# 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 !

## Systèmes à virgule flottante

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

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

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

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

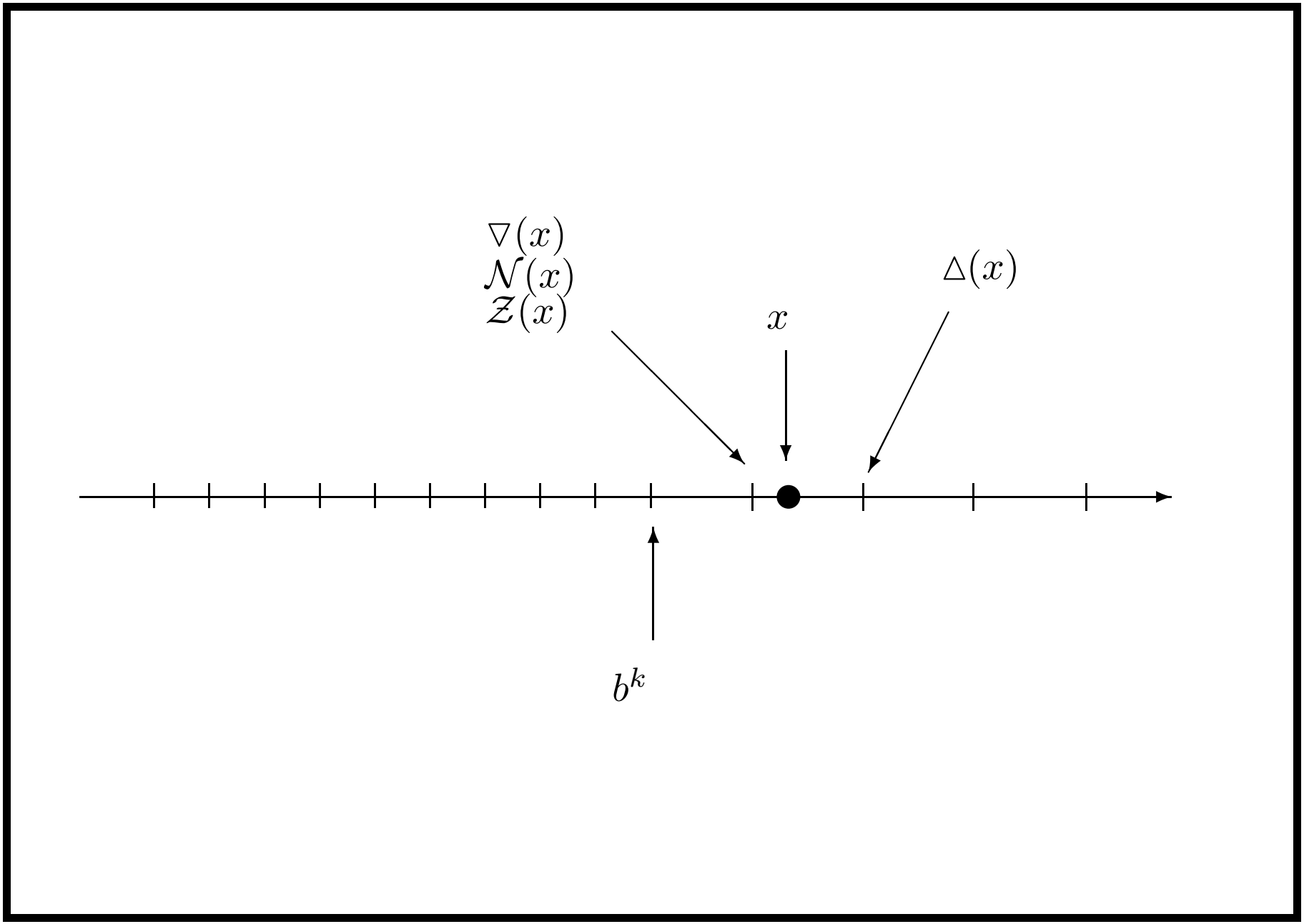
Par exemple :
- Simple précision : $b = 2$, $n = 24$, $E \in [\![-126, +127]\!]$.
- Double précision : $b = 2$, $n = 53$, $E \in [\![-1022, +1023]\!]$.

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

Les 3 premiers : appelés *arrondis dirigés*.


$\nabla(x)$
$\mathcal{N}(x)$
$\mathcal{Z}(x)$
$\triangle(x)$
$x$
$b^k$

## 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 FORTRAN `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 $\leqslant 2^{-m}$.

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,

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

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

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

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

# Exemples en double précision

Pour

$$x = 0.01111111100111011001110111001110011101000011110110 1101$$
$$= \frac{8980155785351021}{18014398509481984},$$

$\sin x$ est égal à :

$$0.01111010011001010100000111001100001100010001101001010 1\,1^{65}\,0000\ldots$$

En considérant la réciproque, on a : pour

$$x = 0.01111010011001010100000111001100001100010001101001011 0$$
$$= \frac{4306410053968715}{9007199254740992},$$

$\arcsin x$ est égal à :

$$0.01111111100111011001110111001110011101000011110110 1101\,0\,0^{64}\,1000\ldots$$

## 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 :

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

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

Nos hypothèses $\rightarrow$ la « probabilité » d'avoir $k \geqslant k_0$ 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)
$\rightarrow m \leqslant$ 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 ?

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

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 !

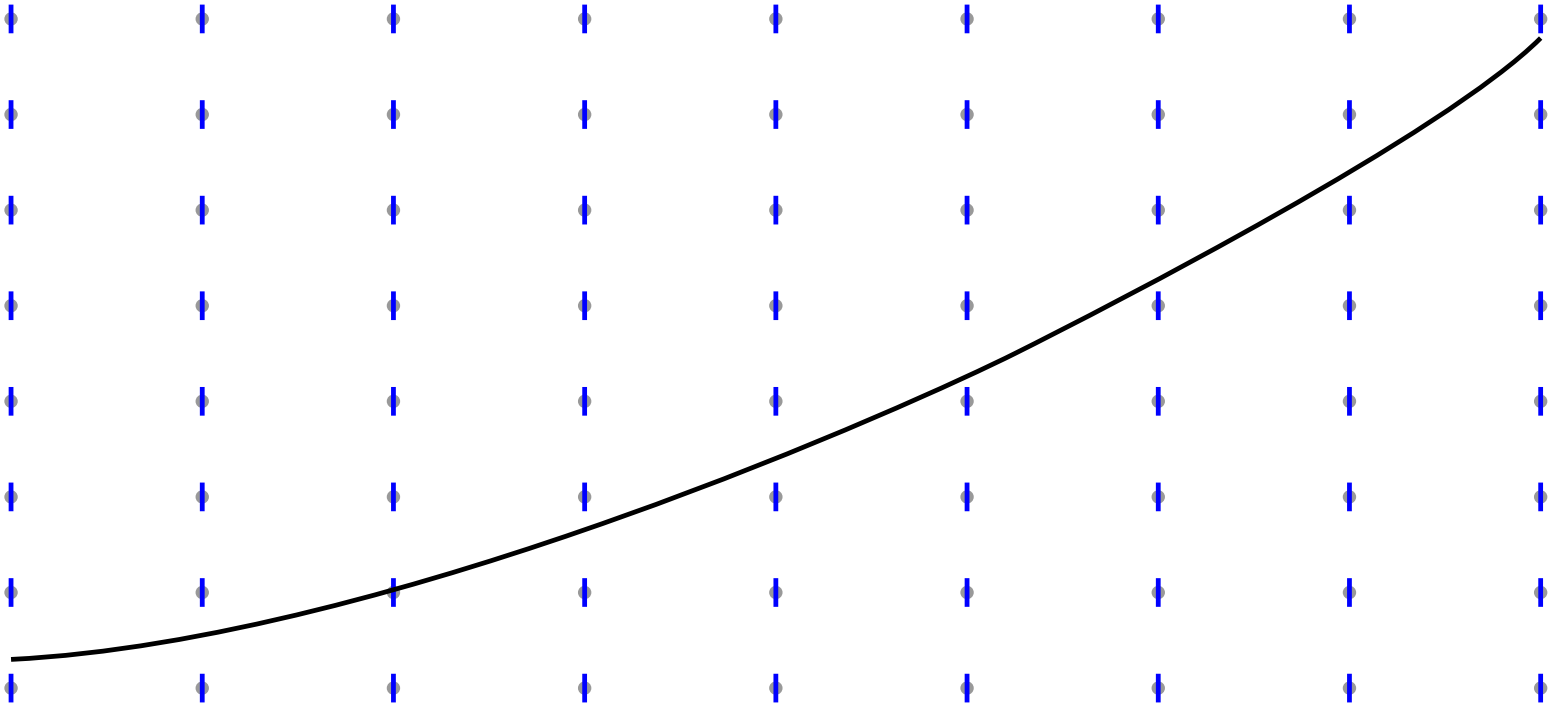
$\rightarrow$ **Nous avons besoin d'algorithmes très rapides.**

## Filtres

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

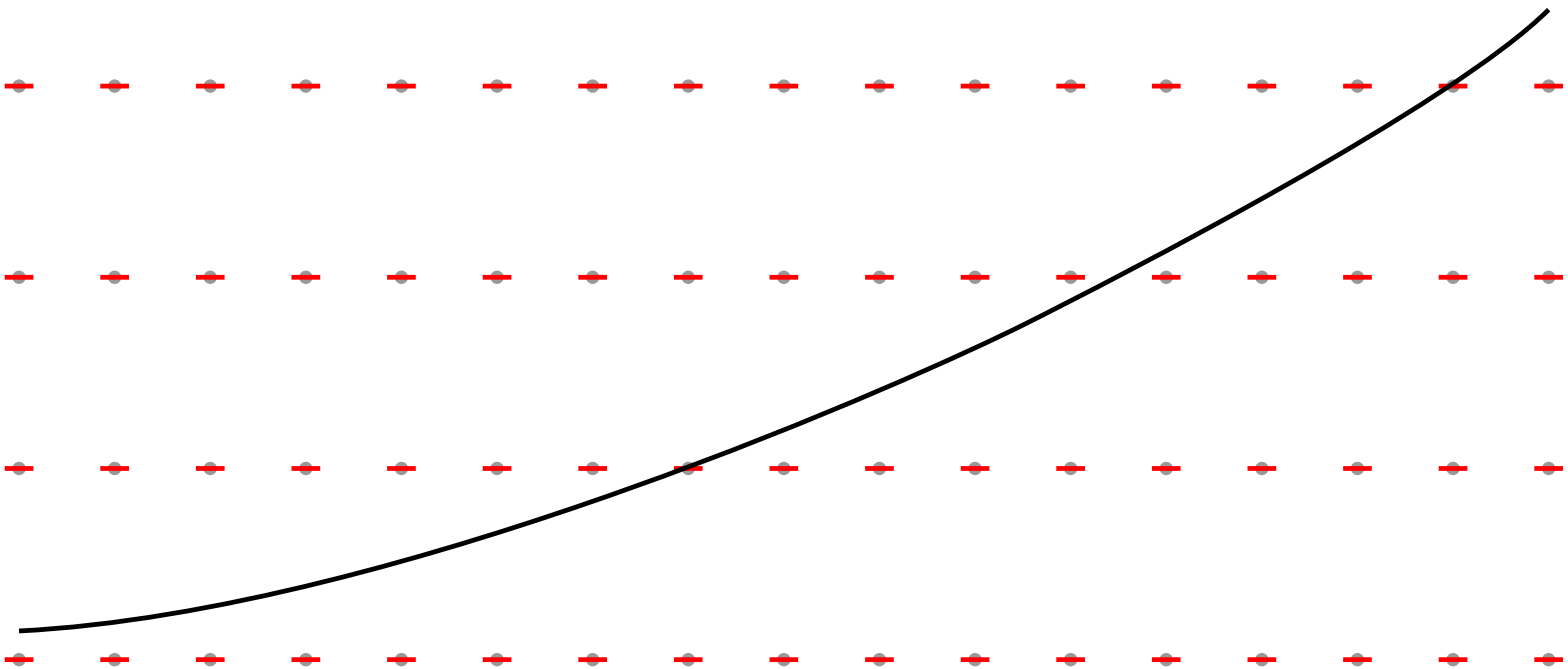
Note :
– on peut utiliser plusieurs filtres ;
– les filtres sont choisis d'après les hypothèses probabilistes.

## Test de $f$ dans un domaine donné



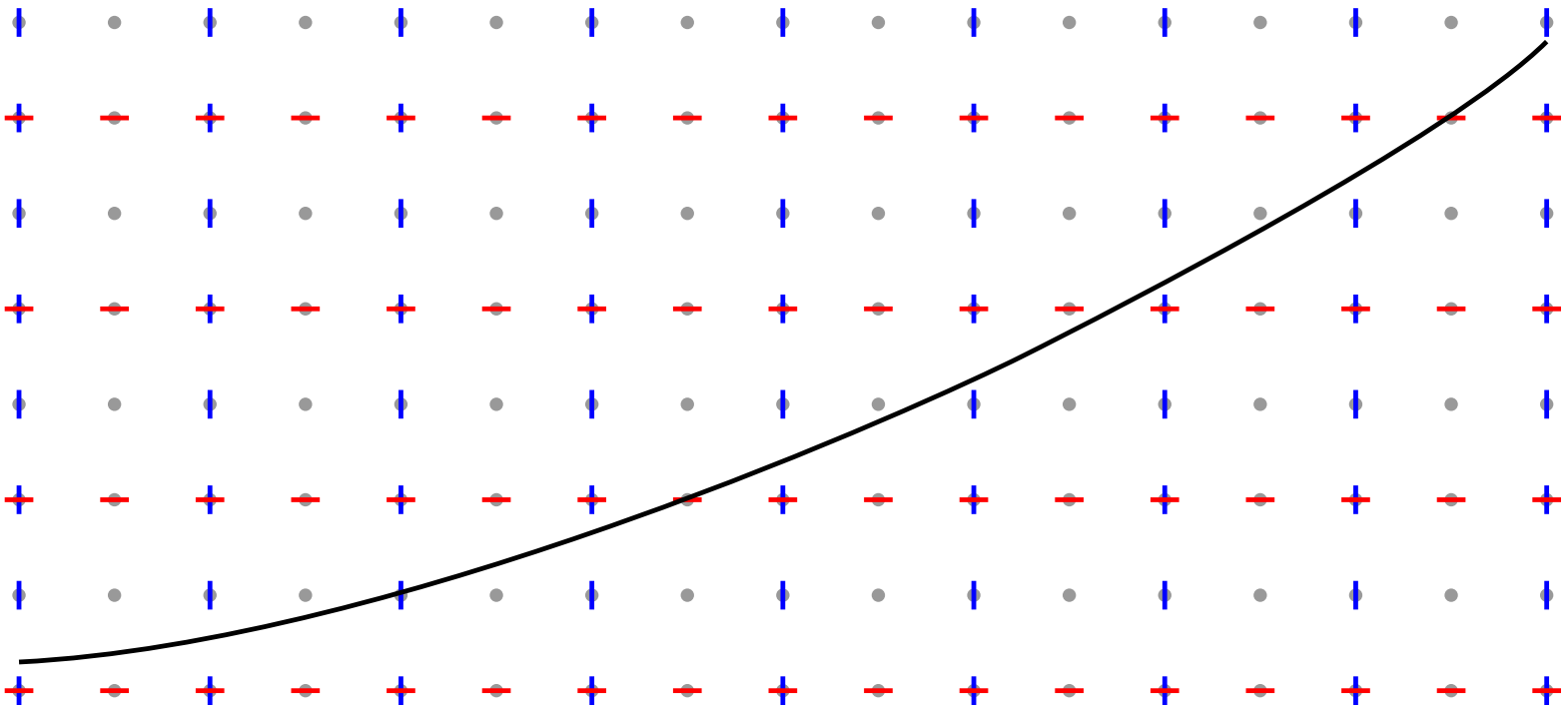
$\rightarrow$ 9 arguments à tester.

## Test de $f^{-1}$ dans le même domaine



$\rightarrow$ 4 arguments à tester.

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



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

## Approximation d'une fonction par un polynôme

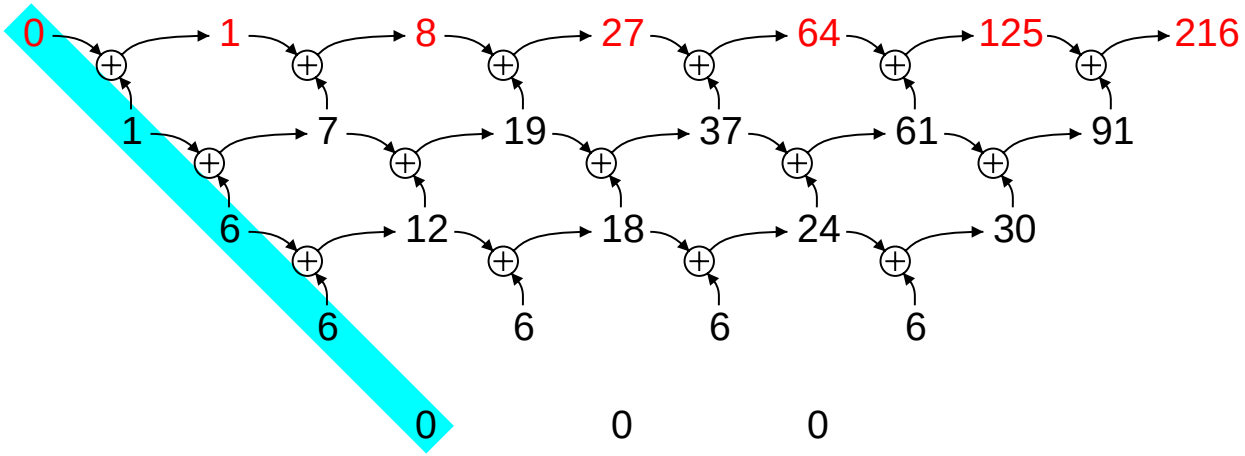
parce que les nombres machine sont espacés régulièrement et calculer les valeurs successives d'un polynôme peut se faire très rapidement (prochain transparent).

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

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

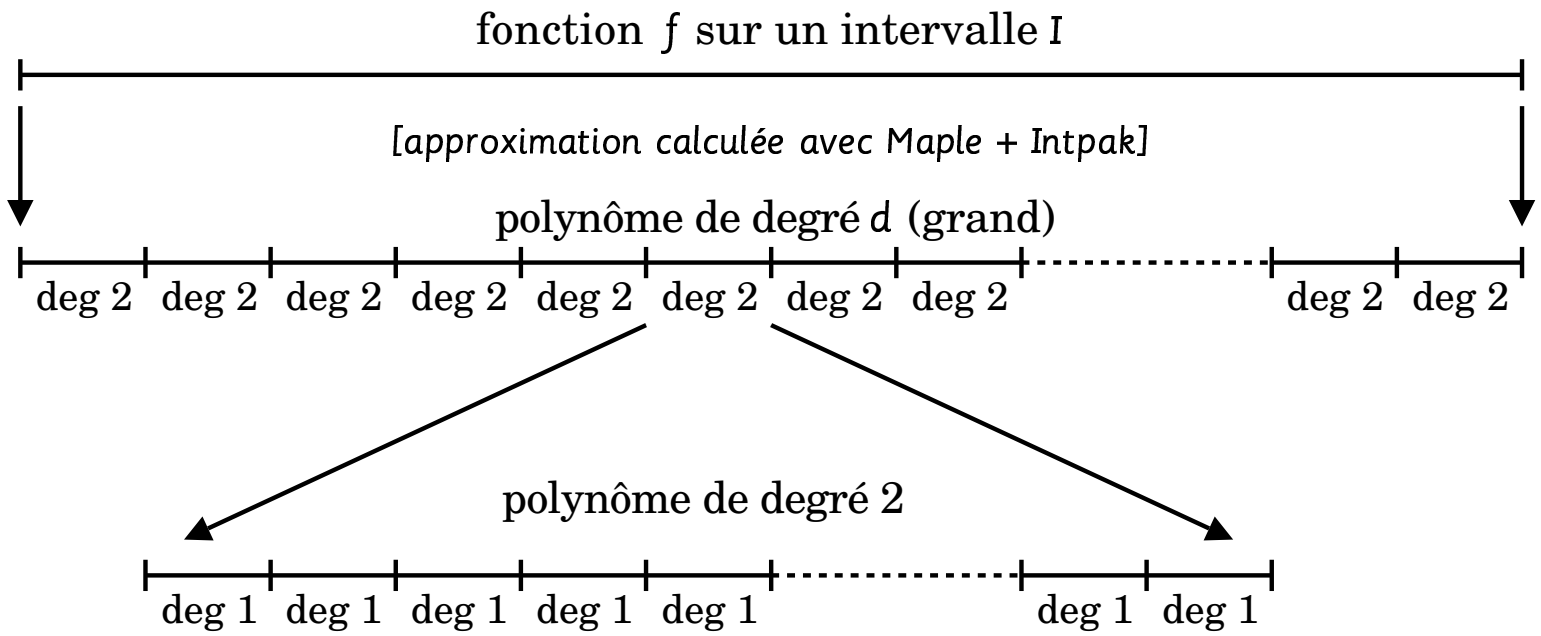
Avantages : rapide, calculs modulo $2^p$ possibles, peu de mémoire.

## 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
$\rightarrow$ approximations hiérarchiques.

# Approximations hiérarchiques




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

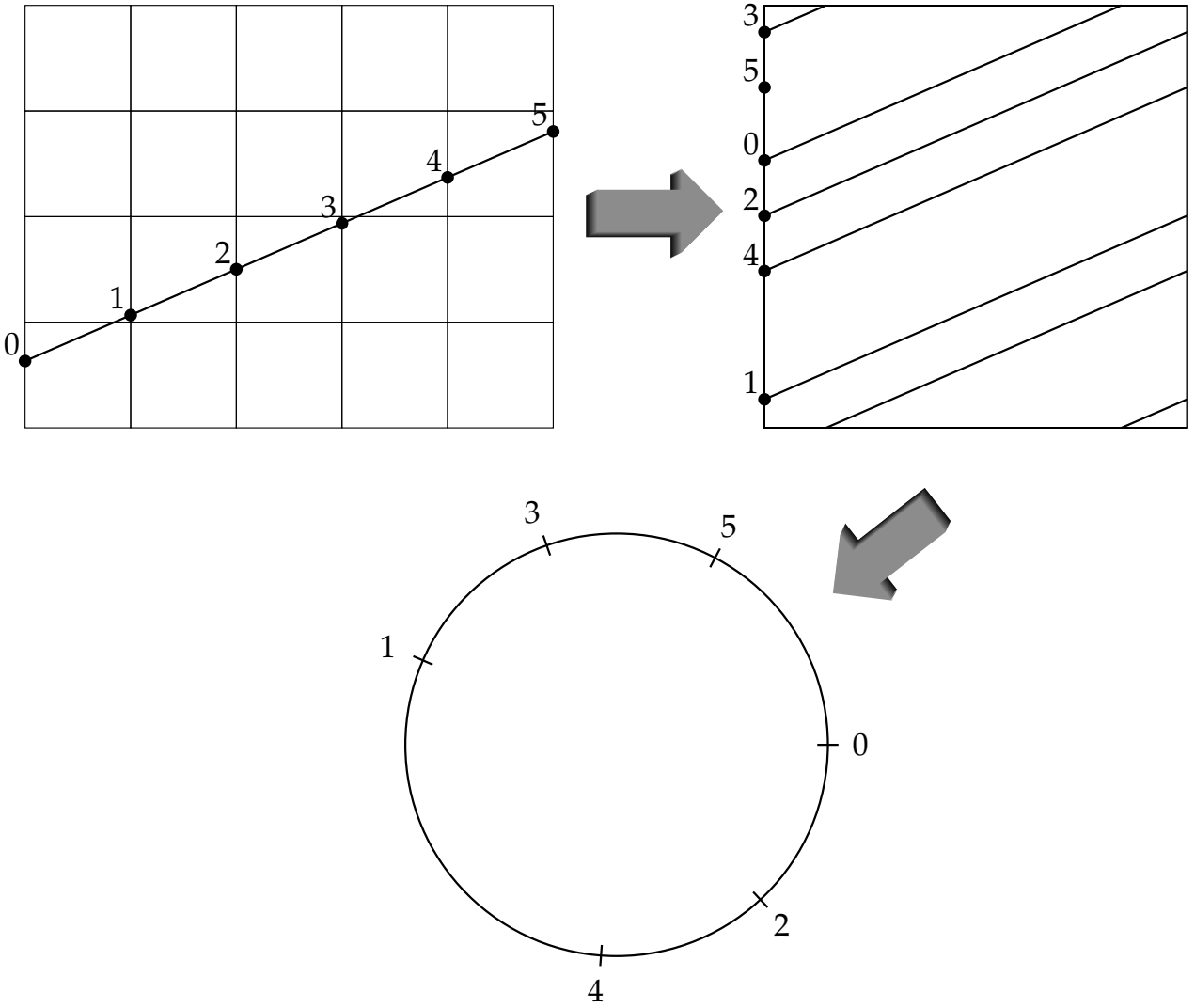
## Distance entre un segment et $\mathbb{Z}^2$

Dans chaque intervalle :
- $f$ approchée par un polynôme de degré $1 \rightarrow$ segment.
- Multiplication des coordonnées par des puissances de 2 $\rightarrow$ grille $= \mathbb{Z}^2$.

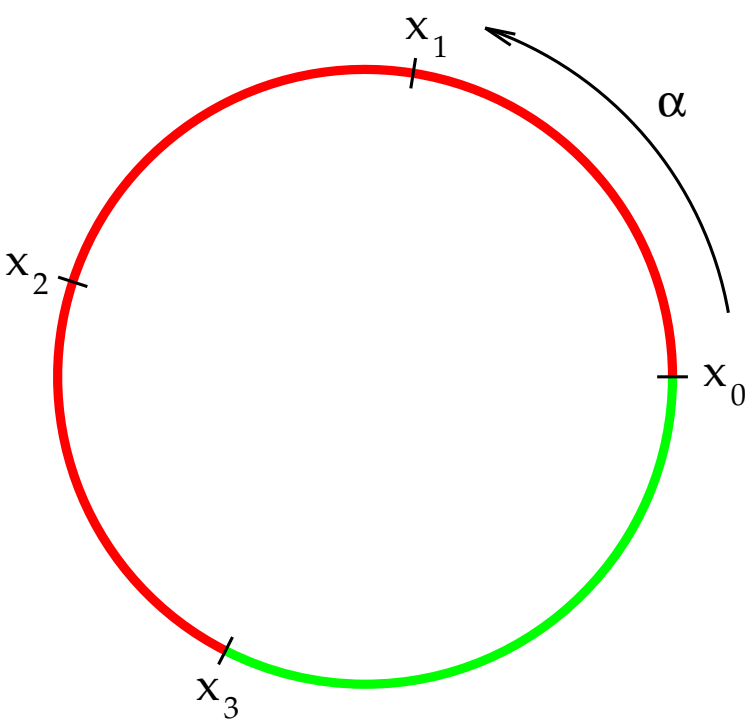
$\rightarrow$ distance entre un segment et $\mathbb{Z}^2$ ?

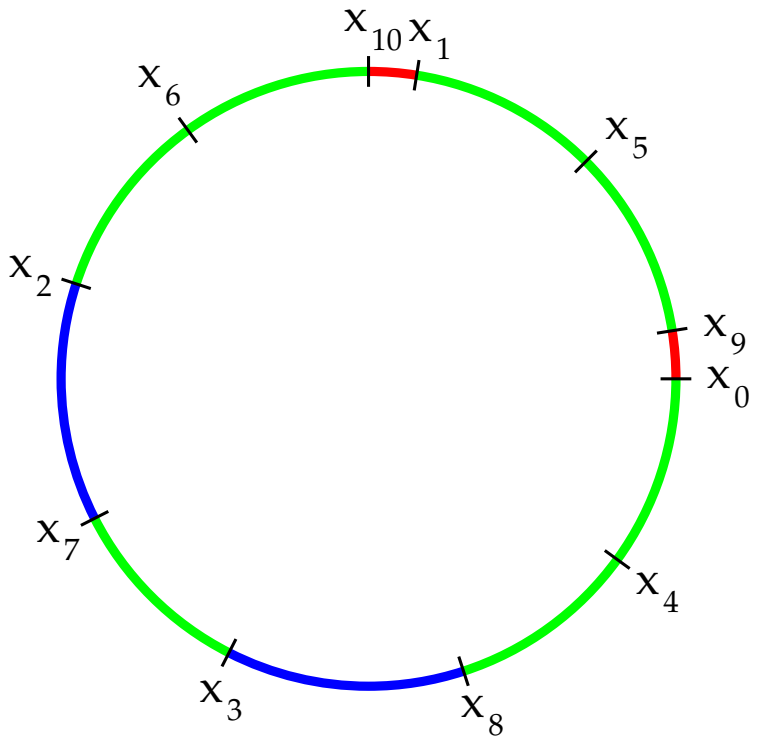
Nous proposons un algorithme beaucoup plus rapide que l'algorithme naïf, mais il ne donne qu'une minoration de la distance – suffisant pour nous.

Relié au théorème des 3 distances...


0
1
2
3
4
5
3
5
0
2
4
1
3
5
1
0
2
4

# Construction des points



## Théorème des trois distances

À tout moment durant ce processus de construction (i.e. pour un $n$ donné, quand les points $x_0$, $x_1$, $x_2$, ..., $x_n$ 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 $\alpha$.

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 $x \in [\![0, N-1]\!]$. 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** $(u + v \geqslant N)$ **fin**
    - $y = y - x;\ u = u + v;$
  - **si** $(u + v \geqslant N)$ **fin**
  - $x = x - y;\ v = v + u;$
- **sinon**
  - $d = d - x;$
  - **tant que** $(y < x)$
    - **si** $(u + v \geqslant N)$ **fin**
    - $x = x - y;\ v = v + u;$
  - **si** $(u + v \geqslant N)$ **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 $d_i$ ($\sim$ 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 $\sim 4$ 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
$\rightarrow$ 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 exp, $m \geqslant 111$

Pour $|x| \geqslant 2^{-32}$ :

| E | mantisse | R | m |
|---|---|---|---|
| 5 | $-1.00010010110100110001101000100000111110110011100010 11$ | N | **112** |
| $-13$ | $-1.10100010111111101111111011111101010110000000110111 11$ | D | **111** |
| $-27$ | $-1.11101101001100011000111011111011011000100111111010 10$ | D | **113** |
| $-29$ | $-1.00110100011101011010110000000101110011101010110101 11$ | D | **111** |
| $-32$ | $1.01111111111111100111111111111111011100000000000100 100$ | D | **111** |
| $-32$ | $1.10000000000000010111111111111111011011111111111011 100$ | D | **111** |
| $-31$ | $1.10011110100111001011101111111101011000001000000010 11$ | N | **111** |
| 2 | $1.10000011110101001011110011011110101110110011111101 00$ | D | **111** |

Sinon, le pire cas est :

| E | mantisse | R | m |
|---|---|---|---|
| $-53$ | $1.11111111111111111111111111111111111111111111111111 1111$ | D | **158** |

# Résultats pour log, $m \geqslant 115$

| E | mantisse | R | m |
|---|---|---|---|
| 86 | 1.10010001111011000100010000010010110000110100010011 11 | D | **115** |
| 245 | 1.11001001000010000001000011010011010101000110000110 00 | D | **117** |
| 656 | 1.10000110011100001101111000001011011010001100101011 01 | D | **116** |
| 678 | 1.01100010101010001000011000010011011000101001101101 10 | D | **118** |
| 732 | 1.11111101000101011101101010011011001110001100110010 | N | **115** |
| 772 | 1.01111011000111011001011111001001000001010011000010 1 | D | **115** |

Pires cas pour $x < 1$ :

| E | mantisse | R | m |
|---|---|---|---|
| $-509$ | 1.11101010011100011101100001011100111011100000001000 00 | D | **114** |
| $-384$ | 1.10010100011101101110001100000100110011010111110001 11 | N | **114** |
| $-232$ | 1.00100110111010011100010011010011001001111001011000 00 | D | **114** |
| $-35$ | 1.01100001001110010101010111011100100000000101111100 0 | 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 $2^x$, $m \geqslant 111$

| E | mantisse | R | m |
|---|---|---|---|
| $-15$ | $-1.00101000011000111010101110101110101011110111110110010$ | D | **111** |
| $-20$ | $-1.01000001011011110110110001100100010001011010111001111$ | D | **111** |
| $-32$ | $-1.00000101010101100000000111001000101010110011111110001$ | D | **111** |
| $-33$ | $-1.00011000010110111000110110110110110101011000000011101$ | D | **111** |
| $-29$ | $1.01010110100011101000110011101100010011110110011011100$ | N | **111** |
| $-27$ | $1.00010010101100010100101000110001100011111001000000100$ | D | **112** |
| $-25$ | $1.10111111101110111101111001000100111011011111111000101$ | D | **113** |
| $-10$ | $1.11100100010110010110010100100110101111111001010011101$ | N | **113** |
| $-10$ | $1.11100111011000000100100100000111001100000110011111111$ | N | **111** |
| $-8$ | $1.11111001100110101111111011111010001100001101011001011$ | D | **111** |
| $-6$ | $1.10001111111010101000000111111011011101010001001011110$ | N | **111** |

# Résultats pour $\log_2(x)$, $m \geqslant 106$

| E | mantisse | R | m |
|---|---|---|---|
| 0 | 1.10110100111010111110010000001100100101011010000000001 | N | **107** |
| 1 | 1.00011011101000111001111111110010100011100011111010 10 | D | **106** |
| 2 | 1.00011011101000111001111111110010100011100011111010 10 | D | **107** |
| 2 | 1.10001001110110010100100010101001010011111110000 10111 | N | **106** |
| 4 | 1.00011011101000111001111111110010100011100011111010 10 | N | **108** |
| 16 | 1.10010101011010110110111100110100000111110101110100 00 | N | **106** |
| 64 | 1.01100001010101010101111101110101100010000101101101 00 | D | **106** |
| 128 | 1.01100001010101010101111101110101100010000101101101 00 | D | **107** |
| 128 | 1.11010011000010100100001101110111001111011101000110 11 | D | **106** |
| 256 | 1.01100001010101010101111101110101100010000101101101 00 | D | **108** |
| 256 | 1.11010011000010100100001101110111001111011101000110 11 | N | **107** |
| 512 | 1.01100001010101010101111101110101100010000101101101 00 | 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 $|x| \geqslant 2^{-32}$.

# Résultats : exp en simple précision, $m \geqslant 51$

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

| 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 |
|---|---|---|---|
| $-24$ | $1.11111111111111111111111$ | D | **71** |

## Résultats : log en simple précision, $m \geqslant 55$

| E | mantisse | R | m |
|---|---|---|---|
| $-66$ | 1.00010000100010100101101 | D | **57** |
| $-65$ | 1.00100010110101010111000 | N | **55** |
| 3 | 1.00101111000111111101011 | 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 : $19 \leqslant n \leqslant 58$. Pires cas :

$n = 21$ :
$f(0.11010010001000101110 0) = 1.01111011111100010100\ 1\ 1^{29}\ 0110...$

$n = 21$ :
$g(1.01111011111100010101) = 0.11010010001000101101 1\ 1^{30}\ 0101...$

$n = 20$ :
$f(0.11010010001000101110) = 1.01111011111100010100\ 0\ 1^{30}\ 0110...$

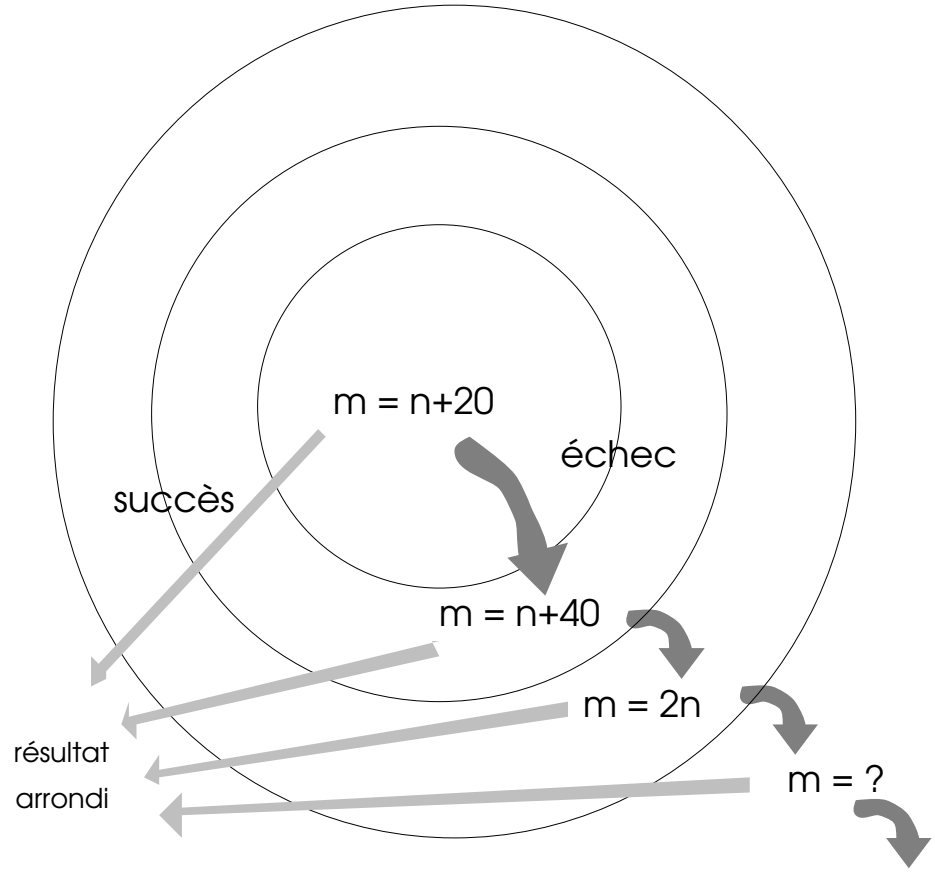
Correspondent à : $1556245 \times 430359^2 = 2^{58} + 101$

## Nos résultats et hypothèses probabilistes

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

| $k_0$ | 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 $m \geqslant m_0$.
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.
$\rightarrow$ **Nous nous sommes concentrés sur les pires cas.**
La stratégie de Ziv *n'est pas* (encore) utilisée.

Réduction d'argument $\rightarrow |x| \leqslant \frac{1}{2}$.

Si $|x| < 2^{-40}$, algorithme spécial à base de tables : $2^x \approx 1 + x.\log(2)$ 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 $\rightarrow$ 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 |

$\rightarrow$ 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/