Arithmétique des Ordinateurs

Jean-Michel Muller

Journées INRIA FIABLE

22-24 septembre 1999

Quels problèmes ?

- Algorithmes et architectures pour les opérateurs
- formats de représentation des nombres
- utiliser au mieux une arithmétique existante
- validation

Validation en virgule flottante

- Validation de l'environnement: architecture, système, compilateur
- Validation des programmes
- Aspects spécifiques de la norme IEEE-754: précision, arrondis, exceptions,...

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**

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…

Propagations d'erreurs d'arrondi

$$v_{n+1} = 3 + \frac{98}{v_n} - \frac{300}{v_n v_{n-1}} + \frac{200}{v_n v_{n-1} v_{n-2}}$$

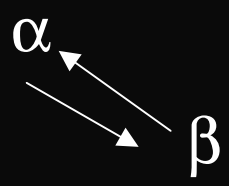
$$v_1 = 3/2$$

$$v_2 = 5/3$$

$$v_3 = 9/5$$

Convergence
« théorique » vers 2

N'importe quelle machine: « convergence » apparente vers un cycle



Les α et β dépendent de la machine, de la précision et du programme, mais ils ont tous en commun: αβ = 100

Généralités sur la virgule flottante

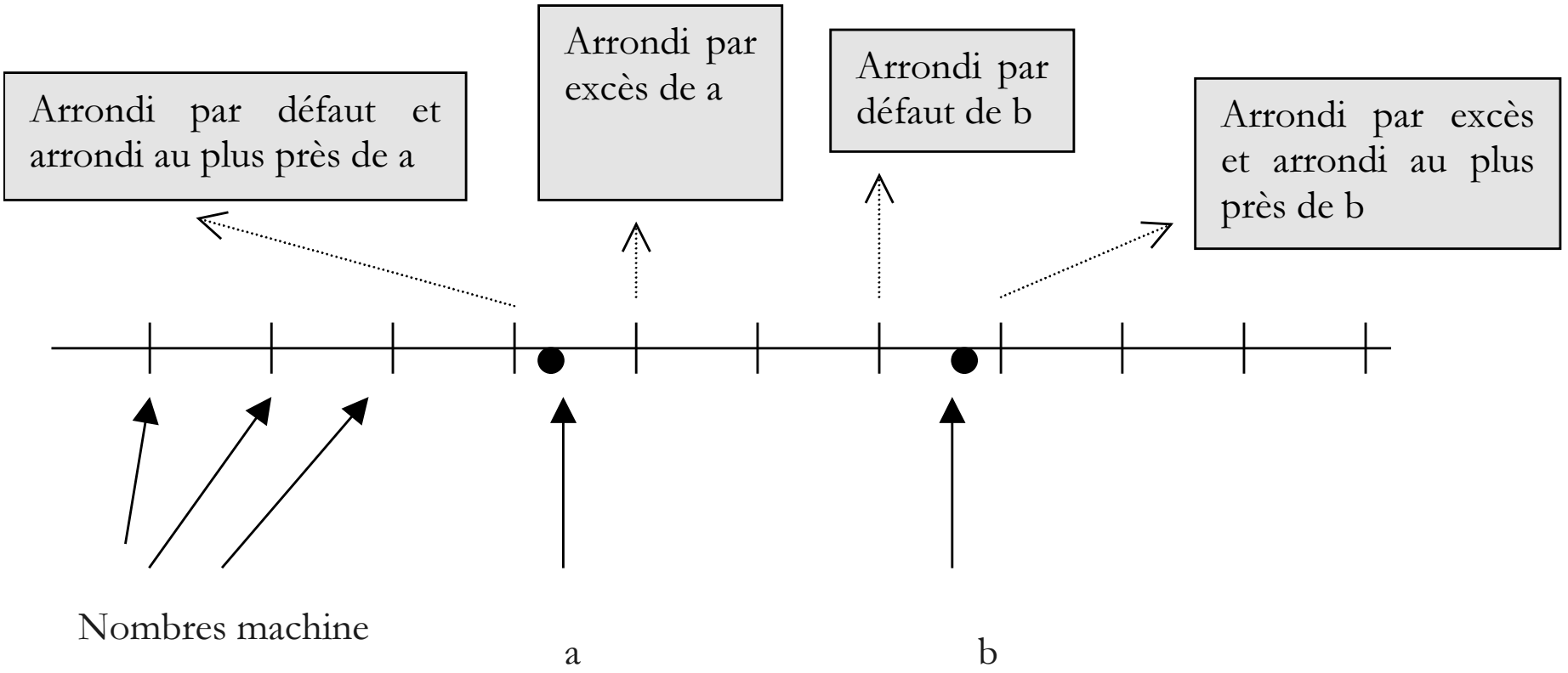
$$(-1)^s \, x_0 . x_1 x_2 \cdots x_{n-1} \times B^e$$

- En pratique $B = 2$
- e est l 'exposant
- mantisse *normalisée* : $x_0 \neq 0$, donne une mantisse entre 1 et B
- 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.


Arrondi par défaut et arrondi 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*.
- Arithmétique d'intervalles, arithmétique stochastique
- Semble difficile à satisfaire pour les fonctions élémentaires.

Standard IEEE-754: formats

Format	*Plus petit dénormalisé*	*Plus petit normalisé*	*+ grand fini*	*Mantisse (bits)*	*Exposant (bits)*
Simple précision	$1.4\ 10^{-45}$	$1.2\ 10^{-38}$	$3.4\ 10^{38}$	23 (+1)	8
Double précision	$4.9\ 10^{-324}$	$2.2\ 10^{-308}$	$1.8\ 10^{308}$	52 (+1)	11
Double étendu x86	$3.6\ 10^{-4951}$	$3.4\ 10^{-4932}$	$1.2\ 10^{4932}$	64	15
Quad SPARC	$6.4\ 10^{-4966}$	$3.4\ 10^{-4932}$	$2.9\ 10^{4932}$	112 (+1)	15

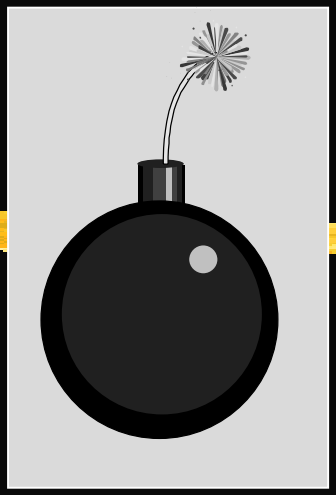
Exceptions : principes

- Idée fondamentale: on n'arrête pas le calcul;
- variables qui représentent le résultat d 'un dépassement, d'une opération invalide…
- drapeaux signalant qu'une exception a eu lieu;
- possibilité de reprogrammer le comportement de l 'arithmétique;

Traitement des exceptions

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

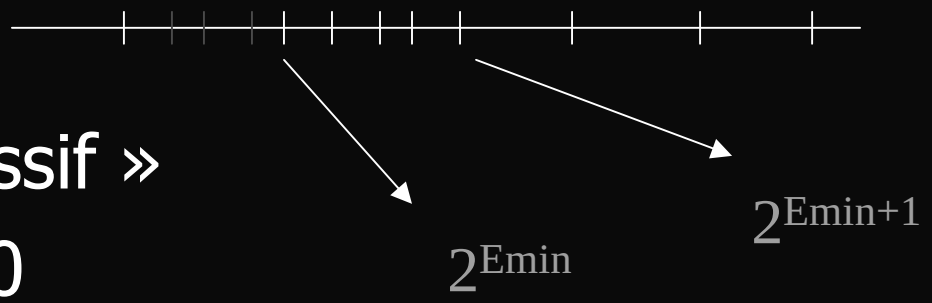
NaN (suite)



- Les NaN se *propagent* (simulation vs embarqué);
- attention aux problèmes logiques que pourraient entraîner des simplifications:
  - si x est un NaN, $x = x$ est faux, tandis que $x \neq x$ est vrai
  - si x ou y est un NaN, les tests $x < y$, $x <= y$, $x == y$, $x >= y$ et $x > y$ retournent « faux »

Nombres dénormalisés

- Lorsque l'exposant est le plus petit possible, la mantisse n'est plus normalisée;
- implantation délicate (donc pas toujours rapide: à tester);
- intérêt :
  - underflow « progressif »
  - si $x \neq y$ alors $x-y \neq 0$



Trois niveaux d'analyse

- on fait comme si les opérations étaient exactes (en tenant compte éventuellement de l'overflow/underflow);
- on fait comme si le calcul x * y donnait $x*y(1+\varepsilon)$, où $|\varepsilon|$ est borné par une valeur connue;
- on essaye de tenir compte de l'arrondi correct et du traitement des exceptions.

Exemple 1

A, B: 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);

Exemple 2 (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 ?

Respect des domaines d'arrivée

- Simple précision IEEE
- nombre machine le + proche de arctg (2^{30}):

$$\frac{13176795}{8388608} > \frac{\pi}{2}$$

- choisir entre précision et cohérence

Exemple 3 (division)

- B : nombre machine;
- y : nombre machine situé à moins d 'un *ulp* de $1/B$
- on applique la suite de calculs

$$z = (1-By)_{rn} \quad t = (y+zy)_{rn}$$

Donne l'arrondi au plus près de 1/B

Division et racine carrée

- Autrefois : opérateur spécifique, et algorithme $\approx$ ce qu'on fait à la main
- IBM RS/6000, AMD K5-K7, … itérations de type Newton utilisant le multiplieur flottant, sans « casser » le pipeline
- garantir l'arrondi correct du résultat (preuve des algorithmes du K5 $\approx$ 50 pages).

Division précision « double étendue »

- $y_0 = 1/b$ err. rel. $< 2^{-8.886}$
- $e_0 = (1-by_0)_{rn}$
- $y_1 = (y_0+e_0y_0)_{rn}$
- $e_1 = e_0^2{}_{rn}$
- $y_2 = (y_1+e_1y_1)_{rn}$
- $e_2 = (1-by_2)_{rn}$
- $y_3 = (y_2+e_2y_2)_{rn}$
- $e_3 = (1-by_3)_{rn}$
- $y_4 = (y_3+e_3y_3)_{rn}$

- $q_0 = (ay_0)_{rn}$
- $r_0 = (a-bq_0)_{rn}$
- $q_1 = (q_0+r_0y_3)_{rn}$
- $r_1 = (a-bq_1)_{rn}$
- $q_2 = (q_1+r_1y_4)_t$

Donne le résultat correct dans le mode d ’arrondi « t » de la dernière opération

Quelques propriétés

- Si x est différent de y alors $x-y \neq 0$ *(dénormalisés nécessaires)*
- Si $\frac{1}{2} \leq y/x \leq 2$, alors le calcul de $x-y$ se fait sans erreur (idem)
- Double précision, A = 2^{600} et B = 2^{-600}, (A*A)*(B*B) donne NaN alors que A*(A*B)*B donne 1.

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)
- tests à faire passer pour vérifier que des simplifications abusives ne sont pas commises

Bons et mauvais élèves

Système	sin (10^{22})
exact	-0.852200849767…
HP 48 GX	-0.852200849767
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

Le dilemme du fabricant de tables

- Problème : arrondi correct des fonctions élémentaires (sin, cos, log, etc.);
- 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 ?

En base 2

- Arrondis « dirigés »:

n

x.xxxxxxxxxxx 0000000000000 1zzzzzz

x.xxxxxxxxxxx 111111111110 zzzzzz

- Arrondi au plus près

x.xxxxxxxxxxx 1000000000000 1zzzzzz

x.xxxxxxxxxxx 011111111110 zzzzzz

Exemple: mantisses 24 bits

- Le nombre qui s'écrit 1.11001100111000110011001 vaut :

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

- Son exponentielle s'écrit en base 2:

110.00001101001111010011 0 11111111111111111111111 01100100001…

Réécriture avec des entiers

$$\exp\left(\frac{15102361}{2^{23}}\right) = 2^{24} \times 126913\mathbf{66.499999981}\ldots$$

Trouver les « pires cas »

- Connaître la bonne valeur de m et donc, si elle est petite, obtenir des algorithmes efficaces;
- Simple précision: aucun problème ;
- meilleures estimations théoriques pour la double précision: $m \approx 1000000$;
- tests exhaustifs… mais 2^{64}, c'est grand !
- IBM/Ziv… sur-dimensionné, et non prouvé!

Méthodologie employée (V. Lefèvre)

- Découpage en petits domaines : approximations d'ordre 1 ;
- distance entre une droite et une grille ;
- algorithme d'Euclide modifié ;
- parallélisme massif ;
- équivalence entre le problème pour f et celui pour f^{-1}.

Quelques pires cas trouvés

$$2^{54} \times \sin\left(\frac{8980155785351021}{2^{54}}\right)$$
$$= 861282010793742.99999999999999999998706....$$

$$2^{51} \times \exp\left(\frac{471483227223279}{2^{49}}\right)$$
$$= 5203087862132336.49999999999999981826665\cdots$$

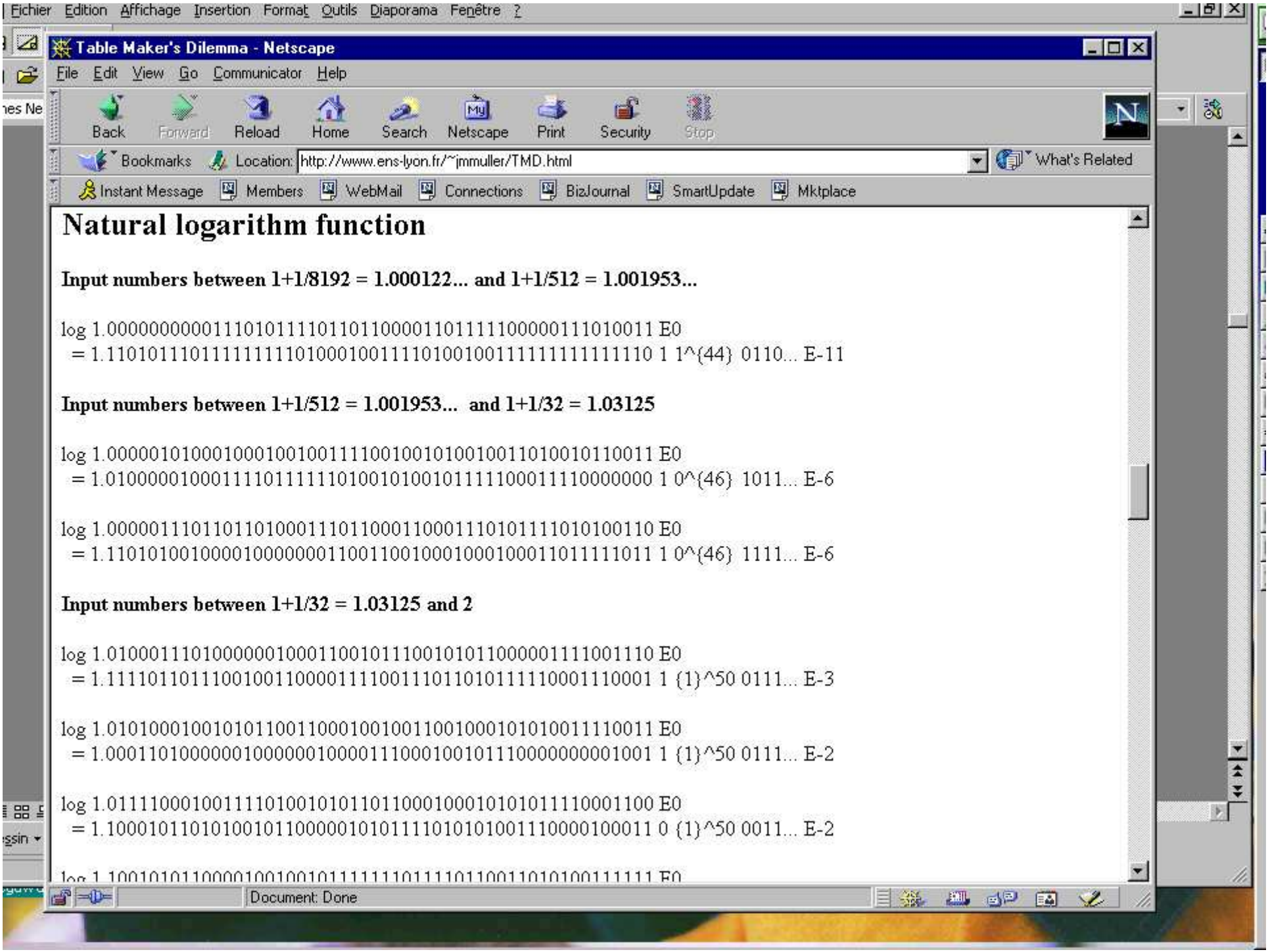
Résultats

- Partie « publique » disponible à l'URL

  `http://www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm`

- Partie confidentielle

- Fonctions: exp (bases e et 2), log (bases e et 2), sin, cos


Fichier Edition Affichage Insertion Format Outils Diaporama Fenêtre ?
Table Maker's Dilemma - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.ens-lyon.fr/~jmmuller/TMD.html What's Related
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Natural logarithm function
Input numbers between 1+1/8192 = 1.000122... and 1+1/512 = 1.001953...
log 1.00000000001110101111011011000011011111000000111010011 E0
= 1.11010111011111111110100010011110100100111111111111110 1 1^{44} 0110... E-11
Input numbers between 1+1/512 = 1.001953... and 1+1/32 = 1.03125
log 1.00000101000100010010011111001001010010011010010110011 E0
= 1.01000001000111101111110100101001011111000111000000 1 0^{46} 1011... E-6
log 1.00000111011011010001110110001100011101011110101001l0 E0
= 1.11010100100001000000011001100100010001000110111111011 1 0^{46} 1111... E-6
Input numbers between 1+1/32 = 1.03125 and 2
log 1.01000111010000001000110010111001010110000011111001110 E0
= 1.11110110111001001100001111001110110101111100011100l1 1 {1}^50 0111... E-3
log 1.01010001001010110011000100100110010001010100111100l1 E0
= 1.00011010000001000000100001110001001011100000000001001 1 {1}^50 0111... E-2
log 1.01111000100111101001010110110001000101010111100011l00 E0
= 1.10001011010100101100000101011110101010011100001000l1 0 {1}^50 0011... E-2
Document: Done

Conséquences

Bientôt, toutes les primitives numériques seront complètement spécifiées:

- meilleure portabilité des programmes
- meilleure fiabilité
- plus grande facilité d'implantation des arithmétiques stochastique, d'intervalles, etc.

Paranoia et UCB Test

- Groupe de W. Kahan à Berkeley
- Principales caractéristiques d'une arithmétique
- pas de garantie absolue de conformité à la norme IEEE, seulement « indications fortes »
- Paranoia devrait être utilisé systématiquement
- inclus dans Aquarels
- Gros intérêt pour les exceptions
- UCB Test teste aussi les fonctions élémentaires: précieux car leur qualité varie beaucoup;

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. précision converti en chaîne décimale d'au moins 17 chiffres, chaîne convertie à nouveau en d.p.