L’arithmétique des ordinateurs

- Algorithmes pour les opérateurs arith.
- formats de représentation des nombres
- rapidité
- fiabilité
- précision

Exercice

- Diviser MMCCCXCIII par CLXXXII
- généralisation: proposer un algorithme de division

Attention: obligation de se mettre dans la peau de quelqu'un qui n'a aucune idée de ce que peut-être un système « de position ».

Deux ou trois choses amusantes

- Calculette de Windows 3.1
  - Demander 2.01-2.00
  - on obtient **0.0**
- Excel versions 3.0 à 7.0 (corrigé depuis)
  - taper **1.40737488355328** , avec une virgule comme séparateur si v.f.
  - on voit apparaître **0.64**

Ces 2 problèmes: conversion base 10 $\rightarrow$ base 2

X en base 10, $C_i = 2^i$ précalculés en base 10, $C_n \leq X < C_{n+1}$

$y_n = 1$; $X \leftarrow X - C_n$

pour i allant de n-1 à 0 faire

si $X \geq C_i$ alors $y_i = 1$; $X \leftarrow X - C_i$ sinon $y_i = 0$

$i = 0$; tant que $X \neq 0$ faire

$y_i = X \bmod 2$; $X := X \operatorname{div} 2$

$i = i+1$

D'autres choses amusantes

- Sur certains Cray, multiplier un nombre représentable par 1 conduit parfois à un dépassement de capacité !
- « Bug » du Pentium: jusqu 'à 3 chiffres significatifs seulement sur certaines divisions. Erreur dans *l'algorithme*, et non pas dans l'implantation.

$$8391667/12582905 = 0.666869455\ldots$$

Avec le logiciel Maple

- Version 6.0, si on entre

  2147483648041364781964379 4

  la quantité affichée et mémorisée sera

  4136478196437 90)+'--.(--.(
- Version 7.0
- (1001!)/ (1000 !) donnera 1

Pas seulement applications scientifiques

- 1982: la bourse de Vancouver a introduit une nouvelle valeur, montant 1000.
- Après chaque transaction, recalcul et troncature après 3ème chiffre fractionnaire (décimal)
- après 22 mois d 'exercice, valeur calculée = 524.881; valeur correcte = 1098.811

cause: biais. Avec un tel système d'arrondi délirant, toutes les erreurs sont de même signe (aucune compensation)

Mauvaise gestion des « exceptions »

- Premier lancement d'Ariane 5 : overflow;
- navire lance-missile USS Yorktown (1998): un membre d'équipage a par erreur entré un 0 comme donnée $\Rightarrow$ division par 0.

  Cascade d'erreurs qui a fini par bloquer le système de propulsion du bateau (*Scientific American*, Novembre 1998)

Erreurs de spécification

- Sonde *Mars Climate Orbiter*: écrasée sur Mars en septembre 1999. 2 équipes de conception des logiciels:
  - l'une supposait que l'unité de mesure était le mètre,
  - l'autre que c'était le pied .

Le banquier infernal

- Je place e -1 = 1.718281828459045… F
- 1ère année: on multiplie mon capital par 1 et on prélève 1 franc pour frais de gestion;
- 2ème année: on multiplie mon capital par 2 et on prélève 1 franc pour frais de gestion;
- …
- $n$ème année: on multiplie mon capital par n et on prélève 1 franc pour frais de gestion;

Avoir au bout de 25 ans ?

Les réponses

- Ma calculette (Casio): **-140302545600000 F**
- un ordinateur xxx: **+4645987753 F**
- Le vrai résultat **Environ 4 centimes**

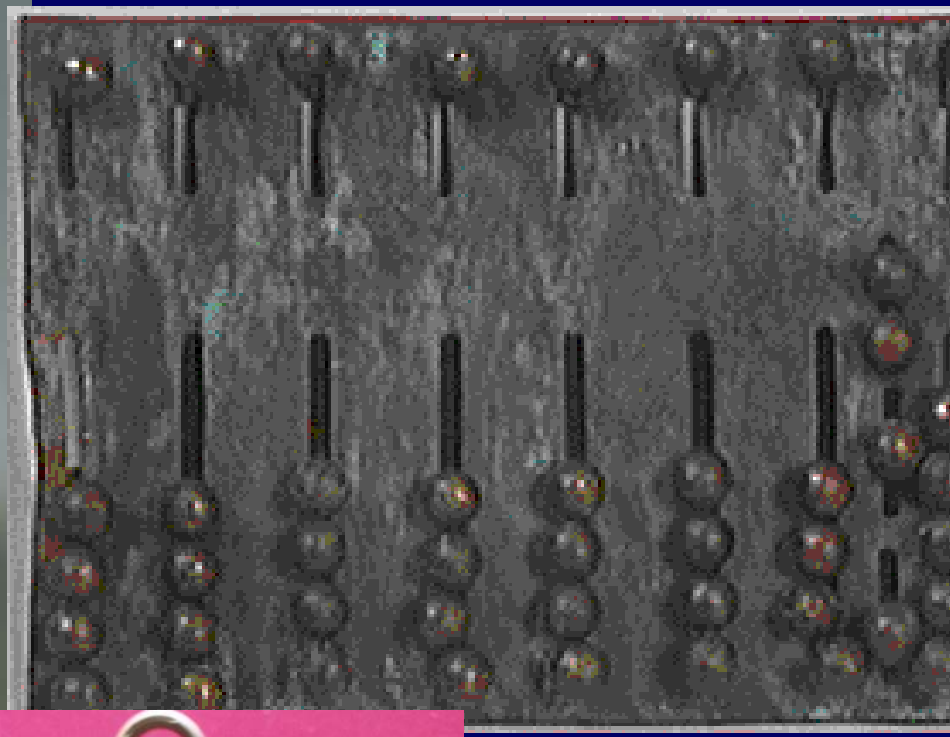
- Conclusion **Ne pas faire confiance à son ordinateur**

  Ne pas faire confiance à son banquier

Systèmes de numération

- « 1, 2, beaucoup »: estimations grossières
- il y a 6000 ans (Sumer): systèmes exacts (mémorisation, pas calcul)
- il y a 4000 ans (Babylone): systèmes « de position » (base 60). Calcul à la main.
- Notre système de base 10: Inde $\rightarrow$ monde arabe $\rightarrow$ Europe (990: Gerbert d'Aurillac)

Question: notre système (base 2 ou 10) est-il adapté au calcul *automatique* ?



Systèmes d'Avizienis

- Base $B > 2$, chiffres $-a, \ldots, +a$
- $2a + 1 \geq B$: on peut représenter tous les entiers
- exemple: $B = 10$, $a = 5$,

$$1247 = 125(-3) = 13(-5)(-3)$$

- systèmes redondants

Algorithme d'Avizienis (1961)

- Base $B > 2$, chiffres $-a, \ldots, +a$
- $a \leq B-1$, $2a \geq B+1$ ($\Rightarrow$ ne marche pas en base 2)
- Algorithme d'Avizienis

$$t_{i+1} = \begin{cases} -1 \text{ si } x_i + y_i \leq -a \\ +1 \text{ si } x_i + y_i \geq +a \\ 0 \text{ sinon} \end{cases}$$

$$w_i = x_i + y_i - Bt_{i+1}$$

$$s_i = w_i + t_i$$

B=10, a=6

x_i	1	-5	3	-2	6
y_i	0	1	4	-5	-2
t_{i+1}	0	0	1	-1	0
w_i	1	-4	-3	3	4
s_i	1	-3	-4	3	4

Conséquence

- Addition « totalement parallèle »
- temps de calcul pour nombres de *n* chiffres: environ le temps de l'algorithme usuel pour nombres de 2 chiffres
- nécessité de conversion, + de place mémoire
- essentiellement utilisé dans des opérateurs « boîte noire » (multiplieurs, diviseurs, fonctions élémentaires)

Généralités sur la virgule flottante

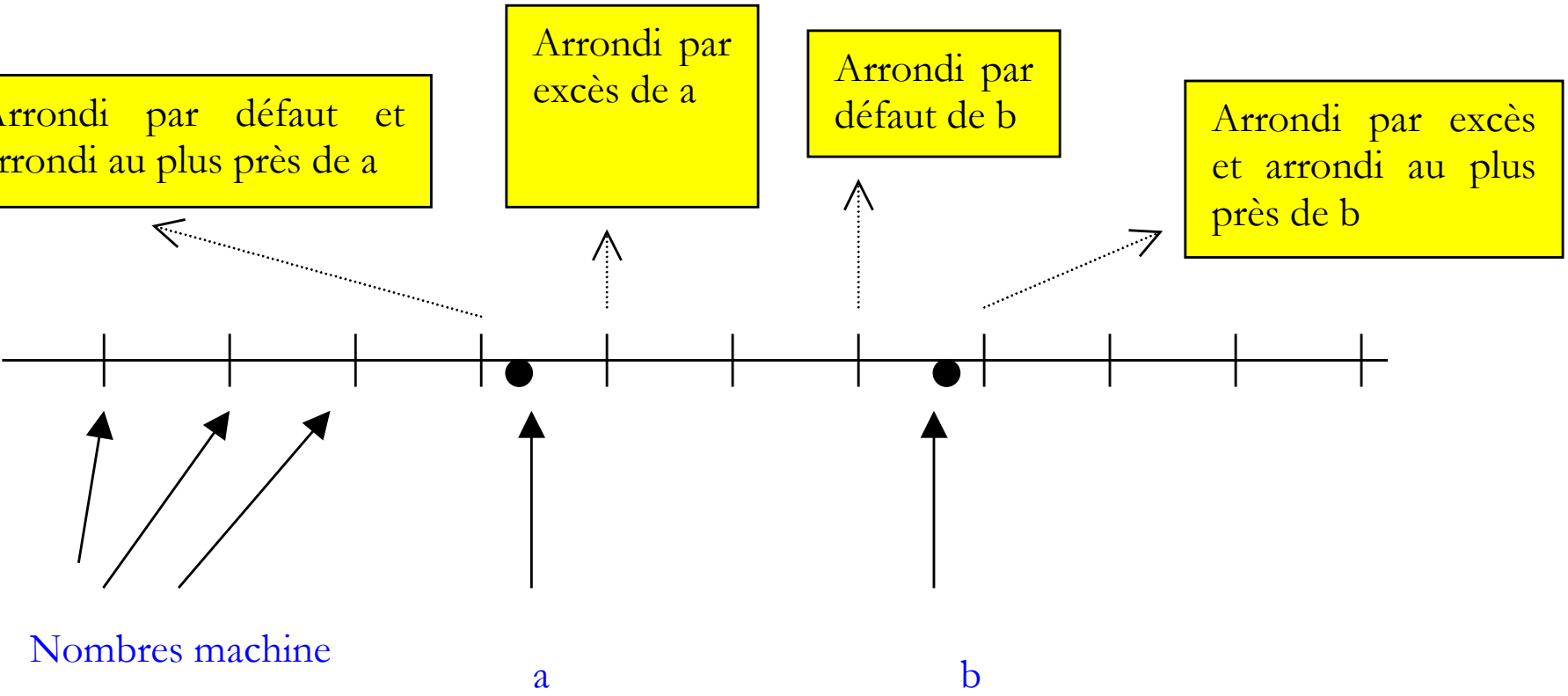
$$(-1)^s x_0 . x_1 x_2 \cdots x_{n-1} \times B^e$$

- En pratique $B = 2$ (calculettes, Maple: 10, certaines machines IBM de gestion: 16)
- e est l'exposant, $m = x_0.x_1x_2\ldots$ est la mantisse
- mantisse *normalisée* : $x_0 \neq 0 \Rightarrow m$ entre 1 et B (demande codage spécial pour $x = 0$)
- en base 2, avec des mantisses normalisées, x_0 vaut forcément 1 : on ne le mémorise pas (« 1 *implicite* »)

Modes d'arrondi (norme IEEE 754)

- Résultat d 'une opération: doit en général être arrondi
- Arrondi « exact », ou « correct »: déterministe, choix parmi 4 modes
  - Au plus proche, avec choix de la valeur paire si on est exactement entre 2 nombres machine
  - Vers $+\infty$
  - Vers $-\infty$
  - Vers zéro

Le système doit se comporter comme si le calcul était fait avec une précision « infinie », puis arrondi.


rrondi par défaut et
rrondi au plus près de a
Arrondi par excès de a
Arrondi par défaut de b
Arrondi par excès et arrondi au plus près de b
Nombres machine
a
b

Propriétés de l'arrondi correct

- Le comportement du système est prévisible et reproductible. On peut élaborer des *algorithmes* et des *preuves*.
- Réduction des coûts de portage des logiciels
- Arithmétique d'intervalles, arithmétique stochastique
- Semble difficile à satisfaire pour les fonctions élémentaires.

Exemple 1 (arrondi au + près)

$$z = x \Big/ \sqrt{x^2 + y^2}$$

$$t = \sqrt{1 - z^2}$$

Problème : si la valeur exacte de z est très proche de 1, peut-on avoir une valeur calculée > 1, ce qui rendrait le 2ème calcul invalide ?

Exemple 2: un algorithme étrange

A, B: **réels virgule flottante**

A ← 1; B ← 1;

tant que ((A+1)-A)-1 = 0 faire A ← 2×A;

tant que ((A+B)-A)-B ≠ 0 faire B ← B+1;

imprimer (B);

Résultat: base de la représentation interne de votre système (2, 10, 16, le reste étant rarissime)

Traitement des exceptions

- Deux infinis $\Leftrightarrow$ deux zéros, avec comportement « intuitif » sauf $\sqrt{(-0)} = -0$;
- Not A Number (NaN): résultats invalides ou indéterminés:
  - $(\pm\infty) \pm (\pm\infty)$ lorsque les signes donnent une indétermination ;
  - $0 * \pm\infty$, $0 / 0$ et $\pm\infty / \pm\infty$
  - Le reste de la division de x par 0 ou le reste de la division de l'infini par x;
  - La racine carrée d'un nombre négatif $\neq 0$.

NaN (suite)

- Les NaN se propagent (très dangereux pour embarqué);
- si x est un NaN, $x = x$ est faux, tandis que x != x est vrai
- si x ou y est un NaN, les tests $x < y$, $x <= y$, $x == y$, $x >= y$ et $x > y$ doivent retourner " faux " ⇨ éventuels problèmes d 'incohérence.

Nombres dénormalisés (std IEEE)

Lorsque l'exposant est le + petit possible, on autorise la mantisse a être non normalisée (i.e. à avoir son premier chiffre nul)

- Si $\frac{1}{2} \leq y/x \leq 2$, alors le calcul de $x-y$ se fait sans erreur
- $x \neq y \Leftrightarrow$ le calcul $x-y$ donne un résultat non nul

Conversions

- Nombre base 10 d'au + 6 chiffres converti en simple précision, puis converti à nouveau en base 10 avec même nombre de chiffres;
- nombre s.p. converti en chaîne décimale d'au moins 9 chiffres, et chaîne convertie à nouveau en s.p.;
- nombre base 10 d'au + 15 chiffres converti en double précision, puis converti à nouveau en base 10 avec même nombre de chiffres;
- nombre d.p. converti en chaîne décimale d'au moins 17 chiffres, et chaîne convertie à nouveau en d.p.

Tous ces cas: pas d'erreur.

Influence du compilateur

- Norme ANSI: il est théoriquement interdit de changer l'ordre d'évaluation des opérations;
- problèmes avec des constantes: $0.1 \times x$
- détection de la dépendance entre changement de mode d'arrondi et opération (DEC Alpha)

Le tableau de la honte

Système	sin (10^{22})
exact	-0.852200849767…
HP 48 GX	-0.8522008497677
HP 700	0.0
IBM 3090/600S-VF AIX 370	0.0
Matlab V.4.2c.1 pour SPARC	-0.8522
Matlab V.4.2c.1 macintosh	0.8740
Silicon graphics Indy	0.87402806
Sharp EL5806	-0.090748172
DEC Station 3100	NaN

Dilemme du fabricant de tables

- Problème : arrondi correct des fonctions algébriques et élémentaires (sin, cos, log, etc.);
- souvent, on ne peut qu'approcher ces fonctions;
- pour une arithmétique à n bits de mantisse, on calcule une approximation de précision relative 2^{-m}, où $m > n$. On arrondit l'approximation;

question : pour quelle valeur de m est-on certain que ceci est équivalent à arrondir le résultat exact ?

Exemple: mantisses de 24 bits

- Le nombre « simple précision » qui s ’écrit en base 2 1.11001100111000110011001 vaut :

$$x = \frac{15102361}{8388608}$$

- son exponentielle est:

110.00001101001111010011 0 11111111111111111111111 01100100001…

Résultats de théorie des nombres

- Baker (1975)
- $\alpha = p/q$, $\beta = r/s$, $p,q,r,s < 2^n$,
- $C = 16^{200}$

$$|\alpha - \log \beta| > \left(n 2^n\right)^{-Cn \log n}$$

Application: pour calculer ln and exp en double précision (n=53), il suffit de calculer une approx. intermédiaire bonne sur environ

10^{244} bits

Théorème (Nesterenko-Waldschmidt, 95)

Rationnel p/q, $q > 0$ et pgcd$(p,q)=1$ ⇨ $H(p/q) = \max(p,q)$

Soient $\alpha, \beta \in \mathbb{K}$, soient A, B et E satisfaisant

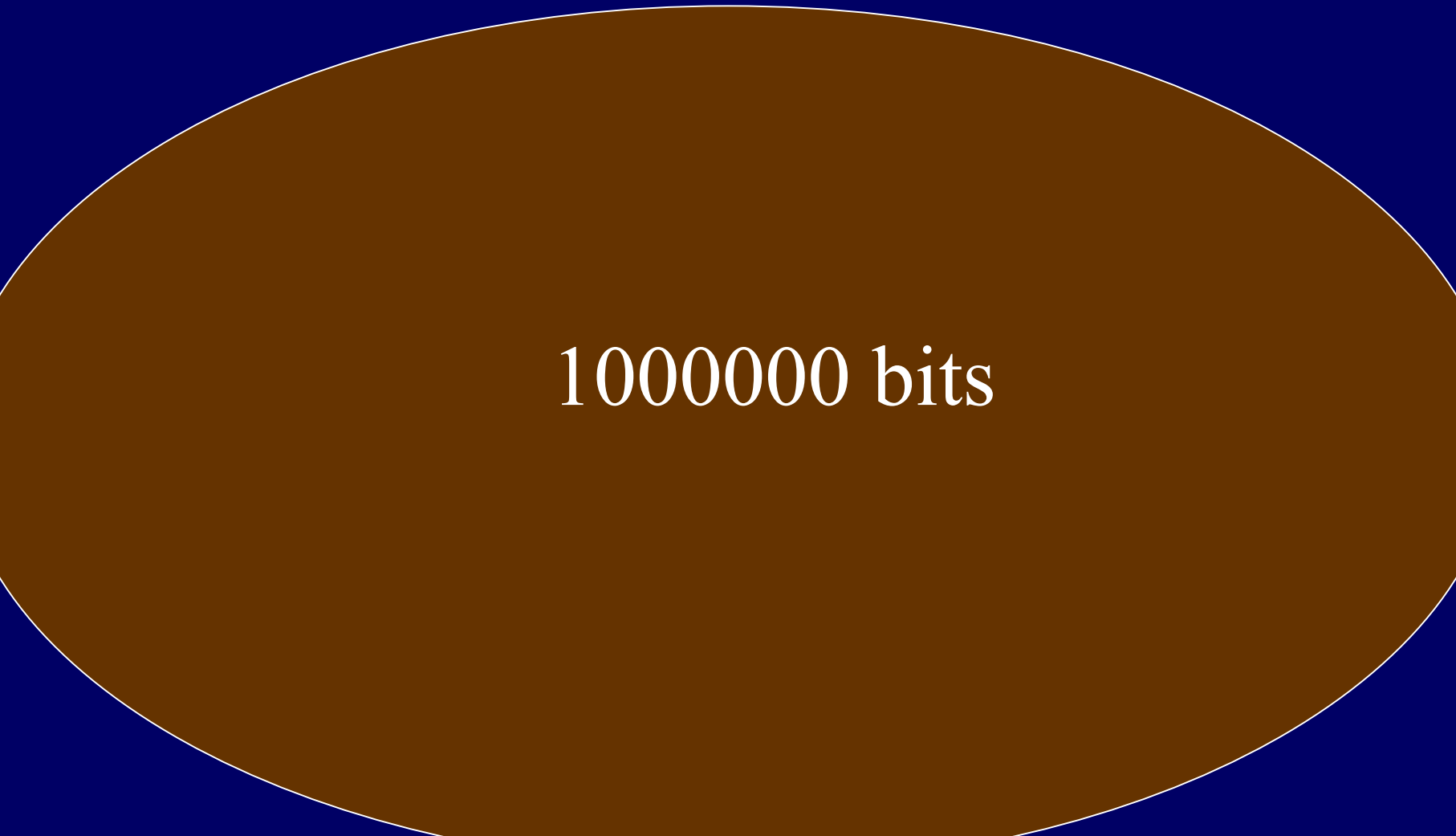
$A \geq \max(H(\alpha),e)$ $\quad$ $B \geq H(\beta)$ $\quad$ $E \geq e$

on a:

$$\left|e^{\beta}-\alpha\right| \geq \exp\left[\begin{array}{l} -211\times\left(\log B+\log\log A+2\log\left(E\max(1,\left|\beta\right|\right)+10\right) \\ \times\left(\log A+2E\left|\beta\right|+6\log E\right)\times\left(3.3\cdot\log 2+\log E\right)\times\left(\log E\right)^{-2}\end{array}\right]$$

Donne $m \approx 10^6$ pour l'exponentielle et le log en double précision

Stratégie de Ziv



Calculer sur 1000000 bits

$$\begin{cases} a_{n+1} = \dfrac{a_n + b_n}{2} \\ b_{n+1} = \sqrt{a_n b_n} \end{cases}$$

Converge *quadratiquement*

limite $A(a_0,b_0)$: *moyenne arithmético-géométrique* (Gauss)

$$A(1,x) = \pi / 2F(x)$$

$$F(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-(1-x^2)\sin^2\theta}}$$

$$F(4/s) = \log(s) + \frac{4\log(s)-4}{s^2} + \frac{36\log(s)-42}{s^4} + \ldots$$

Quelques exemples de « pires cas »

- $\ln(1.01100010101010001000011000010011011000101001101101101 \times 2^{678}) = 111010110.010001111001111010111010011111001001011100010\ 0^{65}\ 11\ldots$
- $2^{(1.11100100010110010110010100100110101111111001010010011 \times 2^{-10})} = 1.000000000101001111111100001011011000010101101010011\ 0\ 1^{59}\ 0100\ldots$
- $\log_2(1.01100001010101010101111101110101100010000101101101001 \times 2^{512}) = 1000000000.011101110000001011010000011110100110111001\ 1^{56}\ 0011$

Arith. Flottante sur Internet

- W. Kahan: `HTTP.CS.Berkeley.EDU/~wkahan/`
- D. Hough: `www.validgh.com/`
- Interval computations: `www.cs.utep.edu/interval-comp/`
- J. Harrison: `www.cl.cam.ac.uk/users/jrh/fpv/index.html`
- `www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm`
- Loria: `cch.loria.fr/documentation/IEEE754/`

Algorithmes à base d'additions et décalages pour l'évaluation des fonctions élémentaires

Jean-Michel Muller

Janvier 2002

Introduction

- algorithmes ne nécessitant pas de $\times$ ou $\div$;
- le plus connu : CORDIC (Volder 59, Walther 71) : 1ères implantations dans HP 35, Intel 8087, Motorola 68881 ;
- introduction à la boulangerie théorique $\Rightarrow$ lien avec les fonctions élémentaires ;
- algorithmes simples ;
- algorithmes plus efficaces utilisant la redondance.

Pesons du pain

Balance de Roberval, poids $w_0, w_1, w_2, \ldots$ vérifiant :

– $\forall i, w_i > 0$;

– la suite w_i décroit et sa somme est finie ;

– $\forall i, w_i \leq \sum_{k=i+1}^{\infty} w_k$.

première règle : soit les poids ne sont pas utilisés, soit on les place dans le plateau qui ne contient pas le pain.

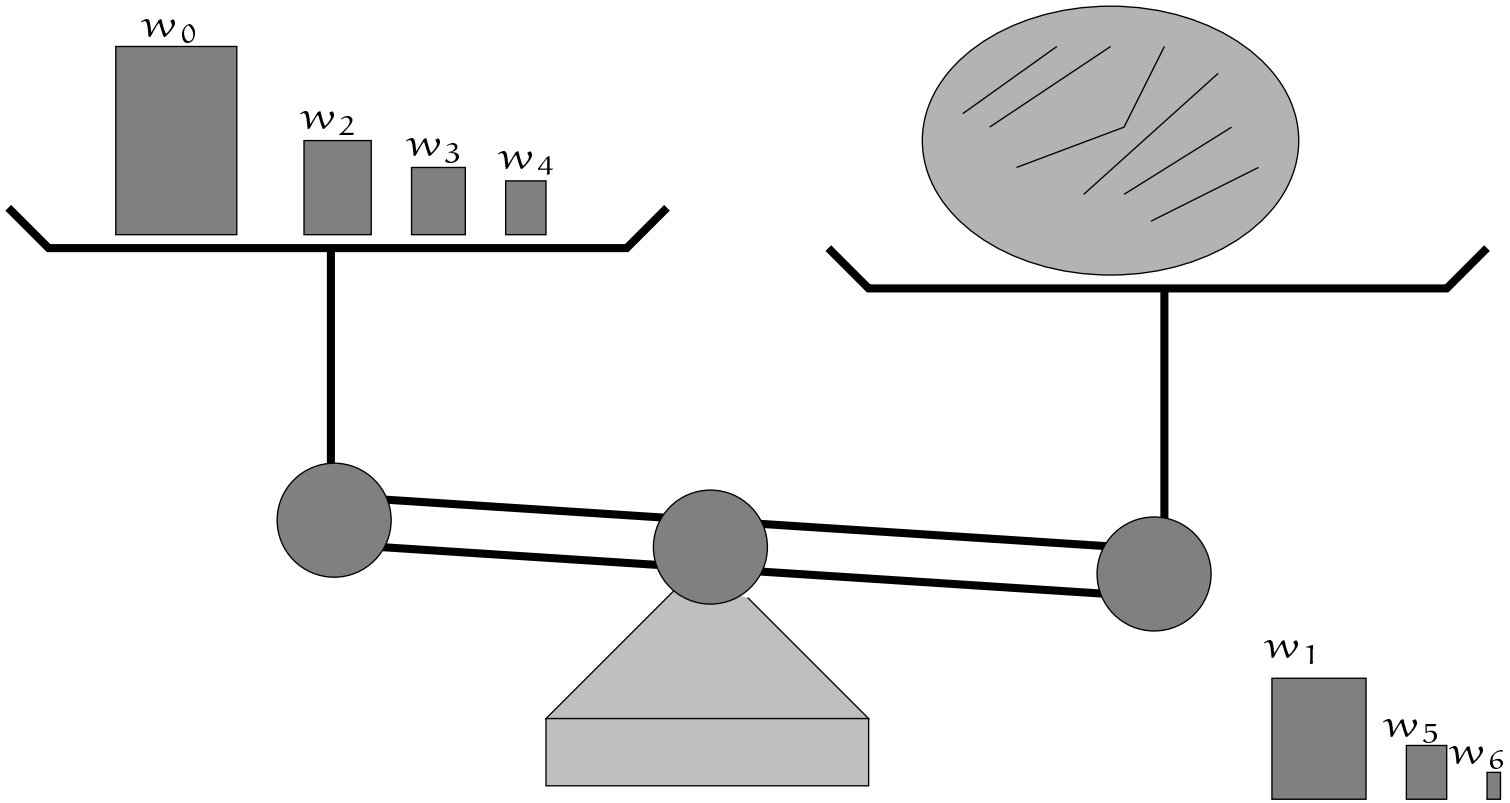


FIG. 1 – Décomposition «restaurante»

Théorème 1 (Décomposition restaurante) *Soit* (w_n) *une suite décroissante de réels* > 0 *telle que* $\sum_{i=0}^{\infty} w_i < \infty$. *Si*

$$\forall n, w_n \leq \sum_{k=n+1}^{\infty} w_k \tag{1}$$

alors $\forall t \in [0, \sum_{k=0}^{\infty} w_k]$, *les suites* (t_n) *et* (d_n) *définies par*

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n w_n \\ d_n &= \begin{cases} 1 & \textit{si } t_n + w_n \leq t \\ 0 & \textit{sinon} \end{cases} \end{aligned} \tag{2}$$

vérifient $t = \sum_{n=0}^{\infty} d_n w_n = \lim_{n \to \infty} t_n$.

Démonstration

Par récurrence, $\forall n,\ 0 \leq t - t_n \leq \sum_{k=n}^{\infty} w_k$.

Théorème 2 *La suite* $(\ln(1 + 2^{-n}))$ *satisfait aux conditions du théorème 1.*

De la boulangerie théorique à l'exponentielle

$w_n = \ln(1+2^{-n})$. Soit $t \in [0, \sum_{k=0}^{\infty} w_k] = [0, 1.56202\cdots]$.
Théorème 1 $\Rightarrow (t_n)\&(d_n)$ définies par

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n \ln(1+2^{-n}) \end{aligned} \qquad d_n = \begin{cases} 1 & \text{si } t_n + \ln(1+2^{-n}) \le t \\ 0 & \text{sinon} \end{cases}$$

vérifient $t = \sum_{n=0}^{\infty} d_n \ln(1+2^{-n}) = \lim_{n\to\infty} t_n$.

Définissons $E_n = e^{t_n}$

- $t_0 = 0 \Rightarrow E_0 = 1$.
- lorsque $t_{n+1} \neq t_n$ (i.e., lorsque $d_n = 1$),
  $t_{n+1} = t_n + \ln(1+2^{-n})$. Pour obtenir E_{n+1}, E_n doit être multiplié par $\exp\ln(1+2^{-n}) = (1+2^{-n})$.

Comme $t_n \to t$, $E_n \to e^t$.

Algorithme 1 (expo-1, entrées : t, N (nb de pas), sortie : E_N)

$t_0 = 0 \qquad E_0 = 1$; **construire t_n et E_n comme suit**

$$\begin{aligned} t_{n+1} &= t_n + \ln(1 + d_n 2^{-n}) \\ E_{n+1} &= E_n(1 + d_n 2^{-n}) = E_n + d_n E_n 2^{-n} \\ d_n &= \begin{cases} 1 & \textit{si } t_n + \ln(1 + 2^{-n}) \leq t \\ 0 & \textit{sinon.} \end{cases} \end{aligned} \tag{3}$$

- seulement $+$, et $\times$ par puissances de 2 ;
- base 2 : ces multiplications $\rightarrow$ décalages ;
- constantes $\ln(1 + 2^{-n})$ précalculées et mémorisées (n bits de précision $\Rightarrow \approx n$ constantes) ;
- remplacer $\ln(1 + 2^{-n})$ par $\log_a(1 + 2^{-n}) \longrightarrow$ algorithme pour a^t.

De l'exponentielle au logarithme

Calculer $\boxed{\ell = \ln(x)}$. On va d'abord supposer que ℓ est connu (!!!)
On calcule son exponentielle (je sais, c'est x) en utilisant :

$$\begin{array}{lcll} t_0 = 0 & & E_1 = 1 & \\ t_{n+1} & = & t_n + d_n \ln(1 + 2^{-n}) & (4) \\ E_{n+1} & = & E_n + d_n E_n 2^{-n} & \end{array}$$

avec $d_n = \begin{cases} 1 & \text{si } t_n + \ln(1 + 2^{-n}) \leq \ell \\ 0 & \text{sinon.} \end{cases}$, on a :

$\boxed{t_n \to \ell,\ E_n \to e^{\ell} = x}$

Inutilisable car nécessite ℓ...

«$E_n = \exp(t_n)$» invariant $\Rightarrow$ la comparaison

$$d_n = \begin{cases} 1 & \text{si } t_n + \ln(1+2^{-n}) \leq \ell \\ 0 & \text{sinon.} \end{cases}$$

peut être remplacée par

$$d_n = \begin{cases} 1 & \text{si } E_n \times (1+2^{-n}) \leq x \\ 0 & \text{sinon.} \end{cases}$$

$\Rightarrow$ même résultats, sans nécessiter la connaissance de ℓ.

Algorithme 2 (logarithme-1)

– **entrées :** x, n, *avec* $1 \leq x \leq \prod_{i=0}^{\infty}\left(1+2^{-i}\right) \approx 4.76$;

– **sortie :** $t_n \approx \ln x$.

$t_0 = 0, E_0 = 1$. **Construire** t_i **et** E_i **comme suit**

$$\begin{aligned} t_{i+1} &= t_i + \ln\left(1+d_i 2^{-i}\right) \\ E_{i+1} &= E_i\left(1+d_i 2^{-i}\right) = E_i + d_i E_i 2^{-i} \\ d_i &= \begin{cases} 1 & \textit{si } E_i + E_i 2^{-i} \leq x \\ 0 & \textit{sinon.} \end{cases} \end{aligned} \tag{5}$$

En remplaçant $\ln\left(1+2^{-n}\right)$ par $\log_a\left(1+2^{-n}\right) \rightarrow$ alg. pour $\log_a$.

Deuxième exercice

Même balance de Roberval & poids, mais maintenant il faut utiliser **tous les poids**. Par contre on peut les mettre dans les deux plateaux.

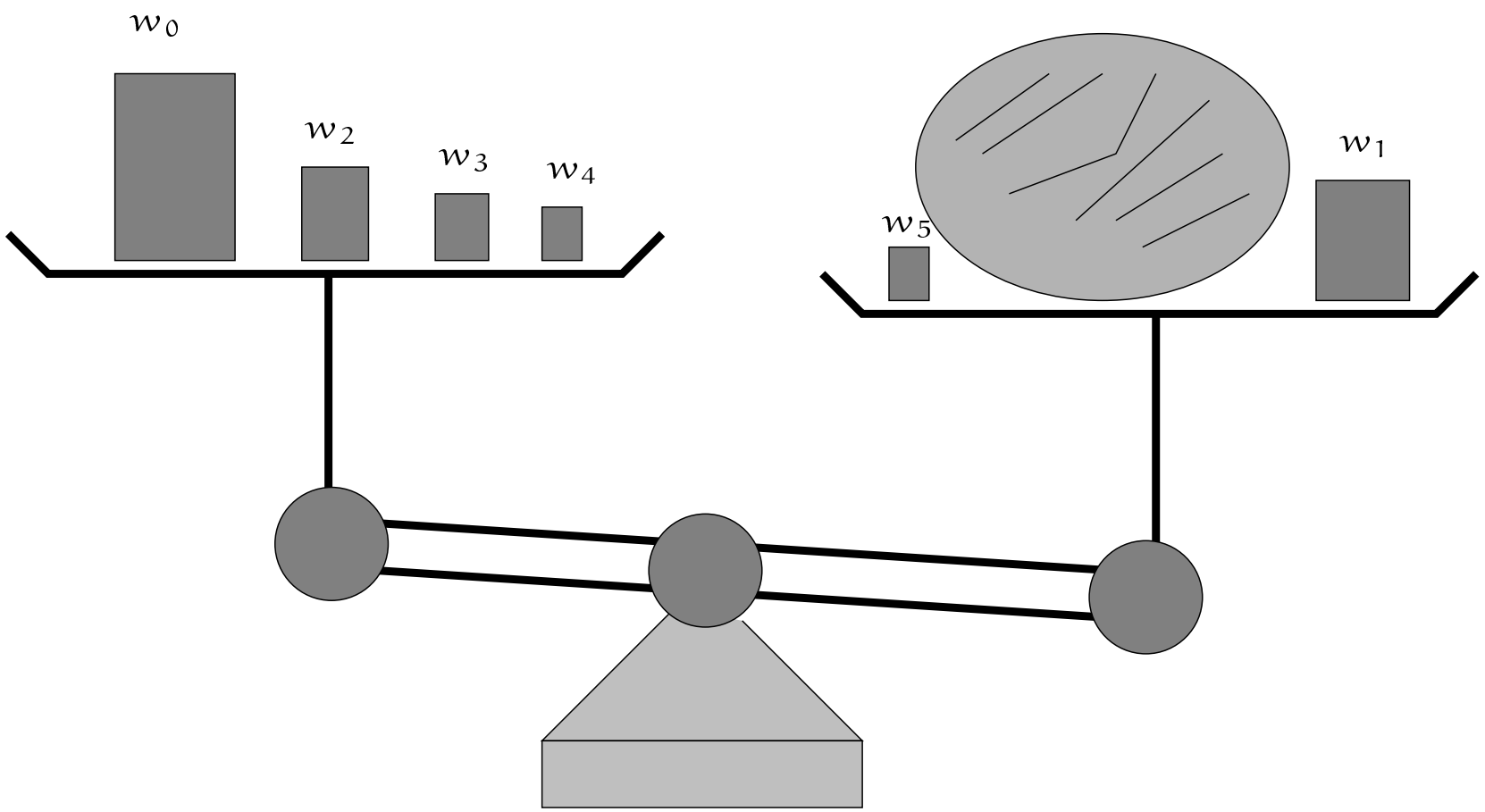


FIG. 2 – Décomposition «non restaurante»

Un autre algorithme glouton :

Théorème 3 (Décomposition «non restaurante») *Soit* (w_n) *une suite vérifiant les conditions du théorème 1.*
$\forall t \in [-\sum_{k=0}^{\infty} w_k, \sum_{k=0}^{\infty} w_k]$, *les suites* (t_n) *et* (d_n) *définies par*

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n w_n \\ d_n &= \begin{cases} 1 & \textit{si } t_n \leq t \\ -1 & \textit{sinon} \end{cases} \end{aligned} \tag{6}$$

vérifient $t = \sum_{n=0}^{\infty} d_n w_n = \lim_{n\to\infty} t_n$.

Théorème 4 *La suite* $(\arctan 2^{-n})$ *satisfait aux conditions des théorèmes 1 et 3*

Fonctions trigonométriques

- Décomposition non restaurante (poids sur les 2 plateaux)
- Suite $w_n = \arctan 2^{-n}$ (Satisfait aux conditions du théorème 3)
- décomposition
  $\Rightarrow \theta = \sum_{k=0}^{\infty} d_k w_k, \quad d_k = \pm 1, \quad w_k = \arctan 2^{-k}.$

Mode Rotation de CORDIC : effectuer une rotation d'angle θ comme composition de rotations d'angles $d_n w_n$. Partir de (x_0, y_0). Obtenir (x_{n+1}, y_{n+1}) à partir de (x_n, y_n) en effectuant une rotation d'angle $d_n w_n$. Donne

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n w_n \end{aligned} \qquad d_n = \begin{cases} 1 & \text{si } t_n \leq \theta \\ -1 & \text{sinon ;} \end{cases} \tag{7}$$

nième rotation

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} \cos(d_n w_n) & -\sin(d_n w_n) \\ \sin(d_n w_n) & \cos(d_n w_n) \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \quad (8)$$

$d_n = \pm 1 \Rightarrow \cos(d_n w_n) = \cos(w_n)$ et $\sin(d_n w_n) = d_n \sin(w_n)$. De plus, $\tan w_n = 2^{-n}$. Donc :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \cos(w_n) \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \quad (9)$$

Arithmétique de base 2 $\rightarrow$ toutes les opérations sont simples sauf multiplication par $\cos(w_n) = 1/\sqrt{1 + 2^{-2n}}$.

Règle d’Or

Quand on ne sait pas résoudre un problème, on l’ignore.

$\Rightarrow$ On ignore le problème et on effectue :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \qquad (10)$$

itération **CORDIC** de base. Au lieu d'une *rotation*, *similitude* d'angle w_n & rapport $1/\cos w_n = \sqrt{1+2^{-2n}}$.

Dernière modification $z_n = \theta - t_n$. Donne $z_0 = \theta$,
$z_{n+1} = z_n - d_n w_n$ et $d_n = 1$ si $z_n \geq 0$, -1 sinon.

$(x_n, y_n) \to$ résultat de la similitude d'angle θ & rapport
$K = 1.646760258121\cdots = \prod \sqrt{1+2^{-2i}}$ appliquée à (x_0, y_0).

Algorithme CORDIC (première version)

$$\begin{cases} x_{n+1} &= x_n - d_n y_n 2^{-n} \\ y_{n+1} &= y_n + d_n x_n 2^{-n} \\ z_{n+1} &= z_n - d_n \arctan 2^{-n}, \end{cases} \tag{11}$$

donne

$$\lim_{n\to\infty} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = K \times \begin{pmatrix} x_0 \cos z_0 - y_0 \sin z_0 \\ x_0 \sin z_0 + y_0 \cos z_0 \\ 0 \end{pmatrix} \tag{12}$$

Par exemple, $x_0 = 1/K$ et $y_0 = 0$ donnent $x_n \to \cos(\theta)$ et $y_n \to \sin(\theta)$.

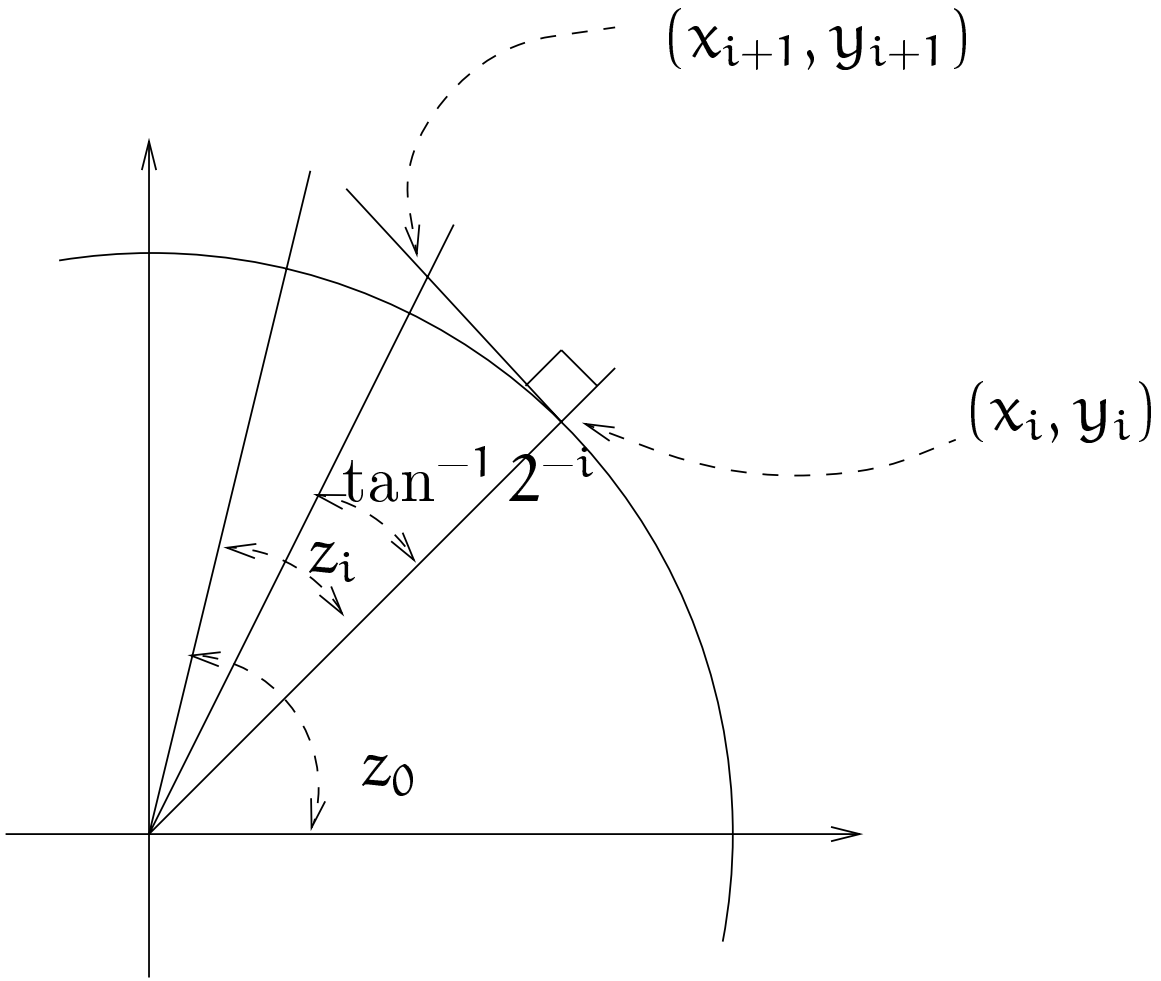


FIG. 3 – *Une itération de CORDIC.*

Algorithme CORDIC «généralisé»

Dû à John Walther. Implanté la première fois sur HP 35. Itération élémentaire :

$$\begin{cases} x_{n+1} &= x_n - m d_n y_n 2^{-\sigma(n)} \\ y_{n+1} &= y_n + d_n x_n 2^{-\sigma(n)} \\ z_{n+1} &= z_n - d_n w_{\sigma(n)}, \end{cases} \tag{13}$$

$m = 1$ et $\sigma(n) = n$ donnent l'algorithme précédent.

m et w_k	$d_n = \text{signe}(z_n)$ (Mode Rotation)	$d_n = -\text{signe}(y_n)$ (Mode «Vectoring»)
1 $\arctan 2^{-k}$	$x_n \to K\,(x_0 \cos z_0 - y_0 \sin z_0)$ $y_n \to K\,(y_0 \cos z_0 + x_0 \sin z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 + y_0^2}$ $y_n \to 0$ $z_n \to z_0 - \arctan \frac{y_0}{x_0}$
0 2^{-k}	$x_n \to x_0$ $y_n \to y_0 + x_0 z_0$ $z_n \to 0$	$x_n \to x_0$ $y_n \to 0$ $z_n \to z_0 - \frac{y_0}{x_0}$
-1 $\tanh^{-1} 2^{-k}$	$x_n \to K'\,(x_1 \cosh z_1 + y_1 \sinh z_1)$ $y_n \to K'\,(y_1 \cosh z_1 + x_1 \sinh z_1)$ $z_n \to 0$	$x_n \to K'\sqrt{x_1^2 - y_1^2}$ $y_n \to 0$ $z_n \to z_1 - \tanh^{-1} \frac{y_1}{x_1}$

TAB. 1 – *Fonctions calculables par CORDIC.*

Trigo ($m = 1$)	$\sigma(n) = n$
Linéaire ($m = 0$)	$\sigma(n) = n$
Hyperbolique ($m = -1$)	$\sigma(n) = n - k$ où k est le + grd entier tq $3^{k+1} + 2k - 1 \leq 2n$

TAB. 2 – *Valeur de* $\sigma(n)$