Résidus quadratiques

Marc Lorenzi - 22 juin 2018

In [1]:
```
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
plt.rcParams['figure.figsize'] = (8, 8)
```

1. De quoi allons-nous parler ?

Dans tout ce notebook, p désigne un nombre premier impair. Rappelons que l'ensemble $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ est un corps. L'ensemble de ses éléments non nuls, $\mathbb{F}_p^*$ est un groupe multiplicatif, que nous noterons $\mathbb{G}$. On notera les entiers naturels et les éléments de $\mathbb{F}_p$ de façon identique, le contexte permettant sans peine de décider si des calculs sont réalisés dans $\mathbb{N}$ ou dans $\mathbb{F}_p$.

Définition : Soit $x \in \mathbb{G}$. On dit que x est un **résidu quadratique** (modulo p) lorsqu'il existe $y \in \mathbb{G}$ tel que $x = y^2$.

Voici les questions auxquelles nous allons répondre :

- Combien y a-t-il de résidus quadratiques modulo p ?
- Étant donné $x \in \mathbb{G}$, comment déterminer si x est un résidu quadratique ?
- Si x est un résidu quadratique, comment trouver $y \in \mathbb{G}$ tel que $y^2 = x$?

Pour faire un parallèle intéressant, remplacez $\mathbb{F}_p$ par $\mathbb{R}$. Les réponses à ces trois questions sont :

- Une infinité. Philosophiquement parlant, on a envie de dire "un sur deux".
- On regarde si le signe de x est positif.
- On appuie sur la touche $\sqrt{\bullet}$ de la calculatrice (il y a d'autres réponses plus satisfaisantes :-))

Nous noterons $\mathcal{Q}$ l'ensemble des résidus quadratiques. Nous dirons de temps en temps "résidu" au lieu de "résidu quadratique".

2. Nombres premiers

Nous aurons besoin de fabriquer des nombres premiers pour tester nos fonctions. Voici tout d'abord une fonction (inefficace) permettant de vérifier si un entier est premier.

In [3]:
```
def premier(n):
    if n <= 1: return False
    else:
        k = 2
        while k * k <= n and n % k != 0: k += 1
        return k * k > n
```

In [4]:
```
print([n for n in range(100) if premier(n)])
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97]
```

Et voici une fonction `premier suivant` qui prend un entier n en paramètre et renvoie le plus petit nombre premier strictement supérieur à n.

In [5]:
```
def premier_suivant(n):
    p = n + 1
    while not premier(p): p = p +1
    return p
```

In [6]:
```
premier_suivant(123456789)
```

Out[6]:
```
123456791
```

Entrons maintenant dans le vif du sujet.

3. La fonction $x \mapsto x^2$

Nous allons étudier la fonction $\varphi : \mathbb{G} \to \mathbb{G}$ définie par $x \mapsto x^2$. Commençons par l'évidence :

Proposition : φ est un endomorphisme du groupe $\mathbb{G}$. On a $\ker \varphi = \{-1, 1\}$ et $Im\ \varphi = \mathcal{Q}$.

Démonstration : la propriété de morphisme est évidente : $(xy)^2 = x^2 y^2$. L'image de φ est $\mathcal{Q}$ par définition même d'un résidu quadratique. Seul le noyau de φ mérite un peu d'attention. Soit $x \in \mathbb{G}$. On a $x \in \ker \varphi$ si et seulement si $x^2 - 1 = 0$, ou encore $(x-1)(x+1) = 0$. N'oublions pas que $\mathbb{G} \subset \mathbb{F}_p$, qui est un corps. Or, dans un corps, un produit est nul si et seulement si l'un des facteurs est nul. D'où le résultat.

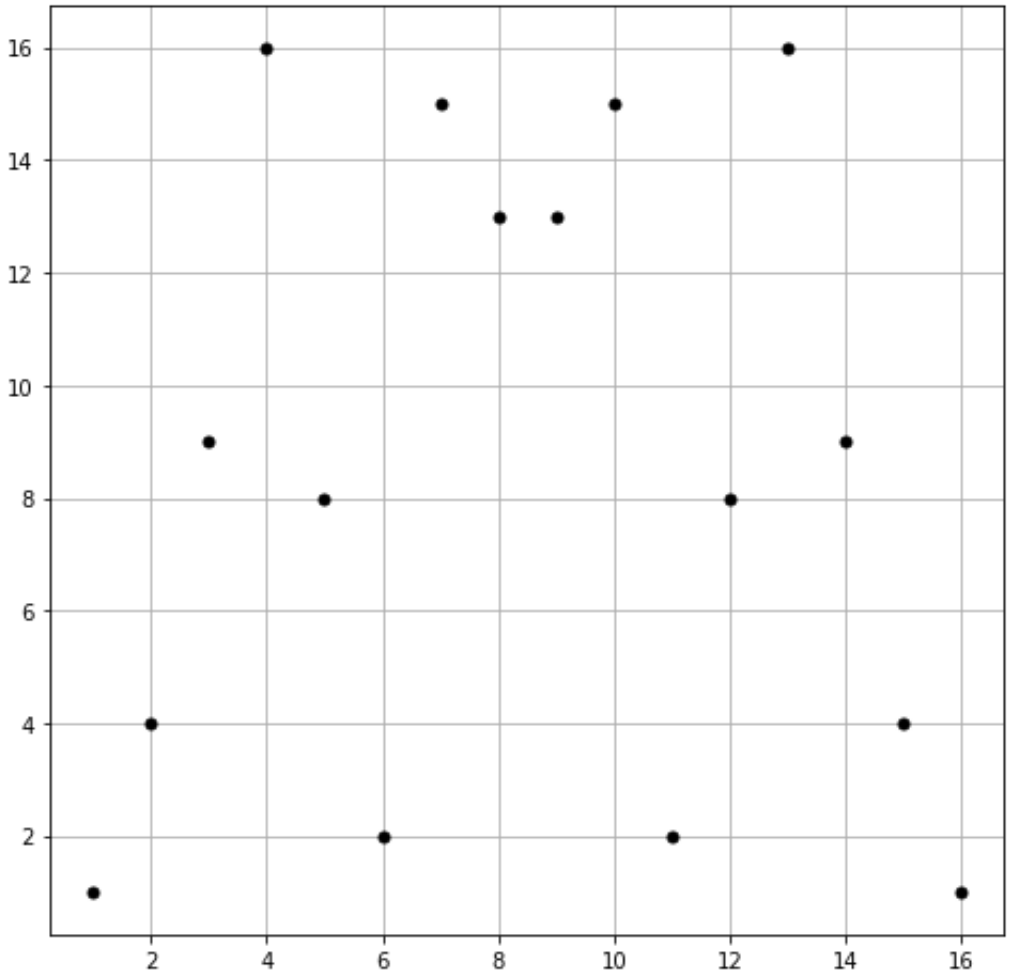
Corollaire : $\mathcal{Q}$ est un groupe.

Démonstration : c'est l'image d'un morphisme de groupes.

Prenons un petit exemple, $p = 17$. Traçons la fonction φ.

In [7]:
```
p = 17
print('p = ', p)
plt.plot(range(1, p), [(k * k) % p for k in range(1, p)], 'ok', marker
plt.grid()
plt.show()
```

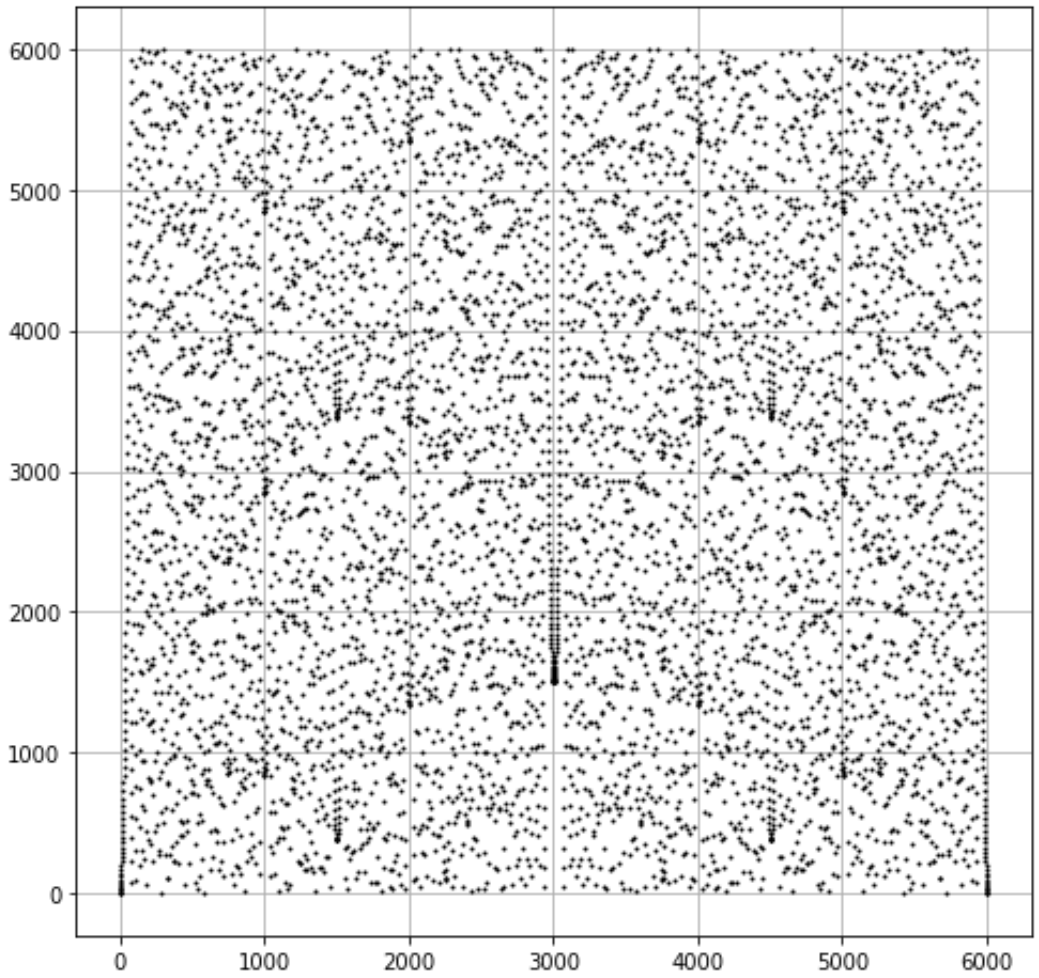
```
p =  17
```



Regardez les ordonnées des points : les résidus modulo 17 sont : 1, 2, 4, 8, 9, 13, 15 et 16. Il y en a donc 8. Pourquoi 8 ? Et pourquoi ces nombres-ci et pas les autres ? Et pourquoi les racines carrées de 2 sont-elles 6 et 11 ??? Pour bien comprendre que la question n'est pas totalement évidente, prenons un plus gros exemple.

In [8]:
```
p = premier_suivant(6000)
print('p = ', p)
rg = range(1, p)
plt.plot(rg, [(k * k) % p for k in rg], 'ok', markersize=1)
plt.grid()
plt.show()
```

```
p =  6007
```



C'est très joli. Cela montre surtout que la fonction $x \mapsto x^2$ a un comportement très compliqué. En y regardant de plus près, on voit des motifs apparaître, des "bouts de paraboles". Changez la valeur de `rg` ci-dessus pour faire des zooms sur des parties du graphe. Valeurs conseillées (rien ne vous empêche d'essayer autre chose) :

- 0-100
- 700-800
- 2500-3500
- etc.

Facile, évidemment, d'obtenir la liste des résidus modulo p. La fonction `liste_residus` fait le travail.

In [9]:
```
def liste_residus(p):
    s = set()
    for k in range(1, p):
            s = s.union(set([(k * k) % p]))
    return s
```

In [10]:
```
print(liste_residus(17))
```

```
{1, 2, 4, 8, 9, 13, 15, 16}
```

4. Combien de résidus ?

Proposition : Il y a exactement $\frac{p-1}{2}$ résidus quadratiques modulo p.

Démonstration : Reprenons notre fonction carré "surjectivisée", $\varphi : \mathbb{G} \to \mathcal{Q}$. On a $\mathbb{G} = \varphi^{-1}(\mathcal{Q}) = \varphi^{-1}(\bigcup_{a\in\mathcal{Q}}\{a\}) = \bigcup_{a\in\mathcal{Q}} \varphi^{-1}(\{a\})$. Les ensembles $\varphi^{-1}(\{a\})$ sont disjoints, donc

$$(1) \quad |\mathbb{G}| = \sum_{a\in\mathcal{Q}} |\varphi^{-1}(\{a\})|$$

où les deux barres verticales désignent le cardinal de l'ensemble.

Soit $a \in \mathcal{Q}$. Soient $x, y \in \mathbb{G}$. Supposons $\varphi(x) = \varphi(y) = a$. On a alors $x^2 = y^2$, ou encore $(x - y)(x + y) = 0$. Donc $y = \pm x$ (corps !). Maintenant, par définition même d'un résidu quadratique, a a un antécédent x par φ. Ainsi, $\varphi^{-1}(\{a\}) = \{-x, x\}$. Peut-on avoir $-x = x$? Non, car dans ce cas $2x = 0$ dans $\mathbb{F}_p$. Mais $2 \neq 0$ car p est impair, et $x \neq 0$ également. Et $\mathbb{F}_p$ est un corps. Ainsi, $|\varphi^{-1}(\{a\})| = 2$. Réinjectons dans l'égalité (1) pour obtenir $|\mathbb{G}| = \sum_{a\in\mathcal{Q}} 2 = 2|\mathcal{Q}|$. D'où le résultat souhaité, puisque $|\mathbb{G}| = p - 1$.

Creusons un peu : dans $\mathbb{F}_p^*$, il y a $\frac{p-1}{2}$ résidus, et donc $\frac{p-1}{2}$ non résidus.

- Si on multiplie entre-eux deux résidus, on obtient un résidu, puisque $\mathbb{G}$ est un groupe.
- Si on multiplie un résidu par un non-résidu, on obtient un non-résidu, toujours grâce à la structure de groupe de $\mathbb{G}$ (supposez le contraire ...).

Mais que se passe-t-il si on multiplie deux non-résidus ? Mais oui, rassurez-vous, cela fait un résidu, mais ce n'est pas totalement immédiat. Soit α un non résidu quadratique. La fonction $f : x \mapsto \alpha x$ est une bijection de $\mathbb{F}_p^*$ sur lui-même. qui envoie les résidus sur DES non-résidus, d'après ce que nous avons dit au point numéro 2. Mais il y a exactement le même nombre de résidus et de non-résidus, donc la fonction f envoie les résidus sur LES non-résidus. Les non-résidus n'ont donc, par l'injectivité de f, plus le choix : leur image EST un résidu. Ainsi :

- Si on multiplie un non-résidu par un non-résidu, on obtient un résidu.

5. Qui est résidu ? Qui ne l'est pas ?

5.1 La formule de Legendre

Soit $x \in \mathbb{G}$. Comment savoir si x est un carré ? On pourrait calculer tous les carrés possibles et voir si on trouve x dans la liste, mais soyons lucides, ce serait totalement inefficace. Il nous faudrait faire $\mathcal{O}(p)$ opérations. Peut-on faire mieux ? Oui.

Proposition : Soit $x \in \mathbb{G}$. Alors x est un résidu quadratique si et seulement si $x^{\frac{p-1}{2}} = 1$.

Démonstration : Les deux implications sont intéressantes.

- $(\Rightarrow)$ Supposons que x est un résidu. Il existe donc $y \in \mathbb{G}$ tel que $x = y^2$. Mais alors $x^{\frac{p-1}{2}} = y^{p-1} = 1$ par le petit théorème de Fermat.
- $(\Leftarrow)$ Considérons le polynôme $P = X^{\frac{p-1}{2}} - 1 \in \mathbb{F}_p[X]$. Ce polynôme est de degré $\frac{p-1}{2}$, il a donc au plus $\frac{p-1}{2}$ racines. Mais nous avons déjà, par l'implication directe montrée juste avant, trouvé $\frac{p-1}{2}$ racines de P : les résidus quadratiques. Il ne peut donc pas y avoir d'autres racines, donc LES racines de P sont LES résidus quadratiques. En prime, P est scindé et toutes ses racines sont simples.

Et si x n'est pas un résidu ? Que vaut $x^{\frac{p-1}{2}}$?

Proposition : Soit $x \in \mathbb{G}$. Si x n'est pas un résidu quadratique alors $x^{\frac{p-1}{2}} = -1$.

Démonstration : Soit $y = x^{\frac{p-1}{2}}$. On a $y^2 = x^{p-1} = 1$ par le petit théorème de Fermat. Donc $y = \pm 1$ (corps !). Mais $y \neq 1$ puisque x n'est pas un résidu.

Nous voici donc en possession d'un test simple pour savoir si x est un résidu. On élève x à la puissance $\frac{p-1}{2}$. La réponse est OUI si et seulement si le résultat est 1 et NON si et seulement si le résultat n'est pas 1. Il est alors -1.

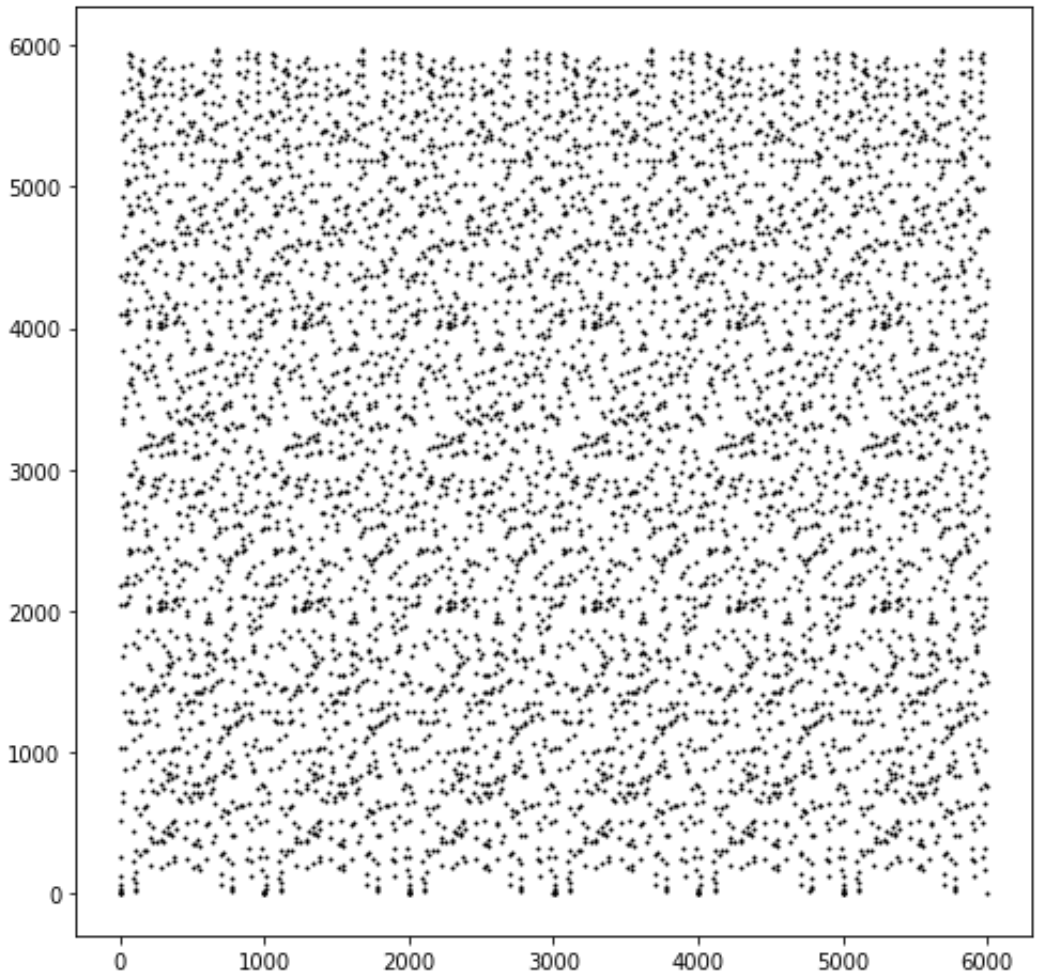
Certes, bon, d'accord, mais cette élévation à une puissance va nous obliger à faire $\mathcal{O}(p)$ multiplications ? Eh non, car l'exponentiation rapide est notre amie. Cet algorithme bien connu permet de calculer une puissance x^n avec une nombre d'opérations de l'ordre de $\log n$. Je ne détaillerai pas ici son fonctionnement, vous l'étudierez (ou l'avez étudié) en cours d'informatique.

La fonction `powermod` prend en paramètres trois entiers x, n, p. Elle renvoie x^n modulo p après avoir fait $\mathcal{O}(\log n)$ multiplications dans $\mathbb{F}_p$. Bien entendu, si p est grand, chaque multiplication est en soi coûteuse, mais je n'entrerai pas plus avant dans les détails.

In [11]:
```
def powermod(x, n, p):
    if n == 0: return 1
    else:
        y = powermod(x, n // 2, p)
        z = (y * y) % p
        if n % 2 == 0: return z
        else: return (x * z) % p
```

Même si l'objet de ce notebook est l'étude de $x \mapsto x^2$, ne résistons pas à la tentation de tracer la courbe de $x \mapsto 2^x$, histoire de tester notre fonction `powermod` .

In [12]:
```
p = premier_suivant(6000)
plt.plot([powermod(2, k, p) for k in range(1, p)], 'ok', markersize=1)
plt.show()
```



On ne distingue pas grand chose, les points ont l'air vraiment répartis de façon très "aléatoire". Imaginons que l'on me donne $x \in \mathbb{F}_p^*$. Supposons que $x = 2^k$ est une puissance de 2. Que vaut k ? Ceci est une question diificile, on ne connaît aucun algorithme permettant d'y répondre en temps raisonnable (pour p grand, cela va de soi). En fait, le problème du calcul de k à partir de x est appelé le problème du **logarithme discret**. Il est à la base d'un système cryptographique appelé le **cryptage ElGamal**.

Revenons à notre sujet. La fonction `est_residu` prend en paramètres deux entiers x et p, où p est premier et impair, et x n'est pas un multiple de p. Elle renvoie `True` lorsque x (sa classe modulo p) est un résidu quadratique, et `False` sinon.

In [13]:
```
def est_residu(x, p):
    return x % p != 0 and powermod(x, (p - 1) // 2, p) == 1
```

Petit test avec petit p ?

In [14]:
```
[x for x in range(17) if est_residu(x, 17)]
```

Out[14]:
```
[1, 2, 4, 8, 9, 13, 15, 16]
```

Mise à l'épreuve avec un nombre premier un peu plus grand.

In [15]:
```
p = premier_suivant(100000)
print(p)
len([x for x in range(1, p) if est_residu(x, p)])
```

```
100003
```

Out[15]:
```
50001
```

Le compte est bon, puisque $\frac{100003-1}{2} = 50001$.

5.2 Peut-on faire mieux ?

En utilisant des résultats mathématiques plus fins on peut écrire un algorithme qui s'exécute plus efficacement que notre algorithme utilisant l'exponentiation rapide. Le lecteur intéressé s'orientera vers les mots clés *symbole de Legendre*, *symbole de Jacobi*, *loi de réciprocité quadratique*.

5.3 Un petit quizz

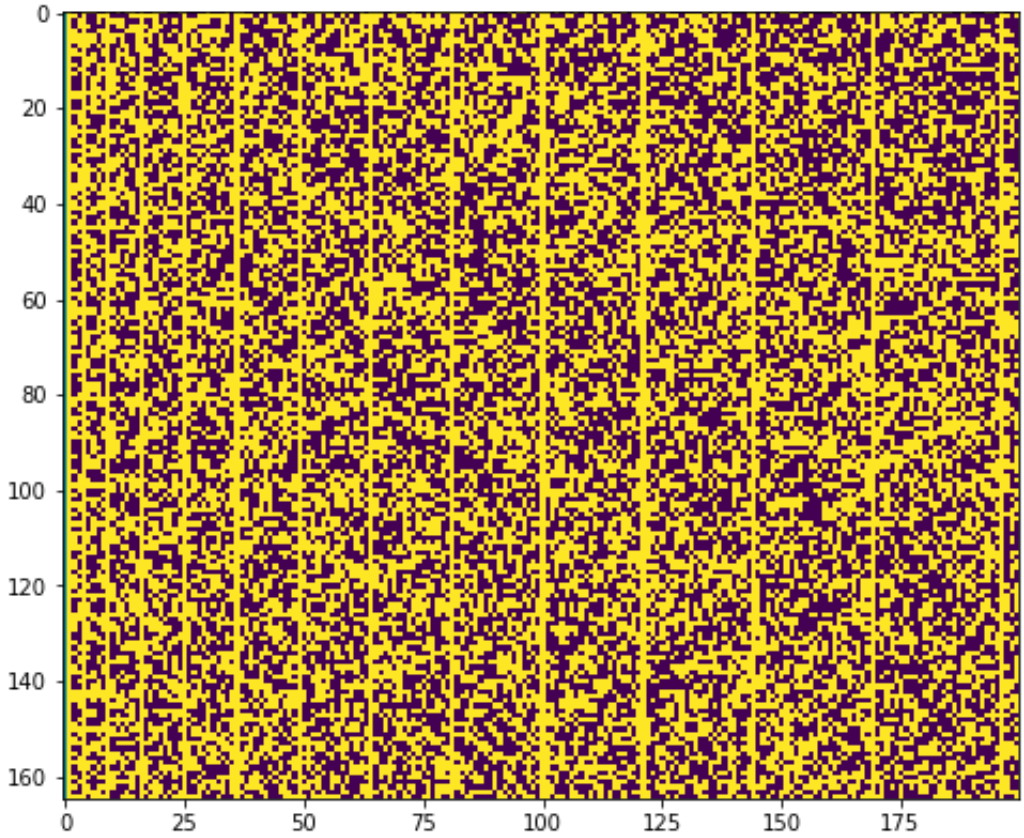
La fonction `chi(x,p)` renvoie 0 si x est multiple de p, et ± 1 sinon, selon que est un résidu modulo p ou pas.

In [16]:
```
def chi(x, p):
    if x % p == 0: return 0
    elif est_residu(x, p): return 1
    else: return -1
```

On trace $\chi(x,p)$ pour p premier, $200 \leq p < 1300$ et $0 \leq x < 200$.

In [17]:
```
rg = [p for p in range(200, 1300) if premier(p)]
A = [[chi(x, p) for x in range(0,200)] for p in rg]
plt.imshow(A)
```

Out[17]: <matplotlib.image.AxesImage at 0x108f7f4e0>



Question : Mais que sont ces bandes jaunes verticales ?

6. Calcul des racines carrées modulo p

J'ai dit un peu plus haut que résoudre dans $\mathbb{F}_p$ l'équation $2^y = x$ d'inconnue y était un problème très difficile pour lequel on ne possède à l'heure actuelle aucune solution satisfaisante. Qu'en est-il de la résolution de l'équation $y^2 = x$? Eh bien ce n'est pas évident, mais ON A une solution satisfaisante à ce problème.

Problème : p est un nombre premier impair, x est un résidu quadratique modulo p. Trouver une racine carrée de x.

Nous allons décrire et écrire l'algorithme de **Tonelli-Shanks**, qui résout ce problème. Voici le principe de l'algorithme. La difficulté essentielle dans ce qui va suivre est le nombre de lettres que l'on doit introduire dans la démonstration. Prenez des notes si vous voulez comprendre !

L'entier p est impair. Il s'écrit donc de façon unique $p = 1 + 2^e q$ où $e \geq 1$ et q est un nombre impair. On a $x^{\frac{p-1}{2}} = (x^q)^{2^{e-1}} = 1$ puisque x est un résidu quadratique. Posons $b = x^q$. Comme l'ensemble $\mathcal{Q}$ des résidus est un groupe, on a $b \in \mathcal{Q}$. De plus, $b^{2^{e-1}} = 1$.

Soit $g \in \mathbb{G}$ un non-résidu. Imaginons que l'on a fabriqué un entier pair k tel que $bg^k = 1$. Posons illico $r = x^{\frac{q+1}{2}} g^{\frac{k}{2}}$. On a $r^2 = x^{q+1} g^k = x x^q g^k = xbg^k = x$. Ainsi, r est une racine carrée de x.

Il nous reste évidemment deux problèmes non triviaux à résoudre.

1. Trouver g.
2. Trouver k.

Commençons par g ...

6.1 Étape 1 : trouver un non-résidu

Si je tire au hasard un entier entre 1 et $p-1$, la probabilité que ce soit un non-résidu est $\frac{1}{2}$. Notre algorithme de recherche de non-résidu sera probabiliste : tirer des entiers au hasard jusqu'à ce qu'on tombe sur un non-résidu. Pourquoi cet algorithme va-t-il fonctionner ? Soit n un entier ≥ 1. Considérons l'univers Ω des n-uplets d'entiers entre 1 et $p-1$ muni de la probabilité uniforme. L'événement A "aucun des éléments du n-uplet n'est un résidu" a une probabilité $(\frac{1}{2})^n$. Tirages avec remise ! Ainsi, par exemple, la probabilité de ne pas avoir obtenu un non-résidu après 20 tirages aléatoires est $\frac{1}{2^{20}} \simeq 10^{-6}$.

Théorème : En prenant n suffisamment grand, la probabilité d'échec est inférieure à celle qu'un yack obtienne un doctorat de physique nucléaire.

Démonstration : La probabilité du yack est strictement positive (quoique petite), et $\frac{1}{2^n}$ tend vers 0 lorsque n tend vers l'infini.

Note aux amis des animaux : je suis un ami des yacks.

Pour être encore plus précis, Soit X le nombre de tirages nécessaires pour obtenir un non-résidu. Pour tout entier $n \in \mathbb{N}^*$, on a $P(X = n) = \frac{1}{2^n}$. La probabilité que l'algorithme ne renvoie JAMAIS de non-résidu est donc $1 - \sum_{n=1}^{\infty} \frac{1}{2^n} = 1 - 1 = 0$. L'événement "la fonction ne termine pas" est de probabilité NULLE. Cela ne veut pas dire qu'il est impossible, mais vous ne l'observerez jamais (avec probabilité 1 :-).

La fonction `trouver_non_residu` fait le travail. Elle prend également un paramètre optionnel `stats` qui permet d'afficher le nombre de tirages aléatoires avant succès. Essayez donc de dépasser 20 tirages !

In [18]:
```
def trouver_non_residu(p, stats=False):
    count = 0
    while True:
        r = random.randint(1, p - 1)
        count=count + 1
        if not est_residu(r, p):
            if stats: print(count)
            return r
```

In [19]:
```
p = premier_suivant(10000000)
trouver_non_residu(p, stats=True)
```

```
4
```

Out[19]:
```
7587897
```

6.2 Trouver k, calculer une racine carrée

Là, ça se corse. On va s'intéresser à des couples (e_1, e_2) tels que

$$(1) \quad x^{e_1} g^{e_2} = 1$$

Je rappelle que x est le résidu dont on cherche la racine carrée, et que g est un non-résidu. Remarquons que e_2 est pair. En effet, $x^{e_1} \in \mathcal{Q}$ et 1 aussi. Comme $\mathcal{Q}$ est un groupe il en résulte que g^{e_2} est un résidu. Mais g n'en est pas un, et une puissance impaire d'un non résidu est un non-résidu.

On part de $e_1 = \frac{p-1}{2}$ et $e_2 = 0$. Tant que e_1 est pair, on le divise par 2 et on adapte e_2 pour maintenir la condition (1). Lorsque la condition de parité sur e_1 n'est plus respectée, on a $e_1 = q$, LE q impair tel que $p = 1 + 2^e q$. Et donc, $x^q g^{e_2} = 1$, ou encore, en posant $b = x^q$ et $k = e_2$, $b g^k = 1$.

À chaque division de e_1 par 2, on cherche un e_2 tel que (1) reste vraie. Remarquons que $(x^{\frac{e_1}{2}} g^{\frac{e_2}{2}})^2 = x^{e_1} g^{e_2} = 1$. Donc (corps !), $x^{\frac{e_1}{2}} g^{\frac{e_2}{2}} = \pm 1$.

- Si c'est 1, c'est le bonheur. En divisant e_1 et e_2 par 2, on obtient un nouveau couple qui satisfait (1).
- Si c'est -1, c'est pas grave. Parce que comme g est un non résidu, on a $g^{\frac{p-1}{2}} = -1$. On prend comme nouvelle valeur de e_2 l'entier $\frac{e_2}{2} + \frac{p-1}{2}$. Le nouveau couple ainsi créé vérifie encore (1) (exercice !).

Voici donc notre fonction qui calcule une racine carrée du résidu x modulo p. Elle reprend les notations de la démonstration.

La présentation que j'ai choisie ici a l'avantage de la clarté (si, si, je vous assure). En revanche, le code ci-dessous effectue des exponentiations rapides dont on pourrait se dispenser, au prix d'un obscurcissement du code. La fonction `racine` garde tout de même une complexité très correcte : la boucle `while` s'exécute en pire cas $\log p$ fois (et souvent (cf l'annexe !) beaucoup moins, juste une ou deux fois, voire pas du tout). Et chaque itération calcule deux puissances qui nécessitent $\mathcal{O}(\log p)$ opérations. Notre fonction est donc de complexité en pire cas $\mathcal{O}((\log p)^2)$. Des nombres premiers de plusieurs centaines de chiffres restent tout à fait envisageables.

En fait, ce qui nous bride dans ce notebook est la fonction `premier` qui, elle, s'exécute en temps $\mathcal{O}(\sqrt{p})$ et ne supportera pas d'être exécutée sur des nombres premiers de plus d'une quinzaine de chiffres. Mais là encore, il existe des algorithmes bien plus performants permettant de s'attaquer à des nombres premiers ayant des milliers de chiffres (voir le notebook sur l'algorithme de Miller-Rabin).

In [20]:
```
def racine(x, p):
    g = trouver_non_residu(p)
    q = (p - 1) // 2
    e1, e2 = (q, 0)
    while e1 % 2 == 0:
        e1, e2 = (e1 // 2, e2 // 2)
        z = (powermod(x, e1, p) * powermod(g, e2, p)) % p
        if z == p - 1: e2 = e2 + q
    return (powermod(x, (e1 + 1) // 2, p) * powermod(g, e2 // 2, p)) %
```

Prenons pour tester un nombre premier pas trop petit (mais pas trop grand :-)).

In [21]:
```
p = premier_suivant(1000000000)
print(p)
```

```
1000000007
```

Prenons un résidu modulo p.

In [22]:
```
est_residu(123456789, p)
```

Out[22]:
```
True
```

Il y avait une chance sur deux que ça marche. Calculons une racine carrée de ce nombre.

In [23]:
```
racine(123456789, p)
```

Out[23]:
```
151347102
```

Vérifions ?

In [24]:
```
(_ * _) % p
```

Out[24]:
```
123456789
```

Youpi.

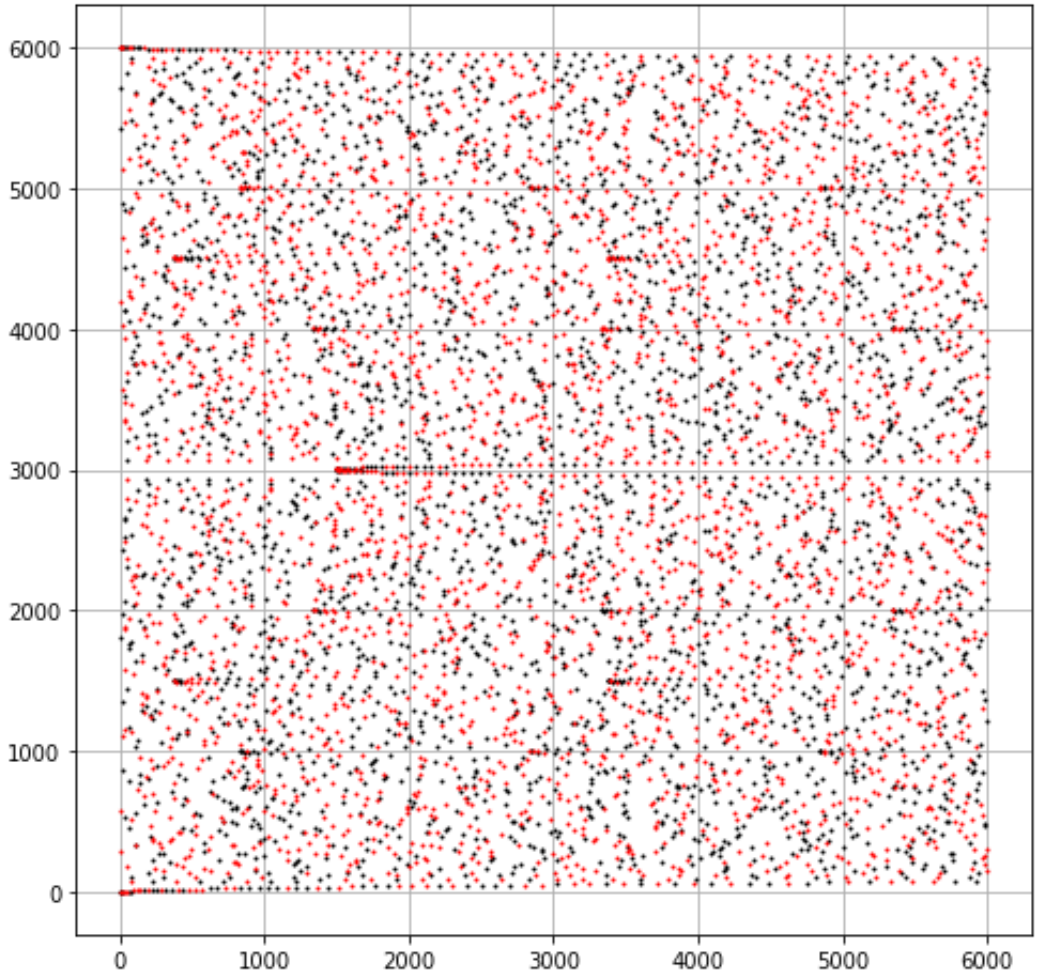
6.3 Le graphe !

Traçons le graphe de la "fonction" racine carrée modulo p. Chaque résidu quadratique a deux racines, qui sont bien entendu opposées (dans $\mathbb{F}_p$). En noir, l'une des deux. En rouge, l'autre. Ce n'est pas moi qui choisis les couleurs, c'est la fonction `racine` .

In [25]:
```
def tracer_racine(p):
    rg = [k for k in range(1, p) if est_residu(k, p)]
    s1 = [racine(k, p) for k in rg]
    s2 = [p - racine(k, p) for k in rg]
    plt.plot(rg, s1, 'ok', markersize=1)
    plt.plot(rg, s2, 'or', markersize=1)
    plt.grid()
    plt.show()
```

In [26]:
```
p = premier_suivant(6000)
print(p)
tracer_racine(p)
```

```
6007
```



Pour tout x entre 1 et $p-1$, il y a

- zéro point d'abscisse x si x n'est pas un résidu.
- Un point rouge et un point noir d'abscisse x sinon.

Et bien entendu, le symétrique orthogonal de ce dessin par rapport à la droite d'équation $y=x$ est le dessin que nous avons fait à la section 3.

Je vous laisse compter les points :-).

6.4 Annexe : Le nombre moyen d'itérations dans l'algorithme de Tonelli-Shanks

Dans ce qui va suivre, j'utiliserai un certain nombre de résultats de théorie des nombres absolument pas triviaux. Nous allons calculer le nombre moyen d'itérations dans la boucle `while` de la fonction `racine` .

Posons, comme nous l'avons fait plus haut, $p = 1 + 2^e q$ où $e \geq 1$ et q est un entier impair. Juste avant le début de la boucle `while` , on a $e_1 = 2^{e-1} q$. Et à chaque itération, e_1 est divisé par 2. Ainsi, le nombre d'itérations de la boucle est $e - 1$.

Bilan : Si $p \equiv 1[2^e]$ et $p \not\equiv 1[2^{e+1}]$, le nombre d'itérations est $e - 1$.

Par exemple, il y a zéro itération si et seulement si $p \equiv 1[2]$, ce qui est toujours vrai, et $p \not\equiv 1[4]$.

Un nombre impair est congru soit à 1, soit à 3 modulo 4. Mais combien y a-t-il de nombres premiers congrus à 3 modulo 4 ? Eh bien une infinité. Ceci peut se montrer de façon élémentaire par l'absurde. Et les nombres premiers congrus à un modulo 4 ? Il y en a une infinité aussi. Cela peut aussi se montrer, mais un peu moins facilement que pour les nombres congrus à 3 modulo 4.

Posons la question autrement : parmi tous les nombres premiers, quelle est la "proportion" des nombres premiers congurs à 3 modulo 4 ? L'idéal serait de pouvoir définir une probabilité uniforme sur l'ensemble $\mathcal{P}$ des nombres premiers, mais ceci est impossible. Il est en revanche possible, je ne le ferai pas, de définir une "quasi-probabilité" (une mesure finiment additive) qui réponde à nos besoins. Nous la noterons μ. Un résultat très général, dû à Dirichlet et de la Vallée Poussin, dit :

Théorème : Soient $a \geq 2$ et $1 \leq b \leq a$ tels que $pgcd(a, b) = 1$. Notons $\mathcal{P}_{a,b}$ l'ensemble des nombres premiers p tels que $p \equiv b[a]$. Alors, $\mu(\mathcal{P}_{a,b}) = \frac{1}{\varphi(a)}$ où φ est la fonction indicatrice d'Euler.

Exemple : Pour $a = 4$ et $b = 1$ ou 3, on a $\mu(\mathcal{P}_{4,1}) = \mu(\mathcal{P}_{4,3}) = \frac{1}{2}$.

Notons $X : \mathcal{P} \to \mathbb{N}$ la fonction qui à tout nombre premier p (impair) associe le nombre d'itérations de la boucle `while` . X n'est pas une variable aléatoire mais il s'en faut de peu. En utilisant les notations standard des probabilités, nous avons donc $\mu(X = 0) = \frac{1}{2}$. Pour la "moitié" des nombres premiers, la boucle ne s'exécute pas !

Prenons maintenant $p \equiv 1[4]$. Il y a deux possibilités : $p \equiv 1[8]$ ou $p \equiv 5[8]$. En termes d'ensembles, cela signifie que $p \in \mathcal{P}_{8,1}$ ou $p \in \mathcal{P}_{8,5}$. $\frac{p-1}{4}$ est impair si et seulement si on est dans le second cas.

Ainsi, $\mu(X = 1) = \mu(\mathcal{P}_{8,5}) = \frac{1}{\varphi(8)} = \frac{1}{4}$. Je ne détaillerai pas le reste, mais on peut montrer assez facilement grâce au théorème de Dirichlet que pour tout entier n, $\mu(X = n) = \frac{1}{2^{n+1}}$.

Quelle est l'"espérance" de X ? Eh bien $E(X) = \sum_{n=0}^{\infty} n\mu(X = n) = \sum_{n=0}^{\infty} \frac{n}{2^{n+1}}$. Il nous reste à calculer la somme de cette série. Posons $f(x) = \frac{1}{2}\sum_{n=0}^{\infty}(\frac{x}{2})^n$. Cette série géométrique est convergente lorsque $|x| < 2$ et sa somme est $f(x) = \frac{1}{2-x}$. Admettons ici que f est dérivable sur $]-2, 2[$ et que sa dérivée est obtenue en dérivant la série terme à terme (résultat bien connu sur les séries entières). On a alors pour tout $x \in]-2, 2[$, $f'(x) = \frac{1}{2}\sum_{n=1}^{\infty}\frac{nx^{n-1}}{2^n}$ et on a donc $f'(1) = E(X)$. Or, pour tout $x \in]-2, 2[$, $f'(x) = \frac{1}{(2-x)^2}$. Donc :

$$E(X) = 1$$

Théorème : le nombre moyen d'itérations de la boucle `while` est ... 1.

In [ ]: