Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi de fonctions mathématiques

Damien STEHLÉ

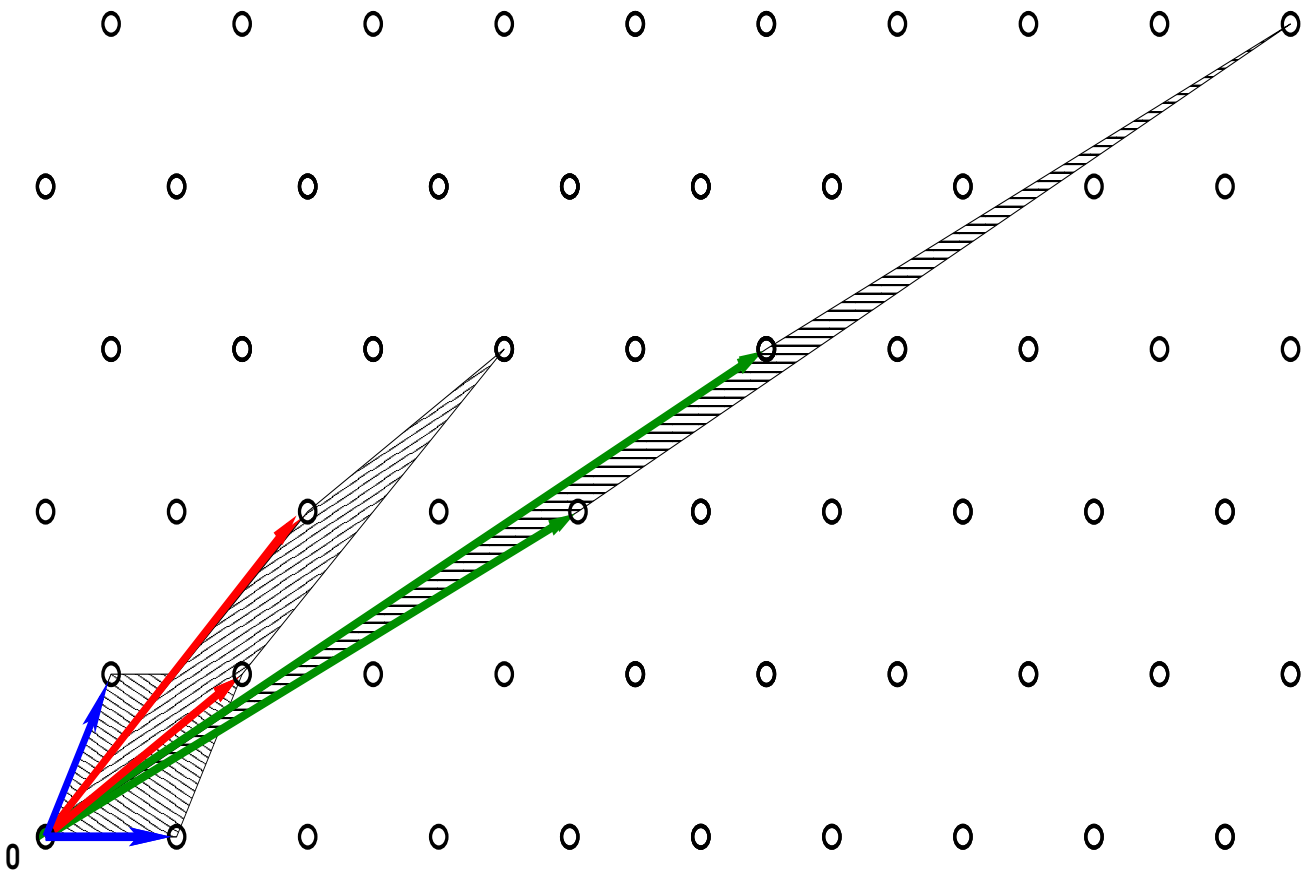
2 décembre 2005

`http://www.loria.fr/~stehle/these.html`

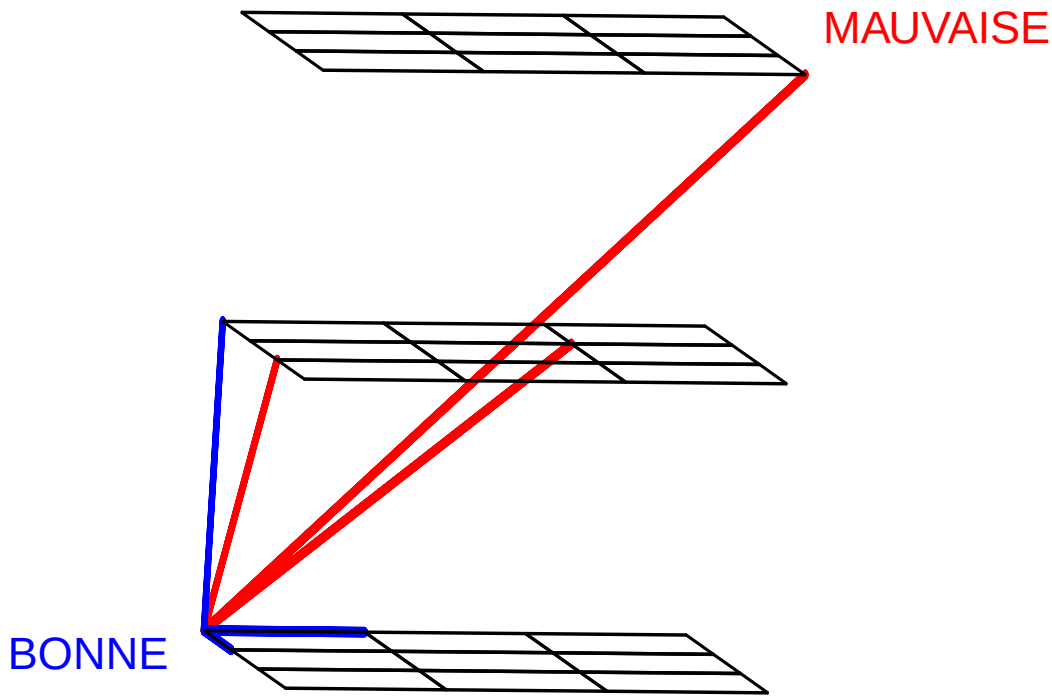
Sommaire.

1) Algorithmique de la réduction de réseaux.

Trois bases d'un réseau de dimension 2.



Deux bases d'un réseau de dimension 3.



Les réseaux euclidiens.

– Réseau : sous-groupe discret de $\mathbb{R}^n$.

$$L[\mathbf{b}_i] = \left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i, x_i \in \mathbb{Z} \right\}.$$

– Infinité de bases, liées entre elles par des relations unimodulaires.

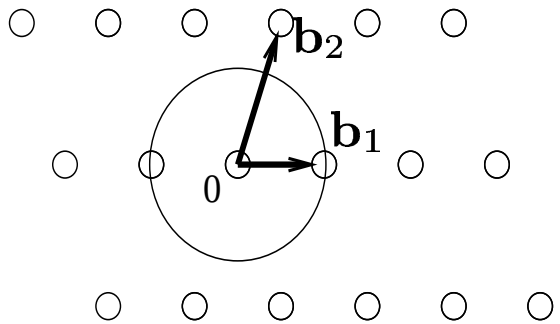
– Leur cardinalité commune d est la dimension.

– $L[\mathbf{b}_i]$ est représenté par la matrice $d \times n$ d'une base quelconque.

Problèmes algorithmiques.

Donnée : une base d'un réseau à coefficients entiers.

– Trouver le vecteur le plus court (SVP).



– Trouver un vecteur assez court.

– Trouver une base avec des vecteurs assez courts et orthogonaux.

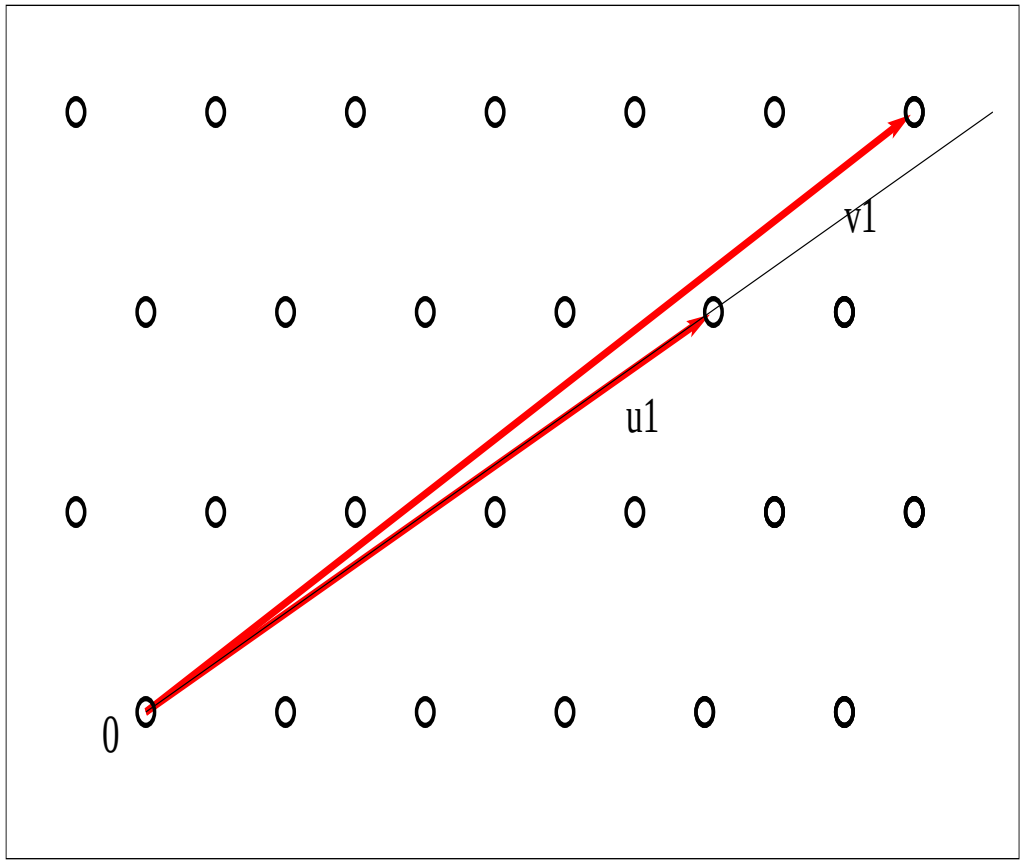
Les réseaux euclidiens, à quoi ça sert ?

- Cryptanalyse : sac-à-dos, RSA, schémas de signature (sous certaines conditions).
- Cryptosystèmes : NTRU, GGH, Ajtai-Dwork.
- Théorie algorithmique des nombres.
- Théorie de la complexité.

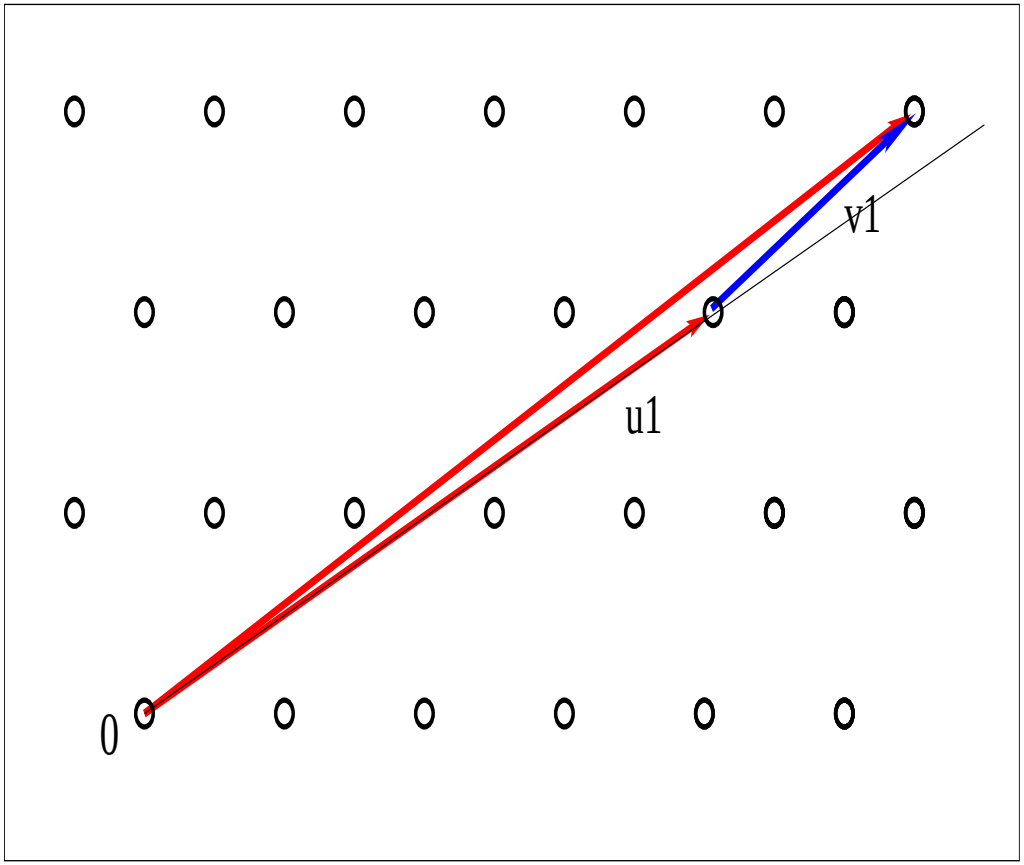
Contributions.

1. Un algorithme de calcul de pgcd de complexité quasi-linéaire : pgcd-BG-rapide.
2. Un algorithme de réduction optimale pour $d \leqslant 4$, de complexité quadratique : glouton.
3. Un algorithme de réduction en dimension quelconque : L^2.
4. Des observations expérimentales sur l'algorithme L^2.

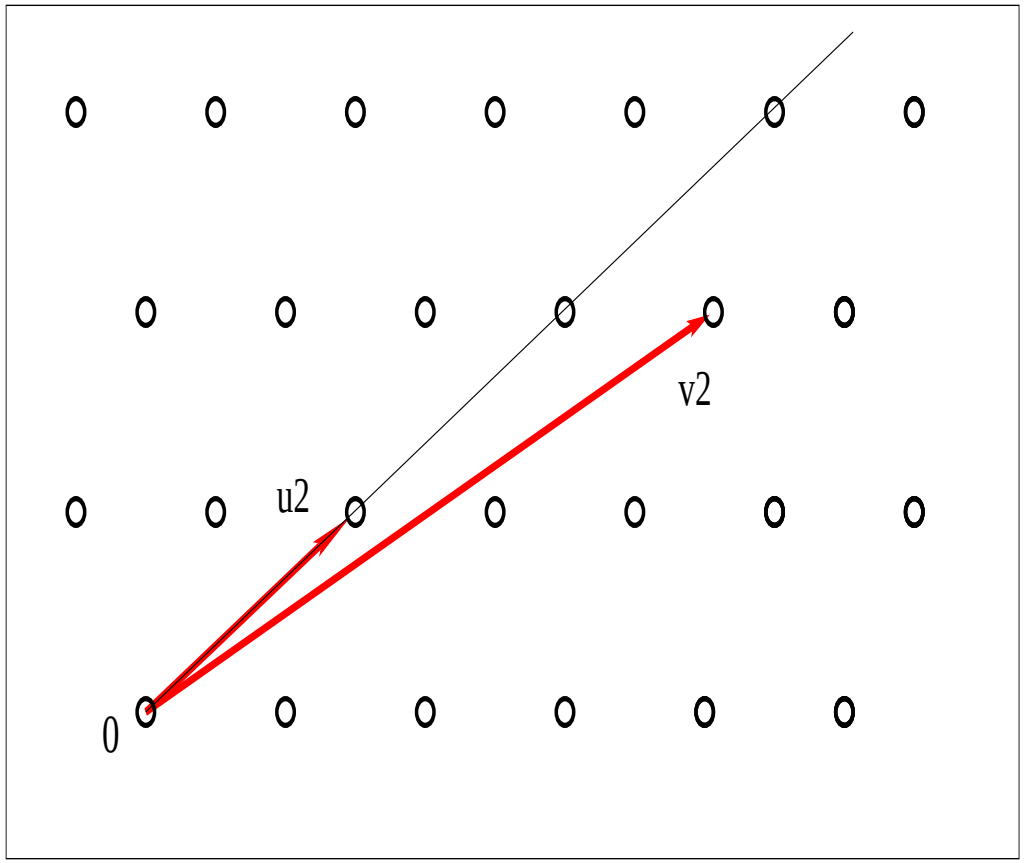
Dimension 2 : algorithme de Lagrange.



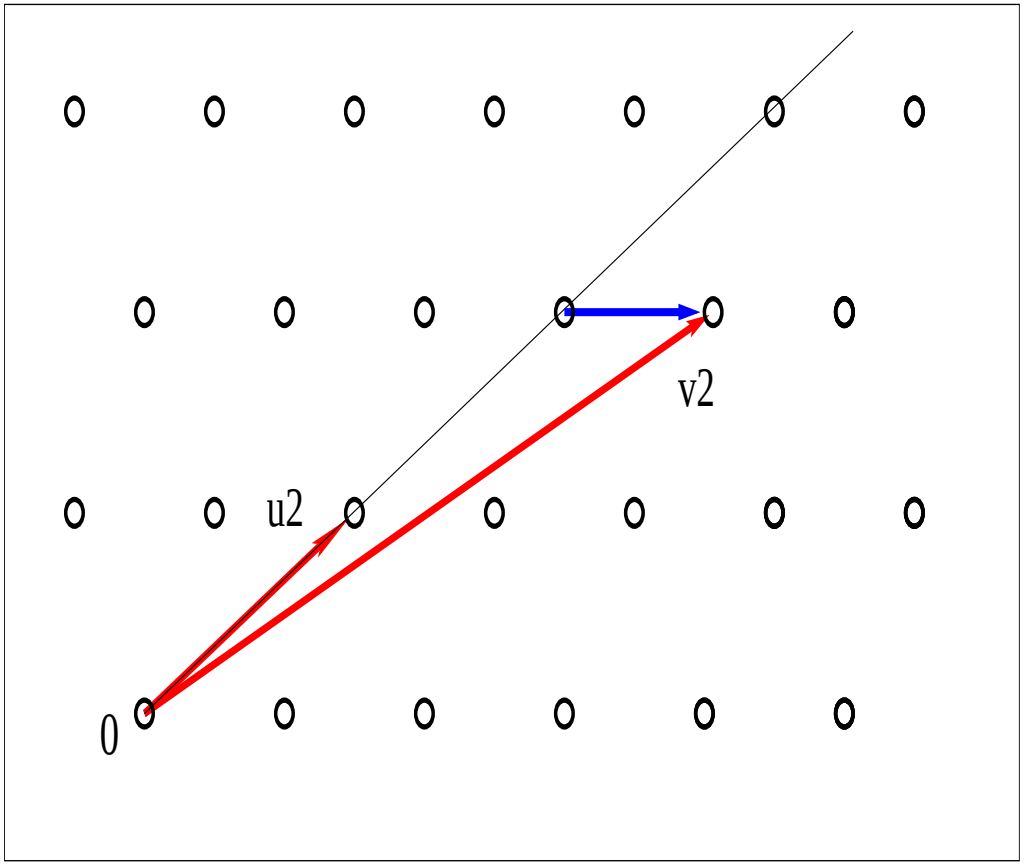
Dimension 2 : algorithme de Lagrange.



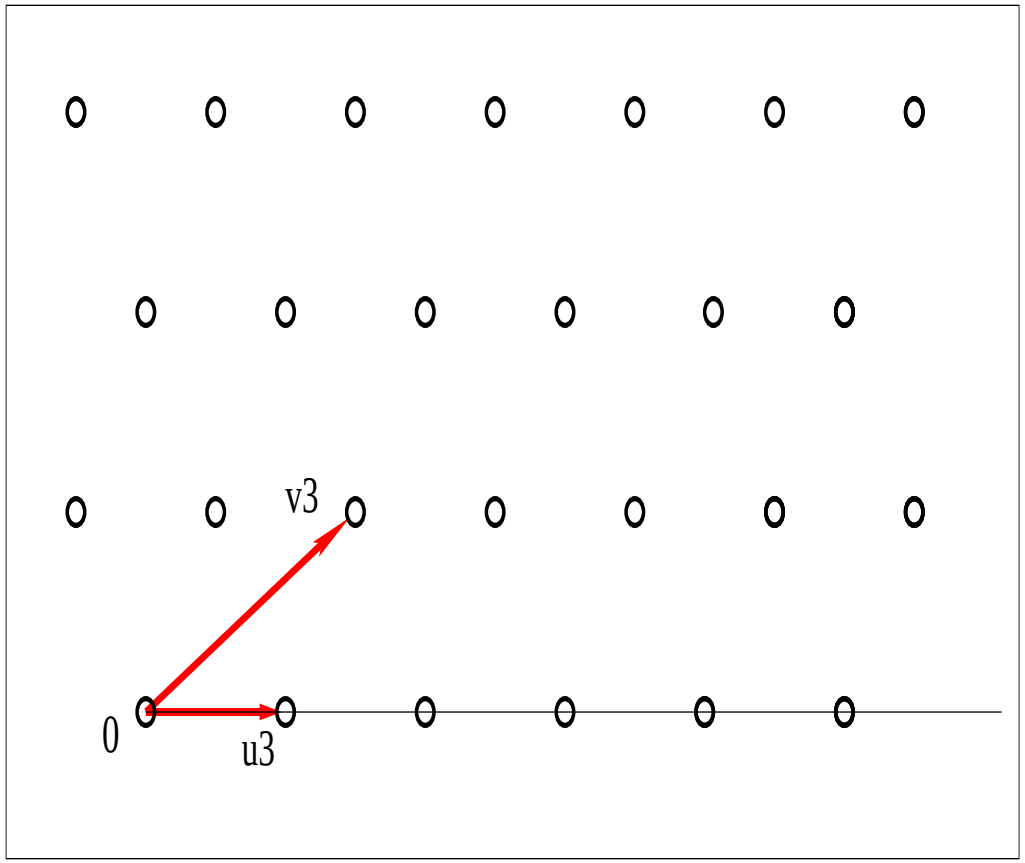
Dimension 2 : algorithme de Lagrange.



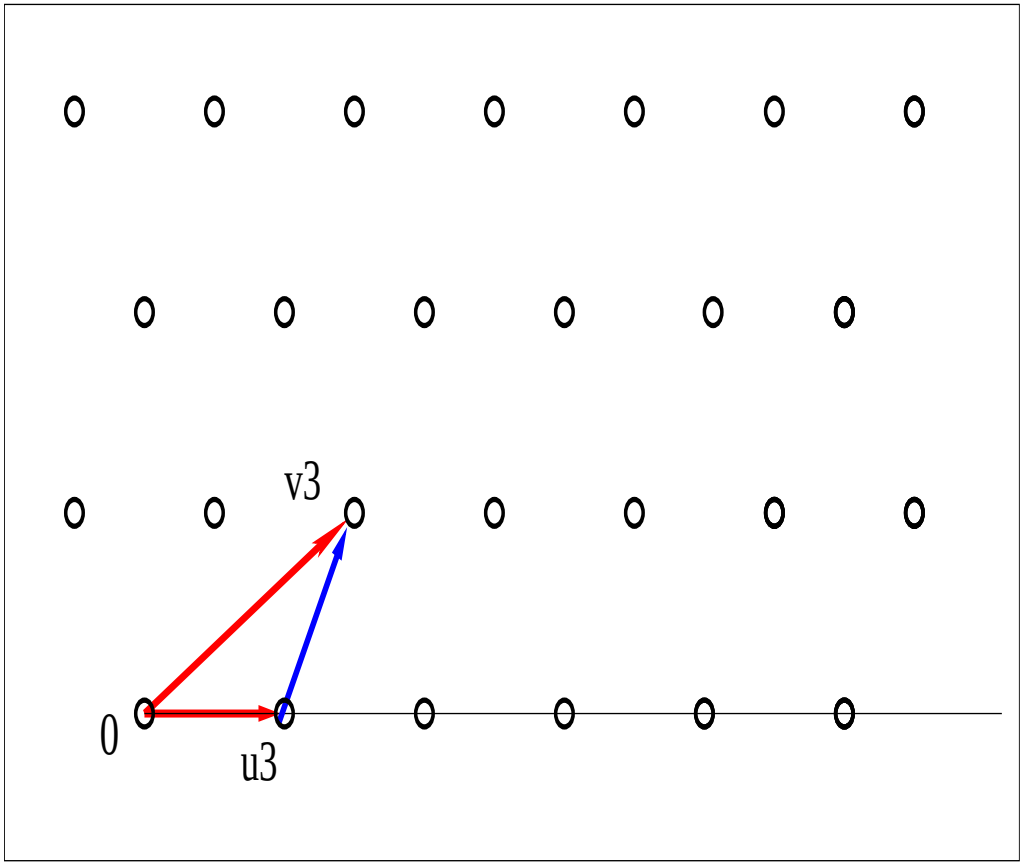
Dimension 2 : algorithme de Lagrange.



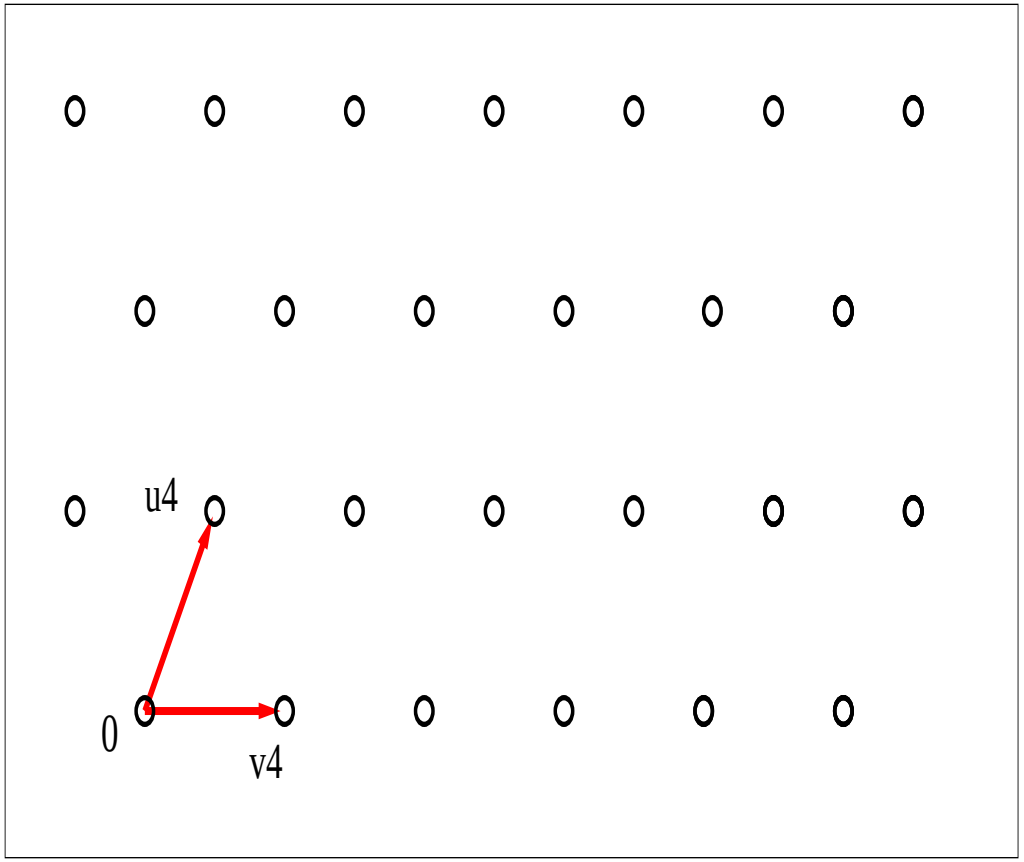
Dimension 2 : algorithme de Lagrange.



Dimension 2 : algorithme de Lagrange.



Dimension 2 : algorithme de Lagrange.



Réduction en dimensions 2 à 4.

– Dimension 2 : algorithme de Lagrange (Gauss). Les vecteurs renvoyés sont les deux plus courts possibles.
Complexité quadratique.

– Dimension 3 : algorithmes de Vallée et Semaev.

– Algorithme glouton : généralisation de l'algorithme de Lagrange pour $d \leqslant 4$. Il renvoie les d vecteurs les plus courts possibles.
Complexité quadratique.

L'algorithme glouton.

Entrée : $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Sortie : Une base « réduite ».

1. Ordonner les vecteurs par longueurs croissantes.
2. Réduire $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$.
3. Raccourcir autant que possible $\mathbf{b}_d$ à l'aide de $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$:

$$\mathbf{b}_d \leftarrow \mathbf{b}_d - (x_1\mathbf{b}_1 + \ldots + x_{d-1}\mathbf{b}_{d-1}).$$

4. Si $\mathbf{b}_d$ est modifié, retourner en 1.

Analyse de l'algorithme glouton.

– Première partie : le nombre d'itérations de boucle est linéaire.

1. Approche globale : quand le produit des longueurs ne décroît plus géométriquement, il reste $O(1)$ itérations de boucle.
2. Approche locale : toutes les d itérations de boucle le produit des longueurs décroît géométriquement.

– Seconde partie : analyse amortie pour sommer finement les coûts des étapes successives.

État de l'art en dimension d.

– L'algorithme LLL (1982) renvoie un vecteur de norme $\leqslant 2^{\frac{d}{4}}(\det L)^{\frac{1}{d}}$. Complexité $O(d^6 \log^3 B)$.

– Très cher en pratique : cas de la factorisation d'un module RSA $N = p \cdot q$ quand la moitié des chiffres de p sont connus :

$$\log B \approx 1000,\ d \approx 64,\ \text{« } d^6 \log^3 B \approx 2^{66} \text{ ».}$$

– Arithmétique rationnelle dans l'orthogonalisation sous-jacente...

État de l'art en dimension d.

– Schnorr'88 : précision des calculs $\approx 12d + 7\log B$.
  Complexité $O(d^4(d+\log B)^2 \log B)$.
– Algorithme difficile à décrire, modèle « arithmétique flottante » non précisé, précision requise élevée. Pas implanté.

– Koy-Schnorr'01 : Givens/Householder plutôt que Gram-Schmidt.
– Schnorr'04 (non publié) : précision $\approx 5d + 2\log B$.

L'algorithme L^2.

Un LLL flottant prouvé dans un modèle précis.

	LLL'82	Schnorr'88	L^2'05
Précision	$O(d \log B)$	$> 12d + 7 \log_2 B$	$d \log_2 3 \approx 1.58d$
Complexité	$O(d^6 \log^3 B)$	$O(d^4(d + \log B)^2 \log B)$	$O(d^5(d + \log B) \log B)$

L^2 est simple à décrire. Bonne version de LLL ?

Les trois idées principales.

– Utilisation exclusive de la matrice de Gram
(pour éviter les pertes de précision liées aux produits scalaires).

– Version paresseuse de l'étape de « proprification »
(pour éviter d'utiliser plus de bits de précision qu'il n'en faut).

– Généralisation en dimension d des analyses amorties des algorithmes d'Euclide, de Lagrange et glouton.

L'algorithme LLL en pratique.

– Bibliothèques ayant un LLL efficace : NTL, Magma, Pari GP, Lidia.

– fplll-1.2 codé en C (GNU MP, MPFR), sous GPL.

	d	$\log B$	NTL	variante rapide	variante prouvée
entrées aléatoires	750	1000	$243s$	$15s$	$2287s$
sac-à-dos	100	20000	$2031s$	$368s$	$1599s$

L'arithmétique flottante dans LLL.

Un réseau de dimension 55 fait boucler le LLL_FP de NTL.

$$\begin{bmatrix} 28915\ldots036 & 0 & 0 & \ldots & 0 & 0 \\ 14457\ldots940 & 25167\ldots693 & 0 & \ldots & 0 & 0 \\ -14457\ldots783 & 12583\ldots053 & 21905\ldots751 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 14457\ldots631 & -12583\ldots224 & 10952\ldots511 & \ldots & 18\ldots616 & 0 \\ 15264\ldots310 & -35953\ldots625 & 19879\ldots178 & \ldots & -37590\ldots437 & 83\ldots884 \end{bmatrix}.$$

Analyse heuristique de la précision requise « en moyenne » :

53 bits devraient suffire jusqu'en dimension 180.

L'arithmétique flottante dans LLL.

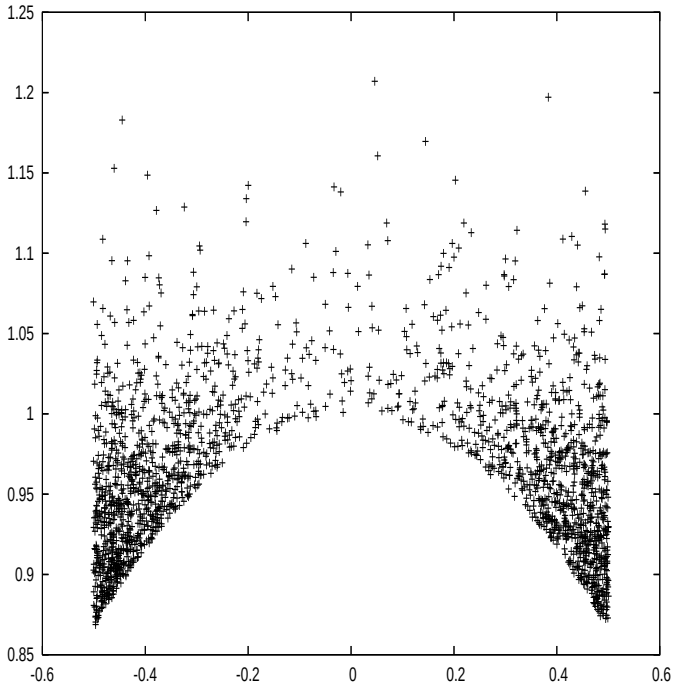
Faut-il remplacer Gram-Schmidt par Givens/Householder ?

Un réseau de dimension 3 fait boucler le G_LLL_FP de NTL.

$$\begin{bmatrix} 1 & 1 & 0 \\ 2^{54}+1 & -2^{54}+1 & 0 \\ 2^{54} & 2^{54}-1 & 1 \end{bmatrix}.$$

Autres remarques sur le comportement pratique de LLL.

- Temps d'exécution asymptotiquement conforme à la borne dans le cas le pire, à une constante près.
- Qualité de la base renvoyée conforme à la borne dans le cas le pire, à une constante près dans l'exposant.



2) Arrondi des fonctions mathématiques.

Rappels sur l'arithmétique flottante.

– $x = \pm m_x \cdot 2^{e_x}$, avec $m_x \in [1/2, 1[$ sur n bits.

– La valeur renvoyée pour « a op b » doit être la plus proche possible de la valeur réelle exacte, pour op $\in \{+, -, \times, \div\}$.

– Double précision : 53 bits de mantisse (IEEE-754).

– Double précision étendue : 64 bits (x87).

– Quadruple précision : 113 bits.

– Précision arbitraire : MPFR, classe RR de NTL, ...

Arrondi des fonctions.

– Une fonction $f : [1/2, 1] \to [a, b]$, transcendante, $\mathcal{C}^\infty$, avec des dérivées bornées : $e^x, 2^x, \sin x, \cos x, \log x, \tan^{-1} x, \ldots$

– But : implanter f, en précision n.

$$\begin{aligned} \exp(0.5000467344) &= 1.6487983244999990\ldots \\ &\to 1.648798324 \\ \exp(0.5274542578) &= 1.6946127655000000003\ldots \\ &\to 1.694612766 \end{aligned}$$

Pires cas pour l'arrondi.

$$\begin{aligned}
\sin(0.1001011111111_2) &= 0.\underbrace{1000111110011001}_{15}\,1\,\underbrace{0000000000000}_{13}\,100\ldots \\
&\to 0.1000111110011010 \\
2^{\frac{15741665614440311501}{2^{64}}} &= \underbrace{1.110\ldots110}_{64}\,1\,\underbrace{00\ldots0}_{63}\,111\ldots \\
&\to 1.110\ldots111
\end{aligned}$$

Pour un exposant donné, on s'attend à trouver les solutions extrêmes $x \in [1/2,1]_n = \left\{\frac{1}{2}, \frac{2^{n-1}+1}{2^n}, \ldots, \frac{2^n-1}{2^n}, 1\right\}$ pour :

$$|(2^n f(x) - 1/2) \mod 1| \lesssim 2^{-n}.$$

Travaux antérieurs.

- Pires cas : Lefèvre, complexité heuristique $2^{\frac{2}{3}n+\varepsilon}$.
- Stehlé-Lefèvre-Zimmermann, complexité heuristique $2^{\frac{4}{7}n+\varepsilon}$.
- Limite pratique : $n = 64$.

- Générateur $1/N$ de Blum-Blum-Shub :
  avec $2\log N + \varepsilon$ bits consécutifs de $1/N$, on peut retrouver N.
- Nesterenko-Waldschmidt : il y a au plus $O(n^2)$ zéros consécutifs dans la suite des bits de $\exp(x)$, pour $x \in [1/2, 1]_n$.

Contributions.

Méthode générale pour trouver les $x \in [1/2, 1]_n$ tels que :

$$\left|(2^n f(x) - c) \bmod 1\right| \leqslant 2^{-m}.$$

1. Pour $m = n$ (pires cas) : complexité heuristique $2^{\frac{n}{2}+\varepsilon}$.
2. Pour $m = n^2$: complexité heuristique polynomiale en n.
3. Variations pour construire les tables de Gal et casser un générateur aléatoire dû à Littlewood.

Inversion d'une fonction quand les premiers bits sont perdus.

On veut retrouver $x \in [1/2, 1]_{64}$ tel que :

$$\exp x = \times\times.\underbrace{\times\times\ldots\times}_{64}0111111111101100101100011010101001010110000110010110101110010000001111010000100000001111010011000111001000000100000111101000000010101010001101010101110100010100111100100001110000101101011000111100111011001000010011111010001101110011110110110011000010101110100111101110110000010000111111101000001110101011000011000011010001010001000011010110100000011011011111000011001110011100110101001\times\times\times\ldots$$

$$x = \frac{17832265507654526358}{2^{64}}$$

Méthode de Coppersmith (1/2).

On cherche les petites racines x_0 de :

$$P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{d-1} x^{d-1} + x^d \ [N].$$

1. On construit des polynômes dont x_0 est racine modulo N^α :
$$N^\alpha, N^{\alpha-1}P, N^{\alpha-1}xP, N^{\alpha-1}x^2P, N^{\alpha-2}P^2, \ldots, P^\alpha.$$
2. On construit un réseau associé à cette famille : les monômes « indicent » les colonnes, les polynômes les lignes.
3. On LLL-réduit ce réseau.
4. On obtient $G_1(x)$ tel que pour x petit, $|G_1(x)| < N^\alpha$.
5. On calcule les racines de G_1 sur $\mathbb{Z}$ pour trouver x_0.

Méthode de Coppersmith (2/2).

– Cas univarié : on peut trouver en temps polynomial déterministe toutes les racines x_0 de P telles que $|x_0| \leqslant N^{1/d}$.

– Cas multivarié :

- Il faut trouver plusieurs vecteurs courts dans le réseau.
- Les polynômes trouvés peuvent avoir des facteurs communs...

– L'analyse est heuristique dans le cas multivarié.

Déroulement général de l'algorithme d'inversion.

Entrée : n, c, m.

Sortie : Les $x_0 \in [1/2, 1]_n$ avec $\left| (2^n f(x_0) - c) \mod 1 \right| \leqslant 2^{-m}$.

Diviser $[1/2, 1]_n$ en $\frac{2^n}{2T}$ intervalles de longueur $\frac{T}{2^n}$. Pour chacun :

1. Approcher f par un polynôme P, avec erreur $\lesssim 2^{-m}$.
2. Chercher les racines de $Q(x, y) = 2^n P(x) - c + y$ modulo 1, avec $|x| \leqslant T/2^n$ et $|y| \lesssim 2^{-m}$.
   Famille de polynômes : $\{x^i Q^j(x, y), i + j \leqslant \alpha\}$
3. Si échec, diviser l'intervalle en deux et recommencer.

Dans l'analyse : deux bornes importantes.

– Borne d'approximation polynomiale :

$$2^n \left(T/2^n\right)^{d+1} \lesssim 2^{-m}.$$

– Borne de Coppersmith :

LLL doit renvoyer au moins deux vecteurs suffisamment courts.

– Pour estimer cela, on utilise « $\|\mathbf{g_1}\| \leqslant 2^{\dim} (\det)^{1/\dim}$ ».

– Le réseau est donné par une matrice rectangulaire...

Résultats expérimentaux.

– Implantation (GNU MP, MPFR) disponible sous GPL : wclr-1.1.

Pour $m = n$, complexité $2^{\frac{n}{2}+\varepsilon}$

Pour $m = n^2$, complexité $O(n^{20})$

} Encore trop cher.

– Avec des fonctions algébriques : la méthode échoue très souvent.

– Pas d'échec observé pour $e^x, 2^x, \sin x, \log x$.

Résultats expérimentaux.

Sur Opteron 2.4GhZ, pour $n = 64$, avec wclr-1.1.

	d	α	m	dims des bases	temps de calcul
$\sim$ Lefèvre	1	1	64	3×3	≈ 19.2 ans
SLZ'04	2	2	64	6×9	≈ 2.7 ans
ici	3	2	64	6×12	≈ 2.3 ans
ici	16	5	400	21×261	2.1 jours

3) Conclusion et perspectives.

Conclusion.

– Améliorations en algorithmique de la réduction des réseaux : petite dimension, dimension quelconque, comportement pratique.
– Utilisation de la méthode de Coppersmith pour trouver les pires cas pour l'arrondi de fonctions.

– Autres contributions : algorithme pgcd-BG-rapide, algorithme de construction des tables de Gal, cryptanalyse d'un générateur aléatoire dû à Littlewood.

Problèmes ouverts (1).

– Améliorer l'algorithme LLL flottant à l'aide des idées de Schönhage, Koy et Schnorr pour diminuer le nombre d'opérations arithmétiques.

– Décrire un algorithme LLL de complexité quasi-linéaire en $\log B$ en conservant une complexité polynomiale en d.

– Étudier les algorithmes permettant d'obtenir des vecteurs plus courts que ceux donnés par LLL.

Problèmes ouverts (2).

– Problème de l'heuristique dans la méthode de Coppersmith : quand les polynômes obtenus sont-ils algébriquement dépendants ?

– Coût prohibitif de l'algorithme de recherche de mauvais cas : améliorer les algorithmes de réduction de réseaux ? diminuer la taille du réseau à réduire ?

– Peut-on faire mieux que $m = n^2$ en temps polynomial ?