1))
    plt.plot(xs, ys, 'sk', markersize=ms)
    plt.savefig('walk.png')
```

[42]:
```
def ajouter_points(xs, ys, zs):
    for z in zs:
        xs.append(z.a)
        ys.append(z.b)
```

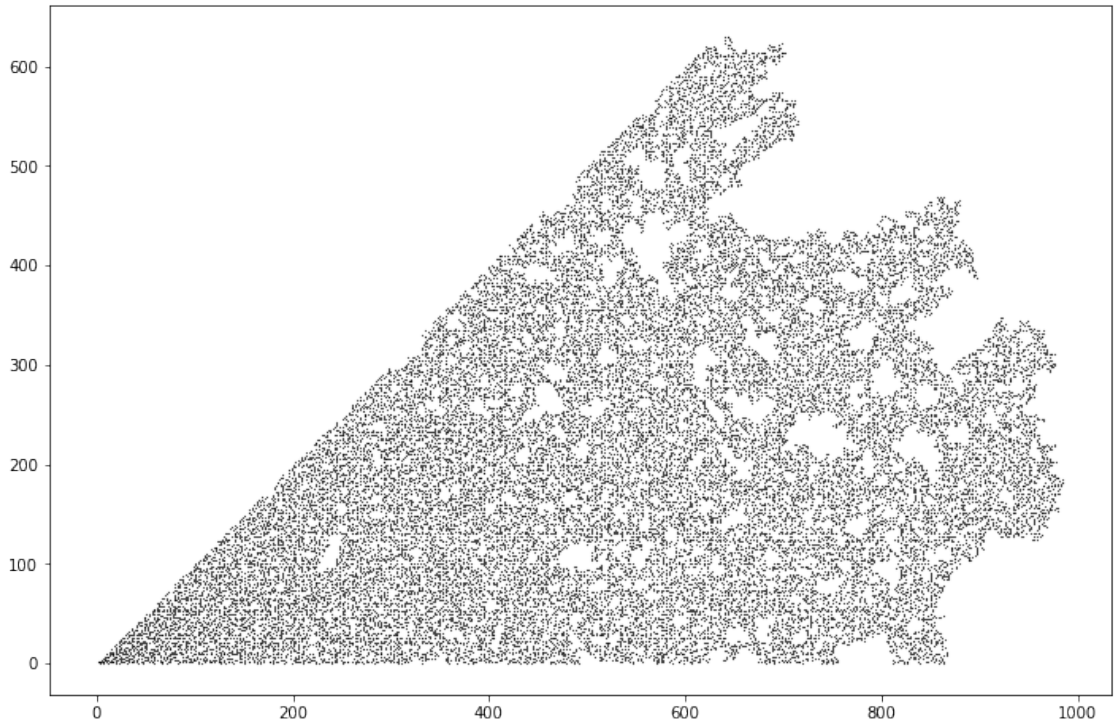
Lançons $d = \sqrt{10}$. Et soyons patients, un peu plus de 500 étapes sont nécessaires (une quarantaine de secondes sur ma machine) …

[43]:
```
plt.rcParams['figure.figsize'] = (12, 8)
```

[44]:
```
promenade2(math.sqrt(10), 1000, ms=0.2)
```

```
0 : 4, 10 : 11, 20 : 16, 30 : 25, 40 : 23, 50 : 23, 60 : 51, 70 : 36, 80 : 47,
90 : 43, 100 : 39, 110 : 48, 120 : 70, 130 : 76, 140 : 55, 150 : 72, 160 : 54,
170 : 61, 180 : 79, 190 : 75, 200 : 74, 210 : 71, 220 : 62, 230 : 73, 240 : 54,
250 : 55, 260 : 53, 270 : 41, 280 : 86, 290 : 74, 300 : 76, 310 : 91, 320 : 112,
330 : 85, 340 : 98, 350 : 86, 360 : 49, 370 : 68, 380 : 86, 390 : 71, 400 : 119,
410 : 116, 420 : 75, 430 : 122, 440 : 69, 450 : 36, 460 : 47, 470 : 47, 480 :
```

```
29, 490 : 41, 500 : 24, 510 : 8, 520 : 11, 530 : 2,
Nombre de points : 31221
Norme max : 1049297
Module max : 1024.35
Temps de calcul :  40.9s
```



Que voyons-nous sur le dessin ci-dessus ? Il y a un certains nombre de « trous », que les auteurs de l'article appellent des *moats* (douve, fossé). La taille de ces fossés est la clé de la réponse à la question posée au début de cette section : L'existence d'une promenade à l'infini est subordonnée à l'exsitence de grands fossés. Question non résolue à ce jour, comme je l'ai dit un peu plus haut.

Vous pouvez tenter $d = 4$ mais il faudra être *très* patients : le calcul prend une dizaine de minutes sur ma machine et nécessite 2820 étapes. Cela dit, le résultat en vaut la peine.

[45]:
```
# promenade2(4, 10000)
```

Pour de plus grandes valeurs de d (la suivante serait $\sqrt{18}$), les temps de calcul deviennent prohibitifs. Inutile d'espérer avoir le dessin en moins de quelques heures …

2.4 4. Sommes de deux carrés

Terminons ce notebook par une question que nous ne pouvons pas passer sous silence.

Soit $n \in \mathbb{N}$. Nous allons répondre aux questions suivantes :

- À quelle condition n est-il la somme de deux carrés ?
- Comment déterminer un couple (a, b) d'entiers tel que $n = a^2 + b^2$?
- Comment déterminer tous les couples (a, b) d'entiers tels que $n = a^2 + b^2$?

Notons Γ l'ensemble des entiers naturels n vérifiant :

$$\exists a, b \in \mathbb{N}, n = a^2 + b^2$$

Remarquons que

$$\Gamma = \{N(z), z \in \mathbb{Z}[i]\}$$

2.4.1 4.1 Une caractérisation

Proposition. L'ensemble Γ est stable pour la multiplication.

Démonstration. Soient $m = N(w)$ et $n = N(z)$ deux éléments de Γ, où $w, z \in \mathbb{Z}[i]$. On a $mn = N(w)N(z) = N(wz) \in \Gamma$.

Proposition. Soit p un entier premier.

- Si $p = 2$, $p \in \Gamma$.
- Si $p \equiv 1 \bmod 4$, $p \in \Gamma$.
- Si $p \equiv 3 \bmod 4$, $p \notin \Gamma$ et $p^2 \in \Gamma$.

Démonstration.

- $2 = 1^2 + 1^2$.
- Si $p \equiv 1 \bmod 4$, nous avons déjà montré que $p \in \Gamma$. Nous avons également vu qu'il existe (à symétrie près) un unique couple (a, b) tel que $p = a^2 + b^2$.
- Si $p \equiv 3 \bmod 4$, nous avons déjà montré que $p \notin \Gamma$. Bien entendu, $p^2 = p^2 + 0^2 \in \Gamma$.

Proposition. Soit $n \in \mathbb{N}$. Écrivons $n = \prod_{p \in \mathcal{P}} p^{\nu_p}$ où $\mathcal{P}$ est l'ensemble des entiers premiers et les ν_p sont des entiers presque tous nuls. Alors, $n \in \Gamma$ si et seulement si pour tout p congru à 3 modulo 4, ν_p est pair.

Démonstration.

($\Rightarrow$) Supposons $n = a^2 + b^2$. Soit $p \equiv 3 \bmod 4$ un nombre premier qui divise n. Nous savons déjà que p est premier dans $\mathbb{Z}[i]$. Dans cet anneau, p divise $(a + ib)(a - ib)$, et par le lemme d'Euclide, p divise $a + ib$ ou p divise $a - ib$. Il en résulte facimement que p divise a et b. En conséquence, p divise $a - ib$ **et** $a + ib$, donc p^2 divise $n = (a - ib)(a + ib)$. On en déduit facilement que $\nu_p(n)$ est pair.

($\Leftarrow$) La proposition précédente et la stabilité de Γ par multiplication montrent qu'un entier n vérifiant la condition est bien dans Γ.

[46]:
```
def est_somme_carres(n):
    if n == 0: return True
    else:
        fs = arith.factor(n)
```

```
        for p, nu in fs:
            if p % 4 == 3 and nu % 2 == 1: return False
        return True
```

[47]:
```
print([n for n in range(1000) if est_somme_carres(n)])
```

```
[0, 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 26, 29, 32, 34, 36, 37, 40,
41, 45, 49, 50, 52, 53, 58, 61, 64, 65, 68, 72, 73, 74, 80, 81, 82, 85, 89, 90,
97, 98, 100, 101, 104, 106, 109, 113, 116, 117, 121, 122, 125, 128, 130, 136,
137, 144, 145, 146, 148, 149, 153, 157, 160, 162, 164, 169, 170, 173, 178, 180,
181, 185, 193, 194, 196, 197, 200, 202, 205, 208, 212, 218, 221, 225, 226, 229,
232, 233, 234, 241, 242, 244, 245, 250, 256, 257, 260, 261, 265, 269, 272, 274,
277, 281, 288, 289, 290, 292, 293, 296, 298, 305, 306, 313, 314, 317, 320, 324,
325, 328, 333, 337, 338, 340, 346, 349, 353, 356, 360, 361, 362, 365, 369, 370,
373, 377, 386, 388, 389, 392, 394, 397, 400, 401, 404, 405, 409, 410, 416, 421,
424, 425, 433, 436, 441, 442, 445, 449, 450, 452, 457, 458, 461, 464, 466, 468,
477, 481, 482, 484, 485, 488, 490, 493, 500, 505, 509, 512, 514, 520, 521, 522,
529, 530, 533, 538, 541, 544, 545, 548, 549, 554, 557, 562, 565, 569, 576, 577,
578, 580, 584, 585, 586, 592, 593, 596, 601, 605, 610, 612, 613, 617, 625, 626,
628, 629, 634, 637, 640, 641, 648, 650, 653, 656, 657, 661, 666, 673, 674, 676,
677, 680, 685, 689, 692, 697, 698, 701, 706, 709, 712, 720, 722, 724, 725, 729,
730, 733, 738, 740, 745, 746, 754, 757, 761, 765, 769, 772, 773, 776, 778, 784,
785, 788, 793, 794, 797, 800, 801, 802, 808, 809, 810, 818, 820, 821, 829, 832,
833, 841, 842, 845, 848, 850, 853, 857, 865, 866, 872, 873, 877, 881, 882, 884,
890, 898, 900, 901, 904, 905, 909, 914, 916, 922, 925, 928, 929, 932, 936, 937,
941, 949, 953, 954, 961, 962, 964, 965, 968, 970, 976, 977, 980, 981, 985, 986,
997]
```

2.4.2 4.2 Écrire un entier comme somme de deux carrés

Soit $n \geq 1$ une somme de deux carrés. Écrivons $n = 2^c P_1 P_3$ où $c \geq 0$,

$$P_1 = \prod_{p|n, p\equiv 1 \bmod 4} p^{\nu_p}$$

$$P_3 = \prod_{p|n, p\equiv 3 \bmod 4} p^{\nu_p}$$

où p désigne dans les produits un nombre premier.

- On a $2^c = (1+i)^c(1-i)^c$
- Pour p congru à 3 modulo 4, ν_p est pair. On a donc $p^{\nu_p} = (p^2)^{\nu_p/2}$.
- Pour p congru à 1 modulo 4, on a $p = a_p^2 + b_p^2$ où $a_p, b_p \in \mathbb{N}$. On a alors $p^{\nu_p} = (a_p + ib_p)^{\nu_p}(a_p - ib_p)^{\nu_p}$.

Ainsi,

$$z = (1+i)^c \prod_{p|n, p\equiv 1 \bmod 4} (a_p + ib_p)^{\nu_p} \prod_{p|n, p\equiv 3 \bmod 4} p^{\nu_p}$$

vérifie $N(z) = n$. La fonction `carres` ci-dessous fait le travail.

[48]:
```
def carres(n):
    if n == 0: return (0, 0)
    fs = arith.factor(n)
    z = 1 + 0 * I
    for (p, nu) in fs:
        if p == 2:
            z = z * (1 + I) ** nu
        elif p % 4 == 3:
            if nu % 2 == 1:
                raise Exception('Pas somme de deux carres')
            else:
                z = z * (p + 0 * I) ** (nu // 2)
        else:
            a, b = somme_carres(p)
            z = z * (a + I * b) ** nu
    return (abs(z.a), abs(z.b))
```

[49]:
```
for n in range(20):
    if est_somme_carres(n):
        print(n, carres(n))
```

```
0 (0, 0)
1 (1, 0)
2 (1, 1)
4 (0, 2)
5 (2, 1)
8 (2, 2)
9 (3, 0)
10 (1, 3)
13 (3, 2)
16 (4, 0)
17 (4, 1)
18 (3, 3)
```

2.4.3 4.3 Combien de décompositions ?

Remarquons que dans la discussion ci-dessus, lorsque p est congru à 1 modulo 4, on a $p = N(a_p+ib_p)$ mais aussi $p = N(a_p - ib_p)$. Ainsi, pour tout $k \in [|0, \nu_p|]$,

$$p^{\nu_p} = N(i^f (a_p + ib_p)^k (a_p - ib_p)^{\nu_p - k})$$

où $f = 0, 1, 2, 3$. Il y a donc $4(\nu_p + 1)$ façons d'écrire p^{ν_p} comme norme d'un entier de Gauss. Par l'unicité de la décomposition en produit de nombres premiers dans $\mathbb{Z}[i]$, cela nous donne effectivement $4(\nu_p + 1)$ décompositions distinctes de p^{ν_p} en somme de deux carrés.

Ce notebook étant déjà bien long, nous admettrons qu'n'y a pas d'autres décompositions possibles. Notons toutefois que notre compte des décompositions inclut la possibilité de nombres négatifs, ainsi que des couples « symétriques » (a, b) et (b, a).

Proposition. Avec les notations ci-dessus, le nombre de décompositions de l'entier n en sommes de deux carrés est

$$4 \prod_{p|n, p\equiv 1 \bmod 4} (\nu_p + 1)$$

Si nous voulons exclure les entiers négatifs des décompositions, il suffit de diviser par 4. Pour exclure les doublons (a, b) et (b, a) il faut faire un peu plus attention : il se peut que l'on ait $a = b$. Mais ceci ne survient que si $n = 2a^2$, c'est à dire lorsque

- La valuation 2-adique de n est impaire, et
- pour tout $p \neq 2$, la valuation p-adique de n est impaire.

Voici donc une fonction renvoyant le nombre de décompositions vraiment distinctes de n en sommes de deux carrés.

[50]:
```
def nombre_decompositions(n):
    if n == 0: return 1
    else:
        fs = arith.factor(n)
        nb = 1
        double_carre = True
        for p, nu in fs:
            if p == 2 and nu % 2 == 0: double_carre = False
            if p % 4 == 3 and nu % 2 == 1: return 0
            elif p % 4 == 1:
                if nu % 2 == 1: double_carre = False
                nb = nb * (nu + 1)
        if double_carre == False: return nb // 2
        else: return (nb + 1) // 2
```

[51]:
```
for n in range(1000):
    nb = nombre_decompositions(n)
    if nb > 1: print(n, nb, end=', ')
```

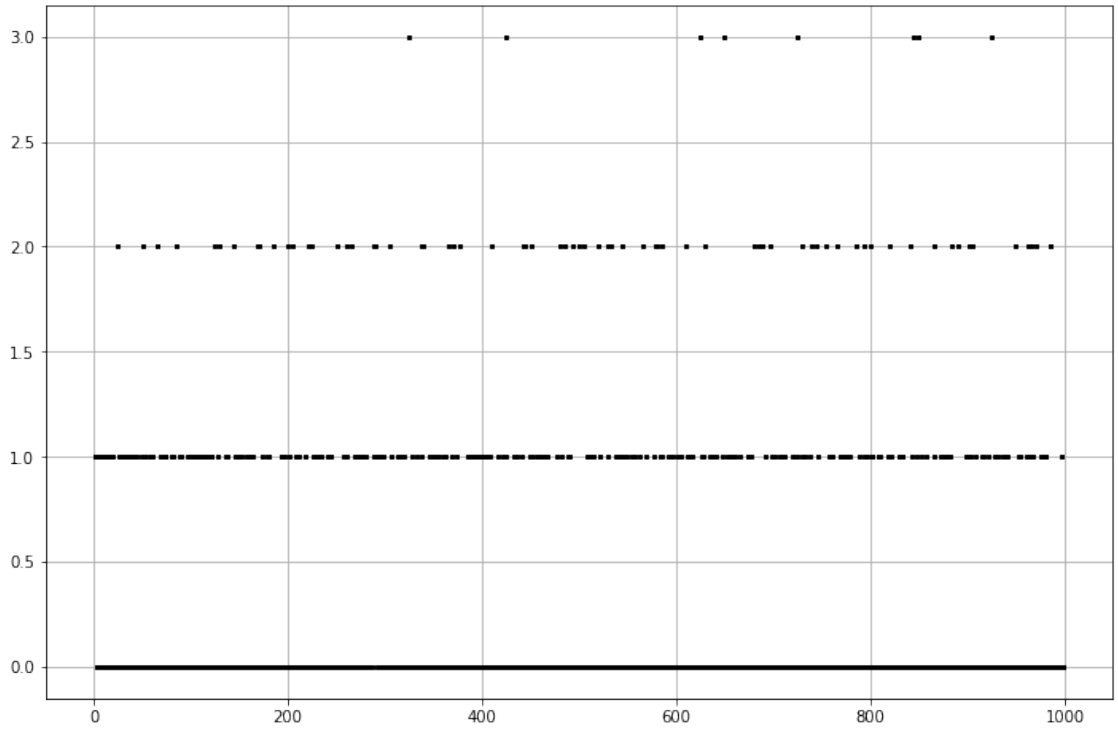
```
25 2, 50 2, 65 2, 85 2, 125 2, 130 2, 145 2, 169 2, 170 2, 185 2, 200 2, 205 2,
221 2, 225 2, 250 2, 260 2, 265 2, 289 2, 290 2, 305 2, 325 3, 338 2, 340 2, 365
2, 370 2, 377 2, 410 2, 425 3, 442 2, 445 2, 450 2, 481 2, 485 2, 493 2, 500 2,
505 2, 520 2, 530 2, 533 2, 545 2, 565 2, 578 2, 580 2, 585 2, 610 2, 625 3, 629
2, 650 3, 680 2, 685 2, 689 2, 697 2, 725 3, 730 2, 740 2, 745 2, 754 2, 765 2,
785 2, 793 2, 800 2, 820 2, 841 2, 845 3, 850 3, 865 2, 884 2, 890 2, 901 2, 905
2, 925 3, 949 2, 962 2, 965 2, 970 2, 985 2, 986 2,
```

[52]:
```
nombre_decompositions(1000922)
```

[52]: 4

Voici un graphe de la répartition du nombre de décompositions pour les entiers $n \leq 1000$.

[53]:
```
ks = range(1, 1000)
ns = [nombre_decompositions(k) for k in ks]
plt.plot(ks, ns, 'sk', ms=2)
plt.grid()
```



Quel est le plus grand nombre possible de décompositions en sommes de deux carrés pour les entiers inférieurs à 200000 ?

[54]:
```
kmax = 0
ndmax = 1
for k in range(200000):
    ndk = nombre_decompositions(k)
    if ndk > ndmax:
        kmax = k
        ndmax = ndk
print(kmax, ndmax)
```

```
160225 12
```

2.4.4 4.4 Trouver toutes les décompositions

Au vu de la discussion précédente, trouver toutes les décompositions d'un entier est maintenant faisable. Je ne commente pas les fonctions ci-dessous.

[55]:
```
def carres_aux(fs):
    if fs == []: return [1+0*I, -1+0*I, I, -I]
    else:
        p, nu = fs[0]
        cs = carres_aux(fs[1:])
        if p == 2:
            pnu = (1 + I) ** nu
            return [pnu * z for z in cs]
        elif p % 4 == 3:
            if nu % 2 != 0: return []
            else:
                pnu = (p + 0 * I) ** (nu / 2)
                return [pnu * z for z in cs]
        else:
            a, b = somme_carres(p)
            return [(a + I * b) ** k * (a - I * b) ** (nu - k) * z for k in␣
↪range(nu + 1) for z in cs]
```

Histoire d'enlever les nombres négatifs, et les couples symétriques ...

[56]:
```
def normaliser(z):
    a = abs(z.a)
    b = abs(z.b)
    if a >= b: return (a, b)
    else: return (b, a)
```

[57]:
```
def carres2(n):
    fs = arith.factor(n)
    cs = carres_aux(fs)
    cs1 = set([normaliser(z) for z in cs])
    return cs1
```

L'entier 50 s'écrit de deux façons comme une somme de deux carrés.

[58]:
```
print(carres2(50))
```

```
{(5, 5), (7, 1)}
```

L'entier 160225 s'écrit de 12 façons comme une somme de deux carrés.

[59]:
```
print(carres2(160225))
```

```
{(384, 113), (360, 175), (392, 81), (311, 252), (337, 216), (329, 228), (400,
15), (393, 76), (399, 32), (300, 265), (356, 183), (375, 140)}
```