ChatGPT

Nombres p-adiques : théorie et implémentations en informatique

I. Propriétés et représentation récursive des nombres p-adiques

Définition et représentation récursive : Un **nombre p-adique** est défini comme une série de chiffres en base p s'étendant *à l'infini vers la gauche* (poids forts) plutôt que vers la droite comme pour les réels [1] [2]. Formellement, les nombres p-adiques $\mathbb{Q}_p$ sont l'achèvement de $\mathbb{Q}$ pour la distance induite par la *valuation p-adique*. Cela revient à considérer une suite d'entiers (a_n) telle que $a_{n+1} \equiv a_n \pmod{p^n}$: on détermine d'abord le reste modulo p , puis modulo p^2 , etc., affinant indéfiniment la précision (*logique de zoom local*) [3]. Intuitivement, connaître un nombre p-adique, c'est connaître son approximation modulo p^n pour tout n . Cette construction garantit une représentation *récursive* par congruences successives, mais implique des expansions infinies en général (impossibles à stocker entièrement) ; en pratique on manipule des développements **troncqués** à une précision finie arbitraire [4].

Distance p-adique et "zoom local" : La norme p-adique $|x|_p = p^{-\nu_p(x)}$ (où $\nu_p(x)$ est l'exposant de p dans x) définit une métrique non-archimédienne. Deux nombres sont *proches p-adiquement* s'ils coïncident sur de nombreux chiffres **de bas poids** (divisible par une grande puissance de p) [5]. Autrement dit, plus leur différence est divisible par p^k avec k grand, plus ils sont proches. Ceci contraste avec les réels, où la proximité se lit sur les premiers chiffres après la virgule. Par exemple, en base 10 réelle $0{,}333\ldots = 1/3$ possède une expansion décimale périodique vers la droite, tandis qu'un nombre rationnel aura en général une **périodicité p-adique vers la gauche** dans son développement p-adique. Une conséquence frappante est que des suites de chiffres « à gauche » constantes ou périodiques représentent des nombres simples : en 5-adique, l'expansion $\ldots444444$ _5 (chiffres 4 répétés à l'infini) représente -1 car $\ldots444444_5 + 1 = 0$ [6]. Plus généralement $\ldots999999_{10} = -1$ en 10-adique [7]. De même, $\ldots313132_5 \times 3 = 1$ dans $\mathbb{Q}_5$ [6], illustrant qu'une expansion p-adique périodique correspond à un nombre rationnel. Ainsi, là où les rationnels ont une période en base 10 à droite de la virgule (0.* périodique), ils ont une périodicité en base p *à gauche* en p-adique (par exemple $1/3$ est périodique en base non divisible par 3).

Stabilisation sur les entiers classiques : Les entiers relatifs $\mathbb{Z}$ sont naturellement inclus dans $\mathbb{Z}_p$ (les entiers p-adiques). Un entier non négatif possède une représentation p-adique *finie* (ses chiffres de gauche deviennent nuls au-delà d'un certain rang). En effet, un entier N s'écrit en base p avec un nombre fini de chiffres, ce qui correspond à une expansion p-adique stationnaire ($N = \ldots000N_k\ldots N_1N_0$ en base p). Pour les entiers négatifs, la représentation p-adique est infinie mais *ultimement constante* : par exemple -1 donne des chiffres $p-1$ répétés indéfiniment comme vu ci-dessus. Plus généralement, tout entier relatif apparaît comme un p-adique dont les coefficients finissent par se stabiliser [8]. Cette propriété fait que les nombres p-adiques peuvent **encoder efficacement les entiers** (y compris négatifs) en évitant toute ambiguïté de signe : il n'y a pas de « -0 » p-adique distinct, puisque $\ldots0000 = 0$. En somme, les nombres p-adiques fournissent un système de numération redondant où l'information « globale » (la valeur entière) se reflète dans la stabilité des chiffres à l'infini à gauche.

Dualité avec les nombres réels : Ostrowski a montré qu'à un scalaire multiplicatif près, il n'existe que deux familles de valeurs absolues sur $\mathbb{Q}$: la valeur usuelle (réelle) et les valeurs p-adiques pour chaque premier p [9]. Les réels $\mathbb{R}$ et les p-adiques $\mathbb{Q}_p$ sont donc deux complétions distinctes de $\mathbb{Q}$. D'un point de vue numérique, on peut les opposer : - **Expansion** : tout nombre réel possède une écriture décimale finie à gauche (partie entière) et infinie à droite (partie fractionnaire). À l'inverse, tout nombre p-adique s'écrit de façon unique sous la forme $\sum_{i=k}^{\infty} a_i p^i$ (avec $a_k \neq 0$), c'est-à-dire fini du côté des puissances négatives (droite) mais infinie vers les puissances positives [10]. - **Convergence** : une série infinie $\sum_{i=k}^{\infty} a_i p^i$ ne converge pas absolument dans $\mathbb{R}$ (si k est négatif, les termes ne tendent pas vers 0), mais elle converge dans $\mathbb{Q}_p$ car p^i tend vers 0 p-adiquement quand i décroît (i.e. les sommes partielles diffèrent d'un multiple de p^n de plus en plus grand) [11]. Ainsi, $0{,}999\ldots = 1$ en réel, alors que $\ldots 999$ (infinité de 9 à gauche) a un sens uniquement en 10-adique et y vaut -1. - **Distances** : la valeur absolue réelle est *archimédienne* (des petites différences s'additionnent), tandis que la p-adique est *ultramétrique* (la « petitesse » d'une somme est dominée par la plus petite des deux en valeur) ce qui donne lieu à des propriétés géométriques très différentes. Par exemple, en p-adique toutes les boules ouvertes sont aussi fermées (et inversement), et deux boules soit sont disjointes, soit l'une contient l'autre (propriétés imbriquées typiques des ultramétriques).

En résumé, les nombres p-adiques fournissent un autre « univers » numérique où la notion de proximité est renversée par rapport aux réels. Cette dualité se reflète dans leurs **expansions** (réciproques l'une de l'autre) et dans les applications : l'arithmétique p-adique excelle pour résoudre des problèmes locaux (congruences, relevés modulo p^n), là où l'analyse réelle traite les phénomènes continus et infinitésimaux.

II. État de l'art des travaux et applications computationnelles des p-adiques

Implémentations en bas niveau (langage machine, C/LLVM) : Représenter un nombre p-adique en mémoire nécessite de **troncquer** son expansion infinie à une certaine précision p^N. La plupart des bibliothèques de calcul arithmétique implémentent $\mathbb{Q}_p$ via une structure contenant un entier (représentant les coefficients significatifs modulaires) et un exposant de valuation. Par exemple, la bibliothèque FLINT (en C) utilise un type `padic_t` stockant $x = p^v \cdot u$ où u est un entier (non divisible par p en forme canonique) et v la valuation, avec une précision maximale N fixée [12]. De même, la librairie Haskell `padic` adopte une représentation interne efficace : un nombre p-adique y est codé par un seul entier N représentant la valeur modulo p^k (pour un certain k couvrant la précision de travail), plutôt que par une liste de chiffres [13]. Les opérations arithmétiques se font alors via l'arithmétique modulaire rapide sur cet entier *« composé »* (quitte à ajuster k dynamiquement). Ce choix garantit un **stockage constant** par rapport au nombre de chiffres manipulés : la dimension mémoire ne dépend que de la précision fixée, et non de la valeur du p-adique. Par contraste, une implémentation naïve en tableau de chiffres serait moins efficace. La bibliothèque FLINT/Nemo en Julia suit cette approche : elle permet de construire un corps $\mathbb{Q}_p$ pour n'importe quel premier p avec une précision choisie [14] [15]. Les éléments p-adiques sont manipulés comme des entités de type FieldElem, avec des opérations sur leurs représentants modulaires et gestion de la « valeur » v (exposant) et de la précision disponible [12] [16].

Concrètement, ces bibliothèques offrent des fonctions pour extraire ou modifier la valuation $v = \nu_p(x)$, la mantisse u (le résidu significatif), et réaliser des « liftings » vers $\mathbb{Z}$ ou $\mathbb{Q}$ si nécessaire [17] [18]. Par exemple, dans Nemo/FLINT on peut obtenir l'entier classique correspondant à un p-adique si celui-ci est suffisamment précis pour être déterminé sans ambiguïté [18]. On peut aussi spécifier des constantes p-adiques via la notation « $O(p^n)$ » pour indiquer qu'on travaille modulo p^n [19] [20]. Ainsi,

sur le plan *langage machine*, les nombres p-adiques sont simulés par des entiers de grande taille (type *bigint*) couplés à des métadonnées (valuation, précision) – ce qui exploite l'**arithmétique binaire** optimisée des grands entiers tout en conservant la logique p-adique.

Kernels symboliques et compilos mathématiques : Les nombres p-adiques ont également été intégrés dans des environnements de calcul formel et de preuve. Dans l'assistant de preuve Coq (bibliothèque *UniMath*), Pelayo, Voevodsky et Warren ont réalisé la **première formalisation** complète des $\mathbb{Q}_p$ [21]. On dispose donc en Coq de la construction des entiers p-adiques $\mathbb{Z}_p$ et rationnels p-adiques, avec leurs propriétés algébriques, ce qui a servi de base à des preuves formelles en théorie des nombres. Plus récemment, en 2022, une formalisation des p-adiques a été effectuée dans Isabelle/HOL [21]. L'assistant Lean intègre lui aussi $\mathbb{Q}_p$ dans sa bibliothèque mathlib : une étape marquante a été la **preuve formelle du lemme de Hensel** (en Lean, 2019) qui démontre de manière constructive comment trouver les racines d'un polynôme dans $\mathbb{Z}_p$ [22] [23]. Ces développements montrent la maturité des outils formels pour manipuler des objets aussi sophistiqués que les nombres p-adiques, alliant aspects analytiques (complétude, convergence) et algébriques (valuation, unité). Par ailleurs, des *compilateurs mathématiques* ou systèmes comme **SageMath**, **PARI/GP** ou **Magma** offrent depuis longtemps un support pour les calculs p-adiques, principalement orientés vers la théorie des nombres (calcul de développements de séries L p-adiques, résolutions d'équations diophantiennes locales, etc.). Magma, en particulier, a récemment bénéficié de packages **ExactpAdics** permettant de représenter *exactement* des nombres p-adiques via des calculs paresseux à précision arbitraire [24]. L'objectif est d'obtenir des résultats **certifiables** (par exemple décider qu'un résultat est exact sans erreur d'arrondi) en ne tronquant la précision qu'au moment voulu : ces nombres p-adiques « lazys » conservent une précision potentiellement infinie en interne [25]. Cela améliore la convivialité (l'utilisateur n'a plus à fixer manuellement la précision) et évite des erreurs, au prix d'un surcoût algorithmique maîtrisé.

Calcul p-adique "en flux" et méthodes récentes : Deux grandes approches se dégagent pour les algorithmes p-adiques : l'approche **zélée (zealous)** et l'approche **paresseuse (lazy)**. L'approche zélée, la plus répandue historiquement, consiste à **fixer à l'avance** une précision de calcul (nombre de chiffres p-adiques) et à effectuer toutes les opérations modulo p^N. C'est le cas par exemple de FLINT/Nemo ou PARI/GP, qui permettent de choisir N et garantissent que toutes les opérations sont exactes mod p^N. Cette stratégie, combinée à des techniques de *Hensel lifting* (itérations à la Newton qui doublent la précision à chaque étape), est **très efficace** asymptotiquement [26]. En revanche, elle nécessite de savoir a priori quelle précision finale est requise.

L'approche lazy prend le contre-pied : on représente un p-adique comme un **générateur de chiffres** (un flux infini), de sorte que les calculs produisent les chiffres au fur et à mesure qu'on en a besoin [27]. Par exemple, une somme p-adique paresseuse attendra qu'on lui demande le résultat mod p^n pour effectuer les calculs nécessaires à cette précision. Ceci est très utile pour résoudre des équations implicites : on peut programmer des opérations sur les p-adiques sans déterminer tout de suite jusqu'où pousser le calcul, ce qui *retarde* la charge de travail jusqu'à ce que la précision soit effectivement requise. Cependant, les algorithmes lazy naïfs souffrent souvent d'une complexité asymptotique moins bonne (du fait de recalculs redondants à chaque nouvel échelon de précision) [28].

C'est pourquoi des recherches récentes ont introduit des méthodes dites **relaxed** (détendues), combinant le meilleur des deux mondes [29]. Proposée initialement par J. van der Hoeven dans le contexte des séries formelles, l'algorithmique *relaxed* applique aux p-adiques des techniques qui permettent d'obtenir la prochaine tranche de chiffres *avec une complexité linéaire* en la taille de l'entrée, tout en conservant l'efficacité des méthodes de Newton pour l'asymptotique. En 2011, Berthomieu, van der Hoeven et Lecerf ont montré comment adapter cette approche aux nombres p-adiques [30] et l'ont implantée dans la bibliothèque C++ *algebramix* de **Mathemagix**. Ils obtiennent des **accélérations significatives** pour la résolution d'équations fonctionnelles p-adiques comparé aux itérations de

Newton classiques [31]. L'idée clé est de permettre des *calculs par blocs* de chiffres et de réutiliser au maximum les informations déjà calculées lorsqu'on accroît la précision, évitant ainsi le recalcul total. Ces avancées relèvent de l'**arithmétique en flux** (streamed arithmetic) où les nombres sont traités comme des flux infinis de données. Notons que l'aspect *parallélisable* de ces méthodes est limité par des dépendances intrinsèques (pour calculer les chiffres de poids très élevé il faut souvent connaître les plus faibles, ou inverser des éléments mod p^n ce qui se fait itérativement). Néanmoins, certaines sous-opérations p-adiques (par exemple le calcul de multiples séries de puissances, ou le traitement de plusieurs premiers en parallèle via le lemme chinois) peuvent bénéficier d'une exécution distribuée. En pratique, la littérature ne met pas en avant d'applications *massivement parallèles* spécifiques aux p-adiques ; l'effort s'est surtout porté sur l'**optimisation algorithmique** et l'intégration dans des systèmes formels.

Autres applications numériques : En calcul scientifique direct, les nombres p-adiques sont peu utilisés (ils servent surtout en théorie des nombres et algèbre). Cependant, ils trouvent place dans des algorithmes de résolution *exacte* ou de vérification. Par exemple, le **lemme de Hensel** – souvent vu comme l'analogue de la méthode de Newton en contexte discret – est implémenté dans les systèmes de calcul formel pour factoriser des polynômes ou trouver des racines entières d'équations en relevant des solutions modulo p en solutions modulo p^n . Cette procédure itérative exploite pleinement l'arithmétique p-adique et est un pilier de nombreuses routines de factorisation (dans PARI, SageMath, etc.). On mentionnera aussi les applications en cryptographie et codage correcteur, où la notion de distance ultramétrique p-adique peut modéliser certaines configurations (par exemple, des travaux ont exploré des réseaux de neurones ou des modèles de physico-chimie en géométrie p-adique, bien que cela sorte du cadre purement arithmétique).

Enfin, dans le domaine du calcul distribué, les p-adiques n'interviennent pas directement comme base de calcul, mais leur théorie inspire parfois des méthodes numériques. Par exemple, la résolution distribuée d'un système d'équations modulaires utilise le lemme chinois, qui est relié à la notion d'adèles (combinaison de plusieurs complétions dont les p-adiques). De même, pour des *FFT* en arithmétique modulaire, il faut tenir compte de la valuation p-adique des coefficients (évitant des divisions par p non contrôlées) [32]. Ces considérations montrent que, même si l'on ne « calcule » pas explicitement avec des nombres p-adiques dans un cluster HPC, la compréhension fine de leur arithmétique influence des algorithmes de calcul exact et modulaire.

III. Conclusion : entre vision mathématique et implémentation logicielle

En conclusion, les nombres p-adiques illustrent l'alliance féconde entre théorie des nombres et informatique. **D'un point de vue mathématique**, ils offrent une vision *locale* des nombres, où « être petit » signifie « être divisible par une grande puissance de p ». Cette idée, purement abstraite au départ (introduite par Hensel à la fin du XIX^e siècle), a profondément enrichi la théorie des nombres et la géométrie algébrique. **Sur le plan de l'implémentation**, le défi est de dompter ces objets à infinité de chiffres dans un contexte fini. Les solutions développées – stockage modulo p^N , algorithmes de lifting, calcul paresseux ou détendu – montrent que l'informatique sait s'adapter aux concepts mathématiques en trouvant des représentations appropriées. Le *fil rouge* est de préserver l'exactitude arithmétique tout en contrôlant les ressources (temps de calcul, mémoire). À cet égard, les nombres p-adiques sont un cas d'école : on a appris à les représenter de façon compacte et efficace (grâce à l'**arithmétique modulaire** et aux bigints) et à effectuer des calculs complexes (inversion, exponentiation, résolution d'équations) en garantissant une précision donnée.

Le bilan des travaux récents indique une **maturité croissante** dans la manipulation des p-adiques par les logiciels. Des bibliothèques bas niveau jusqu'aux assistants de preuve formels, les p-adiques sont désormais outillés pour être utilisés de manière *pratique et fiable*. Cela ouvre la voie à des applications plus intégrées – par exemple, des solveurs hybrides qui combineraient calcul p-adique et calcul réel pour aborder des problèmes globaux/localisés, ou des vérifications formelles de propriétés p-adiques dans du code critique. En somme, la *vision mathématique* (où les p-adiques sont un univers parallèle fascinant) et l'*implémentation logicielle* (qui les rend concrets et utilisables) convergent pour faire des nombres p-adiques à la fois un objet d'étude théorique profond et un outil computationnel de plus en plus courant en algorithmique du nombre et en vérification numérique.

Références : Les affirmations et exemples ci-dessus s'appuient sur la littérature existante, notamment les notes et ouvrages sur la définition des p-adiques [1] [5], des exemples illustratifs de leur arithmétique [6], la documentation de bibliothèques comme FLINT/Nemo (Julia) [14] [12] ou la librairie Haskell Padic [13], ainsi que des travaux de recherche sur les algorithmes p-adiques paresseux et relaxés [27] [31]. Des contributions récentes en Coq/Isabelle [21] et Lean [22] ont été mentionnées pour souligner l'aspect formel et fiable des développements autour de $\mathbb{Q}_p$. Ces références offrent un panorama plus détaillé pour le lecteur désirant approfondir chaque point évoqué.

---

[1] [5] [7] [8] p-adic.md
https://github.com/Bricktech2000/Notes/blob/30f6c86713f87fe2cd671b21f91e0f9b2905333c/notes/p-adic.md

[2] [6] [9] [10] [11] [22] [23] A Formal Proof of Hensel's Lemma over the p-adic Integers
https://robertylewis.com/padics/padics.pdf

[3] [4] [26] [27] [28] [29] [30] [31] Relaxed algorithms for p-adic numbers
https://www.texmacs.org/joris/padic/padic.html

[12] [16] padic.h – p-adic numbers — FLINT 3.4.0-dev documentation
https://flintlib.org/doc/padic.html

[13] padic: Fast, type-safe p-adic arithmetic
https://hackage.haskell.org/package/padic

[14] [15] [17] [18] [19] [20] Padics · Nemo.jl
https://nemocas.github.io/Nemo.jl/latest/padic/

[21] arxiv.org
https://arxiv.org/pdf/2306.17234

[24] [25] [1805.09794] ExactpAdics: An exact representation of p-adic numbers
https://arxiv.org/abs/1805.09794

[32] arxiv.org
https://arxiv.org/pdf/1701.06794