

Le Dilemme du Fabricant de Tables

Vincent LEFÈVRE

INRIA Lorraine, projet SPACES

Mars 2001

Plan

1. Introduction.
2. Tests exhaustifs.
3. Temps de calcul et résultats.
4. Application à l'implémentation de 2^x .
5. Conclusion.

Représentation des nombres

Les programmeurs de logiciels numériques veulent :

- calculs rapides ;
- résultats précis et cohérents ;
- programmes portables et calculs reproductibles.

Multiprécision, arithmétiques exactes, calcul symbolique (quand c'est possible)

→ résultats exacts ou très précis, mais souvent trop lent !

Arithmétique virgule flottante (presque toujours utilisée)

→ rapide, mais résultats souvent imprécis ; a besoin d'être améliorée !

Systemes à virgule flottante

Caractéristiques : base b , taille de mantisse n , plage des exposants $E_{\min} \dots E_{\max}$.

En général, $b = 2$ (mais 16 et 10 utilisés par certaines machines).

Nombre machine x : signe $s_x = \pm 1$, mantisse $m_x = x_0.x_1x_2\dots x_{n-1}$, exposant $E_x \in \llbracket E_{\min}, E_{\max} \rrbracket$.

$$x = s_x \times m_x \times b^{E_x}$$

Par exemple :

- Simple précision : $b = 2, n = 24, E \in \llbracket -126, +127 \rrbracket$.
- Double précision : $b = 2, n = 53, E \in \llbracket -1022, +1023 \rrbracket$.

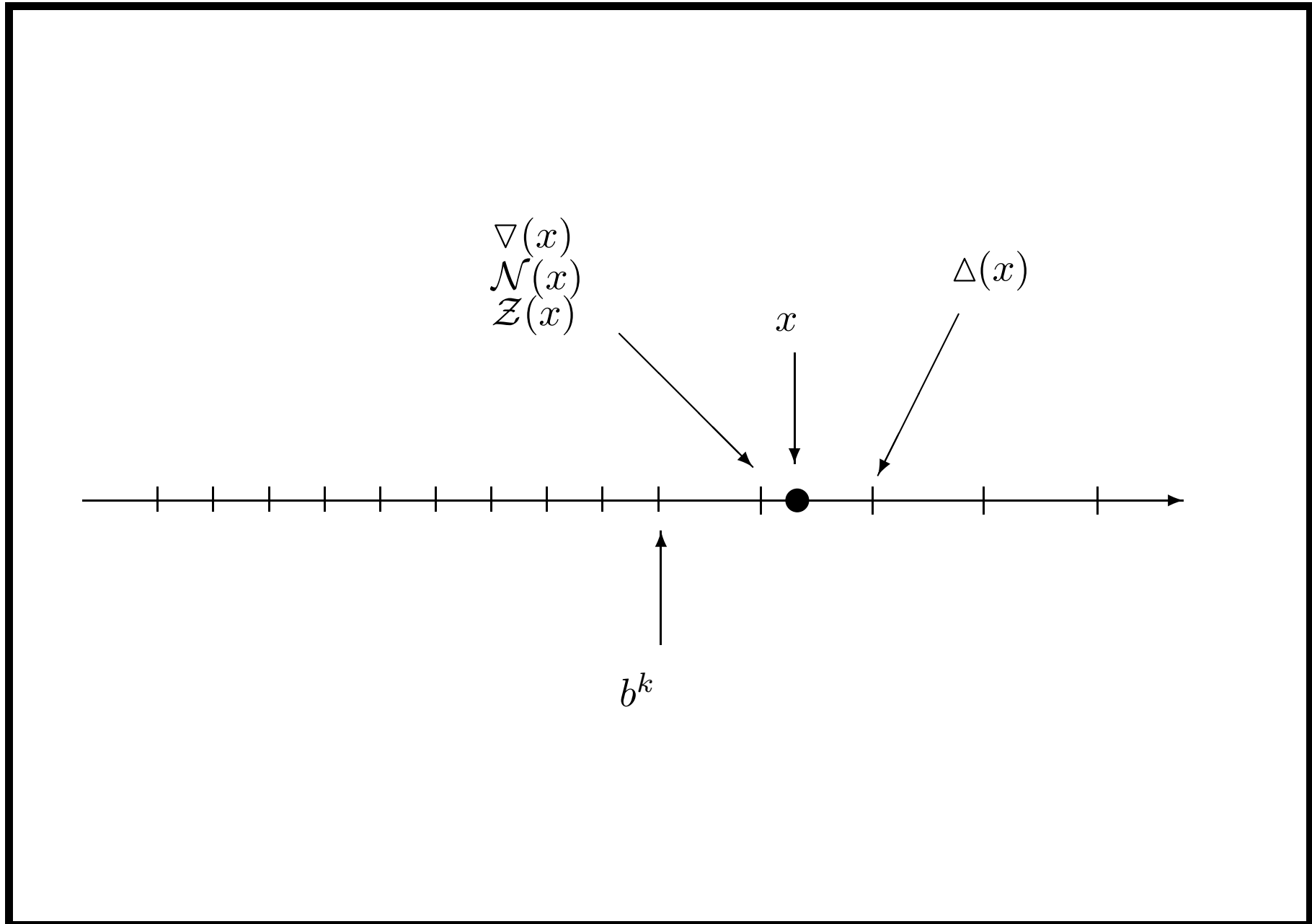
Modes d'arrondi

En général, la somme, le produit ou le quotient de 2 nombres machine *n'est pas* un nombre machine : il doit être **arrondi**.

Modes d'arrondi :

- vers $-\infty$: $\nabla(x)$ est le plus grand nombre machine $\leq x$;
- vers $+\infty$: $\Delta(x)$ est le plus petit nombre machine $\geq x$;
- vers 0 : $\mathcal{Z}(x)$ est égal à $\nabla(x)$ si $x \geq 0$, et à $\Delta(x)$ si $x < 0$;
- au plus près : $\mathcal{N}(x)$ est le nombre machine le plus proche de x .

Les 3 premiers : appelés *arrondis dirigés*.



Norme IEEE-754 (1985)

- Caractéristiques les plus connues : *formats* des représentations.
- Caractéristique la plus intéressante : **arrondi exact**.

C'est-à-dire :

- Mode d'arrondi *actif* : \diamond
- x et y : nombres machine.

Quand on calcule $x \star y$ (\star étant $+$, $-$, \times ou \div), le résultat obtenu doit *toujours* être $\diamond(x \star y)$, i.e. *l'arrondi du résultat exact*.

Même exigence pour \sqrt{x} .

Conséquences

- compatibilité et portabilité : même programme → mêmes résultats sur différentes machines ;
- des algorithmes (et des preuves) peuvent être construits en utilisant cette propriété (arithmétique à précision arbitraire, somme de plusieurs nombres machine, prise de décisions en géométrie algorithmique...);
- préservation de propriétés (e.g. monotonie);
- arithmétique d'intervalles.

Malheureusement : rarement accessible dans un langage de haut niveau, et pas encore de spécification pour les fonctions élémentaires (exp, log, sin...).

Exemples de propriétés

Avec l'arrondi exact :

- En arrondi au plus près, le calcul de

$$X' = X / \sqrt{X^2 + Y^2}$$

où X et Y sont des nombres machine, donnera toujours un résultat compris entre -1 et $+1$ (Kahan).

- Quand le résultat exact est un nombre machine (e.g. un entier), on l'obtient sans erreur.

Sur certaines machines non conformes à la norme IEEE, l'instruction `FORTAN I = 14.0/7.0` donnera comme résultat 1.

Le Dilemme du Fabricant de Tables

- système à virgule flottante en base 2, mantisses de n bits ;
- fonction élémentaire f (exp, log, sin, cos, etc.) ;
- nombre machine x ;
- pour $m > n$, on peut calculer une approximation y' de $y = f(x)$ dont l'erreur sur sa mantisse est $\leq 2^{-m}$.

Problème : **Obtient-on l'arrondi de $f(x)$ en arrondissant y' ?**

Pas toujours possible si y est de la forme :

– en arrondi au plus près,

$$\underbrace{1.xx\dots xx}_{n \text{ bits}} \overbrace{1000\dots 00}^{m \text{ bits}} xx\dots \quad \text{ou} \quad \underbrace{1.xx\dots xx}_{n \text{ bits}} \overbrace{0111\dots 11}^{m \text{ bits}} xx\dots$$

– avec les autres modes d'arrondi (arrondi *dirigé*),

$$\underbrace{1.xx\dots xx}_{n \text{ bits}} \overbrace{0000\dots 00}^{m \text{ bits}} xx\dots \quad \text{ou} \quad \underbrace{1.xx\dots xx}_{n \text{ bits}} \overbrace{1111\dots 11}^{m \text{ bits}} xx\dots$$

Ce problème est appelé le **Dilemme du Fabricant de Tables (TMD)**.

Exemples en double précision

Pour

$$\begin{aligned}
 x &= 0.011111111001110110011101110011100111010000111101101101 \\
 &= \frac{8980155785351021}{18014398509481984},
 \end{aligned}$$

$\sin x$ est égal à :

$$0.011110100110010101000001110011000011000100011010010101 \ 1 \ 1^{65} \ 0000\dots$$

En considérant la réciproque, on a : pour

$$\begin{aligned}
 x &= 0.011110100110010101000001110011000011000100011010010110 \\
 &= \frac{4306410053968715}{9007199254740992},
 \end{aligned}$$

$\arcsin x$ est égal à :

$$0.011111111001110110011101110011100111010000111101101101 \ 0 \ 0^{64} \ 1000\dots$$

Résoudre le TMD

- Lindemann, 1882 : l'exponentielle d'un nombre algébrique $\neq 0$ n'est pas algébrique ;
 - les nombres virgule flottante sont algébriques ;
- $\Rightarrow \exp(x), \sin(x), \cos(x), \arctan(x)$ pour $x \neq 0$, et $\log x$ pour $x \neq 1$ ne peuvent pas avoir une infinité de 0 ou 1 consécutifs dans leur écriture binaire.
- \Rightarrow Pour tout x , il existe m tel que le TMD ne peut pas se produire.
- Le nombre de nombres machine est fini \Rightarrow il existe m tel que pour tout x le TMD ne peut pas se produire.
- Problème : **trouver cet m** (précision intermédiaire).

Quelques estimations

Expériences \rightarrow il semble que $m \approx 2n$.

Attention! approche non rigoureuse. On cherche à comprendre intuitivement d'où vient la relation $m \approx 2n$. On suppose :

- arrondi au plus près ;
- quand x est un nombre machine, les bits de $f(x)$ après la n -ième position peuvent être vus comme des *suites aléatoires de 0 et de 1*, le 0 et le 1 étant équiprobables ;
- ces suites peuvent être considérées « indépendantes » pour deux nombres machine différents.

La mantisse de $y = f(x)$ est de la forme :

$$y_0.y_1y_2\dots y_{n-1} \overbrace{01111\dots 11}^{k \text{ bits}} \dots \quad \text{ou} \quad y_0.y_1y_2\dots y_{n-1} \overbrace{10000\dots 00}^{k \text{ bits}} \dots$$

avec $k \geq 1$. Plus grande valeur de k ?

Nos hypothèses \rightarrow la « probabilité » d'avoir $k \geq k_0$ est de 2^{1-k_0} .

n bits de mantisse et n_e exposants : $N = n_e \times 2^{n-1}$ nombres machine
 $\Rightarrow m_{\max} = n + k_{\max} \approx n + \log_2(N) = 2n + \log_2(n_e) - 1$ (non prouvé).

Meilleur théorème connu (Nesterenko et Waldschmidt, 1995)

$\rightarrow m \leq$ plusieurs millions ou milliards pour les fonctions liées à l'exponentielle complexe (exp, log, fonctions trigonométriques et hyperboliques).

Bornes non satisfaisantes. \rightarrow **Tests exhaustifs**

Tests exhaustifs

Problème : système à virgule flottante, fonction f sur un intervalle I et entier m_0 donnés. Quels sont les nombres machine $x \in I$ tels que la mantisse de $f(x)$ soit de la forme suivante ?

$$\underbrace{1.\overbrace{xx\dots xx}^{m_0 \text{ bits}} r\overbrace{bb\dots bb}^{n \text{ bits}} xx\dots}_{n \text{ bits}}$$

où tous les bits b ont la même valeur.

Estimation du temps de calcul pour une fonction élémentaire f , $n = 53$ (double précision), $m_0 \approx 90$, machine à 500 MHz, un algorithme classique (200 cycles) : 2^{52} mantisses \rightarrow **57 ans** pour chaque exposant !

\rightarrow **Nous avons besoin d'algorithmes très rapides.**

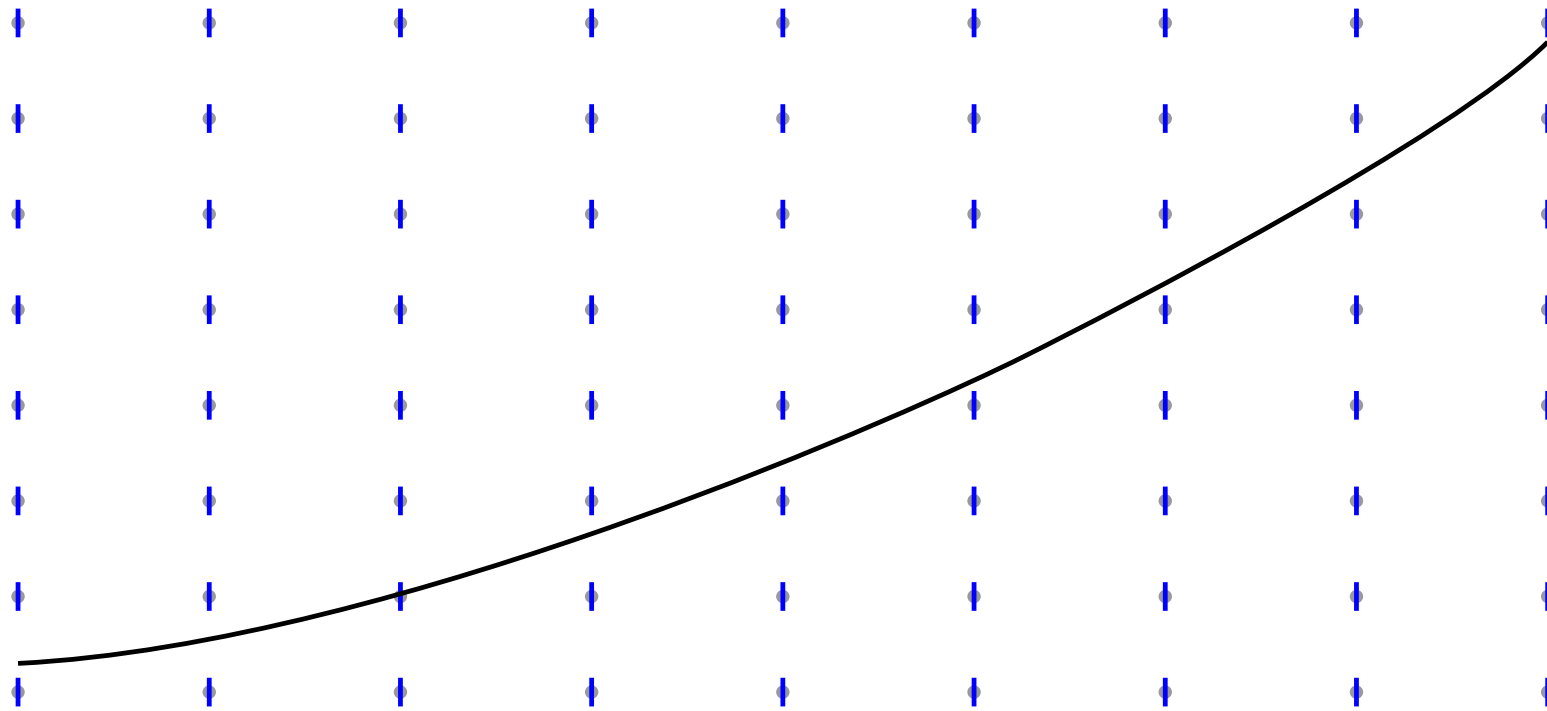
Filtres

1. Filtre : algorithme très rapide (basse précision) qui sélectionne un ensemble S contenant tous les « pires cas » (arguments tels que $m \geq m_0$).
2. On teste ensuite chaque nombre machine de S avec un algorithme plus précis, mais qui peut être beaucoup plus lent.

Note :

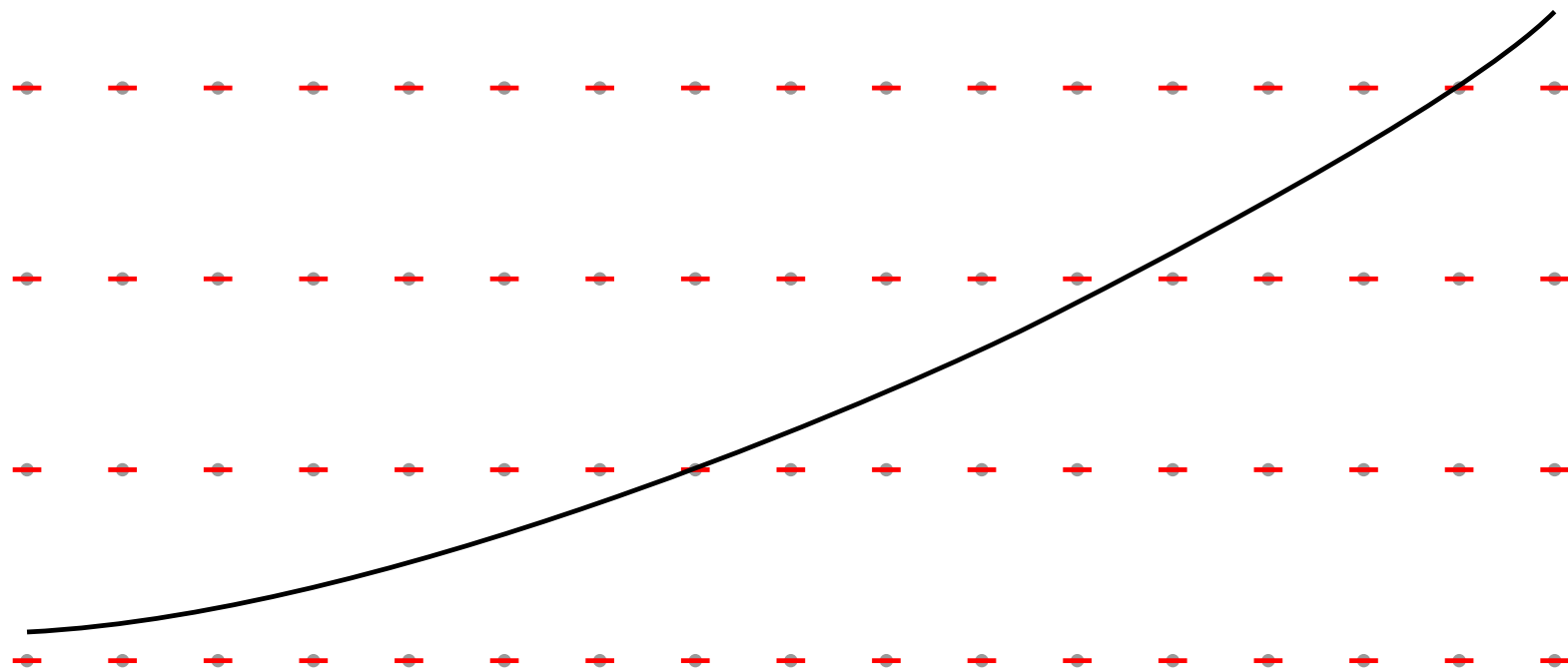
- on peut utiliser plusieurs filtres ;
- les filtres sont choisis d'après les hypothèses probabilistes.

Test de f dans un domaine donné



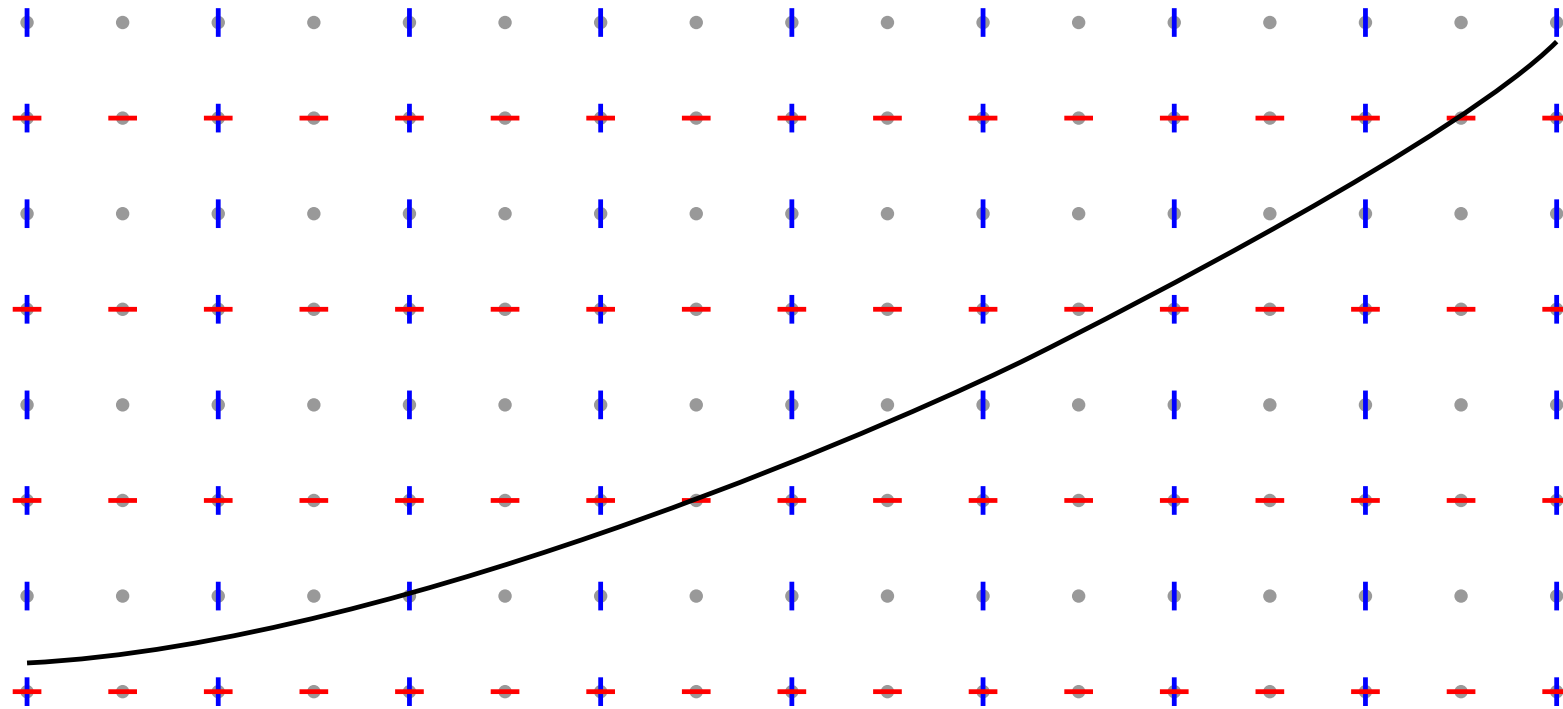
→ 9 arguments à tester.

Test de f^{-1} dans le même domaine



→ 4 arguments à tester.

Équivalence $f \leftrightarrow f^{-1}$



→ 7 arguments à tester (test de f^{-1}) au lieu de $9 + 4 = 13$.

Approximation d'une fonction par un polynôme

parce que les nombres machine sont espacés régulièrement et calculer les valeurs successives d'un polynôme peut se faire très rapidement (prochain transparent).

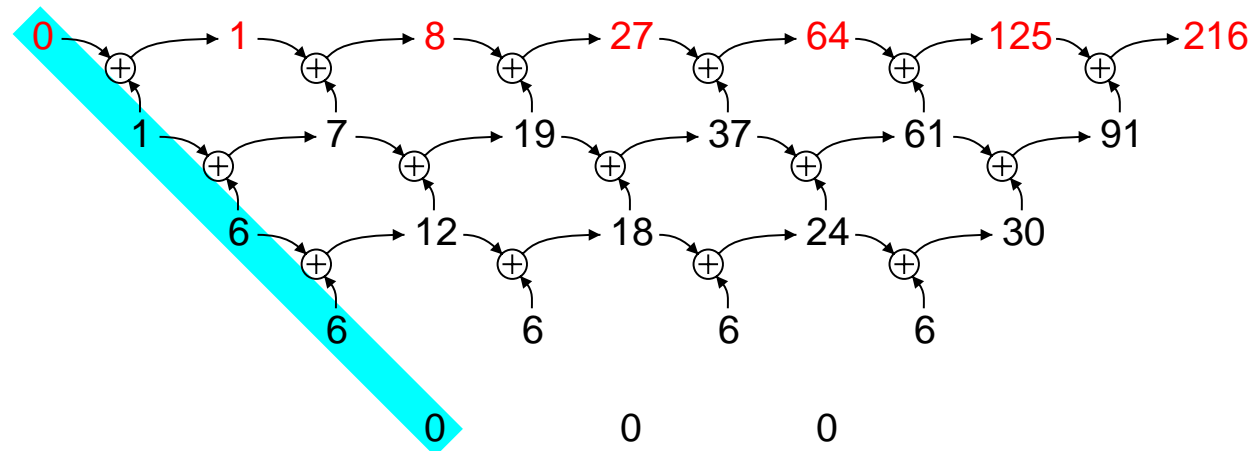
Formule de Taylor. Pas la meilleure approximation, mais :

- facile à calculer ;
- l'erreur peut être facilement majorée ;
- la précision est suffisante pour nous ;
- on peut choisir le degré du polynôme dynamiquement.

Calculs avec Maple + Intpak (arithmétique d'intervalles).

Calcul des valeurs successives d'un polynôme

Polynôme $P(X) = X^3$. Table des différences :



Coefficients du polynôme dans la base

$$\left\{ 1, X, \frac{X(X-1)}{2}, \frac{X(X-1)(X-2)}{3!}, \frac{X(X-1)(X-2)(X-3)}{4!}, \dots \right\}.$$

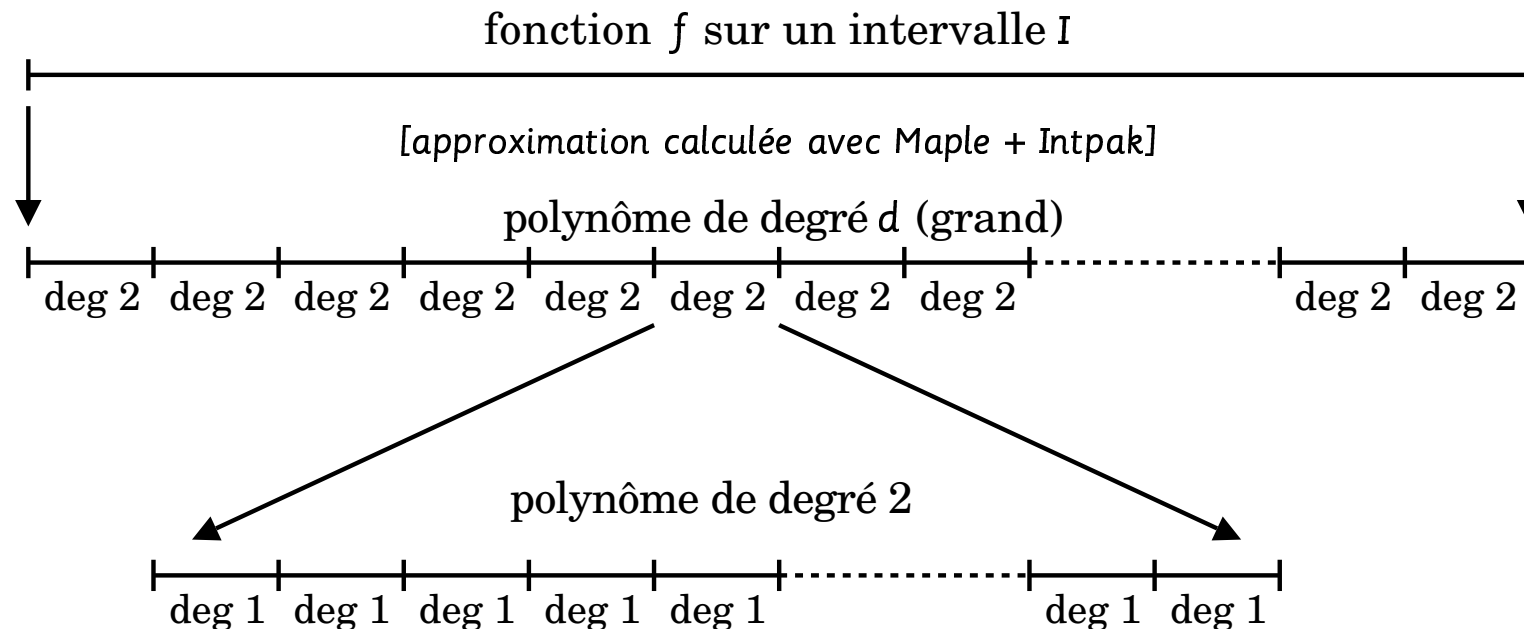
Avantages : rapide, calculs modulo 2^p possibles, peu de mémoire.

Petit vs grand degré

- Petit degré:
 - Calcul rapide des valeurs successives (degré $d \rightarrow d$ add.);
 - mais l'approximation est valide dans un petit intervalle seulement : le calcul des coefficients des polynômes (pour chaque intervalle) prendrait beaucoup de temps.
- Grand degré:
 - Le temps de calcul des approximations est négligeable ;
 - mais le calcul des valeurs successives prend plus de temps.

On voudrait les avantages des petits et des grands degrés à la fois
→ approximations hiérarchiques.

Approximations hiérarchiques



Calcul de l'approximation suivante : basée sur la méthode des différences finies → rapide.

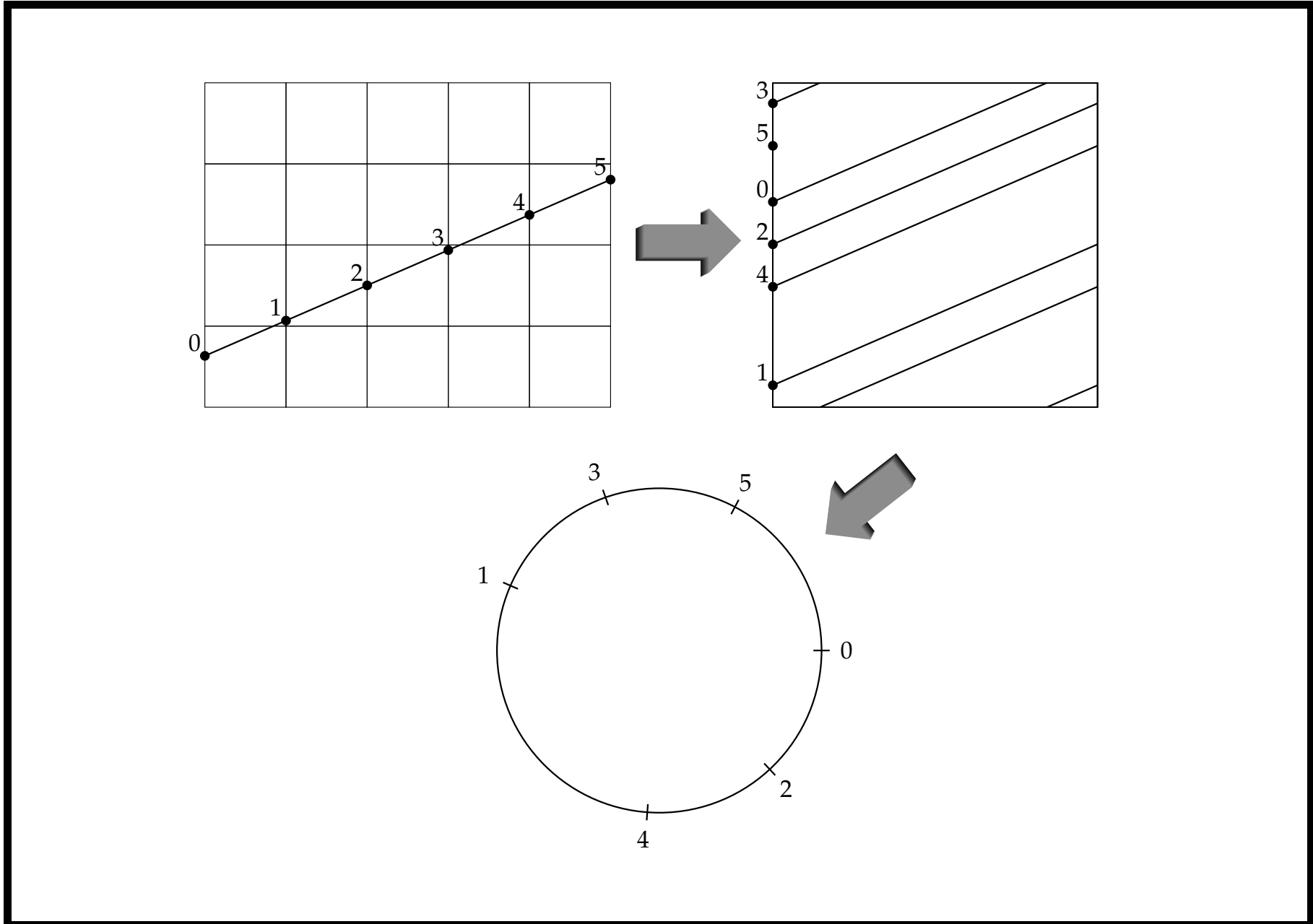
Distance entre un segment et \mathbb{Z}^2

Dans chaque intervalle :

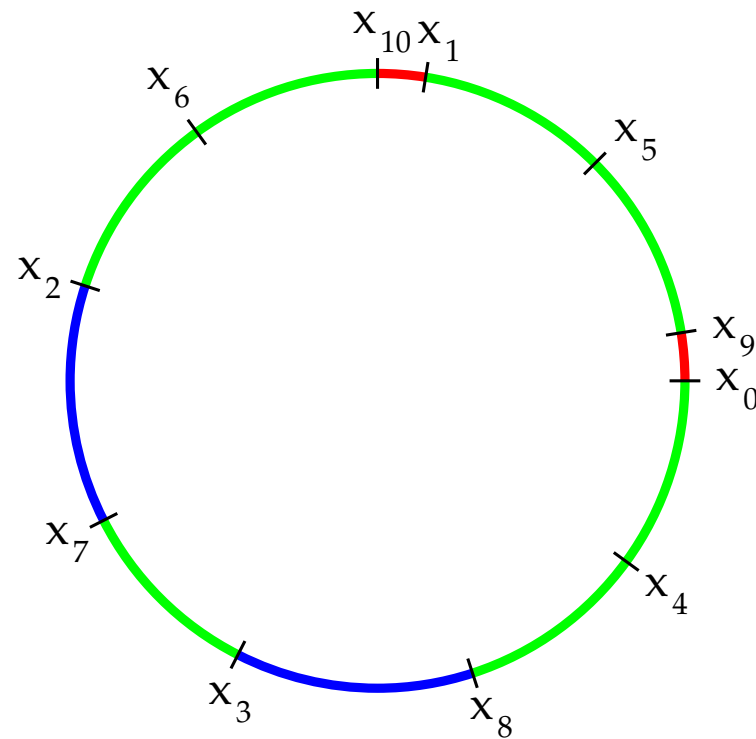
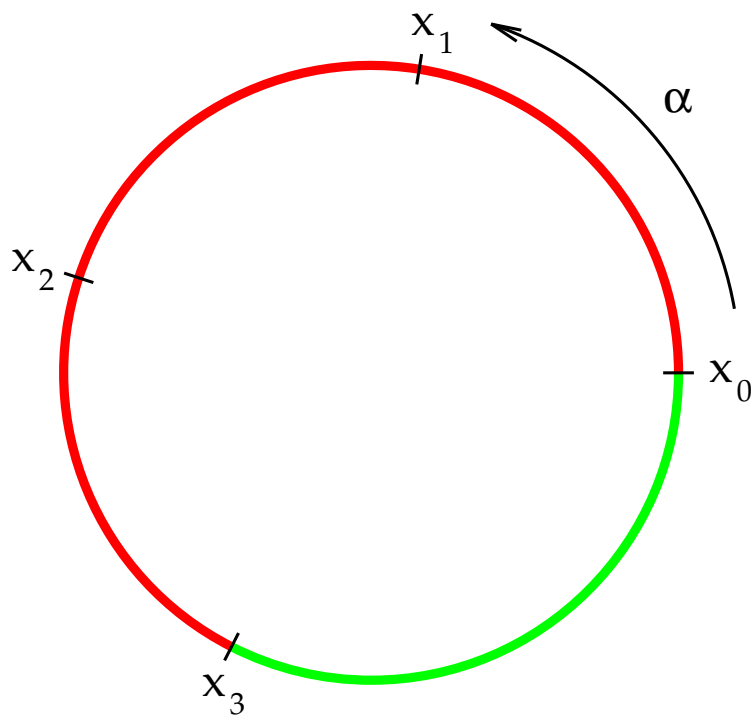
- f approchée par un polynôme de degré 1 \rightarrow segment.
- Multiplication des coordonnées par des puissances de 2
 \rightarrow grille = \mathbb{Z}^2 .
- \rightarrow distance entre un segment et \mathbb{Z}^2 ?

Nous proposons un algorithme beaucoup plus rapide que l'algorithme naïf, mais il ne donne qu'une minoration de la distance – suffisant pour nous.

Relié au théorème des 3 distances...



Construction des points



Théorème des trois distances

À tout moment durant ce processus de construction (i.e. pour un n donné, quand les points $x_0, x_1, x_2, \dots, x_n$ sont construits), l'angle (distance) entre deux points voisins sur le cercle ne peut prendre qu'au plus **3 valeurs possibles**. Ces valeurs dépendent de n et α .

De plus, il y a une infinité de valeurs de n pour lesquelles cette distance ne prend que **2 valeurs possibles**. Ces *configurations intéressantes* sont utilisées par notre algorithme.

L'algorithme suivant calcule une minoration de $\{b - ax\}$, où $x \in \llbracket 0, N - 1 \rrbracket$. Lié à l'algorithme d'Euclide (PGCD).

Algorithme

Initialisation: $x = \{a\}$; $y = 1 - \{a\}$; $d = \{b\}$; $u = v = 1$;

Boucle infinie:

```

si ( $d < x$ )
  tant que ( $x < y$ )
    si ( $u + v \geq N$ ) fin
     $y = y - x$ ;  $u = u + v$ ;
  si ( $u + v \geq N$ ) fin
   $x = x - y$ ;  $v = v + u$ ;
sinon
   $d = d - x$ ;
  tant que ( $y < x$ )
    si ( $u + v \geq N$ ) fin
     $x = x - y$ ;  $v = v + u$ ;
  si ( $u + v \geq N$ ) fin
   $y = y - x$ ;  $u = u + v$ ;

```

Valeur de retour: d

Implémentation (double précision)

3 étapes :

1. filtres (calculs en assembleur, résultats temporaires) ;
2. tests plus lents (en Maple) pour obtenir les résultats finaux, stockés sur disque ;
3. tests pour restreindre le nombre de pires cas et obtenir les résultats de la fonction réciproque.

Première étape : algorithmes les plus rapides et implémentation très efficace. Prend quand même beaucoup de temps, même parallélisée.

Après la première étape, le nombre d'arguments est divisé (en moyenne) par $2^{32} \approx 4 \times 10^9$.

Approximations

(Valeurs données : cas général.)

- Fonction f sur un intervalle I (1 exposant, 2^{53} mantisses).
- Fonction f approchée par des polynômes P_i de degré d_i (~ 4 à 20) sur 2^{13} sous-intervalles $J_i \subset I$ (avec Maple + Intpak).
- Polynômes P_i approchés par des polynômes $Q_{i,j}$ de degré 2 sur des sous-intervalles $K_{i,j} \subset J_i$ ayant 2^{15} arguments.
- *Minoration...* sur $K_{i,j}$ (approximation de degré 1).
- En cas d'échec :
 - $K_{i,j}$ est découpé en ~ 4 sous-intervalles $L_{i,j,k}$.
 - *Minoration...* sur $L_{i,j,k}$ (approximation de degré 1).
 - En cas d'échec : *méthode des différences finies* sur $L_{i,j,k}$.

Parallélisation des calculs

Cible : réseau de stations de travail à l'ENS-Lyon.

Serveur + clients. Les clients se connectent au serveur pour obtenir un numéro d'intervalle (i) et d'autres paramètres
→ en général, 5 minutes à 2 heures de calcul.

Clients implémentés de telle façon à ce qu'ils

- tournent à une basse priorité (*nice*) ;
- puissent s'arrêter automatiquement après un temps donné ;
- détectent automatiquement quand une machine est utilisée (clavier, souris... ou autre processus de calcul en tâche de fond) et s'arrêtent dans ce cas.

Temps de calcul

En pratique, à l'ENS-Lyon, quelques jours à quelques semaines par exposant (2^{53} mantisses).

Jusqu'à 35 arguments testés par cycle en moyenne (sur une Sun Ultra-5). Peut être amélioré dans de futures implémentations.

Le choix de la taille des sous-intervalles $K_{i,j}$ et $L_{i,j,k}$ est très important. Par exemple (exp, $x_0 = 16$, 2^{40} arguments) :

$\#K_{i,j}$	$\#L_k$	temps
32768	32768	9530 s
4096	4096	930 s
32768	8192	430 s
32768	4096	360 s
32768	2048	500 s

Résultats : exp et log en double précision

- $\exp(x)$ est testé pour x entre $1/2$ et $\log(2^{1024})$, et pour x entre $\log(2^{-1074})$ et $-1/2$ (dénormalisés pris en compte).
- $\log(x)$ est testé pour x entre $1/2$ et 2 .

Résultats pour $\log, m \geq 115$

E	mantisse	R	m
86	1.1001000111101100010001000001001011000011010001001111	D	115
245	1.1100100100001000000100001101001101010100011000011000	D	117
656	1.1000011001110000110111100000101101101000110010101101	D	116
678	1.0110001010101000100001100001001101100010100110110110	D	118
732	1.1111110100010101110110101010011011001110001100110010	N	115
772	1.0111101100011101100101111100100100000010100110000101	D	115

Pires cas pour $x < 1$:

E	mantisse	R	m
-509	1.1110101001110001110110000101110011101110000000100000	D	114
-384	1.1001010001110110111000110000010011001101011111000111	N	114
-232	1.0010011011101001110001001101001100100111100101100000	D	114
- 35	1.0110000100111001010101011101110010000000001011111000	N	114

Résultats : 2^x et $\log_2(x)$ en double précision

Plus simple que exp et log, car $\log_2(2^k t) = k + \log_2(t)$.

Si k est un entier :

- $2^k t$ et t ont la même mantisse ;
- $k + \log_2(t) \rightarrow$ décalage dans la mantisse.

$\log_2(x)$ testé dans $[1/2, 2)$.

2^x testé dans $[1, 2)$ et $[32, 33)$.

Résultats pour $2^x, m \geq 111$

E	mantisse	R	m
-15	-1.0010100001100011101010111010111010101111011110110010	D	111
-20	-1.0100000101101111011011000110010001000101101011001111	D	111
-32	-1.000001010101010110000000011100100010101011001111110001	D	111
-33	-1.0001100001011011100011011011011011010101100000011101	D	111
-29	1.0101011010001110100011001110110001001111011001101100	N	111
-27	1.0001001010110001010010100011000110001111100100000100	D	112
-25	1.1011111110111011110111100100010011101101111111000101	D	113
-10	1.1110010001011001011001010010011010111111100101001101	N	113
-10	1.1110011101100000010010010000011100110000011001111111	N	111
- 8	1.1111100110011010111111101111101000110000110101100101	D	111
- 6	1.1000111111101010100000011111101101110101000100101110	N	111

Résultats pour $\log_2(x)$, $m \geq 106$

E	mantisse	R	m
0	1.1011010011101011111001000000110010010101101000000001	N	107
1	1.000110111010001110011111111001010001110001111101010	D	106
2	1.000110111010001110011111111001010001110001111101010	D	107
2	1.1000100111011001010010001010100101001111111000010111	N	106
4	1.000110111010001110011111111001010001110001111101010	N	108
16	1.1001010101101011011011110011010000011111010111010000	N	106
64	1.0110000101010101010111110111010110001000010110110100	D	106
128	1.0110000101010101010111110111010110001000010110110100	D	107
128	1.1101001100001010010000110111011100111101110100011011	D	106
256	1.0110000101010101010111110111010110001000010110110100	D	108
256	1.1101001100001010010000110111011100111101110100011011	N	107
512	1.0110000101010101010111110111010110001000010110110100	D	109

(E : -1, 0 ou puissance de 2 ; utiliser $\log_2(2^k t) = k + \log_2(t)$ pour autres exposants.)

Résultats pour la double précision (résumé)

N : arrondi au plus près. D : arrondis dirigés. Valeurs de m maxi :

f	domaine	f, N	f, D	f^{-1}, N	f^{-1}, D
exp	résultats complets	112*	113*	115	118
2^x	résultats complets	113	113	108	109
$1/x^2$	résultats complets	106	105	111	107
sin	2^{-5} à 2	110	119	108	118
cos	2^{-6} à 12867/8192	108	109	111	116
tan	2^{-5} à arctan(2)	111	109	108	109
sinh	2^{-4} à 2^5	108	109	112	113
cosh	2^{-6} à 2^5	111	109	115	111

* pour $|x| \geq 2^{-32}$.

Résultats : exp en simple précision, $m \geq 51$

Pour $|x| \geq 2^{-15}$:

E	mantisse	R	m
5	-1.01101101011110110001100	D	51
3	-1.11010010001001011001101	N	52
- 2	-1.10101100111111110010101	D	51
- 8	-1.11100001110110111110001	N	51
- 9	-1.01100101100111101100100	D	52
-10	-1.11000001110001001011100	N	51
-10	1.01100010011110101001111	D	52

Sinon, le pire cas est :

E	mantisse	R	m
-24	1.111111111111111111111111	D	71

Résultats : log en simple précision, $m \geq 55$

E	mantisse	R	m
-66	1.00010000100010100101101	D	57
-65	1.00100010110101010111000	N	55
3	1.0010111100011111101011	N	55
25	1.10111010110010110100101	N	57
27	1.11000000100111010111110	N	56
76	1.10110001001000011010011	N	58
78	1.01010001100100001100000	N	55
117	1.00101111111001100001010	D	56

Résultats : $f(x) = 1/x^2$ et $g(x) = 1/\sqrt{x}$

Tests pour diverses précisions : $19 \leq n \leq 58$. Pires cas :

$n = 21$:

$$f(0.110100100010001011100) = 1.01111011111100010100 \ 1 \ 1^{29} \ 0110\dots$$

$n = 21$:

$$g(1.01111011111100010101) = 0.110100100010001011011 \ 1 \ 1^{30} \ 0101\dots$$

$n = 20$:

$$f(0.11010010001000101110) = 1.0111101111110001010 \ 0 \ 1^{30} \ 0110\dots$$

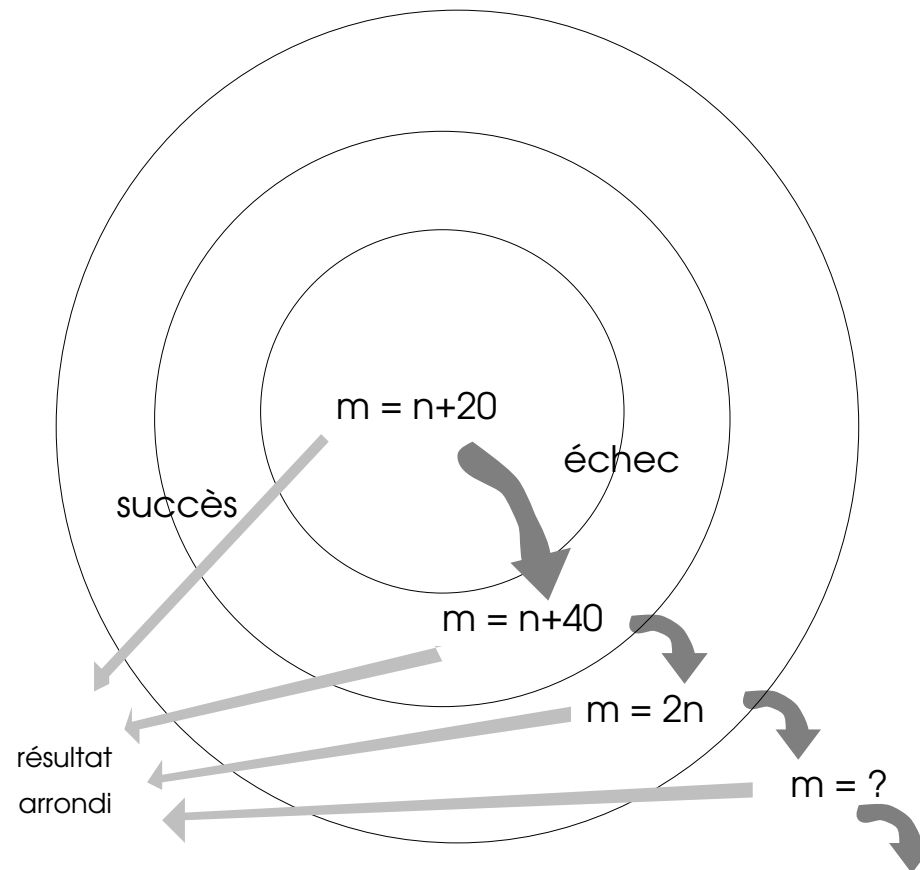
Correspondent à : $1556245 \times 430359^2 = 2^{58} + 101$

Nos résultats et hypothèses probabilistes

Nombre moyen de pires cas par exposant tels que $k \geq k_0$:

k_0	estimation	$\exp(x)$	$\exp(-x)$	$\sin(x)$	$\cos(x)$
45	768	788.7	761.0	774.8	762.5
46	384	393.4	377.9	398.0	396.5
47	192	200.3	190.1	198.7	190.0
48	96	98.2	95.9	104.5	94.5
49	48	52.9	46.8	52.8	41.5
50	24	27.4	23.5	26.2	18.0
51	12	14.0	11.4	12.2	6.5
52	6	7.0	5.2	6.3	3.0
53	3	2.6	2.4	3.3	2.0
exposants		-1 à 8	0 à 7	-5 à 0	-2 à -1

Stratégie de Ziv



Utilisation des résultats des tests

1. Construire un algorithme qui calcule f , en utilisant la stratégie de Ziv, jusqu'à une précision de m_0 bits.
2. Tester l'algorithme sur tous les pires cas vérifiant $m \geq m_0$.
3. Garder les pires cas pour lesquels le TMD se produit (seront stockés dans une table) et compléter l'implémentation.

m_0 petit \rightarrow implémentation rapide, mais grande table.

- L'algorithme détermine si x est un vrai pire cas ou non.
- L'algorithme calcule une approximation de $f(x)$.

\rightarrow Pour chaque pire cas x , on n'a pas besoin de stocker x en entier (un code de hachage suffit), ni le résultat arrondi (1 bit suffit).

Implémentation de 2^x

Nous voulons montrer que l'arrondi exact est possible à bas coût.

→ **Nous nous sommes concentrés sur les pires cas.**

La stratégie de Ziv *n'est pas* (encore) utilisée.

Réduction d'argument → $|x| \leq \frac{1}{2}$.

Si $|x| < 2^{-40}$, algorithme spécial à base de tables : $2^x \approx 1 + x \cdot \log(2)$ et les arguments frontière entre deux valeurs arrondies différentes sont stockés dans une table ($\approx 17\,000$ valeurs).

Sinon 2^x est calculé avec une erreur $< 2^{-97} + 2^{-101}$.

Tous les pires cas possibles sont testés → vrais pires cas.

Mode d'arrondi	Pires cas	Exact	Inexact
vers $-\infty$	52 231	51 148	1 083
vers $+\infty$	52 231	51 028	1 203
au plus près	52 224	26 174	26 050

→ seuls les pires cas correspondant à un arrondi inexact sont mémorisés. Dans l'algorithme : premier code de hachage (12 bits) pour réduire l'ensemble ; second code de hachage (2 octets) pour les comparaisons (au plus 4 pour les arrondis dirigés, et 17 pour l'arrondi au plus près).

Temps : au plus $4.10 \mu s$ sur une Sun Ultra-5 à 333 MHz.

(ml4j d'IBM : jusqu'à $4.4 ms$ pour exp.)

Conclusion

Notre but : avoir une bibliothèque de fonctions élémentaires arrondies exactement qui ne prenne pas trop de ressources (temps et mémoire).

→ Extension de la norme IEEE-754.

Conséquences :

- Résultats plus précis en général.
- Compatibilité et portabilité.
- Algorithmes (et preuves) basés sur l'arrondi exact.
- Certaines propriétés mathématiques préservées.
- Arithmétique d'intervalles, minorations / majorations de résultats.

Références / URL

- Ma thèse *Moyens arithmétiques pour un calcul fiable* :
<ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD2000/PhD2000-02.ps.Z>
- Page sur le *dilemme du fabricant de tables* :
<http://www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm>
- Ma page « recherche » :
<http://www.vinc17.org/research/>
ou
<http://www.ens-lyon.fr/~vlefevre/research/>