Nombres_Padiques

December 17, 2018

1 Nombres p-adiques

Marc Lorenzi

29 novembre 2018

Tout le monde le sait, un nombre réel c'est un nombre avec éventuellement une infinité de chiffres après la virgule. Alors c'est quoi un nombre avec une infinité de chiffres **AVANT** la virgule ? Tout le monde le sait, ça n'existe pas ! Eh bien si, ça existe. Cela s'appelle un nombre p-adique.

Nous allons nous intéresser seulement à une partie de l'ensemble des nombres p-adiques : l'ensemble $\mathbb{Z}_p$ des *entiers p-adiques* (ce sont les nombres p-adiques qui n'ont pas de chiffres **APRÈS** la virgule).

Dans tout le notebook, p désigne un nombre premier (sauf une fois de temps en temps, juste pour voir, où nous prendrons $p = 10$). Pour tout $x \in \mathbb{Z}$ et tout entier $n \geq 2$, nous noterons $x \bmod n$ le reste de la division de x par n.

Soit $E_p = \prod_{n=1}^{\infty} \mathbb{Z}/p^n\mathbb{Z}$. Un élément de E_p est une suite $(x_n)_{n\geq 1}$ telle que pour tout $n \geq 1$ on ait $x_n \in \mathbb{Z}/p^n\mathbb{Z}$.

Évidemment, modéliser les éléments de E_p en Python est un tantinet délicat, vu qu'il faudrait pouvoir stocker une infinité de nombres ! En revanche, si $x = (x_n)_{n\geq 1} \in E_p$, il est tout à fait possible de manipuler dans notre langage préféré la liste des n premiers termes de x, $[x_1, x_2, \ldots, x_n]$.

Vocabulaire : Nous appellerons $[x_1, x_2, \ldots, x_n]$ **l'approximation** de la suite x à l'ordre n. Pourquoi *approximation* ? Patience ...

Remarquer le subtil décalage entre les indices Python qui commencent à 0, et les indices des termes de la suite qui commencent à 1. Nous essaierons de ne pas l'oublier.

1.1 1. Entiers p-adiques

1.1.1 1.1 Suites cohérentes

Seuls certains éléments de E_p nous intéresseront. Nous allons nous focaliser sur les *suites cohérentes*. Remarquons tout d'abord que si $x, x' \in \mathbb{Z}$ et $n \geq 1$, alors

$$x \equiv x'[p^{n+1}] \Rightarrow x \equiv x'[p^n]$$

On peut donc considérer l'application $\varphi_n : \mathbb{Z}/p^{n+1}\mathbb{Z} \to \mathbb{Z}/p^n\mathbb{Z}$ qui à la classe de x modulo p^{n+1} associe la classe de x modulo p^n. Comme on peut le montrer facilement, cette application est un morphisme surjectif d'anneaux.

Notation : Nous noterons pour $x \in \mathbb{Z}/p^{n+1}\mathbb{Z}$, $\varphi_n(x) = x \bmod p^n$.

Remarque : En composant, $\varphi_{n+1} \circ \varphi_n$ est un morphisme surjectif de $\mathbb{Z}/p^{n+2}\mathbb{Z}$ sur $\mathbb{Z}/p^n\mathbb{Z}$. Et on peut généraliser. Nous nous autoriserons en fait à noter x mod p^n pour tout $x \in \mathbb{Z}/p^k\mathbb{Z}$ où $k \geq p$.

Définition : Une suite $(x_n)_{n\geq 1} \in E_p$ est dite **cohérente** lorsque pour tout $n \geq 1$ on a x_{n+1} mod $p^n = x_n$.

Exemple : Pour $p = 2$, la suite $(0, 2, 2, 10, 10, 10, \ldots)$ est cohérente. Je vous laisse le vérifier. Ou alors attendez 5 minutes, une fonction Python le fera pour nous.

Remarque essentielle : Si $(x_n)_{n\geq 1}$ est cohérente, et qu'on connaît la valeur d'un certain terme x_n de la suite, alors on connaît aussi $x_{n-1} = x_n$ mod p^{n-1}. Puis on peut calculer $x_{n-2}, x_{n-3}, \ldots, x_1$. En revanche, un doute subsiste sur $x_{n+1}, x_{n+2}, \ldots$. Ainsi, pour avoir une approximation à l'ordre n d'une suite cohérente, il suffit de posséder le $n$ième terme de la suite. Les termes précédents se retrouvent à partir de celui-ci.

La fonction `est_coherente` ci-dessous prend en paramètre une liste d'entiers s et un nombre premier p. Elle renvoie `True` si cette liste est une (approximation d'une) suite cohérente, et `False` sinon. La fonction affiche (horreur !!!) les étapes du calcul.

```
In [1]: def est_cohérente(s, p):
            n = len(s)
            for k in range(len(s) - 1, 0, -1):
                print('%3d : %-5d mod %5d^%d = %-5d' %(k + 1, s[k], p, k, s[k - 1]))
                if s[k] % (p ** k) != s[k - 1]: return False
            return True
```

```
In [2]: est_cohérente([0, 2, 2, 10, 10, 10, 10], 2)

  7 : 10    mod     2^6 = 10
  6 : 10    mod     2^5 = 10
  5 : 10    mod     2^4 = 10
  4 : 10    mod     2^3 = 2
  3 : 2     mod     2^2 = 2
  2 : 2     mod     2^1 = 0


Out[2]: True
```

La fonction `faire_coherente` prend en paramètres un entier x, un nombre premier p et un entier $N \geq 1$. Elle renvoie l'approximation à l'ordre N de la suite cohérente de E_p dont le N-ième terme est x.

```
In [3]: def faire_coherente(x, p, N):
            return [x % (p ** k) for k in range(1, N + 1)]

In [4]: faire_coherente(10, 2, 7)

Out[4]: [0, 2, 2, 10, 10, 10, 10]

In [5]: faire_coherente(1000000, 7, 10)

Out[5]: [1, 8, 155, 1184, 8387, 58808, 176457, 1000000, 1000000, 1000000]
```

```
In [6]: est_cohérente(_, 7)

 10 : 1000000 mod     7^9 = 1000000
  9 : 1000000 mod     7^8 = 1000000
  8 : 1000000 mod     7^7 = 176457
  7 : 176457 mod     7^6 = 58808
  6 : 58808 mod     7^5 = 8387
  5 : 8387  mod     7^4 = 1184
  4 : 1184  mod     7^3 = 155
  3 : 155   mod     7^2 = 8
  2 : 8     mod     7^1 = 1


Out[6]: True

In [7]: faire_coherente(-1, 2, 10)

Out[7]: [1, 3, 7, 15, 31, 63, 127, 255, 511, 1023]

In [8]: est_cohérente(_, 2)

 10 : 1023  mod     2^9 = 511
  9 : 511   mod     2^8 = 255
  8 : 255   mod     2^7 = 127
  7 : 127   mod     2^6 = 63
  6 : 63    mod     2^5 = 31
  5 : 31    mod     2^4 = 15
  4 : 15    mod     2^3 = 7
  3 : 7     mod     2^2 = 3
  2 : 3     mod     2^1 = 1


Out[8]: True
```

Faites d'autres essais pour bien comprendre ce qu'est une suite cohérente. Il va y en avoir dans les 25 prochaines pages :-).

1.1.2 1.2 L'anneau $\mathbb{Z}_p$

Définition : On appelle *entier p-adique* toute suite cohérente de E_p. On note $\mathbb{Z}_p$ l'ensemble des entiers p-adiques.

Résumons-nous. Un entier p-adique est une suite $x = (x_n)_{n\geq 1}$ telle que

- Pour tout entier $n \geq 1$, $x_n \in \mathbb{Z}/p^n\mathbb{Z}$
- Pour tout entier $n \geq 1$, $x_{n+1} \bmod p^n = x_n$.

Nous verrons des éléments de $\mathbb{Z}_p$ de plus en plus intéressants au fur et à mesure que nous avancerons dans notre étude.

Définissons maintenant une addition et une multiplication dans E_p.

Définition : Soient (x_n) et (y_n) deux éléments de E_p. On pose

$$x + y = (x_n + y_n)_{n \geq 1}$$

et

$$x \times y = (x_n \times y_n)_{n \geq 1}$$

Proposition : $(E_p, +, \times)$ est un anneau commutatif.

Proposition : La somme et le produit de deux entiers p-adiques sont encore des entiers p-adiques.

Proposition : $\mathbb{Z}_p$ est un sous-anneau de E_p.

Démonstrations : En exercice, ce sont de simples vérifications. Remarquons tout de même que le neutre pour l'addition est la suite nulle, et le neutre pour la multiplication est la suite "constante" égale à 1.

Exercice : Pourquoi ai-je mis des guillemets autour du mot "constante" ? Indication : c'est qui, 1 ?

1.1.3 1.3 Une injection de $\mathbb{Z}$ dans $\mathbb{Z}_p$

Proposition : L'application $\varphi : \mathbb{Z} \to \mathbb{Z}_p$ définie par $\varphi(x) = (x \bmod p^n)_{n \geq 1}$ est un morphisme d'anneaux injectif.

Nous identifierons dorénavant les entiers relatifs avec leur image dans $\mathbb{Z}_p$. Voici par exemple le nombre 10, vu comme élément de $\mathbb{Z}_2$. Plus précisément, voici les 12 premiers termes de la suite $\varphi(10)$. Je vous laisse deviner les suivants.

```
In [9]: faire_coherente(10, 2, 12)
```

```
Out[9]: [0, 2, 2, 10, 10, 10, 10, 10, 10, 10, 10, 10]
```

Et voici l'entier -1.

```
In [10]: faire_coherente(-1, 2, 12)
```

```
Out[10]: [1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095]
```

Exercice :

1- Soit $x = (x_n)_{n \geq 1} \in \mathbb{Z}_p$. Montrer que la suite est *stationnaire* si et seulement si il existe $n \in \mathbb{N}$ tel que $\varphi(n) = x$.

2- Critiquez la question 1. Comment une suite peut-elle être stationnaire alors que ses termes appartiennent à des ensembles différents ??? Reformulez correctement la question.

2- Quelles sont les images par φ des entiers négatifs ? Je vous rappelle que vous êtes dans un notebook et que vous disposez de la fonction `faire_coherente`.

```
In [11]: faire_coherente(-7, 2, 12)
```

```
Out[11]: [1, 1, 1, 9, 25, 57, 121, 249, 505, 1017, 2041, 4089]
```

Pas d'idée ? C'est pourtant facile, regardez ci-dessous.

```
In [12]: [2 ** k - 7 for k in range(1, 13)]
```

```
Out[12]: [-5, -3, 1, 9, 25, 57, 121, 249, 505, 1017, 2041, 4089]
```

```
In [13]: [(2 ** k - 7) % (2 ** k) for k in range(1, 13)]
```

```
Out[13]: [1, 1, 1, 9, 25, 57, 121, 249, 505, 1017, 2041, 4089]
```

1.1.4 1.4 Représentation des entiers p-adiques en Python

Nous allons créer une classe `Padic`. Un objet de cette classe est la représentation d'un entier p-adique, ou plutôt d'une approximation de celui-ci à un certiain ordre.

Rappelons que si $x = (x_n) \in \mathbb{Z}_p$ et que, pour un certain entier $k \geq 2$, nous connaissons x_k, alors nous possédons aussi x_{k-1} qui est le reste de la division de x_k (enfin, d'un des représentants de la classe x_k) par p^{k-1}. Pour un entier N donné, pour stocker l'approximation à l'ordre N, $[x_1, x_2, \ldots, x_N]$, il suffit donc de connaître x_N. Et aussi, bien entendu, p et N.

Voici la classe `Padic`. Un objet de cette classe possède trois champs :

- Un champ p qui contient la valeur de … p.
- Un champ N contenant la "précision" à laquelle notre entier p-adique est stocké.
- Un champ n qui contient x_N.

Si un tel objet est la représentation d'un entier p-adique $x = (x_n)_{n\geq 1}$, on peut donc à partir de notre objet retrouver les valeurs de $x_1, x_2, \ldots, x_N$.

```
In [14]: class Padic:

             def __init__(self, n, p, N):
                 q = p ** N
                 self.n = n % q
                 self.p = p
                 self.N = N
                 self.q = q

             def __repr__(self):
                 n = self.n
                 s = ''
                 for k in range(self.N):
                     if self.p <= 10: s = str(n % self.p) + s
                     else: s = str(n % self.p) + ' ' + s
                     n = n // self.p
                 return '...' + s

             def termes(self):
                 s = [self.n % (self.p ** k) for k in range(1, self.N + 1)]
                 return s
```

La méthode `__repr__` permet un affichage joli des objets. Un exemple ? Oui, je, sais, 10 n'est pas un nombre premier. Mais les choses seront plus claires avec ce nombre parce que, je ne sais pas pourquoi, les gens ont moins de mal avec l'écriture en base 10 :-).

```
In [15]: x = Padic(123456, 10, 9)
         print(x)
```

```
...000123456
```

Euh, oui, ce sont les chiffres de notre nombre en base 10. Remarquez les points de suspension à gauche : il y a une infinité de chiffres, tous nuls dans notre exemple. Remarquez également la méthode `termes` qui renvoie les N premiers termes de la suite cohérente.

```
In [16]: x.termes()
```

```
Out[16]: [6, 56, 456, 3456, 23456, 123456, 123456, 123456, 123456]
```

Nous retrouvons ici les N premiers termes de la suite (x_n). Comprenez-vous bien ce qu'est une suite cohérente modulo 10 ? Comprenez-vous pourquoi j'ai dit au début du notebook que nous allions travailler avec des nombres qui ont une infinité de chiffres **AVANT** la virgule ? Parce que ça y est, nous y sommes. Le nombre -1, par exemple, est l'entier 2-adique qui s'écrit avec une infinité de 1.

```
In [17]: x = Padic(-1, 2, 20)
         print(x)
```

```
...11111111111111111111
```

Évidemment, si $p > 10$ cette façon d'afficher les chiffres du nombre en base p ne convient pas. Voilà pour quoi la méthode `__repr__` de la classe `Padic` envisage deux possibiltés. Tentons avec $p = 11$:

```
In [18]: x = Padic(5678, 11, 9)
         print(x)
```

```
...0 0 0 0 0 4 2 10 2
```

```
In [19]: x.termes()
```

```
Out[19]: [2, 112, 354, 5678, 5678, 5678, 5678, 5678, 5678]
```

Remarque philosophique : Tout cela est bien beau, me direz vous, mais nous aimerions voir un vrai entier p-adique, et pas juste un entier relatif écrit de façon très compliquée. Mais vous me demandez l'impossible !

Imaginez que vous me disiez : "Je veux voir un réel, un vrai, pas juste un nombre décimal" ? Je vous répondrais la même chose. Un "vrai" réel ça ne se voit pas, cela s'approche avec des décimaux (montrez-moi $\sqrt{2}$!). Eh bien un vrai nombre p-adique c'est pareil, cela s'approche avec des entiers naturels. Et comme nous verrons d'ici quelque temps que $\mathbb{N}$ est **dense** dans $\mathbb{Z}_p$, ce que je raconte ici est bien plus qu'une analogie. La situation des réels et celle des nombres p-adiques sont identiques.

1.1.5 1.5 Les opérations d'anneau

Implémenter addition, opposé, soustraction et multiplication est immédiat. Il n'y a qu'à opérer modulo p^N. J'ai rajouté également une fonction de division ... qui ne marche pas (enfin pas encore). Diviser dans un anneau n'est pas toujours possible ni facile, et deux questions restent à résoudre (et non des moindres) :

- Quels sont les éléments inversibles de $\mathbb{Z}_p$?
- Comment calculer l'inverse d'un élément inversible ?

La classe `Padic` ci-dessous recopie le code déjà vu plus haut et rajoute celui les opérations de base. Aucune difficulté.

```
In [20]: class Padic:

             def __init__(self, n, p, N):
                 q = p ** N
                 self.n = n % q
                 self.p = p
                 self.N = N
                 self.q = q

             def __repr__(self):
                 n = self.n
                 s = ''
                 for k in range(self.N):
                     if self.p <= 10: s = str(n % self.p) + s
                     else: s = str(n % self.p) + ' ' + s
                     n = n // self.p
                 return '...' + s

             def termes(self):
                 s = [self.n % (self.p ** k) for k in range(1, self.N + 1)]
                 return s

             def __add__(self, y):
                 assert(self.p == y.p)
                 N = min(self.N, y.N)
                 return Padic(self.n + y.n, self.p, N)

             def __mul__(self, y):
                 assert(self.p == y.p)
                 N = min(self.N, y.N)
                 return Padic(self.n * y.n, self.p, N)

             def __neg__(self):
                 return Padic(-self.n, self.p, self.N)

             def __sub__(self, y):
                 return self + (- y)

             def __truediv__(self, y):
                 z = inverse(y)
                 return self * z
```

```python
    def __pow__(self, n):
        if n < 0:
            return inverse(self ** (-n))
        else:
            y = Padic(1, self.p, self.N)
            for k in range(n):
                y = y * self
            return y
```

Nous pouvons enfin montrer que $8 \times 7 \simeq 56$:-).

```
In [21]: x = Padic(8, 3, 10) * Padic(7, 3, 10)
         print(x)
         print(x.termes())
```

```
...0000002002
[2, 2, 2, 56, 56, 56, 56, 56, 56, 56]
```

Que vaut 1547 dans $\mathbb{Z}_3$?

```
In [22]: Padic(1547, 3, 10)
```

```
Out[22]: ...0002010022
```

Que vaut $3^4 \times 1547$?

```
In [23]: x = Padic(1547, 3, 10) * Padic(3, 3, 20) ** 4
         print(x)
```

```
...0100220000
```

Règle : Multiplier dans $\mathbb{Z}_p$ par p^k revient à ajouter k zéros au nombre.

Je vous laisse prouver la règle. Testez sur d'autres exemples, jusqu'à ce que vous ayez bien compris comment fonctionnent les opérations, la multiplication par p, le passage à l'opposé, la soustraction, les puissances ...

L'intérêt d'un notebook est de tester, tester, et ... tester. Si on ne teste rien et qu'on se contente de lire, alors autant prendre un PDF.

1.2 2. Valuation et valeur absolue p-adiques

Nous allons maintenant définir dans $\mathbb{Z}_p$ une notion de distance, qui sera très différente de la distance usuelle entre les entiers relatifs. Les nombres divisibles par de grandes puissances de p seront très proches de 0.

1.2.1 2.1 Valuation p-adique

Définition : Soit $x = (x_n) \in \mathbb{Z}_p$. Si $x \neq 0$ on appelle valuation p-adique de x l'entier $v_p(x) = \min\{k \in \mathbb{N}^*, x_k \neq 0\} - 1$. On pose également $v_p(0) = +\infty$.

La fonction `valuation` ci-dessous prend un entier p-adique x (ou du moins une approximation de celui-ci). Si l'approximation de x est non nulle, la fonction renvoie v, la valuation de x. Si, en revanche, l'approximation de x est nulle, cela ne veut pas forcément dire que x est nul. La fonction renvoie `N+1` où N est l'ordre d'approximation de x. Cela signifie que $v(x) \geq N+1$, et on ne peut pas dire mieux.

Remarquez le parallèle avec $\mathbb{R}$. Si $x = 0.00000\ldots$ cela ne veut pas dire que $x = 0$, mais seulement que $|x| < 10^{-5}$, parce qu'il y a 5 zéros après la virgule.

La fonction `valuation` fonctionne comme suit : elle cherche la plus grande puissance de p qui divise x (ou plus exactement `x.n`). Quel rapport avec la définition de la valuation ??? Voyez le théorème ci-dessous.

```
In [24]: def valuation(x):
             v = 0
             n = x.n
             while n % x.p == 0 and v <= x.N:
                 v += 1
                 n = n // x.p
             return v
```

Par exemple, la valuation de $63 = 3^2 \times 7$ dans $\mathbb{Z}_3$ est 2 parce que ce nombre finit par 2 zéros.

```
In [25]: x = Padic(63, 3, 5)
         print(x)
         print(valuation(x))
```

```
...02100
2
```

Et la valuation de 0 "à l'approximation 3^5" est au moins 6.

```
In [26]: valuation(Padic(0, 3, 5))
```

```
Out[26]: 6
```

Proposition : Soit $x = (x_n)_{n\geq 1} \in \mathbb{Z}_p \setminus \{0\}$. On a - Pour tout $n \geq 1$, $p^{v_p(x)}$ divise x_n. - Il existe $n \geq 1$ tel que $p^{v_p(x)+1}$ ne divise pas x_n (en l'occurrence, $n = v_p(x) + 1$). - $v_p(x)$ est le seul entier vérifiant les deux points précédents.

Démonstration : Plaçons nous d'abord dans le cas où $v_p(x) = 0$. Le premier point est trivial, puisqu'il faut montrer que pour tout $n \geq 1$, 1 divise x_n. Le second point l'est aussi (trivial) puisque, comme $v_p(x) = 0$, $x_1 \neq 0$. Donc p^1 ne divise pas x_1.

Supposons maintenant que $v = v_p(x) \geq 1$. On a donc $x_1 = \ldots = x_v = 0$ et $x_{v+1} \neq 0$. Comme $x_{v+1} \bmod p^v = x_v = 0$, on en déduit que p^v divise x_{v+1}. Pour tout $k \geq v+1$, on a $x_k \bmod p^{k-1} = x_{k-1}$ donc aussi $x_k \bmod p^v = x_{k-1}$. On en déduit facilement que pour tout $k \geq v+1, x_k \bmod p^v = 0$. Si $k \leq v$ la congruence est évidente puisque $x_k = 0$. Concernant le second point, $x_{v+1} \neq 0$ et donc $x_{v+1} \bmod p^{v+1} \neq 0$.

Le dernier point est clair, en y réfléchissant un peu :-).

Exercice : Réfléchissez.

Corollaire : Soit $x \in \mathbb{Z}_p \setminus \{0\}$. On a $x = p^{v_p(x)}u$ où $v_p(u) = 0$. Et $v_p(x)$ est le seul entier vérifiant cette propriété.

Proposition : On a pour tous entiers p-adiques x et y :

- $v_p(xy) = v_p(x) + v_p(y)$
- $v_p(x + y) \geq \min(v_p(x), v_p(y))$
- Si $v_p(x) \neq v_p(y)$ alors $v_p(x + y) = v_p(x) + v_p(y)$

On fait bien entendu les conventions évidentes dans le cas où l'une des valuations est infinie.

Démonstration : Si $x = 0$ ou $y = 0$ la propriété est évidente. Supposons donc x et y non nuls. Par le corollaire précédent, on a $x = p^{v_p(x)}u$ où $v_p(u) = 0$. De même, $y = p^{v_p(y)}u'$ où $v_p(u') = 0$. - $xy = p^{v_p(x)+v_p(y)}uu'$. Maintenant, $u_1 \neq 0$ et $u_1' \neq 0$ puisque u et u' sont de valuation nulle. Mais alors $u_1u_1' \neq 0$ car $u_1, u_1' \in \mathbb{Z}/p\mathbb{Z}$, qui est un corps. Voilà où intervient le fait que p est premier. Donc, $v_p(uu') = 0$.

- Supposons par exemple $v(x) \leq v(y)$. On a $x + y = p^{v_p(x)}u + p^{v_p(y)}u' = p^{v_p(x)}u''$ où $u'' = u + p^{v_p(y)-v_p(x)}u'$. Ainsi, $p^{v_p(x)}$ divise $x + y$, et $v_p(x + y) \geq v_p(x) = \min(v_p(x), v_p(y))$.

- Supposons en plus que $v_p(x) < v_p(y)$. Reprenons $u'' = u + p^{v_p(y)-v_p(x)}u' = u + p^k u'$ où $k > 0$. On a alors $u_1'' = u_1 \neq 0$ donc $v_p(u'') = 0$. Ainsi, $v_p(x + y) = v_p(x) = \min(v_p(x), v_p(y))$.

Corollaire : L'anneau $\mathbb{Z}_p$ est intègre.

Démonstration : Soient $x, y \in \mathbb{Z}_p$. Supposons que $xy = 0$. On a donc $v_p(xy) = v_p(x) + v_p(y) = \infty$. Ceci n'est possible que si $v_p(x) = \infty$ ou $v_p(y) = \infty$, c'est à dire $x = 0$ ou $y = 0$.

Remarque : Que se passe-t-il si on prend pour p un entier non premier ? Eh bien $\mathbb{Z}_p$ est toujours un anneau mais on perd l'intégrité. Nous verrons à la fin du notebook un exemple de deux entiers 10-adiques non nuls très intéressants dont le produit est nul.

1.2.2 2.2 Les inversibles de $\mathbb{Z}_p$

Proposition : Soit $x \in \mathbb{Z}_p$. x est inversible pour la multiplication si et seulement si $v_p(x) = 0$.

Démonstration : Supposons x inversible. Il existe $y \in \mathbb{Z}_p$ tel que $xy = 1$. On a donc $v_p(xy) = v_p(x) + v_p(y) = v_p(1) = 0$. Ainsi, $v_p(x) = v_p(y) = 0$.

Inversement, supposons $v_p(x) = 0$. On a donc $x_1 \neq 0$. La suite x est cohérente, donc p ne divise aucun des x_n (facile), ce qui prouve que les x_n sont premiers avec p^n. Rappelons que p est premier ! Pour tout $n \geq 1$, notons y_n l'inverse de x_n dans $\mathbb{Z}/p^n\mathbb{Z}$. Pourquoi un tel y_n existe-t-il ? Eh bien, d'après le théorème de Bézout, comme x_n et p^n sont premiers entre eux, il existe deux entiers y_n et z_n tels que $x_ny_n + p^nz_n = 1$. En réduisant l'égalité modulo p^n il vient $x_ny_n = 1$. Soit $y = (y_n)_{n\geq 1}$. Admettons provisoirement que la suite (y_n) est cohérente. On a alors $x.y = (x_n.y_n) = (1)$, le neutre pour la multiplication dans $\mathbb{Z}_p$.

Soyons courageux et montrons que la suite (y_n) est cohérente. Soit $n \geq 1$. On a $x_{n+1}y_{n+1} = 1 \equiv 1[p^{n+1}]$. Par ailleurs, la suite (x_n) étant cohérente, il existe $\lambda \in \mathbb{Z}$ tel que $x_{n+1} = x_n + \lambda p^n$. De là $x_{n+1}y_{n+1} = (x_n + \lambda p^n)y_{n+1} \equiv 1[p^n]$ ou encore $x_ny_{n+1} \equiv 1[p^n]$. Mais on a $x_ny_n \equiv 1[p^n]$. En soustrayant, on obtient $x_n(y_{n+1} - y_n) \equiv 0[p^n]$. Mais comme x_n est inversible dans $\mathbb{Z}/p^n\mathbb{Z}$, on en déduit que $y_{n+1} - y_n \equiv 0[p^n]$.

Ça y est, nous tenons les inversibles ! Nous savons qui inverser, reste à savoir comment. La démonstration de la proposition nous le dit, il suffit d'appliquer le théorème de Bézout pour trouver des inverses dans $\mathbb{Z}/p^n\mathbb{Z}$. Comment appliquer le théorème de Bézout ? En utilisant l'algorithme d'Euclide. La fonction `invmod` ci-dessous prend en paramètres deux entiers x et n tels que x est premier avec n. Elle effectue l'algorithme d'Euclide pour renvoyer un entier y tel que $xy \equiv 1[n]$.

```
In [27]: def invmod(x, n):
             u1, v1, r1 = 1, 0, x
             u2, v2, r2 = 0, 1, n
             while r2 != 0:
                 q = r1 // r2
                 u = u1 - q * u2
                 v = v1 - q * v2
                 r = r1 - q * r2
                 u1, v1, r1 = u2, v2, r2
                 u2, v2, r2 = u, v, r
             assert r1 == 1
             return u1 % n
```

```
In [28]: invmod(2, 3 ** 10)
```

```
Out[28]: 29525
```

```
In [29]: (2 * _) % (3 ** 10)
```

```
Out[29]: 1
```

Et voici la fonciton `inverse`. Elle prend un entier p-adique x en paramètre et renvoie x^{-1} dans le cas où $v(x) = 0$. Sinon, la fonction lève une exception.

```
In [30]: def inverse(x):
             v = valuation(x)
             if v != 0:
                 raise Exception('Non inversible')
             else:
                 y = invmod(x.n, x.p ** x.N)
                 return Padic(y, x.p, x.N)
```

Maintenant nous pouvons inverser et diviser dans $\mathbb{Z}_p$. Essayons. Quel est l'inverse de 17 dans $\mathbb{Z}_3$?

```
In [31]: x = Padic(1, 3, 20) / Padic(17, 3, 20)
         print(x)
```

```
...20212211020100112022
```

```
In [32]: print(x.termes())
```

```
[2, 8, 8, 62, 143, 386, 386, 386, 6947, 6947, 125045, 125045, 656486, 2250809, 11816747, 4051456
```

Bigre . . . Comme on n'est pas obligé d'y croire, vérifions.

```
In [33]: x * Padic(17, 3, 20)
```

```
Out[33]: ...00000000000000000001
```

Maintenant, nous voilà convaincus que $\mathbb{Z}_p$ contient des objets intéressants. Par exemple, $\frac{1}{17}$ est un **ENTIER** 3-adique. Et de façon plus générale, tout rationnel $\frac{p}{q}$ où q n'est pas un multiple de 3 est aussi un entier 3-adique. Voici les 1000 derniers chiffres de l'entier 3-adique égal à la célèbre fraction $\frac{355}{113}$.

```
In [34]: x = Padic(355, 3, 1000) / Padic(113, 3, 1000)
         print(x)
```

```
...0000201100120001102201010002212102020012201211110102110122220211221022211200212122200101202O2
```

1.2.3 2.2 Valeur absolue p-adique

Définition : Pour tout $x \in \mathbb{Z}_p$, on pose $|x|_p = p^{-v_p(x)}$, avec la convention que $p^{-\infty} = 0$. La quantité $|x|_p$ est appelée la valeur absolue p-adique de x.

```
In [35]: def vabs(x):
             v = valuation(x)
             return x.p ** (- v)
```

```
In [36]: vabs(Padic(63, 3, 8))
```

```
Out[36]: 0.1111111111111111
```

Proposition : On a les propriétés suivantes : - Pour tout $x \in \mathbb{Z}_p$, $|x|_p \in \mathbb{R}_+$ - Pour tout $x \in \mathbb{Z}_p$, $|x|_p = 0$ si et seulement si $x = 0$. - Pour tous $x, y \in \mathbb{Z}_p$, $|xy|_p = |x|_p|y|_p$ - Pour tous $x, y \in \mathbb{Z}_p$, $|x+y|_p \leq \max(|x|_p, |y|_p)$ (inégalité ultramétrique)

L'inégalité ultramétrique implique évidemment que $|x+y|_p \leq |x|_p + |y|_p \ldots$ et bien d'autres choses. C'est elle qui fait de $\mathbb{Z}_p$ un ensemble bizarre.

Démonstration : Laissée en exercice, il suffit d'utiliser les propriétés de la valuation.

1.2.4 2.3 Distance p-adique

Munis d'une valeur absolue, nous pouvons maintenant définir une *distance* sur $\mathbb{Z}_p$ pour en faire un *espace métrique*. Pour tous $x, y \in \mathbb{Z}_p$, posons $d_p(x, y) = |y - x|_p$.

Proposition : La fonction d_p est une *distance* sur l'ensemble $\mathbb{Z}_p$. Précisément :

- Pour tous $x, y \in \mathbb{Z}_p$, $d_p(x, y) \in \mathbb{R}_+$
- Pour tous $x, y \in \mathbb{Z}_p$, $d_p(x, y) = 0$ si et seulement si $x = y$.
- Pour tous $x, y, z \in \mathbb{Z}_p$, $d_p(x, z) \leq \max(d_p(x, y), d_p(y, z))$ (inégalité ultramétrique)

L'inégalité ultramétrique entraîne bien évidemment l'inégalité triangulaire : $d_p(x, z) \leq d_p(x, y) + d_p(y, z)$.

```
In [37]: def distance(x, y):
             return vabs(y - x)
```

```
In [38]: distance(Padic(8, 3, 10), Padic(-1, 3, 10))
```

```
Out[38]: 0.1111111111111111
```

1.2.5 2.4 Suites convergentes dans $\mathbb{Z}_p$

Considérons la somme $S_n = \sum_{k=0}^{n} 2^k = 2^{n+1} - 1$. Considérons cet entier comme un élément de $\mathbb{Z}_2$.

```
In [39]: S20 = Padic(2 ** 20 - 1, 2, 30)
         print(S20)
```

```
...000000000011111111111111111111
```

La distance entre S_{20} et -1 est inférieure à 10^{-6}.

```
In [40]: print(distance(S20, Padic(-1, 2, 30)))
```

```
9.5367431640625e-07
```

Plus généralement, on a $|S_n - (-1)|_2 = |2^{n+1}| = \frac{1}{2^{n+1}}$. Cette quantité tend vers 0 lorsque n tend vers l'infini.

Définition : Soit $(u_n)_{n \geq 0}$ une suite d'éléments de $\mathbb{Z}_p$. Soit $\ell \in \mathbb{Z}_p$. On dit que u_n tend vers ℓ lorsque n tend vers l'infini lorsque $|u_n - \ell|_p$ tend vers 0.

Munis de notre définition, nous concluons que $1 + 2 + 2^2 + \ldots + 2^n \to -1$ lorsque $n \to \infty$. Dit autrement,

$$\sum_{n=0}^{\infty} 2^n = -1$$

Impossible, me direz-vous ? Éduqués dans la $\mathbb{R}$-attitude, nous avons du mal à concevoir le monde de $\mathbb{Z}_p$. Mais le fait est là, l'anneau $\mathbb{Z}_p$ a été construit, une notion de convergence a été définie. Le reste n'est que raisonnements. Et si le résultat nous déplait, nous n'y pouvons rien :-).

Revenons sur le vocabulaire introduit au début du notebook. Si $x = (x_n)_n \geq 1$ nous avons appelé la liste $[x_1, x_2, \ldots, x_N]$ **l'approximation** de x à l'ordre N. Pourquoi "approximation" ? Posons $u_N = (x_1, \ldots, x_N, 0, 0, 0, \ldots)$. On a $x - u_N = (0, \ldots, 0, x_{N+1}, x_{N+1}, \ldots)$. Ainsi, $v_p(x - u_N) \geq N + 1$ et donc

$$d_p(u_N, x) \leq \frac{1}{p^{N+1}}$$

On en déduit que $u_N \to x$ lorsque N tend vers l'infini. Le mot "approximation" est donc tout à fait mérité : u_N est une approximation de x à p^{-N-1} près.

Remarquons que pour tout N, u_N est lui-même une suite. Cette suite est stationnaire (elle finit par des zéros), donc u_N est un **entier naturel**. Que venons nous de montrer ? Nous venons de prouver que tout entier p-adique est la limite d'une suite d'entiers naturels.

Proposition : $\mathbb{N}$ est **dense** dans $\mathbb{Z}_p$.

Remarquez que cela entraîne que tout entier négatif est la limite d'une suite d'entiers naturels :-).

Exercice : On prend $p = 2$. Trouvez une suite d'entiers naturels qui tend vers -13.

1.2.6 2.5 Le monde étrange des nombres p-adiques

La vie dans $\mathbb{Z}_p$ est vraiment étrange. Pour vous donner un vague aperçu de vos prochaines vacances je me contenterai de deux résultats.

Proposition : Dans $\mathbb{Z}_p$, tous les triangles sont isocèles.

Démonstration : Soient $a, b, c \in \mathbb{Z}_p$. Soient $d = |a-b|_p$, $d' = |b-c|_p$ et $d = |a-c|_p$. Pour fixer les idées, supposons par exemple $d \leq d' \leq d''$. On a $d'' = |a-c|_p = |(a-b)+(b-c)|_p$. L'inégalité ultramétrique vue au début du paragraphe sur la valeur absolue p-adique nous prouve que

$$d'' = |a-c|_p = |(a-b)+(b-c)|_p \leq \max(d, d') = d'$$

Ainsi, $d \leq d' \leq d'' \leq d'$. Dans un triangle, *les deux côtés les plus grands* sont égaux.

Définition : Soient $a \in \mathbb{Z}_p$ et $r > 0$. On appelle boule de centre a et de rayon r l'ensemble

$$B(a, r) = \{x \in \mathbb{Z}_p, |x-a|_p < r\}$$

Rien de bien original, sauf que ...

Proposition : Pour tout $b \in B(a, r)$ on a $B(b, r) = B(a, r)$. Dit autrement, tout point dans une boule est **UN** centre d'une boule. Les boules ont des tonnes de centres.

Démonstration : Soit $x \in B(b, r)$. Le triangle (abx) est isocèle, ses deux plus grands côtés étant égaux. Or $d_p(b, a) < r$ et $d_p(b, x) < r$. Donc $d_p(a, x) < r$ sinon le plus grand côté du triangle serait plus grand que les deux autres. Ainsi, $x \in B(a, r)$ et donc $B(b, r) \subset B(a, r)$.

Comme $b \in B(a, r)$, on a $d_p(a, b) < r$ et donc $a \in B(b, r)$. La situation étant symétrique en a et b, on a donc aussi $B(a, r) \subset B(b, r)$.

Exercice : Montrer que si deux boules (pas forcément de même rayon) ont UN point commun, alors l'une des deux est incluse dans l'autre.

Exercice : Montrer que $\mathbb{Z}_p$ est la réunion de p boules disjointes de rayon 1. Considérer pour cela les boules $B(k, 1)$ pour $k = 0, 1, \ldots, p-1$.

1.3 3. La méthode de Newton

1.3.1 3.1 Introduction

Le nombre 2 a-t-il une racine carrée ? Ben oui, c'est $\sqrt{2}$. Dans $\mathbb{R}$, certes. Mais dans ... $\mathbb{Z}_7$?? Il y aurait donc des irrationnels dans $\mathbb{Z}7$? Nous allons voir que oui. Un algorithme ébauché par Newton (dans un cadre très différent !) au XVIIe siècle permet de trouver des solutions à des équations dans $\mathbb{Z}_p$, comme par exemple l'équation $x^2 - 2 = 0$. Nous terminerons ce notebook par une tentative audacieuse, prendre $p = 10$ pour trouver un objet x tel que $x^2 = x$ dans $\mathbb{Z}_{10}$, cet x n'étant ni 0, ni 1. Nous en trouverons même deux.

Remarquons tout d'abord que si $x \in \mathbb{Z}_p \setminus \{0\}$ possède une racine carrée y, il en possède exactement deux. En effet, si $z^2 = x$, alors $z^2 = y^2$ donc $(z-y)(z+y) = 0$. Mais $\mathbb{Z}_p$ est intègre, propriété essentielle, donc n'admet pas de diviseurs de 0. Ainsi, $z = \pm y$. Inversement, $(\pm y)^2 = y^2 = x$. Bien entendu, si p n'est premier, rien n'empêche x d'avoir 5 ou 6 racines carrées ...

1.3.2 3.2 Remontée modulaire

Soit $f : \mathbb{Z}_p \to \mathbb{Z}_p$, disons un polynôme pour fixer les idées.

Suppposons connu x tel que $v_p(f(x)) \geq k$, où $k \geq 1$. Peut-on trouver x' tel que $v_p(f(x')) \geq k+1$? Oui si $f'(x) \not\equiv 0[p^k]$.

Soit $x' = x - f(x)f'(x)^{-1}$. Nous avons $f(t) = f(x) + (t-x)f'(x) + (t-x)^2 Q(t)$ où Q est un polynôme. Prenons $t = x'$:

$$f(x') = f(x) + (x'-x)f'(x) + (x'-x)^2 Q(x') = f(x)^2 f'(x)^2 Q(x')$$

Ainsi,

$$v_p(f(x')) = 2v_p(f(x)) + 2v_p(f'(x)) + 2v_p(Q(x')) \geq 2v_p(f(x)) \geq 2k \geq k+1$$

puisque $k \geq 1$.

Remarquons également que

$$v_p(x'-x) = v_p(-f(x)f'(x)^{-1}) = v_p(f(x)) + v_p(f'(x)^{-1}) = v_p(f(x)) \geq k$$

Ainsi, $|x'-x|_p < \frac{1}{p^k}$.

La fonction `lift` prend en paramètres une fonction f, sa dérivée f_1, un entier p-adique x. Elle suppose que $v_p(f(x)) \geq k$, où $k \geq k$. La fonction renvoie un entier p-adique x' tel que $v_p(f(x')) \geq k+1$ et $|x'-x|_p < \frac{1}{p^k}$.

```
In [41]: def lift(f, f1, x):
             return x - f(x) / f1(x)
```

Prenons un petit exemple. Soit $f : x \mapsto x^2 - 2$. Soit $p = 7$. Nous avons $3^2 \equiv 2[7]$. Donc $v_7(f(3)) \geq 1$.

```
In [42]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, Padic(3,7,20))

Out[42]: ...11111111111111111113
```

Le nombre x' renvoyé ci-dessus vérifie donc $v_7(f(x')) \geq 2$. Rien ne nous empêche de recommencer :

```
In [43]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, _)

Out[43]: ...35233062113523306213

In [44]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, _)

Out[44]: ...06401623525321216213

In [45]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, _)

Out[45]: ...33302011266421216213

In [46]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, _)

Out[46]: ...64112011266421216213

In [47]: lift(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, _)

Out[47]: ...64112011266421216213
```

Ça ne bouge plus … automatisons tout cela !

1.3.3 3.3 Le lemme de Hensel

Proposition : Soit $f : \mathbb{Z}_p \to \mathbb{Z}_p$ une fonction polynôme (pour fixer les idées). Soit $x_1 \in \mathbb{Z}_p$ tel que $v_p(f(x_1)) \geq 1$ et $v_p(f'(x_1)) = 0$. Soit la suite $(x_n)_{n \geq 1}$ définie par $x_{n+1} = x_n - f(x_n)f'(x_n)^{-1}$. Alors :

- La suite (x_n) est une suite qui est bien définie et qui tend vers $x \in \mathbb{Z}_p$ tel que $f(x) = 0$.
- x est l'unique racine x de f telle que $x \bmod p = x_1$.

Ce notebook est déjà bien long. Je ne prouverai pas ce résultat et vous dirai simplement que nous avons en déjà beaucoup montré (mais pas tout) dans le paragraphe **3.2**.

Écrivons une fonction qui calcule tout cela. La fonction `newton` prend en paramètres une fonction $f : \mathbb{Z}_p \to \mathbb{Z}_p$, sa dérivée f_1, un nombre premier p et un entier N. Elle renvoie une racine de l'équation $f(x) = 0$ à l'approximation N. Dit autrement, elle renvoie $x \in \mathbb{Z}_p$ tel que $v_p(x) > N$, ou encore $|f(x)|_p < \frac{1}{p^N}$.

La fonction `newton` prend également un paramètre optionnel `guess`. Si `guess` est différent de `None`, ce doit être un entier p-adique k tel que $f(k) \bmod p = 0$. Si `guess` vaut `None`, un appel à `find_root` trouve un tel entier.

```
In [48]: def newton(f, f1, p, N, guess=None):
             if guess == None:
                 x = find_root(f, p, N)
             else:
                 x = guess
             while valuation(f(x)) <= N:
                 x = lift(f, f1, x)
             return x
```

Que fait `find_root` ? C'est en fait le point problématique. Elle teste tous les entiers k de 0 à $p-1$ jusqu'à ce qu'elle en trouve (ou pas) un tel que $f(k) \bmod p = 0$. Ceci est totalement inefficace lorsque p est grand, mais je ne discuterai pas plus avant de ce sujet. Pour les personnes intéressées, le notebook sur les résidus quadratiques permet de répondre à la question pour la fonction $x \mapsto x^2 - a$: l'algorithme de Tonnelli-Shanks y est détaillé, il permet un calcul très efficace d'une racine carrée modulo p.

```
In [49]: def find_root(f, p, N):
             for k in range(p):
                 x = Padic(k, p, N)
                 if valuation(f(x)) > 0: return x
             return None
```

```
In [50]: x = newton(lambda x:x ** 2 - Padic(2, 7, 20), lambda x:Padic(2, 7, 20) * x, 7, 20)
         print(x)
```

```
...64112011266421216213
```

1.3.4 3.4 Racine carrée d'un entier p-adique

L'exemple ci-dessus nous montre que 2 a une racine carrée (et donc deux) dans $\mathbb{Z}_7$. Automatisons le procédé : la fonction `sqrt` prend un entier p-adique x en paramètre et renvoie une des deux racines carrées de x (à la même précision que x) si x possède une racine carrée.

```
In [51]: def sqrt(x):
             return newton(lambda t:t ** 2 - x, lambda t: Padic(2, x.p, x.N) * t, x.p, x.N)
```

Voici une racine carrée de 2 dans $\mathbb{Z}_7$ avec 1000 chiffres avant la virgule.

```
In [52]: x = sqrt(Padic(2, 7, 1000))
         print(x)
```

```
...513661665651640022423522243403205345214020511300265531323533262550542051014342314550160125254
```

Un autre exemple : $i \in \mathbb{Z}_5$. Dit autrement, -1 a une racine carrée dans $\mathbb{Z}_5$.

```
In [53]: x = sqrt(Padic(-1, 5, 1000))
         print(x)
```

```
...443231120214413101214243132240241144144201331024312314430143034103102231044014421102030020421
```

```
In [54]: print(- x * x)
```

```
...000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

1.3.5 3.5 Deux nombres bizarres dans $\mathbb{Z}_{10}$

Trouvons tous les entiers x tels que $x^2 = x$.

0 et 1, me direz-vous ? Oui, certes, mais si je vous parle d'entiers 10-adiques ? Alors il y en a 4. En effet il y a 4 entiers entre 0 et 9 tels que $x^2 \equiv x[10]$, à savoir 0, 1, 5 et 6. Car $5 \times 5 = 25 \equiv 5[10]$ et $6 \times 6 = 36 \equiv 6[10]$.

Lançons la fonction `newton` avec $f(x) = x^2 - x$ et, hasardeusement, $p = 10$ (je rappelle que 10 n'est pas premier). Prenons une valeur du paramètre guess égale à 5.

```
In [55]: N = 100
```

```
In [56]: a = newton(lambda x:x ** 2 - x, lambda x: Padic(2, 10, N) * x - Padic(1, 10, N), 10, N,
         print(a)
```

```
...395300731910816980293850989006216650958086381100055742342323089610900410661997739225625991821
```

Relançons avec guess=6.

```
In [57]: b = newton(lambda x:x ** 2 - x, lambda x: Padic(2, 10, N) * x - Padic(1, 10, N), 10, N,
         print(b)
```

```
...604699268089183019706149010993783349041913618899944257657676910389099589338002260774374008178
```

Juste pour voir, calculons $a + b$ et ab.

```
In [58]: a + b
```

```
Out[58]: ...000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
In [59]: a * b
```

```
Out[59]: ...000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Diantre, aurait-on $a + b = 1$ et $ab = 0$? Cette deuxième égalié prouverait que $\mathbb{Z}_{10}$ n'est pas un anneau intègre !

Proposition : $a + b = 1$ et $ab = 0$.

Démonstration : $(ab)^2 = a^2b^2 = ab$. Donc ab est solution de l'équation $x^2 = x$. En remarquant que $a = ...5$ et $b = ...6$ nous voyons que ab mod $10 = 0$. En espérant qu'il reste quelque chose du lemme de Hensel pour $p = 10$, on en déduit que $ab = 0$.

Par ailleurs $(a + b)^2 = a^2 + 2ab + b^2 = a + 0 + b = a + b$ donc $a + b$ est aussi solution de l'équation $x^2 = x$. Mais $a + b$ mod $10 = 5 + 6 = 1$ (on est dans $\mathbb{Z}/10\mathbb{Z}$) et donc, toujours par le lemme de Hensel, $a + b = 1$.

Exercice : Montrer que $\mathbb{Z}_{10} \sim \mathbb{Z}_5 \times \mathbb{Z}_2$, en considérant l'application $x \mapsto (ax, bx)$, a et b étant nos deux nombres magiques.

La conséquence sur les entiers normaux, me direz-vous ? Eh bien nous venons de trouver deux nombres u et v de N chiffres tels que leurs N derniers chiffres sont les nombres de départ. Et nous savons faire cela pour tout N.

```
In [60]: u, v = a.n, b.n
         print(u)
         print(u ** 2)

         print(v)
         print(v ** 2)
```

```
395300731910816980293850989006216650958086381100055742342323089610900410661997739225625991821289
156262668649227598064331353110070152175866099862641585667341245283019813138017644302379296760923
604699268089183019706149010993783349041913618899944257657676910389099589338002260774374008178710
365661204827593637476629375097636850259693337662530100982695066061218991814022165851127313118345
```

Vous préférez 1000 chiffres ? Ce sera notre test ultime.

```
In [61]: N = 1000
```

```
In [62]: a = newton(lambda x:x ** 2 - x, lambda x: Padic(2, 10, N) * x - Padic(1, 10, N), 10, N,
         b = newton(lambda x:x ** 2 - x, lambda x: Padic(2, 10, N) * x - Padic(1, 10, N), 10, N,
         print('a =', a)
         print('b =', b)
```

```
a = ...12781254001336900860348890843640238757659368219796261819178335204927041993248752378258671
b = ...87218745998663099139651109156359761242340631780203738180821664795072958006751247621741328
```

Et le fait que $a+b=1$ et $ab=0$, que nous raconte-t-il sur leurs approximations u et v ? Regardez, vu les valeurs de u et v on n'aurait pas pu le deviner sans notre petite théorie.

```
In [63]: u = a.n
         v = b.n
         print('u + v =', u + v)
         print('uv    =', u * v)
```

```
u + v = 1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
uv    = 1114764946286999547095226375471941918863419914313922201345616185451472221921433806566591
```

1.4 4. Et ensuite ?

L'ensemble $\mathbb{Z}_p$ des entiers p-adiques est un anneau intègre, tout comme $\mathbb{Z}$. Un théorème sur les anneaux nous dit que tout anneau intègre $\mathbb{A}$ possède un **corps de fractions**, c'est à dire qu'il existe un corps $\mathbb{K}$ contenant $\mathbb{A}$ et tel que, dans un certain sens, $\mathbb{K}=\{\frac{x}{y}, x, y \in \mathbb{A}, y \neq 0\}$.

Le corps des fractions de $\mathbb{Z}$ est bien connu, il s'agit de $\mathbb{Q}$. Celui de $\mathbb{Z}_p$ est appelé le corps des nombres p-adiques, et est noté $\mathbb{Q}_p$. On peut montrer que

$$\mathbb{Q}_p=\{\frac{x}{p^n}, x \in \mathbb{Z}_p, n \in \mathbb{N}\}$$

Programmer une classe "nombre p-adique" n'est pas bien compliqué, il suffit de rajouter à la classe `Padic` un champ qui contient la puissance n du dénominateur de nos fractions. Les opérations sur $\mathbb{Q}_p$ sont également très faciles à écrire. Faites-le !

Malgré la notation, $\mathbb{Q}_p$ est un analogue de $\mathbb{R}$ (et pas de $\mathbb{Q}$). En fait, tout comme $\mathbb{R}$, $\mathbb{Q}_p$ contient le corps $\mathbb{Q}$ des nombres rationnels, $\mathbb{Q}$ est dense dans $\mathbb{Q}_p$, et $\mathbb{Q}_p$ est **complet**, en un sens que je ne préciserai pas plus ici (suites de Cauchy).

Et ensuite ? Peut-on faire dans $\mathbb{Q}_p$ de l'analyse ? Suites ? Séries ? Fonctions continues ? Dérivées ? Intégrales ? Nombres complexes p-adiques ? Oui, mais ceci est une autre histoire.