Entiers_Gauss1

January 26, 2021

1 Entiers de Gauss (I)

Marc Lorenzi

26 janvier 2021

[1]:
```
import matplotlib.pyplot as plt
import math
import random
import arith
```

1.1 1. L'anneau des entiers de Gauss

1.1.1 1.1 C'est quoi ?

On pose

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

On vérifie aisément que $\mathbb{Z}[i]$ est un sous-anneau de $\mathbb{C}$. On l'appelle l'anneau des *entiers de Gauss*.

Clairement, $\mathbb{Z}[i]$ est stable par l'application $z \mapsto \overline{z}$. On pose également pour tout $z \in \mathbb{Z}[i]$,

$$N(z) = |z|^2 = z\overline{z}$$

Par les propriétés du module d'un nombre complexe, on a bien entendu pour tout $z \in \mathbb{Z}[i]$, $N(z) \in \mathbb{N}$. De plus, pour tous $z, z' \in \mathbb{Z}[i]$, $N(zz') = N(z)N(z')$.

1.1.2 1.2 La classe `Gauss`

On représente les entiers de Gauss en Python par des objets de la classe `Gauss` définie ci-dessous. Un entier de Gauss $x + iy$ est créé par l'instruction `Gauss(x, y)`. Si y est omis, il est pris par défaut égal à zéro.

Cette classe implémente toutes les opérations dans $\mathbb{Z}[i]$:

- addition (`+`), soustraction, opposé (`-`)

- multiplication (`*`)

L'addition, la soustraction et la multiplication peuvent éventuellement avoir l'un de leurs deux paramètres entier (i.e. de type `int`). On définit également :

- conjugué (`conj`)
- module au carré (`norm`)
- inversibilité (`inversible`)
- inverse lorsqu'il existe (`inverse`)
- puissance entière, éventuellement négative en cas d'inversibilité (`**`)

Toutes les méthodes sont *évidentes*. Les seules qui méritent une remarque sont celles qui concernent l'inversibilité.

Proposition. Soit $z \in \mathbb{Z}[i]$. On a équivalence entre

- z est inversible
- $N(z) = 1$
- $z = \pm 1$ ou $z = \pm i$

Démonstration.

- Supposons z inversible. Il existe donc $z' \in \mathbb{Z}[i]$ tel que $zz' = 1$. En passant aux normes (module au carré), il vient $N(z)N(z') = 1$. $N(z)$ est donc un entier naturel (évident) inversible dans $\mathbb{Z}$, ainsi $N(z) = 1$.
- Posons $z = a + ib$, où $a, b \in \mathbb{Z}$. Supposons $N(z) = a^2 + b^2 = 1$. Il est clair que $a = \pm 1$ et $b = 0$, ou $a = 0$ et $b = \pm 1$.
- ± 1 est inversible, égal à son propre inverse, et $\pm i$ est inversible, d'inverse $\mp i$.

Notation. Nous noterons dorénavant $\mathcal{U} = \{1, i, -1, -i\}$ l'ensemble des inversibles de $\mathbb{Z}[i]$.

Vous pouvez lire le code ci-dessous, mais il n'a rien de particulièrement subtil. On prépare également le terrain pour une future division euclidienne, qui permettra d'utiliser les opérateurs `//` et `%`.

[2]:
```
class Gauss:

    def __init__(self, a, b=0):
        self.a = a
        self.b = b

    # Affichage

    # Choix assez arbitraire de < , >
    def __repr__(self): return '<%d, %d>' % (self.a, self.b)

    # Égalité

    def __eq__(self, z):
        if isinstance(z, int): z = Gauss(z)
        return self.a == z.a and self.b == z.b

    def __req__(self, z):
```

```
        return self == z

    # Hachage

    def __hash__(self):
        return hash((self.a, self.b))

    # Addition

    def __add__(self, z):
        if isinstance(z, int): z = Gauss(z)
        return Gauss(self.a + z.a, self.b + z.b)

    def __radd__(self, z): return self + z

    # Opposé

    def __neg__(self): return Gauss(-self.a, -self.b)

    # Soustraction

    def __sub__(self, z):
        if isinstance(z, int): z = Gauss(z)
        return Gauss(self.a - z.a, self.b - z.b)

    def __rsub__(self, z):
        return -(self - z)

    # Multiplication

    def __mul__(self, z):
        if isinstance(z, int): z = Gauss(z)
        a = self.a * z.a - self.b * z.b
        b = self.a * z.b + self.b * z.a
        return Gauss(a, b)

    def __rmul__(self, z):
        return self * z

    # Conjugué

    def conj(self): return Gauss(self.a, -self.b)

    # Norme

    def norm(self): return (self * self.conj()).a
```

```
    # Inversibilité

    def inversible(self): return self.norm() == 1

    def inverse(self):
        if self.inversible():
            return self.conj()
        else: raise Exception ('Non inversible')

    # Puissances entières (exponentiation rapide)

    def __pow__(self, n):
        if n < 0:
            if self.inversible():
                return (self ** (-n)).inverse()
            else:
                raise Exception('Non inversible')
        elif n == 0: return Gauss(1, 0)
        else:
            y = (self * self) ** (n // 2)
            if n % 2 == 0:
                return y
            else:
                return self * y

    # Division euclidienne (voir plus loin pour la fonction division)

    def __divmod__(self, z):
        if isinstance(z, int): z = Gauss(z)
        return division(self, z)

    def __floordiv__(self, z):
        return self.__divmod__(z)[0]

    def __mod__(self, z):
        return self.__divmod__(z)[1]

    def __rdivmod__(self, z):
        z = Gauss(z)
        return z.__divmod__(self)

    def __floordiv__(self, z):
        return self.__divmod__(z)[0]

    def __rfloordiv__(self, z):
        return Gauss(z) // self
```

```
    def __mod__(self, z):
        return self.__divmod__(z)[1]

    def __rmod__(self, z):
        return Gauss(z) % self
```

Définissons un élément bien particulier de $\mathbb{Z}[i]$.

[3]:
```
I = Gauss(0, 1)
```

[4]:
```
I ** 2
```

[4]:
```
<-1, 0>
```

[5]:
```
(1 + 2 * I) ** 3
```

[5]:
```
<-11, -2>
```

1.2 2. Une division euclidienne dans $\mathbb{Z}[i]$

1.2.1 2.1 Le corps des fraction de $\mathbb{Z}[i]$

Soit $\mathbb{K} = \{x + iy, x, y \in \mathbb{Q}\}$. Nous avons là un sous-corps de $\mathbb{C}$ qui contient $\mathbb{Z}[i]$. Plus précisément, $\mathbb{K}$ est le *corps des fractions* de $\mathbb{Z}[i]$.

Proposition. $\mathbb{K} = \left\{\frac{a}{b}, a, b \in \mathbb{Z}[i], b \neq 0\right\}$.

Démonstration. Soient $a, b \in \mathbb{Z}[i]$, $b \neq 0$. On a

$$\frac{a}{b} = \frac{1}{N(b)}c$$

où $c = a\bar{b} \in \mathbb{Z}[i]$. Clairement, $\frac{a}{b} \in \mathbb{K}$.

Inversement, soit $z = x + iy \in \mathbb{K}$. En réduisant x et y au même dénominateur, on peut écrire z sous la forme $\frac{c}{n}$ où $c \in \mathbb{Z}[i]$ et $n \in \mathbb{Z} \subset \mathbb{Z}[i]$. Ainsi, z est le quotient de deux éléments de $\mathbb{Z}[i]$.

1.2.2 2.2 Division

Soient $a, b \in \mathbb{G}$, $b \neq 0$. Soit $z = \frac{a}{b} \in \mathbb{K}$. Soit $q \in \mathbb{Z}[i]$ tel que $|z - q| < 1$ (en lisant la suite vous comprendrez pourquoi un tel q existe. Il peut même y en avoir jusqu'à quatre). Posons également $r = a - bq$. On a alors

$$\left|\frac{r}{b}\right| = \left|\frac{a}{b} - q\right| = |z - q| < 1$$

et donc

$$N(r) = |r|^2 < |b|^2 = N(b)$$

Remarquons qu'un tel couple (q, r) n'est pas unique. Nous y reviendrons un peu plus loin.

La fonction `arrondi` prend en paramètres deux entiers relatifs a et b. Elle renvoie un entier q tel que

$$\left|\frac{a}{b} - q\right| \leq \frac{1}{2}$$

[6]:
```
def arrondi(a, b):
    if b < 0 :
        a, b = -a, -b
    q, r = a // b, a % b
    if 2 * r <= b: return q
    else: return q + 1
```

La fonction de division euclidienne dans $\mathbb{Z}[i]$ s'en déduit facilement. La fonction `division` prend en paramètres deux éléments x, y de $\mathbb{Z}[i]$ tels que $y \neq 0$. Elle renvoie une couple $(q, r) \in \mathbb{Z}[i]^2$ tel que $a = bq + r$ et $N(r) < N(b)$.

Exercice. Montrer que l'on a mieux que cela : $N(r) \leq \frac{1}{2}N(b)$.

[7]:
```
def division(x, y):
    z = x * y.conj()
    N = y.norm()
    alpha = arrondi(z.a, N)
    beta = arrondi(z.b, N)
    q = alpha + I * beta
    r = x - q * y
    return (q, r)
```

On peut maintenant utiliser de façon naturelle sur les entiers de Gauss les opérateurs Python `//` et `%`.

[8]:
```
a = 16 + 2 * I
b = 3 + 11 * I
print(a // b)
print(a % b)
```

```
<1, -1>
<2, -6>
```

Faisons quelques vérifications ...

[9]:
```
def random_gauss(N):
    return random.randint(-N, N) + I * random.randint(-N, N)
```

[10]:
```
N = 10 ** 10
a = random_gauss(N)
b = random_gauss(N)
print('a, b =', a, b)
q, r = a // b, a % b
print('q, r =', q, r)
print('a = bq + r :', a == b * q + r)
print('|N(r)|<|N(b)| :', abs(r.norm())< abs(b.norm()))
```

```
a, b = <-7868134803, 7724213591> <8760180110, 8472889697>
q, r = <0, 1> <604754894, -1035966519>
a = bq + r : True
|N(r)|<|N(b)| : True
```

1.2.3 2.2 La non unicité du quotient et du reste

Le quotient et le reste de la division euclidienne dans $\mathbb{Z}[i]$ ne sont pas uniques. Allons un peu plus loin. Soient $a, b \in \mathbb{Z}[i]$ tels que $b \neq 0$. Recherchons tous les couples $(q, r) \in \mathbb{Z}[i]^2$ vérifiant $a = bq + r$ et $N(r) < N(b)$. Posons $z = \frac{a}{b}$. Un tel couple (q, r) convient si et seulement si $r = a - bq$ et $|z - q| < 1$. Nous sommes donc ramenés au problème suivant :

Étant donné $z \in \mathbb{K}$, déterminer tous les entiers de Gauss q tels que $|z - q| < 1$.

Posons $z = u + iv$ où $u, v \in \mathbb{Q}$. Soient $\alpha = \lfloor u \rfloor$ et $\beta = \lfloor v \rfloor$. Clairement, seuls 4 éléments q de $\mathbb{G}$ sont candidats : $q = (\alpha + \delta) + i(\beta + \delta')$, où $\delta, \delta' \in \{0, 1\}$.

Le schéma ci-dessous montre le nombre de solutions à notre problème. Inutile de lire le code, on trace 4 quarts de cercles et on écrit quelques nombres.

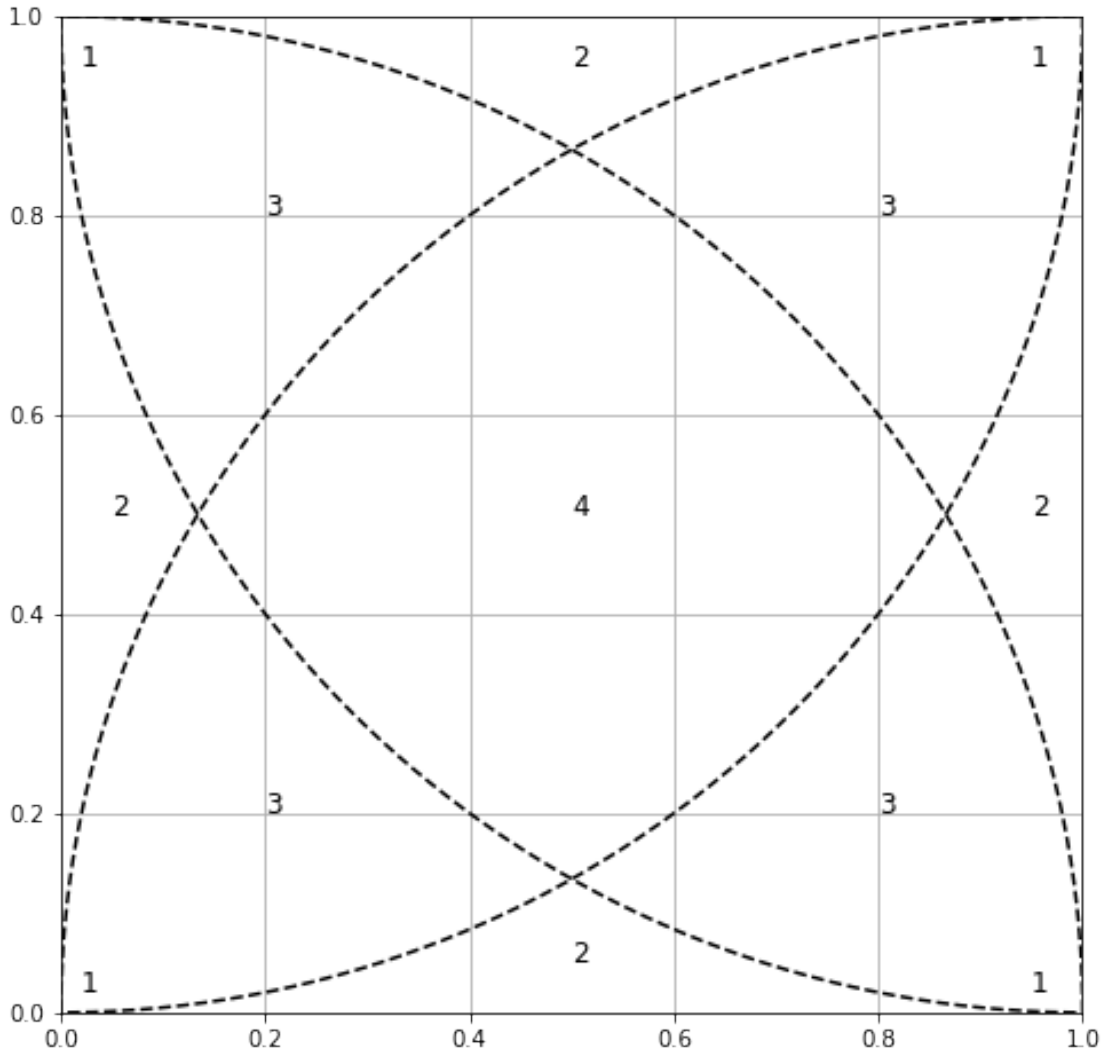
[11]:
```
def tracer():
    N = 500
    ts = [k * math.pi / (2 * N) for k in range(N + 1)]
    xs = [math.cos(t) for t in ts]
    ys = [math.sin(t) for t in ts]
    plt.plot(xs, ys, '--k')
    xs = [math.cos(t) for t in ts]
    ys = [1 - math.sin(t) for t in ts]
    plt.plot(xs, ys, '--k')
    xs = [1 - math.cos(t) for t in ts]
    ys = [math.sin(t) for t in ts]
    plt.plot(xs, ys, '--k')
    xs = [1 - math.cos(t) for t in ts]
    ys = [1 - math.sin(t) for t in ts]
    plt.plot(xs, ys, '--k')
    plt.text(0.5, 0.5, '4', fontsize=12)
    plt.text(0.8, 0.8, '3', fontsize=12)
    plt.text(0.8, 0.2, '3', fontsize=12)
    plt.text(0.2, 0.2, '3', fontsize=12)
```

```
    plt.text(0.2, 0.8, '3', fontsize=12)
    plt.text(0.5, 0.95, '2', fontsize=12)
    plt.text(0.5, 0.05, '2', fontsize=12)
    plt.text(0.95, 0.5, '2', fontsize=12)
    plt.text(0.05, 0.5, '2', fontsize=12)
    plt.text(0.02, 0.02, '1', fontsize=12)
    plt.text(0.02, 0.95, '1', fontsize=12)
    plt.text(0.95, 0.02, '1', fontsize=12)
    plt.text(0.95, 0.95, '1', fontsize=12)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.grid()
```

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

Le schéma représente le carré $[0,1] \times [0,1]$, mais ce serait exactement la même chose dans le carré $[\alpha, \alpha+1] \times [\beta, \beta+1]$. On voit apparaître un certain nombre de zones, limitées par des quarts de cercles de rayon 1. Le nombre dans chacune des zones est le nombre de côtés du carrés dont la distance à un point de la zone est strictement inférieure à 1. Pour les points sur les frontières en pointillés, c'est le plus petit nombre dans les zones voisines qui gagne. Les sommets du carré sont étiquetés 1, car il sont à distance strictement inférieure à 1 d'eux mêmes, et d'aucun autre sommet.

[13]:
```
tracer()
```



Les nombres indiqués dans les zones indiquent ainsi le nombre de quotients possibles pour une division euclidienne. Écrivons une fonction `division2` qui renvoie la liste des couples $(q, r) \in \mathbb{Z}[i]^2$ tels que $a = bq + r$ et $N(r) < N(b)$.

[14]:
```
def part_int(a, b):
    if b < 0 : a, b = -a, -b
    return a // b
```

[15]:
```
def division2(x, y):
    z = x * y.conj()
    N = y.norm()
    alpha = part_int(z.a, N)
    beta = part_int(z.b, N)
    s = []
```

```
    for a in [alpha, alpha + 1]:
        for b in [beta, beta + 1]:
            q = a + I * b
            r = x - q * y
            if r.norm() < N: s.append((q, r))
    return s
```

[16]:
```
division2(16 + 2 * I, 3 + 11 * I)
```

[16]:
```
[(<0, -2>, <-6, 8>),
 (<0, -1>, <5, 5>),
 (<1, -2>, <-9, -3>),
 (<1, -1>, <2, -6>)]
```

Faisons quelques tests.

[17]:
```
def check_div2():
    N = 10 ** 10
    a = random_gauss(N)
    b = random_gauss(N)
    print('a, b =', a, b)
    s = division2(a, b)
    k = 1
    for q, r in s:
        print('solution numéro', k)
        print('    q, r =', q, r)
        print('    a = bq + r :', a == b * q + r)
        print('    |N(r)|<|N(b)| :', abs(r.norm())< abs(b.norm()))
        k += 1
```

[18]:
```
check_div2()
```

```
a, b = <6907054758, -3989385896> <7687963488, 3183272370>
solution numéro 1
    q, r = <0, -1> <3723782388, 3698577592>
    a = bq + r : True
    |N(r)|<|N(b)| : True
solution numéro 2
    q, r = <0, 0> <6907054758, -3989385896>
    a = bq + r : True
    |N(r)|<|N(b)| : True
solution numéro 3
    q, r = <1, -1> <-3964181100, 515305222>
    a = bq + r : True
    |N(r)|<|N(b)| : True
solution numéro 4
    q, r = <1, 0> <-780908730, -7172658266>
    a = bq + r : True
```

```
|N(r)|<|N(b)| : True
```

1.2.4 2.3 Un peu de probabilités

Si vous avez évalué plusieurs fois la cellule ci-dessus, vous avez dû voir que très souvent il y a trois solutions, un peu moins souvent 4 solutions, très peu souvent 2 solutions, et « jamais » une seule réponse. Soyons plus précis.

Soient $a, b \in \mathbb{Z}[i]$, $b \neq 0$. Quelle est la probabilité que `check_div2` renvoie une solution ? Deux ? Trois ? Quatre ? Cette probabilité ne dépend que de la partie fractionnaire (les chiffres après la virgule) de la partie réelle et de la partie imaginaire de $\frac{a}{b}$. Mettons la probabilité uniforme $\mathbb{P}$ sur le carré $\mathcal{C} = [0,1] \times [0,1]$. Pour « toute » partie A de $\mathcal{C}$, $\mathbb{P}(A)$ est l'aire de A.

Considérons la variable aléatoire $X : \mathcal{C} \to \{1,2,3,4\}$ qui à tout élément de $\mathcal{C}$ associe le nombre de sommets de $\mathcal{C}$ qui sont à distance de ce point strictement inférieure à 1. Notre problème est ainsi le calcul de $\mathbb{P}[X = k]$ pour $k = 1,2,3,4$.

Ces probabilités sont égales aux aires des zones du dessin vu plus haut étiquetées par le nombre correspondant. La probabilité de tomber sur un sommet est nulle (l'aire d'un point est 0), on a donc $\mathbb{P}[X = 1] = 0$. De même, la probabilité de « tomber » sur une courbe en pointillés est nulle car l'aire totale de ces courbes est 0.

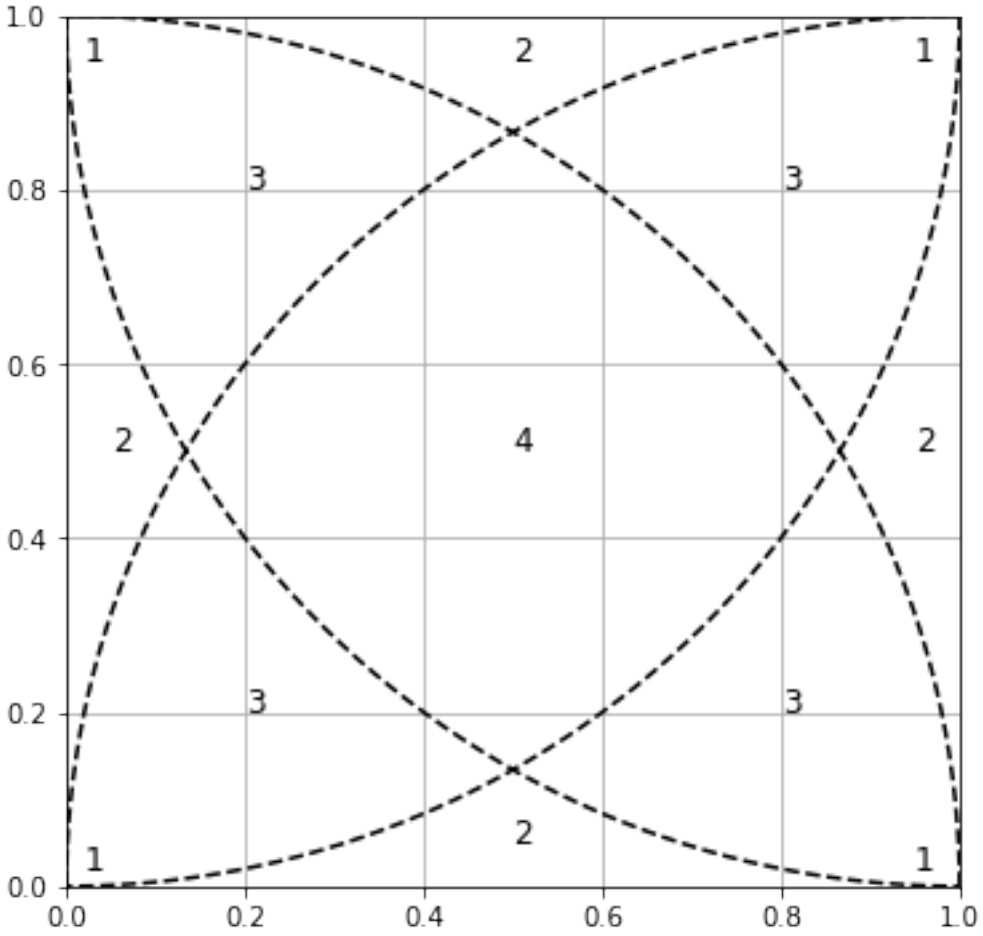
Définissons un dictionnaire P qui contiendra les probabilités en question. Mettons $P[1]$ à 0, et $P[2]$, $P[3]$ et $P[4]$ à `None`. Nous allons dans ce qui suit affecter les bonnes valeurs aux clés 2, 3, 4.

[19]:
```
P = {1: 0, 2:None, 3:None, 4:None}
```

Rappelons nous le dessin :

[20]:
```
plt.rcParams['figure.figsize'] = (6, 6)
```

[21]:
```
tracer()
```



Quelle est l'aire de la zone 2 tout en haut du carré ? C'est 1 - l'aire de ce qui est sous les cercles centrés en $(0,0)$ et $(1,0)$. Pour des raisons de symétrie, l'aire de la zone 2 du haut du carré est donc

$$\mathcal{A} = 1 - 2\int_0^{\frac{1}{2}} \sqrt{1-x^2}\,dx$$

Posons $x = \sin t$ dans cette intégrale. Il vient

$$\begin{aligned}
\mathcal{A} &= 1 - 2\int_0^{\frac{\pi}{6}} \cos^2 t\,dt \\
&= 1 - \int_0^{\frac{\pi}{6}} (1+\cos 2t)\,dt \\
&= 1 - \left[t + \tfrac{1}{2}\sin 2t\right]_0^{\frac{\pi}{6}} \\
&= 1 - \tfrac{\pi}{6} - \tfrac{\sqrt{3}}{4}
\end{aligned}$$

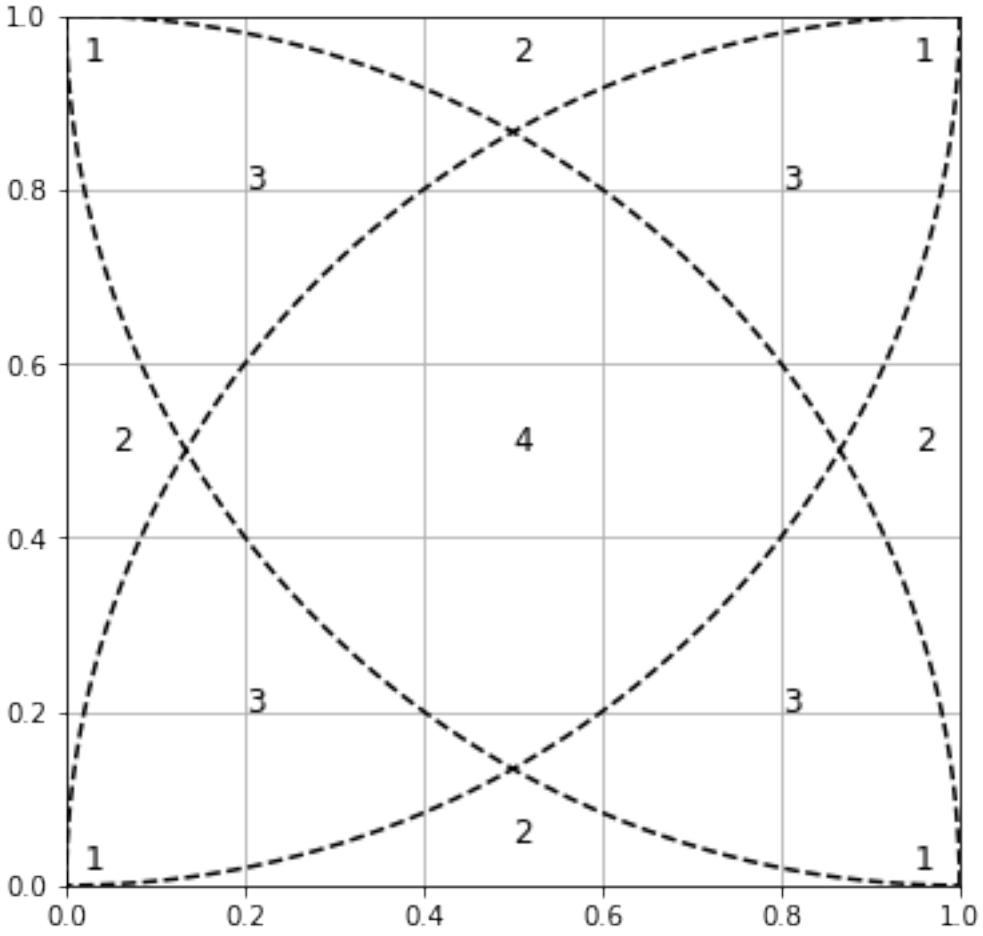
L'aire numéro 2 totale est donc 4 fois cette valeur :

$$\mathbb{P}[X=2] = 4 - \frac{2\pi}{3} - \sqrt{3}$$

[22]:
```
P[2] = 4 - 2 * math.pi / 3 - math.sqrt(3)
print(P)
```

```
{1: 0, 2: 0.17355409003792754, 3: None, 4: None}
```

[23]:
```
tracer()
```



Passons à l'aire de la zone 4. Le point « anguleux « à gauche de la zone 4 a pour ordonnée $\frac{1}{2}$ et se trouve sur le cercle de centre $(1,0)$ et de rayon 1. Son abscisse a vérifie donc

$$(a-1)^2 + \frac{1}{4} = 1$$

d'où

$$a = 1 \pm \frac{\sqrt{3}}{2}$$

Comme $a < 1$, on a donc $a = 1 - \frac{\sqrt{3}}{2}$. Pour des raisons de symétrie, l'aire numéro 4 est

$$\mathbb{P}[X = 4] = 2 \int_a^{\frac{1}{2}} \left(2\sqrt{1 - (x-1)^2} - 1 \right) dx$$

Posons $x = 1 - \sin t$ dans l'intégale. Il vient

$$\begin{aligned} \mathbb{P}[X = 4] &= 2 \int_{\frac{\pi}{6}}^{\frac{\pi}{3}} (2\cos t - 1) \cos t \, dt \\ &= \frac{\pi}{3} - \sqrt{3} + 1 \end{aligned}$$

[24]:
```
P[4] = math.pi / 3 - math.sqrt(3) + 1
print(P)
```

```
{1: 0, 2: 0.17355409003792754, 3: None, 4: 0.31514674362772044}
```

L'aire de la zone 3 s'en déduit aussitôt :

$$\mathbb{P}[X = 3] = 1 - \mathbb{P}[X = 1] - \mathbb{P}[X = 2] - \mathbb{P}[X = 4] = \frac{\pi}{3} + 2\sqrt{3} - 4$$

[25]:
```
P[3] = math.pi / 3 + 2 * math.sqrt(3) - 4
print(P)
```

```
{1: 0, 2: 0.17355409003792754, 3: 0.5112991663343518, 4: 0.31514674362772044}
```

Faisons un petit test. Effectuons M divisions euclidiennes aléatoires, et regardons les proportions de divisions qui comportent 1, 2, 3 ou 4 solutions.

[26]:
```
def check_div2_bis(M):
    N = 10 ** 10
    s1 = {1:0, 2:0, 3:0, 4:0}
    for k in range(M):
        a = random_gauss(N)
        b = random_gauss(N)
        s = division2(a, b)
        m = len(s)
        s1[m] = s1[m] + 1
    for k in range(1, 5):
        s1[k] = s1[k] / M
    return s1
```

[27]:
```
print(check_div2_bis(10000))
print(P)
```

```
{1: 0.0, 2: 0.1681, 3: 0.5273, 4: 0.3046}
{1: 0, 2: 0.17355409003792754, 3: 0.5112991663343518, 4: 0.31514674362772044}
```

Merci à la loi des grands nombres … les résultats obtenus sont tout à fait conformes aux probabilités calculées.

1.3 3. Pgcd

1.3.1 3.1 Pgcd de deux entiers de Gauss

Puisque l'anneau $\mathbb{Z}[i]$ est un anneau euclidien, il est donc *principal* : ses idéaux sont les ensembles de la forme $a\mathbb{Z}[i]$, où $a \in \mathbb{Z}[i]$.

Étant donnés deux éléments a et b de $\mathbb{Z}[i]$, $a\mathbb{Z}[i] + b\mathbb{Z}[i]$ est un idéal de $\mathbb{Z}[i]$. Il existe donc $\delta \in \mathbb{Z}[i]$ tel que

$$a\mathbb{Z}[i] + b\mathbb{Z}[i] = \delta\mathbb{Z}[i]$$

Un tel δ est unique à inversible près, c'est un pgcd de a et b. Il peut évidemment être calculé par l'algorithme d'Euclide. Pour varier un peu les plaisirs, écrivons un *générateur* qui engendre les reste des divisions successives de l'algorithme d'Euclide.

[28]:
```
def gen_pgcd(a, b):
    yield a
    while b != 0:
        a, b = b, a % b
        yield a
    return a
```

[29]:
```
for x in gen_pgcd(3 + 11 * I, 16 + 2 * I):
    print(x)
```

```
<3, 11>
<16, 2>
<5, -5>
<1, -3>
```

Le pgcd de a et b est obtenu en écartant les restes intermédiaires et en renvoyant le dernier reste.

[30]:
```
def pgcd(a, b):
    for x in gen_pgcd(a, b): pass
    return x
```

Si l'on veut tous les pgcds de a et b, il suffit d'en multipler un par ± 1 et $\pm i$.

[31]:
```
def pgcds(a, b):
    d = pgcd(a, b)
    return {d, -d, I * d, -I * d}
```

[32]:
```
pgcds(3 + 11 * I, 16 + 2 * I)
```

[32]:
```
{<-1, 3>, <-3, -1>, <1, -3>, <3, 1>}
```

1.3.2 3.2 Algorithme d'Euclide étendu

Soient $a, b \in \mathbb{Z}[i]$ de pgcd $a \wedge b = \delta$. Il existe alors $u, v \in \mathbb{Z}[i]$ tels que $ua + vb = \delta$. Un tel couple (u, v) peut être calculé *via* l'algorithme d'Euclide.

Ici encore nous écrivons un générateur, l'intérêt étant que l'on peut examiner les résultats intermédiaires de l'algorithme.

[33]:
```
def gen_AEE(a, b):
    u1, v1, r1 = (Gauss(1), Gauss(0), a)
    u2, v2, r2 = (Gauss(0), Gauss(1), b)
    yield (u1, v1, r1)
    while r2 != 0:
        q = r1 // r2
        u, v, r = u1 - q * u2, v1 - q * v2, r1 - q * r2
        u1, v1, r1 = u2, v2, r2
        u2, v2, r2 = u, v, r
        yield (u1, v1, r1)
```

[34]:
```
a = 3 + 11 * I
b = 16 + 2 * I
for u, v, r in gen_AEE(a, b):
    print('%s * %s + %s * %s = %s' % (u, a, v, b, r))
```

```
<1, 0> * <3, 11> + <0, 0> * <16, 2> = <3, 11>
<0, 0> * <3, 11> + <1, 0> * <16, 2> = <16, 2>
<1, 0> * <3, 11> + <0, -1> * <16, 2> = <5, -5>
<-1, -2> * <3, 11> + <-1, 1> * <16, 2> = <1, -3>
```

La fonction `AEE` (Algorithme d'Euclie Étendu) renvoie un triplet (u, v, δ) tel que $ua + vb = \delta$ et $\delta = a \wedge b$.

[35]:
```
def AEE(a, b):
    for s in gen_AEE(a, b): pass
    return s
```

[36]:
```
print(AEE(3 + 11 * I, 16 + 2 * I))
```

```
(<-1, -2>, <-1, 1>, <1, -3>)
```

Nous ne poursuivrons pas plus avant. L'existence de l'algorithme d'Euclide implique que tout ce que l'on fait dans l'anneau $\mathbb{Z}$ peut être refait dans l'anneau $\mathbb{Z}[i]$:

- ppcm
- équations diophantiennes de degré 1

- congruences
- restes chinois
- etc.