Le flottant pour les nuls

Sylvie Boldo

IRIF – 1er avril 2019

Merci à...

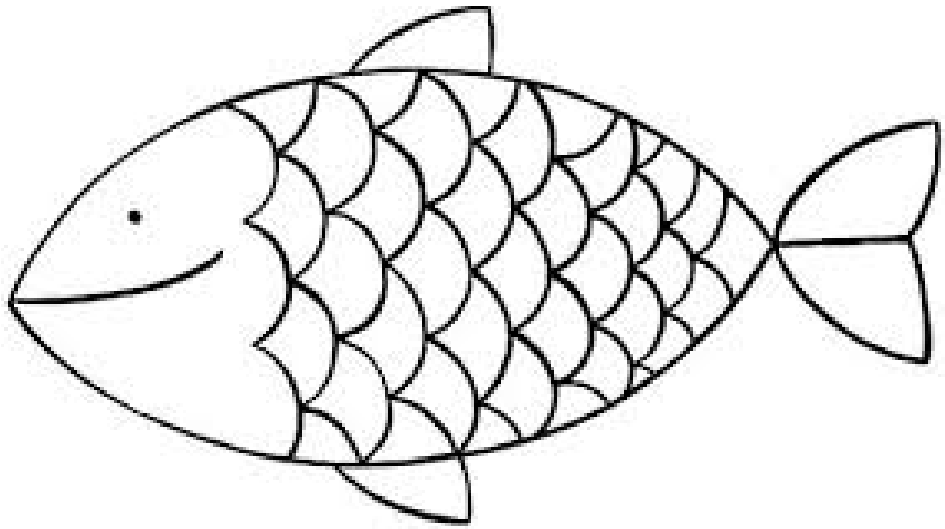
- Valérie Berthé pour l'invitation

Merci à...

- Valérie Berthé pour l'invitation
- Guillaume Melquiond pour les slides en commun (HDR cet après-midi)

Merci à...

- Valérie Berthé pour l'invitation
- Guillaume Melquiond pour les slides en commun (HDR cet après-midi)



Plan

Les nombres, je sais ce que c'est!

0, 1, 2,

Les nombres, je sais ce que c'est!

$0,\ 1,\ 2,\ \frac{2}{3},$

Les nombres, je sais ce que c'est!

$$0,\ 1,\ 2,\ \frac{2}{3},\ \sqrt{2},$$

Les nombres, je sais ce que c'est!

$$0,\ 1,\ 2,\ \frac{2}{3},\ \sqrt{2},\ \pi$$

De tout temps, on a compté...

De tout temps, on a compté...

Os d'Ishango (23 000 BC)

De tout temps, on a compté...

Os d'Ishango (23 000 BC)

les troupeaux ou les impôts...

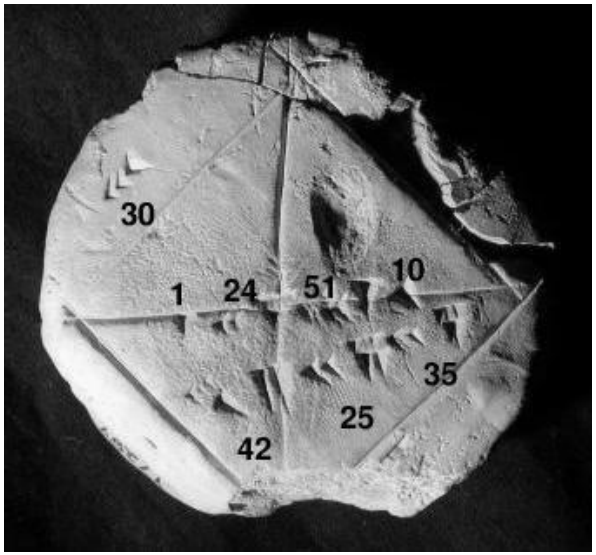
De tout temps, on a compté...

Os d'Ishango (23 000 BC)

les troupeaux ou les impôts...

et de la géométrie!

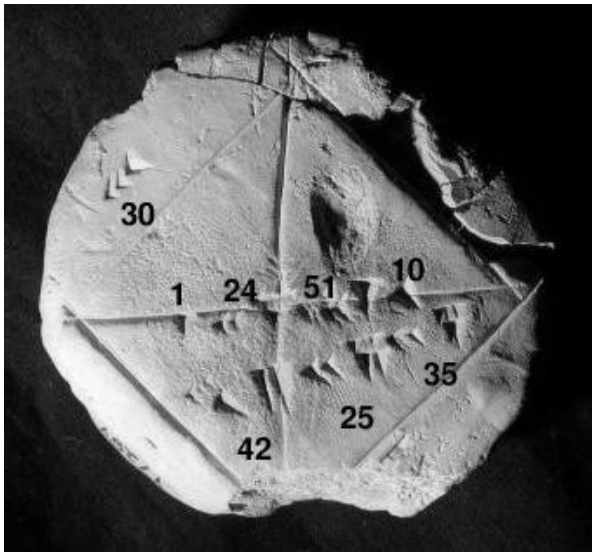
Babyloniens, 2000 avant JC



Tablette YBC 7289 (1600–1800 BC)

- les quatre opérations
- extraction de racines carrées, racines cubiques
- résolution d'équations du second degré...

Babyloniens, 2000 avant JC

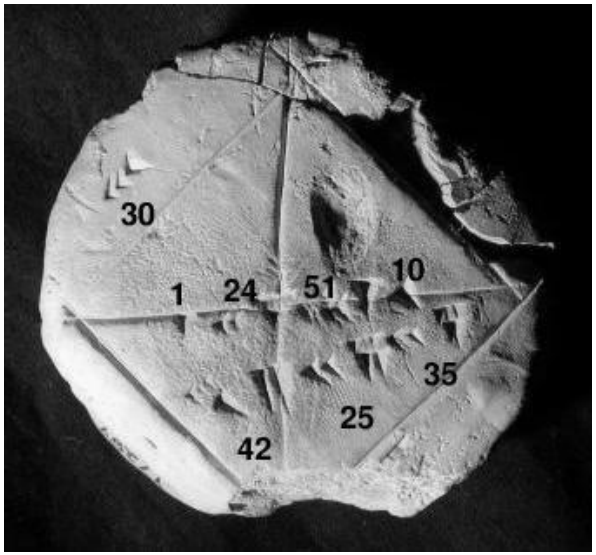


Tablette YBC 7289 (1600–1800 BC)

- les quatre opérations
- extraction de racines carrées, racines cubiques
- résolution d'équations du second degré...

En base 60!

Babyloniens, 2000 avant JC



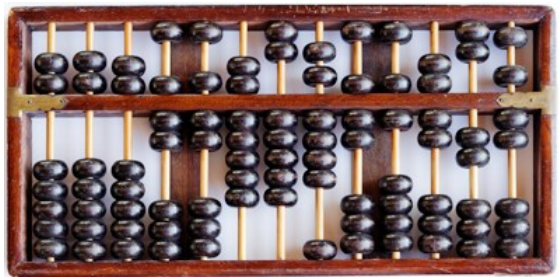
Tablette YBC 7289 (1600–1800 BC)

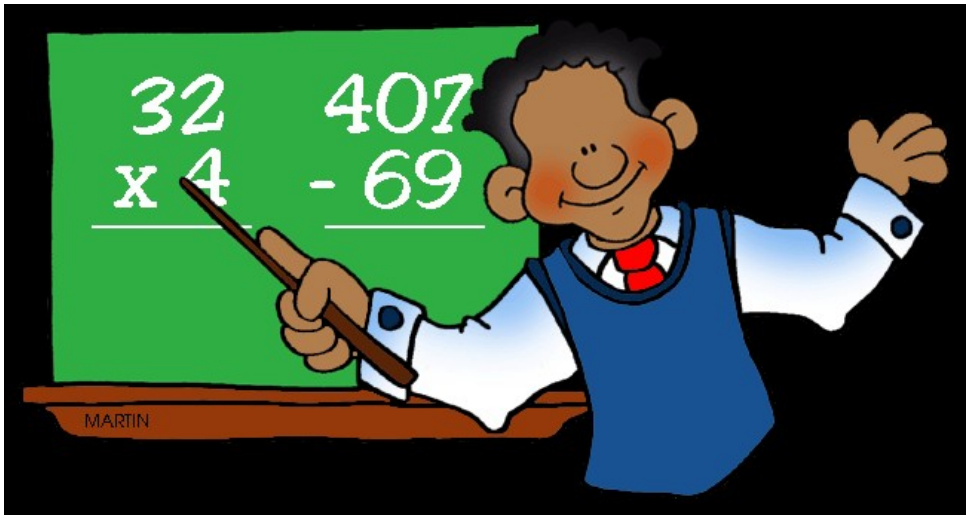
- les quatre opérations
- extraction de racines carrées, racines cubiques
- résolution d'équations du second degré...

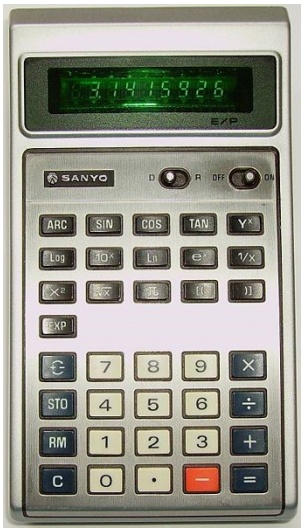
En base 60!

La photo représente $\sqrt{2}$ avec quatre chiffres sexagésimaux significatifs soit près de six chiffres décimaux!

Le calcul humain assisté (G. Berry ©)





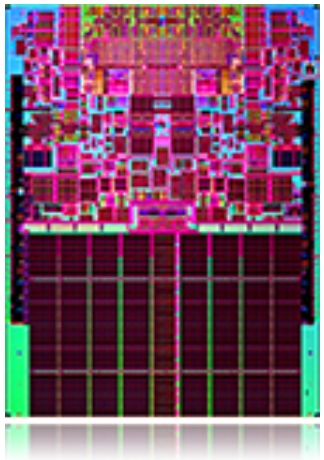


L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

Il peut y avoir :

- un bug matériel (le processeur est faux).

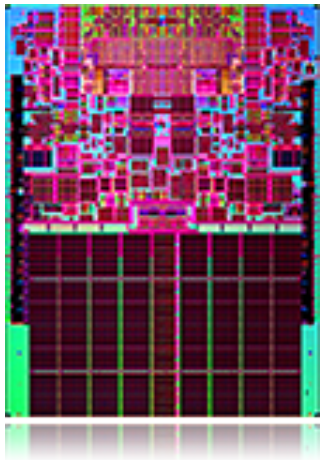


L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

Il peut y avoir :

- un bug matériel (le processeur est faux).

  Ça arrive, mais c'est très rare.



L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

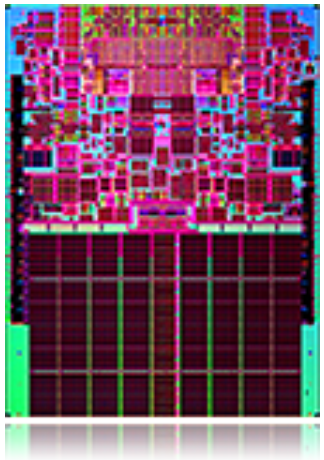
Il peut y avoir :

- un bug matériel (le processeur est faux).

  Ça arrive, mais c'est très rare.

  Bug du Pentium : certaines divisions fausses à partir du 5e chiffre

  475 millions de $.



L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

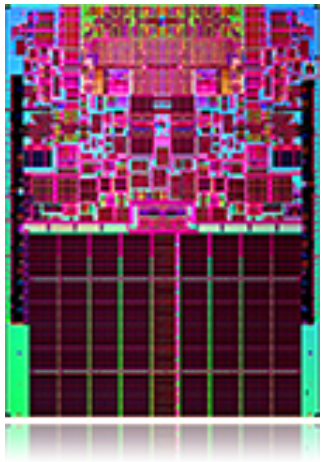
Il peut y avoir :

- un bug matériel (le processeur est faux).

  Ça arrive, mais c'est très rare.

  Bug du Pentium : certaines divisions fausses à partir du 5e chiffre

  475 millions de $.



- un bug logiciel.

L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

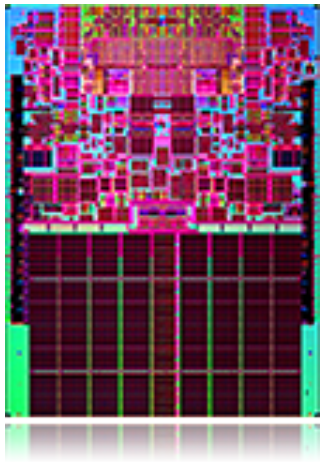
Il peut y avoir :

- un bug matériel (le processeur est faux).

  Ça arrive, mais c'est très rare.

  Bug du Pentium : certaines divisions fausses à partir du 5e chiffre

  475 millions de $.



- un bug logiciel.

  Là, par contre. . .

L'erreur est humaine, mais pour provoquer une vraie catastrophe, il faut un ordinateur.

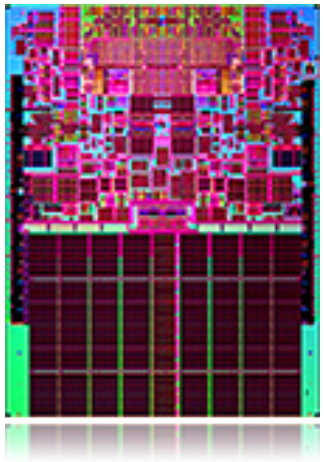
Il peut y avoir :

- un bug matériel (le processeur est faux).

  Ça arrive, mais c'est très rare.

  Bug du Pentium : certaines divisions fausses à partir du 5e chiffre

  475 millions de $.



- un bug logiciel.

  Là, par contre...

  Ariane 5 : explosion due à un dépassement de capacité

  500 millions de $.

Un exemple (28/09/07)

Prenons Microsoft Excel 2007.

Un exemple (28/09/07)

Prenons Microsoft Excel 2007.

- Dans la case A1, inscrivez 850
- Dans la case A2, inscrivez 77,1 =PRODUIT(A1: A2)
- Dans la case A3, inscrivez la formule suivante :

Un exemple (28/09/07)

Prenons Microsoft Excel 2007.

- Dans la case A1, inscrivez 850
- Dans la case A2, inscrivez 77,1 =PRODUIT(A1: A2)
- Dans la case A3, inscrivez la formule suivante :

Excel vous répond 100 000.

Un exemple (28/09/07)

Prenons Microsoft Excel 2007.

- Dans la case A1, inscrivez 850
- Dans la case A2, inscrivez 77,1 =PRODUIT(A1: A2)
- Dans la case A3, inscrivez la formule suivante :

Excel vous répond 100 000.
La réponse juste est 65 535.

Un exemple (28/09/07)

Prenons Microsoft Excel 2007.

- Dans la case A1, inscrivez 850
- Dans la case A2, inscrivez 77,1 =PRODUIT(A1: A2)
- Dans la case A3, inscrivez la formule suivante :

Excel vous répond 100 000.
La réponse juste est 65 535.

Mais ce n'est ≪ que ≫ un bug d'affichage.

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.
- 9.5*x vaut

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.
- 9.5*x vaut
  - Avec Maple 11 à 13, **Segmentation fault**!

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.
- 9.5*x vaut
  - Avec Maple 11 à 13, **Segmentation fault**!
  - Avec Maple 14 et 15, $0.10\,10^{-92233720368547758 08}$!

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.
- 9.5*x vaut
  - Avec Maple 11 à 13, **Segmentation fault**!
  - Avec Maple 14 et 15, $0.10\,10^{-92233720368547758 08}$!
  - Avec Maple 17, 0.10!

Un exemple plus récent (25/05/13) de Vincent Lefèvre

Prenons Maple sur une architecture 64 bits.

- Calculons x := 10.^3995480790*10.^92233720328592950 16;
- Calculons 10.*x puis 9.4*x puis 9.5*x.

- x := $0.1\,10^{92233720368547758 07}$.
- 10.*x vaut Float(infinity).
- 9.4*x vaut $0.9\,10^{92233720368547758 07}$.
- 9.5*x vaut
  - Avec Maple 11 à 13, **Segmentation fault**!
  - Avec Maple 14 et 15, $0.10\,10^{-92233720368547758 08}$!
  - Avec Maple 17, 0.10!

Pour une machine 32 bits, on peut considérer 95.*10.^2147483645 ($\rightarrow$ out of memory, petit nombre).

En fait, les bugs y sont rarement pour quelque chose...

En fait, les bugs y sont rarement pour quelque chose...

Votre ordinateur calcule toujours (un peu) faux

En fait, les bugs y sont rarement pour quelque chose...

Votre ordinateur calcule toujours (un peu) faux

...mais vous ne vous en rendez pas compte.

En fait, les bugs y sont rarement pour quelque chose...

Votre ordinateur calcule toujours (un peu) faux

...mais vous ne vous en rendez pas compte.

Comment l'ordinateur fait pour calculer (faux)?

Les nombres à virgule flottante

Le mémoire de mon ordinateur étant limitée, on limite la précision (le nombre de chiffres) qu'on utilise. Les valeurs ainsi représentées sont appelées nombres flottants.

$$\pi \quad \hookrightarrow \quad 3{,}14$$

Les nombres à virgule flottante

Le mémoire de mon ordinateur étant limitée, on limite la précision (le nombre de chiffres) qu'on utilise. Les valeurs ainsi représentées sont appelées nombres flottants.

$$\pi \quad \hookrightarrow \quad 3{,}14$$

$$123\ 456 \quad \hookrightarrow \quad 1{,}23 \times 10^5$$

$$\frac{1}{17} = 0{,}058\ 823\ 5\ldots \quad \hookrightarrow \quad 5{,}88 \times 10^{-2}$$

Les nombres à virgule flottante

Le mémoire de mon ordinateur étant limitée, on limite la précision (le nombre de chiffres) qu'on utilise. Les valeurs ainsi représentées sont appelées nombres flottants.

$$\pi \quad \hookrightarrow \quad 3{,}14$$

$$123\ 456 \quad \hookrightarrow \quad 1{,}23 \times 10^5$$

$$\frac{1}{17} = 0{,}058\ 823\ 5\ldots \quad \hookrightarrow \quad 5{,}88 \times 10^{-2}$$

Ces exemples sont en base 10 avec 3 chiffres.
Le processeur utilise la base 2 avec 24 ou 53 chiffres.

Les calculs (version base 10 avec 3 chiffres)

Chaque calcul simple est ensuite fait au mieux : le processeur renvoie le meilleur résultat possible étant donnés ses impératifs.

$$3{,}14^2 = 9{,}859\ 6 \quad \hookrightarrow \quad 9{,}86$$

Chaque résultat de calcul est donc arrondi.

Beaucoup de calculs

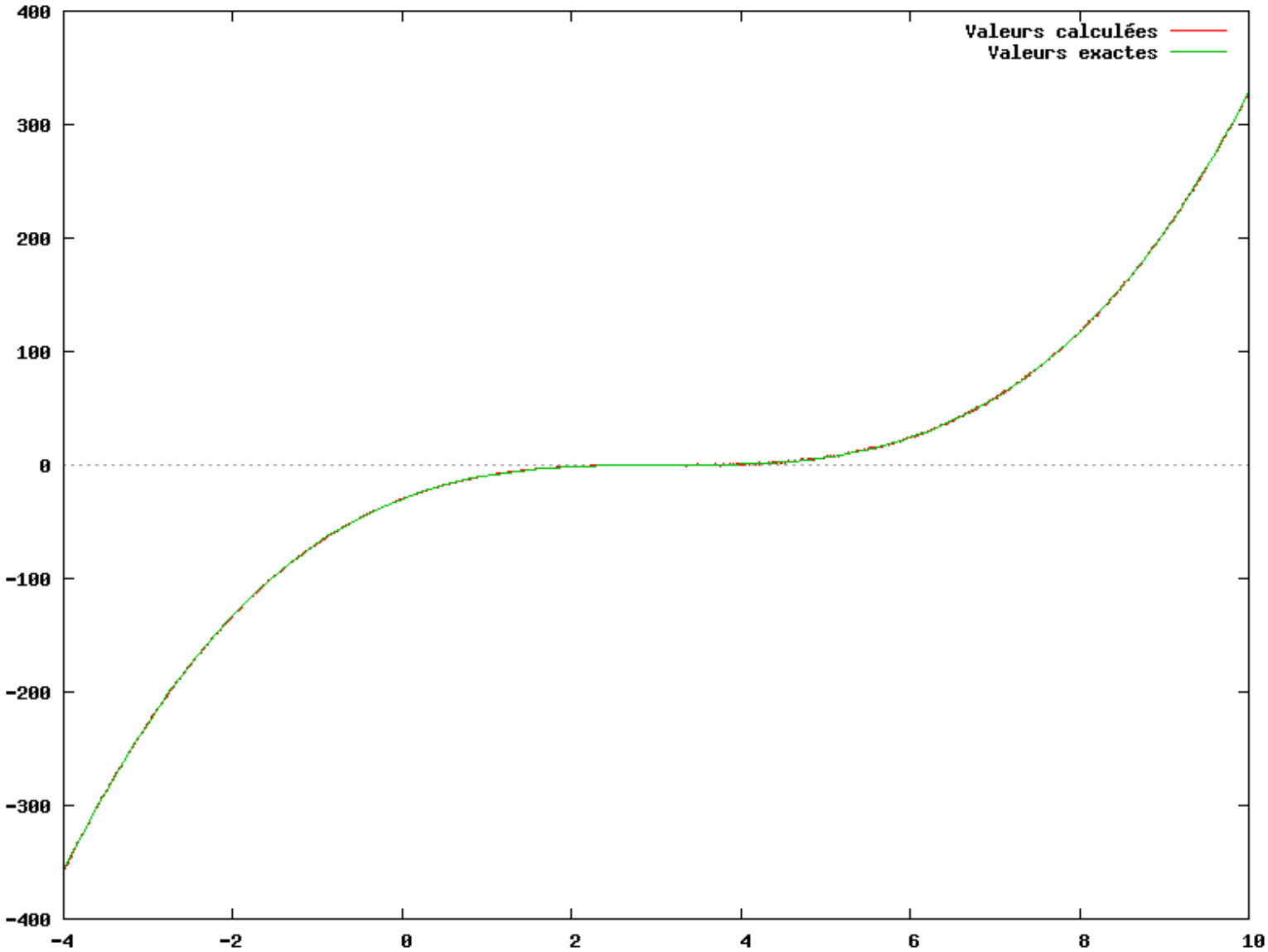
Même si un calcul est presque juste, une succession de calculs est parfois fausse :

$$\pi^2 \quad \hookrightarrow \quad 3{,}14^2 \quad \hookrightarrow \quad 9{,}86$$

Mais $\pi^2 = 9{,}869\ 604\ldots$ donc le nombre flottant le plus proche est 9,87.

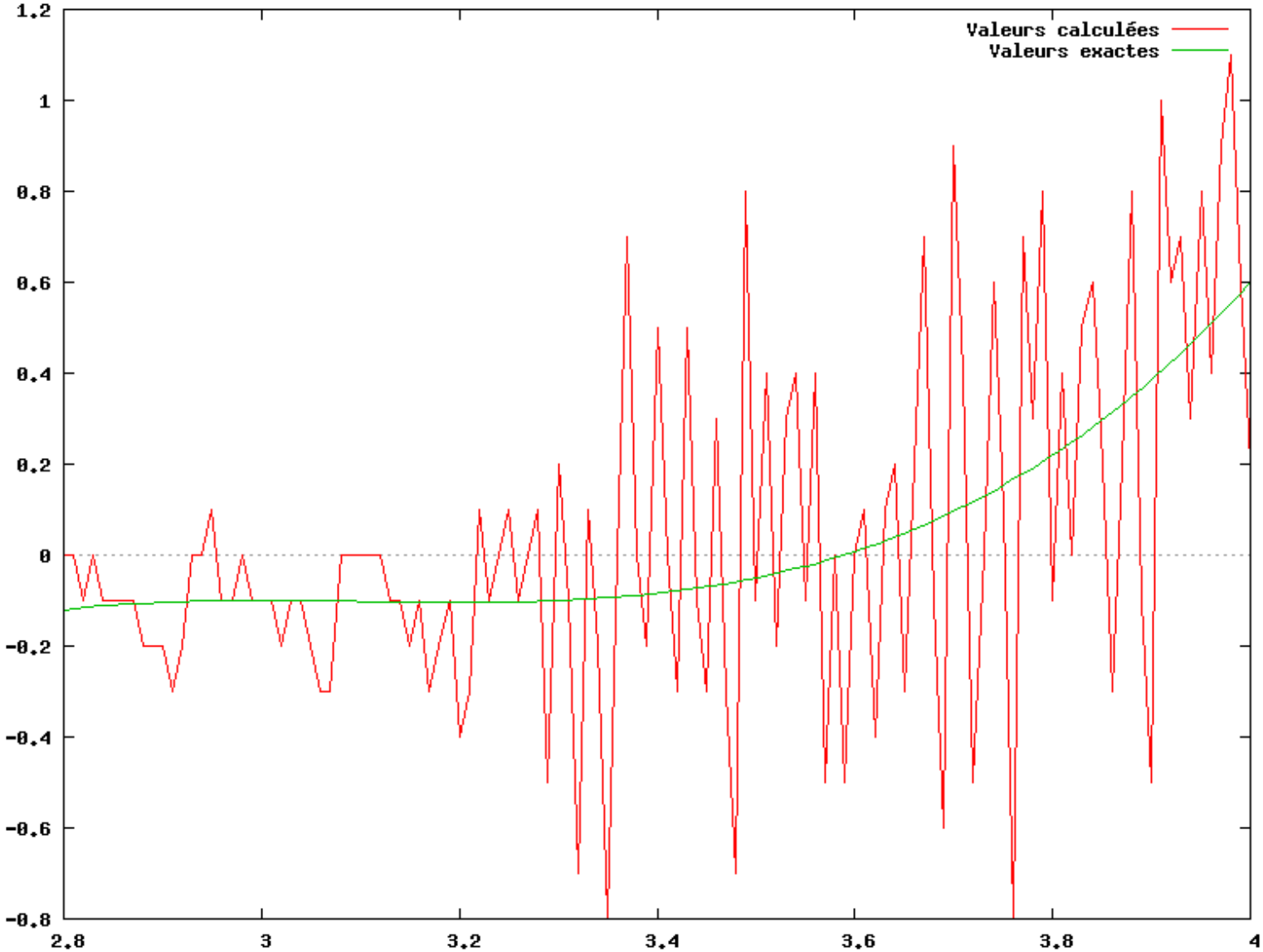
Vraiment beaucoup de calculs

Soit $P(x) = x^3 - 9{,}3x^2 + 28{,}8x - 29{,}8$.



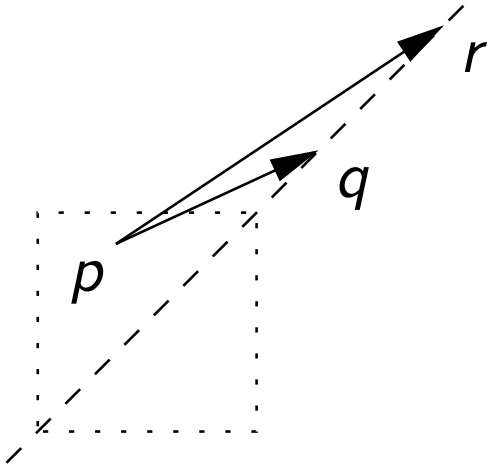
Vraiment beaucoup de calculs

Soit $P(x) = x^3 - 9{,}3x^2 + 28{,}8x - 29{,}8$.



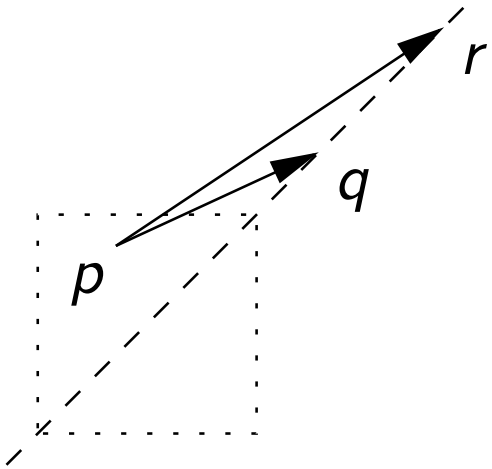
Orientation de 3 points

Étant donnés 3 points du plan p, q et r. On veut savoir si pqr sont alignés dans le sens horaire ou dans le sens anti-horaire.



Orientation de 3 points

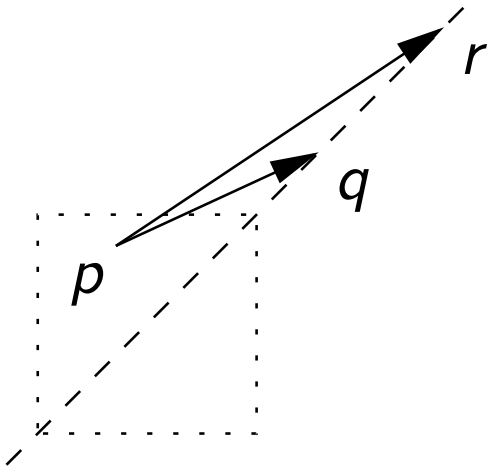
Étant donnés 3 points du plan p, q et r. On veut savoir si pqr sont alignés dans le sens horaire ou dans le sens anti-horaire.



$$\text{orient}_2(p,q,r) = \text{signe} \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix}$$

Orientation de 3 points

Étant donnés 3 points du plan p, q et r. On veut savoir si pqr sont alignés dans le sens horaire ou dans le sens anti-horaire.

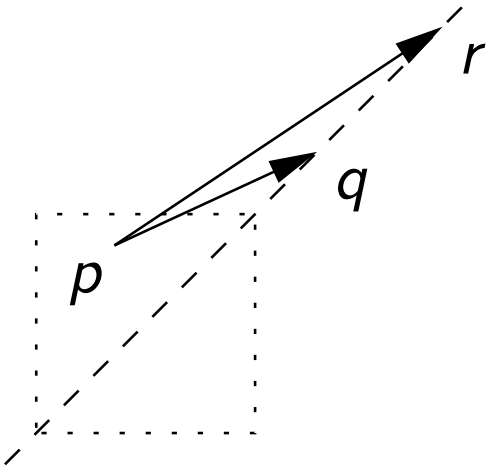


$$\text{orient}_2(p,q,r) = \text{signe} \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix}$$

```
float det = (qx - px) * (ry - py)
          - (qy - py) * (rx - px);
if (det > 0) return POSITIVE;
if (det < 0) return NEGATIVE;
return ZERO;
```

Orientation de 3 points - calculs exacts

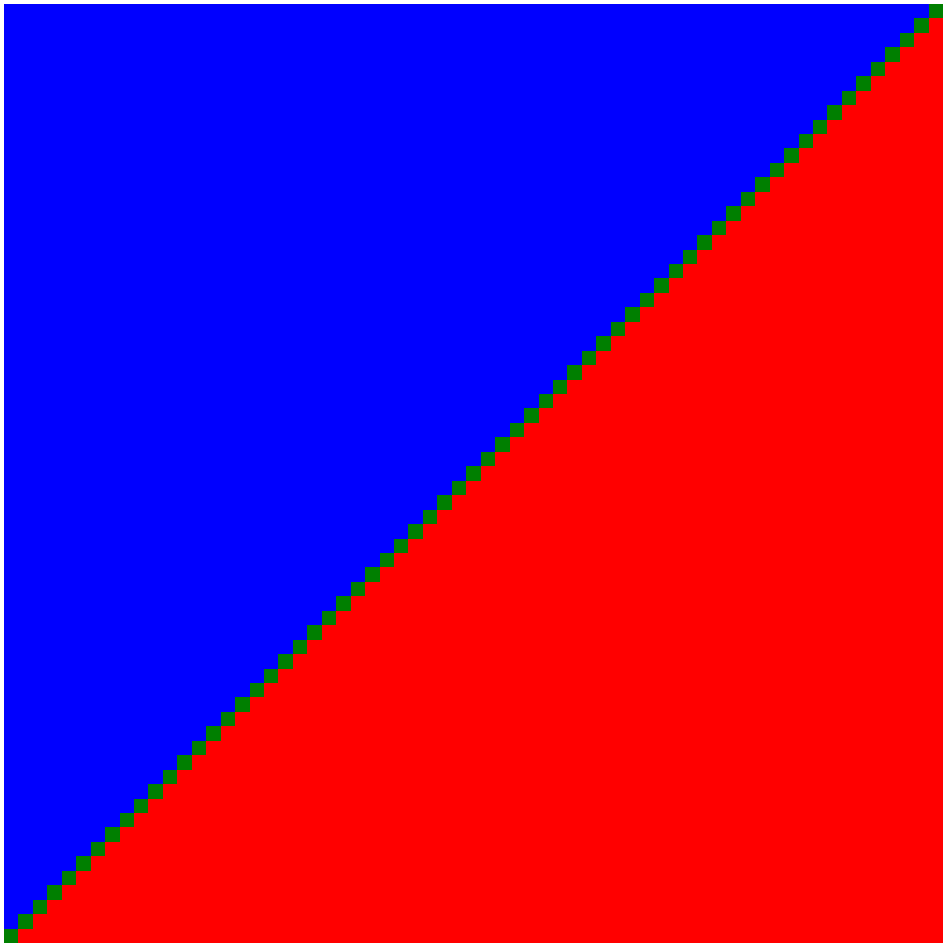
Pour $q = (8.1, 8.1)$ et $r = (12.1, 12.1)$ et p autour de (1,5; 1,5), le signe du déterminant devrait être :



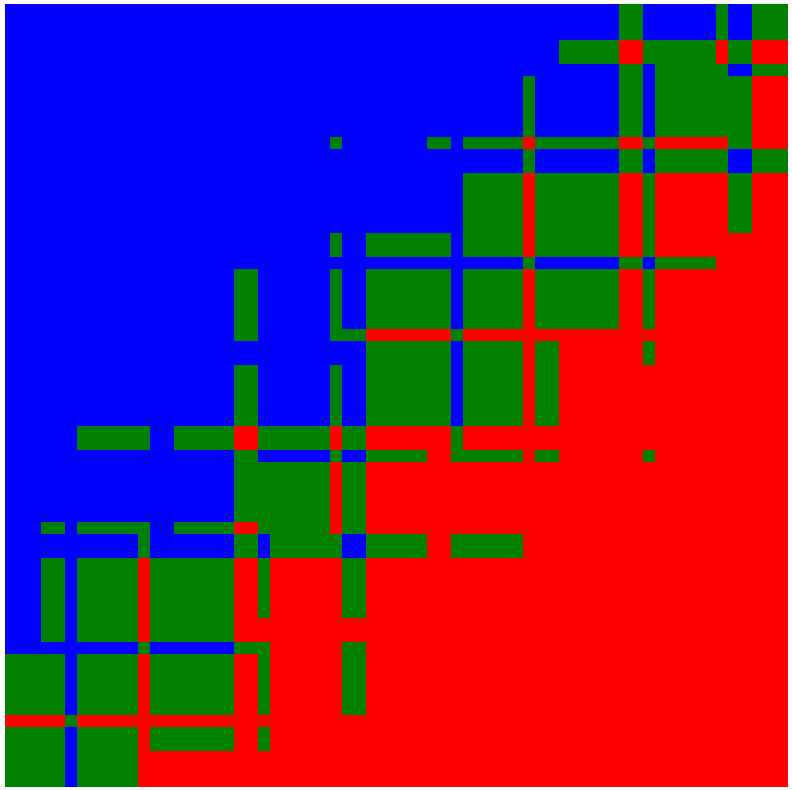
aligné

orienté

orienté



Orientation de 3 points - calculs flottants simple précision

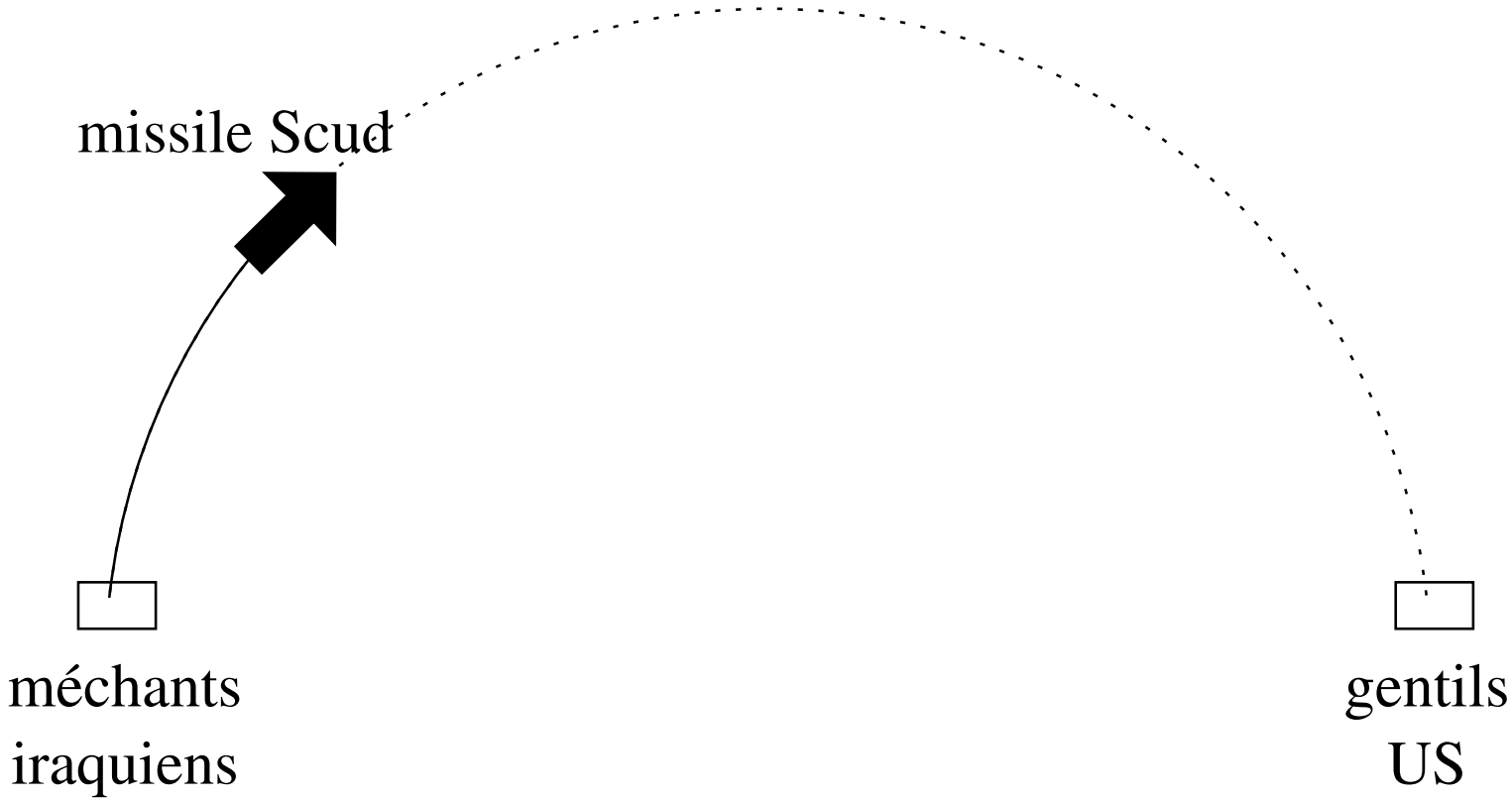


Première guerre du Golfe (1991)

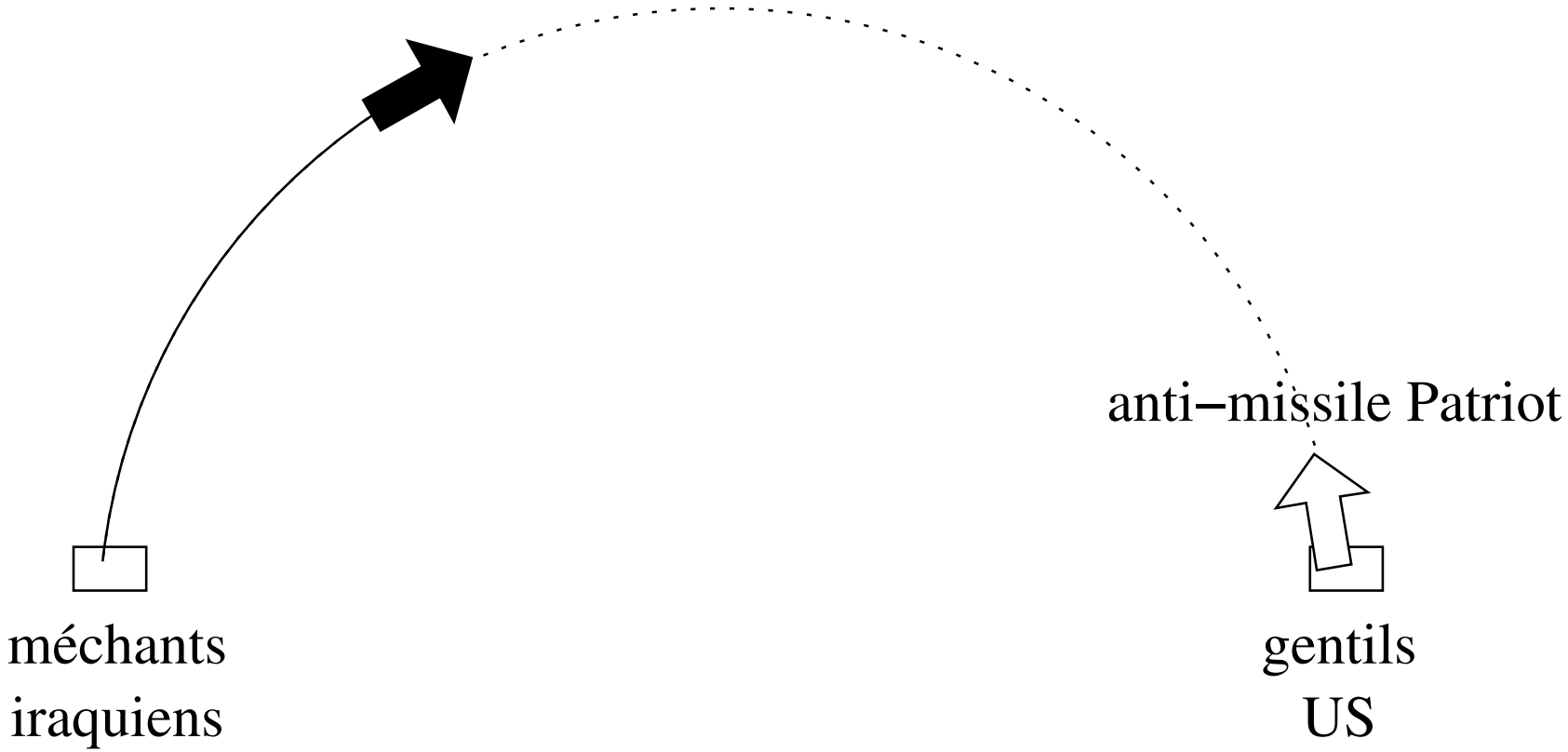
méchants
iraquiens

gentils
US

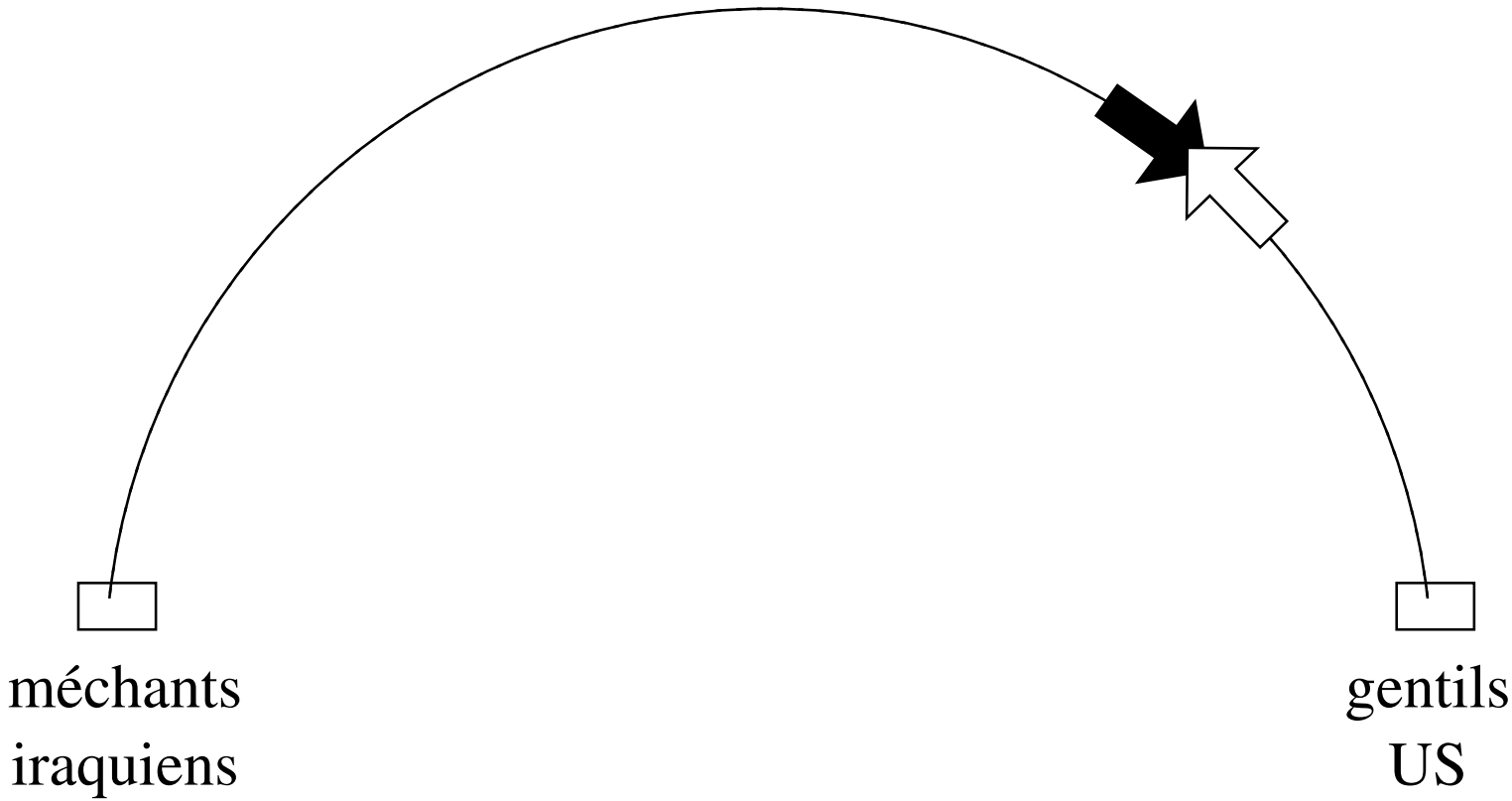
Première guerre du Golfe (1991)



Première guerre du Golfe (1991)



Première guerre du Golfe (1991)



Première guerre du Golfe (1991)

méchants
iraquiens

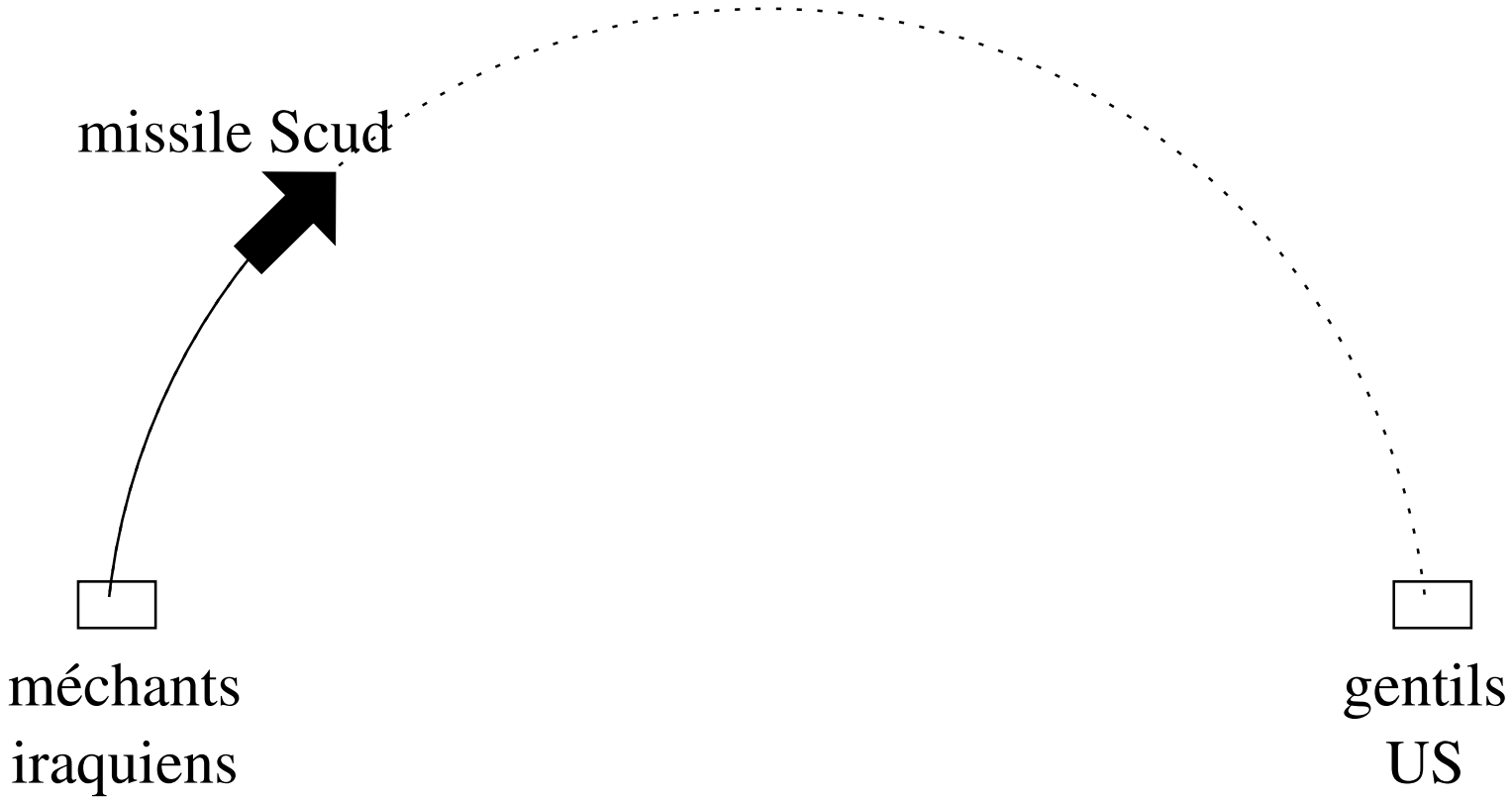
gentils
US

Première guerre du Golfe - 100h plus tard

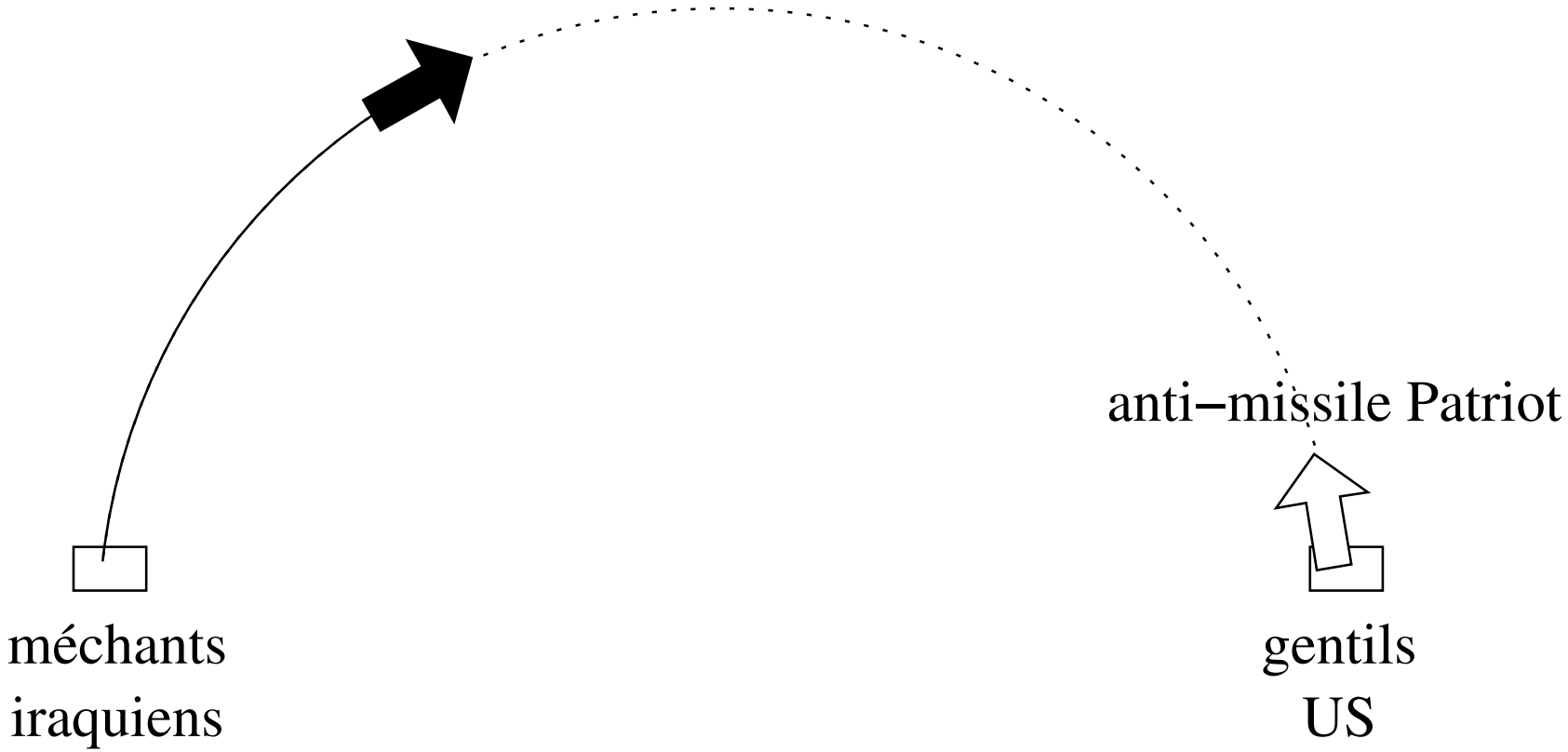
méchants
iraquiens

gentils
US

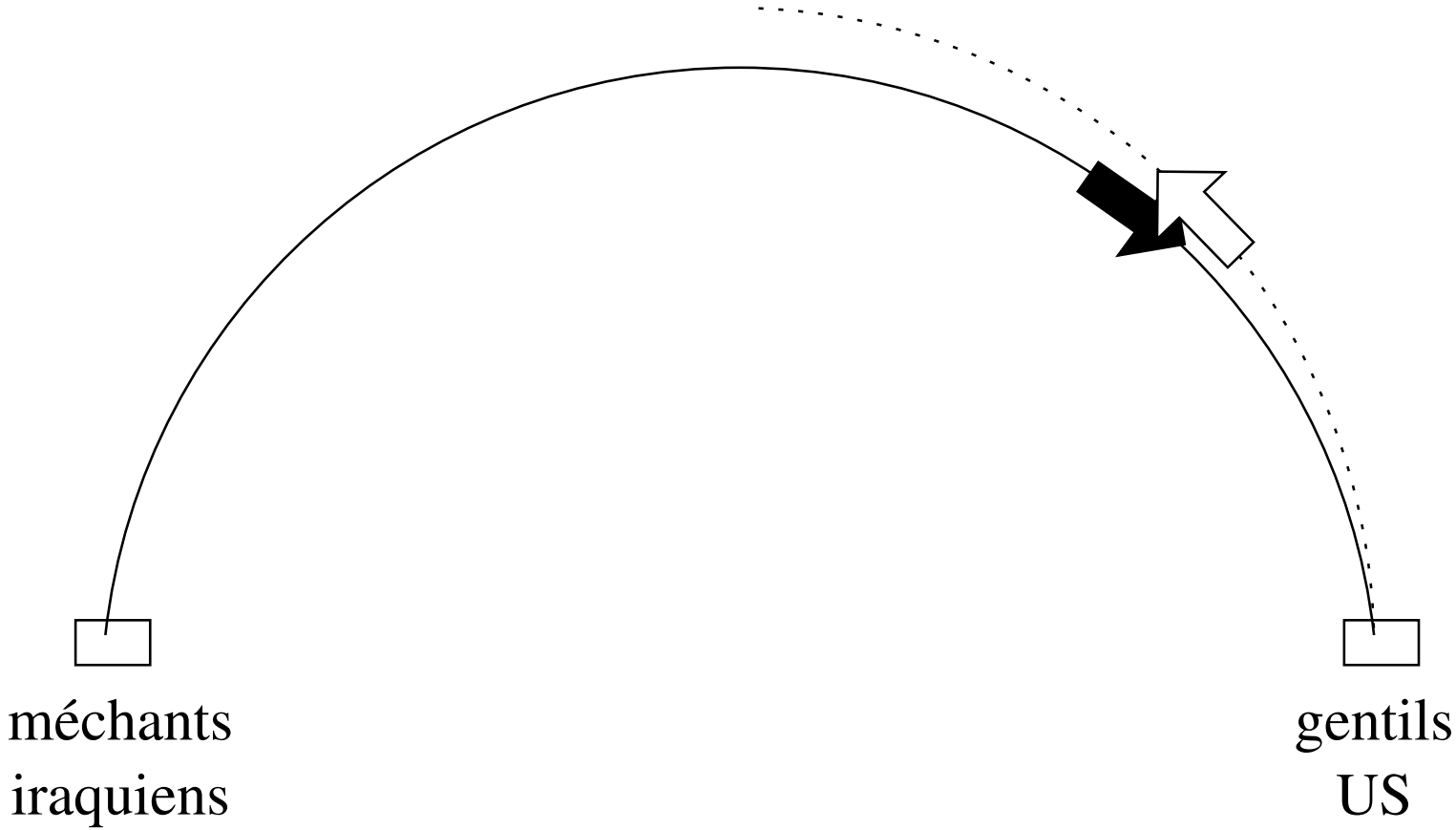
Première guerre du Golfe - 100h plus tard



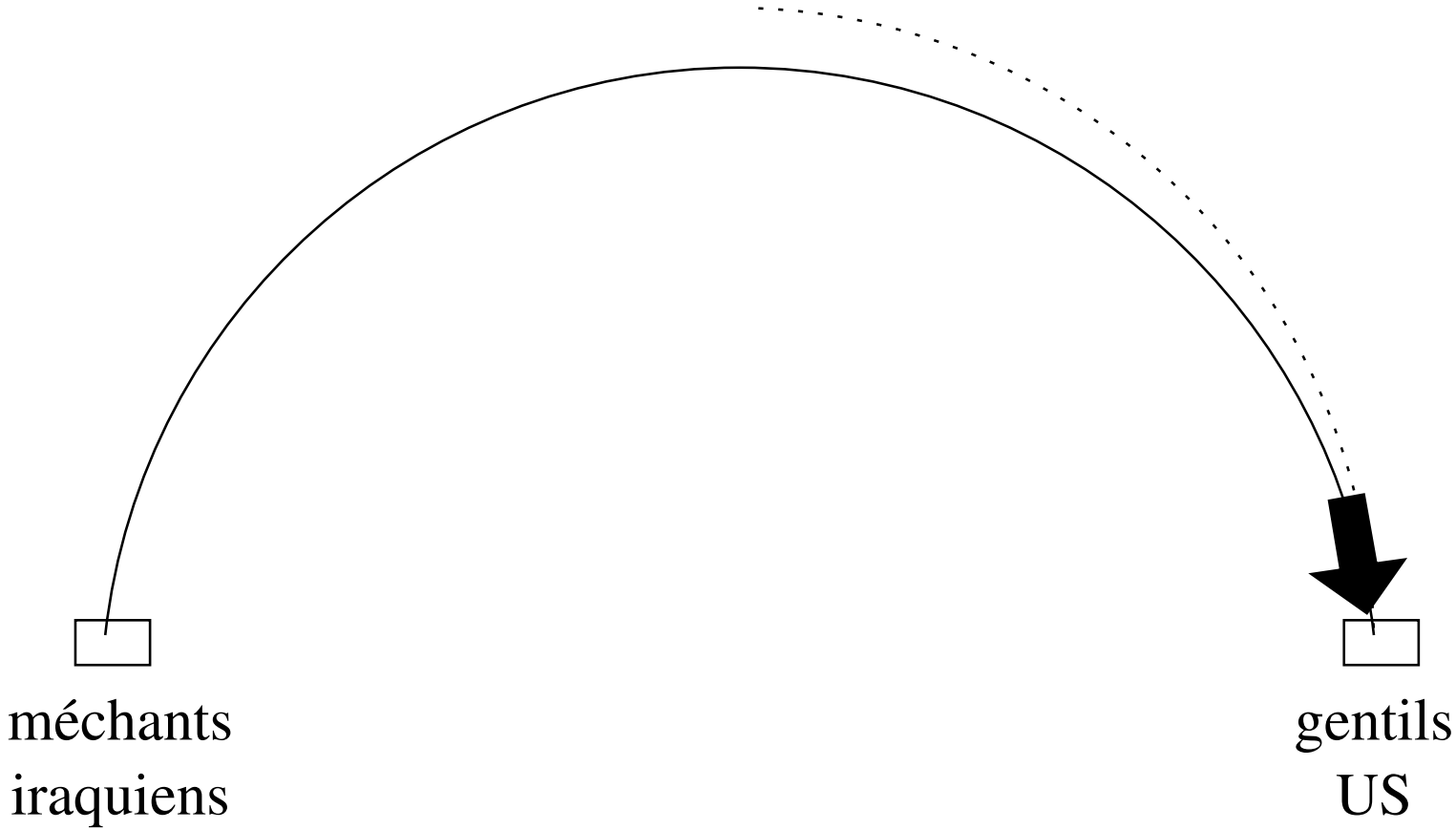
Première guerre du Golfe - 100h plus tard



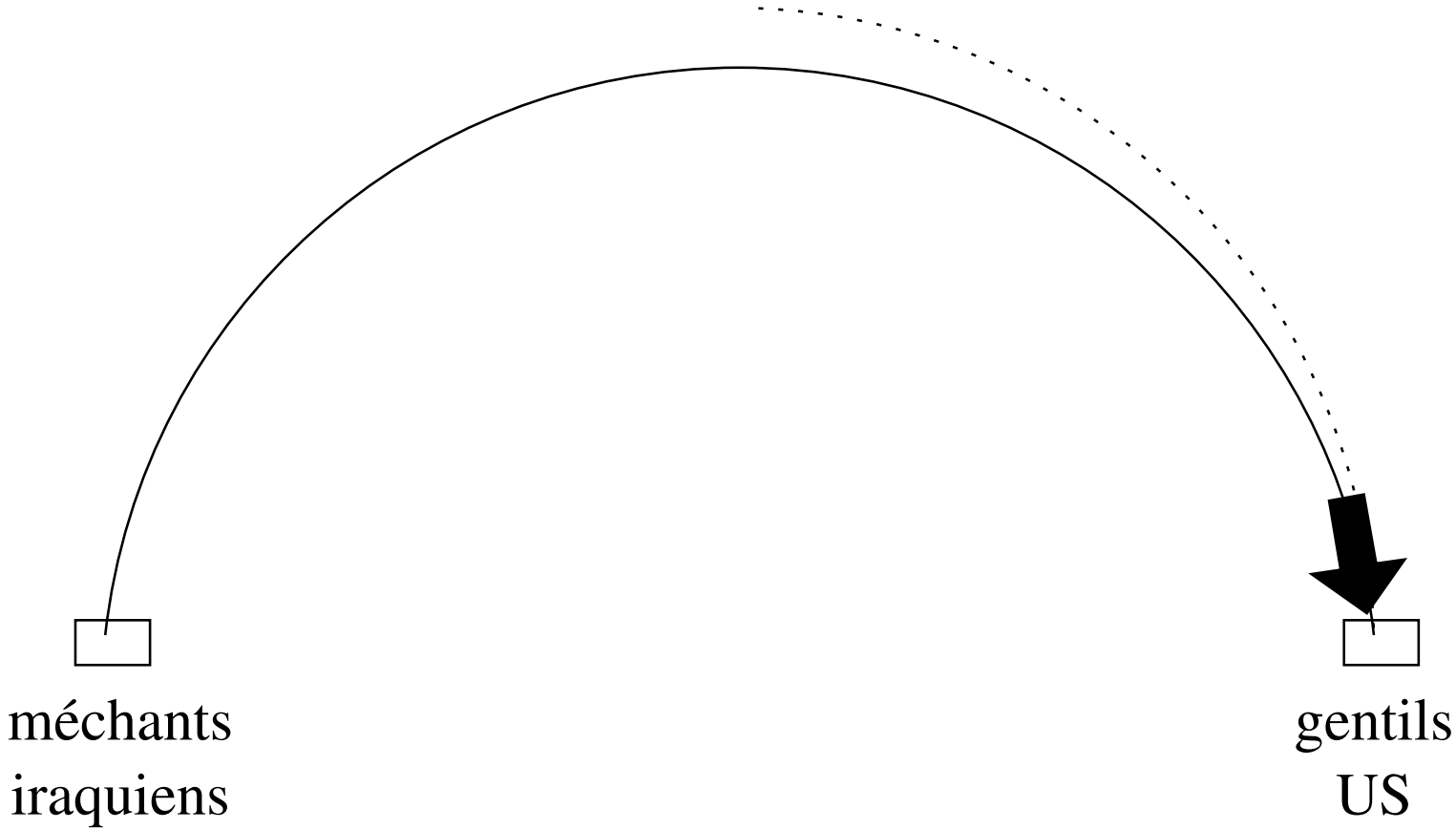
Première guerre du Golfe - 100h plus tard



Première guerre du Golfe - 100h plus tard



Première guerre du Golfe - 100h plus tard



$\Rightarrow$ 28 GI morts et 98 blessés

Première guerre du Golfe - explication

L'anti-missile Patriot est prévu pour fonctionner pendant quelques heures.

Mais ça marchait tellement bien qu'ils l'ont laissé branché.

Première guerre du Golfe - explication

L'anti-missile Patriot est prévu pour fonctionner pendant quelques heures.

Mais ça marchait tellement bien qu'ils l'ont laissé branché.

L'horloge interne ajoute 0,1s à chaque tic.
Mais 0,1 n'est pas exact en binaire :
$\Rightarrow$ une petite erreur à chaque tic
$\Rightarrow$ les erreurs, toutes dans le même sens, se sont accumulées
$\Rightarrow$ au bout de 100h, une erreur suffisante pour rater le missile

Plan

Plan

Nombres entiers

Nombres entiers positifs sur n bits :

$$a_n a_{n-1} \cdots a_1 a_0 \text{ représente la valeur } \sum_{i=0}^{n} a_i 2^i.$$

Nombres entiers

Nombres entiers positifs sur n bits :

$$a_n a_{n-1} \cdots a_1 a_0 \text{ représente la valeur } \sum_{i=0}^{n} a_i 2^i.$$

Nombres entiers signés sur n bits :

Premier bit s et les $n-1$ bits suivants $\hookrightarrow$ valeur positive v

$\hookrightarrow$ v si $s = 0$

$\hookrightarrow$ $-2^{n-1} + v$ si $s = 1$.

Nombres entiers

Nombres entiers positifs sur n bits :

$$a_n a_{n-1} \cdots a_1 a_0 \text{ représente la valeur } \sum_{i=0}^{n} a_i 2^i.$$

Nombres entiers signés sur n bits :

Premier bit s et les $n-1$ bits suivants $\hookrightarrow$ valeur positive v

$\hookrightarrow$ v si $s = 0$

$\hookrightarrow$ $-2^{n-1} + v$ si $s = 1$.

$\Rightarrow$ valeurs de -2^{n-1} à $2^{n-1} - 1$

$\Rightarrow$ un seul zéro

$\Rightarrow$ le premier chiffre donne le signe

Nombres entiers et overflow

Sur 32 bits, les valeurs signées vont de -2^{31} à $2^{31} - 1$. Et au-delà ?

Nombres entiers et overflow

Sur 32 bits, les valeurs signées vont de -2^{31} à $2^{31}-1$. Et au-delà ?

Plusieurs possibilités selon le langage et l'environnement :

- arithmétique modulo
- saturation
- échec

Nombres entiers et overflow

Sur 32 bits, les valeurs signées vont de -2^{31} à $2^{31}-1$. Et au-delà ?

Plusieurs possibilités selon le langage et l'environnement :

- arithmétique modulo
- saturation
- échec

$\Rightarrow$ pour être certain du comportement, on doit prouver que les valeurs restent entre -2^{31} à $2^{31}-1$!

Plan

Historique

1941 Premier vrai processeur avec unité flottante, le Z3 de Konrad Zuse

Historique

1941 Premier vrai processeur avec unité flottante, le Z3 de Konrad Zuse

1980 Coprocesseur flottant Intel 8087 (50 000 flops !)

Historique

1941 Premier vrai processeur avec unité flottante, le Z3 de Konrad Zuse

1980 Coprocesseur flottant Intel 8087 (50 000 flops !)

1985 Norme IEEE-754 régissant les formats et les calculs flottants

Historique

1941 Premier vrai processeur avec unité flottante, le Z3 de Konrad Zuse

1980 Coprocesseur flottant Intel 8087 (50 000 flops !)

1985 Norme IEEE-754 régissant les formats et les calculs flottants

2008 Révision de la norme IEEE-754 (dont ajout du décimal)

Nombre à virgule flottante

Ce n'est qu'une suite de bits

11100011010010011110000111000000

Nombre à virgule flottante

Ce n'est qu'une suite de bits

$$11100011010010011110000111000000$$

à laquelle on donne un sens selon des tailles données
pour s (signe), e (exposant) et f (fraction)

1	11000110	10010011110000111000000
1	11000110	10010011110000111000000
s	e	f

Nombre à virgule flottante

et une valeur réelle

$$\underset{s}{\boxed{1}} \qquad \underset{e}{\boxed{11000110}} \qquad \underset{f}{\boxed{10010011110000111000000}}$$

$$\downarrow \qquad\qquad \downarrow \qquad\qquad \downarrow$$

$$(-1)^s \times 2^{e-B} \times 1 \bullet f$$

$$(-1)^1 \times 2^{198-127} \times 1.10010011110000111000000_2$$

$$-2^{54} \times 206727 \approx -3{,}7 \times 10^{21}$$

Nombre à virgule flottante

et une valeur réelle

$$\underset{s}{\boxed{1}} \qquad \underset{e}{\boxed{11000110}} \qquad \underset{f}{\boxed{10010011110000111000000}}$$

$$\downarrow \qquad \downarrow \qquad \downarrow$$

$$(-1)^s \times 2^{e-B} \times 1 \bullet f$$

$$(-1)^1 \times 2^{198-127} \times 1.10010011110000111000000_2$$

$$-2^{54} \times 206727 \approx -3{,}7 \times 10^{21}$$

sauf valeurs spéciales de e : 0 et le maximum $e_{\max}$

Types de nombre à virgule flottante selon la norme IEEE-754

- si $0 < e < e_{\max}$, nombre normal de valeur $(-1)^s \cdot 1.f \cdot 2^{e-B}$
- si $e = 0$, nombre dénormalisé de valeur $(-1)^s \cdot 0.f \cdot 2^{1-B}$
  $\Rightarrow$ on a $+0$ et -0
- si $e = e_{\max}$ et $f = 0$, deux valeurs $+\infty$ et $-\infty$
- si $e = e_{\max}$ et $f \neq 0$, NaN (*Not-a-Number*)

Quelques définitions

précision p : nombre de chiffre de la mantisse.
$\Rightarrow$ double précision IEEE-754 (64 bits) : $p = 53$
$\Rightarrow$ simple précision IEEE-754 (32 bits) : $p = 24$.

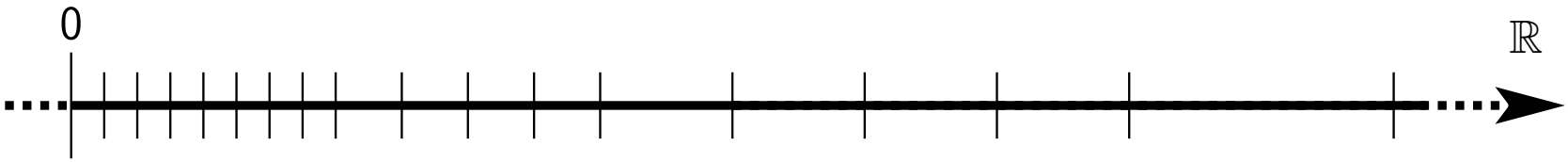
Quelques définitions

précision p : nombre de chiffre de la mantisse.
$\Rightarrow$ double précision IEEE-754 (64 bits) : $p = 53$
$\Rightarrow$ simple précision IEEE-754 (32 bits) : $p = 24$.

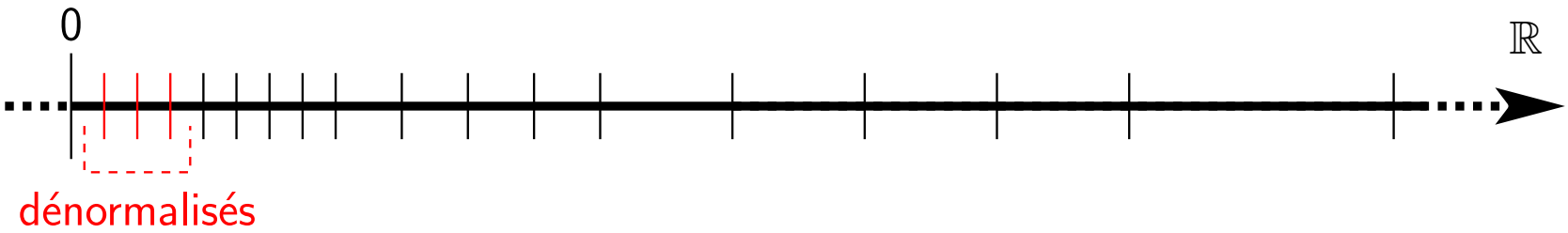
On appelle ulp la valeur du dernier bit de la mantisse d'un flottant.
Ainsi en double précision, on a $\text{ulp}(1) = 2^{-52}$ et $\text{ulp}\left(2^{-1074}\right) = 2^{-1074}$.

Lorsque l'on obtient un dénormalisé, on parle d'underflow.

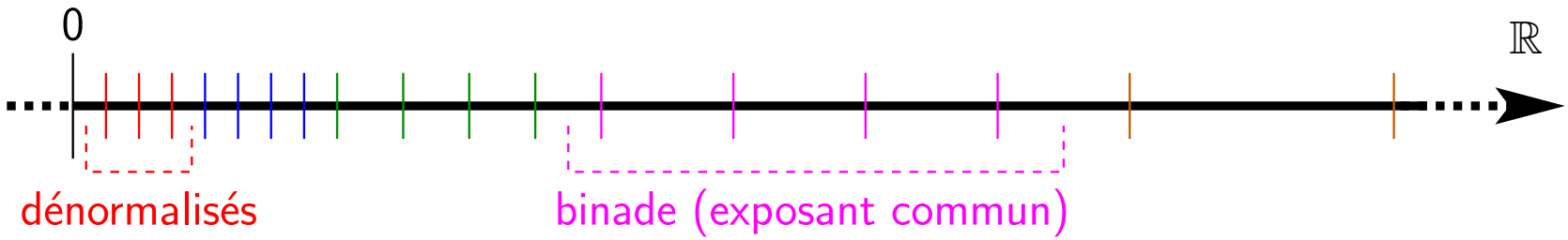
Répartition des flottants



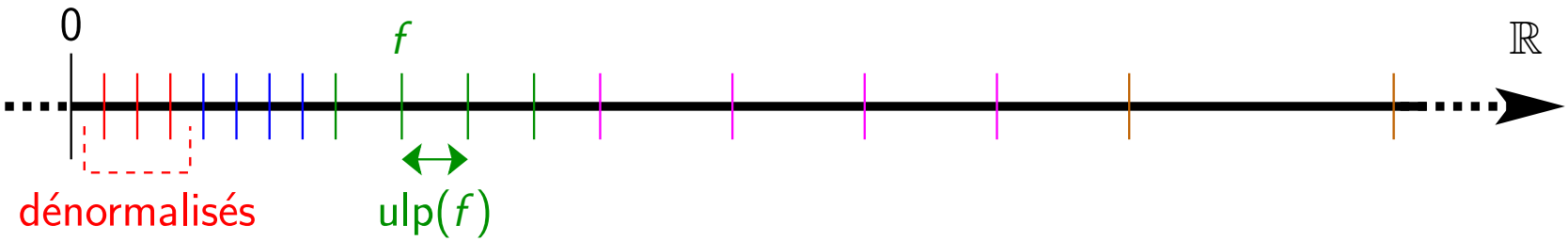
Répartition des flottants



Répartition des flottants



Répartition des flottants



Calcul et arrondis

La norme IEEE-754 définit 5 modes d'arrondi :

- arrondi au plus proche pair, noté $\circ$. Il renvoie le flottant le plus proche. Au milieu, il choisit celui qui a la mantisse paire.
- arrondi vers $-\infty$
- arrondi vers $+\infty$
- arrondi vers zéro
- arrondi au plus proche avec choix loin de zéro. Au milieu, il choisit celui qui a la plus grande valeur absolue.

Calcul et arrondis

La norme IEEE-754 définit 5 modes d'arrondi :

- arrondi au plus proche pair, noté $\circ$. Il renvoie le flottant le plus proche. Au milieu, il choisit celui qui a la mantisse paire.
- arrondi vers $-\infty$
- arrondi vers $+\infty$
- arrondi vers zéro
- arrondi au plus proche avec choix loin de zéro. Au milieu, il choisit celui qui a la plus grande valeur absolue.

Chaque calcul élémentaire (addition, soustraction, multiplication, division, racine carrée) est parfait : on obtient le même résultat que si l'on avait calculé avec une précision infinie et arrondi ensuite au format choisi.

Vers l'infini et au-delà !

On peut calculer avec les infinis et les zéros de façon intuitive : Ainsi, $1 + \infty = +\infty$ et $-3 * +\infty = -\infty$

Vers l'infini et au-delà !

On peut calculer avec les infinis et les zéros de façon intuitive : Ainsi, $1 + \infty = +\infty$ et $-3 * +\infty = -\infty$

Mais

$$\left(10^{100}\right)^2 \hookrightarrow +\infty$$

Vers l'infini et au-delà !

On peut calculer avec les infinis et les zéros de façon intuitive : Ainsi, $1 + \infty = +\infty$ et $-3 * +\infty = -\infty$

Mais

$$\left(10^{100}\right)^2 \hookrightarrow +\infty$$
$$\frac{1}{\left(10^{100}\right)^2} \hookrightarrow 0$$

Vers l'infini et au-delà !

On peut calculer avec les infinis et les zéros de façon intuitive : Ainsi, $1 + \infty = +\infty$ et $-3 * +\infty = -\infty$

Mais

$$\left(10^{100}\right)^2 \hookrightarrow +\infty$$

$$\frac{1}{\left(10^{100}\right)^2} \hookrightarrow 0$$

$$\frac{1}{\frac{1}{\left(10^{100}\right)^2}}$$

$\hookrightarrow$ BOUM

Vers l'infini et au-delà !

On peut calculer avec les infinis et les zéros de façon intuitive : Ainsi, $1 + \infty = +\infty$ et $-3 * +\infty = -\infty$

Mais

$$\left(10^{100}\right)^2 \hookrightarrow +\infty$$

$$\frac{1}{\left(10^{100}\right)^2} \hookrightarrow 0$$

$$\frac{1}{\frac{1}{\left(10^{100}\right)^2}}$$

$\hookrightarrow$ BOUM

(en fait $\hookrightarrow +\infty$ mais le programme s'arrête sur "division by zero").

NaN!

En fait, le processeur me renvoie toujours un résultat, même quand le calcul demandé n'a aucun sens.

Si je calcule $\sqrt{-1}$ ou $+\infty - \infty$, j'obtiens NaN (Not-a-Number).

NaN!

NaN apparaît aux endroits les plus incongrus :

NaN!

NaN apparaît aux endroits les plus incongrus :

NaN!

NaN apparaît aux endroits les plus incongrus :

NaN!

NaN apparaît aux endroits les plus incongrus :

NaN!

NaN apparaît aux endroits les plus incongrus :

Plan

Répartition des flottants

Définition (Nombre représentable)

Pour un format $\mathbb{F}$ de précision p et d'exposant minimal $e_{\min}$,
un nombre $x \in \mathbb{R}$ est représentable
s'il existe $m_x, e_x \in \mathbb{Z}$ tels que $x = m_x \cdot 2^{e_x}$
avec $|m_x| < 2^p$ et $e_x \geq e_{\min}$.

Répartition des flottants

Définition (Nombre représentable)

Pour un format $\mathbb{F}$ de précision p et d'exposant minimal $e_{\min}$,
un nombre $x \in \mathbb{R}$ est représentable
s'il existe $m_x, e_x \in \mathbb{Z}$ tels que $x = m_x \cdot 2^{e_x}$
avec $|m_x| < 2^p$ et $e_x \geq e_{\min}$.

Définition (Représentation canonique)

Une représentation $m \cdot 2^e$ est canonique si

- $2^{p-1} \leq |m| < 2^p$ (nombre normal)
- ou $e = e_{\min}$ (nombre dénormal)

Répartition des flottants

Lemme (Successeur)

Étant donné un nombre représentable $x = m_x \cdot 2^{e_x} \geq 0$,

1. $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}$,
2. $m_x \cdot 2^{e_x}$ *représentation canonique* $\Rightarrow \nexists z \in \mathbb{F},\ x < z < y$.

Répartition des flottants

Lemme (Successeur)

Étant donné un nombre représentable $x = m_x \cdot 2^{e_x} \geq 0$,

1. $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}$,
2. $m_x \cdot 2^{e_x}$ *représentation canonique* $\Rightarrow \nexists z \in \mathbb{F},\ x < z < y$.

Démonstration.

1. $0 \leq m_x < 2^p$ et $e_x \geq e_{\min}$,
   si $|m_x + 1| < 2^p$, alors $(m_x + 1) \cdot 2^{e_x}$ représente y,
   sinon $m_x + 1 = 2^p$ et $1 \cdot 2^{e_x + p}$ est une représentation valide de y.
2. $2^{p-1} \leq m_x < 2^p$ ou $e_x \geq e_{\min}$,
   donc $x < z < y \Rightarrow e_x > e_z \wedge m_z > 2^{e_x - ez} m_x \geq 2 m_x$.

□

Propriétés des modes d'arrondi

Lemme (Arrondi fidèle)

- $\triangledown(x) = \max\{y \in \mathbb{F} \mid y \leq x\}$,
- $\triangle(x) = \min\{y \in \mathbb{F} \mid y \geq x\}$,
- $\square(x) = \triangledown(x)$ *ou* $\square(x) = \triangle(x)$.

Propriétés des modes d'arrondi

Lemme (Arrondi fidèle)

- $\triangledown(x) = \max\{y \in \mathbb{F} \mid y \leq x\}$,
- $\triangle(x) = \min\{y \in \mathbb{F} \mid y \geq x\}$,
- $\square(x) = \triangledown(x)$ *ou* $\square(x) = \triangle(x)$.

Lemme (Idempotence)

$$\forall x \in \mathbb{F},\ \square(x) = x.$$

Propriétés des modes d'arrondi

Lemme (Arrondi fidèle)

- $\triangledown(x) = \max\{y \in \mathbb{F} \mid y \leq x\}$,
- $\triangle(x) = \min\{y \in \mathbb{F} \mid y \geq x\}$,
- $\square(x) = \triangledown(x)$ *ou* $\square(x) = \triangle(x)$.

Lemme (Idempotence)

$$\forall x \in \mathbb{F},\ \square(x) = x.$$

Lemme (Monotonie locale)

$$\forall x, y \in \mathbb{R},\ y \in [\square(x), x] \Rightarrow \square(y) = \square(x).$$

Propriétés des modes d'arrondi

Lemme (Monotonie)

$$\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \square(x) \leq \square(y).$$

Propriétés des modes d'arrondi

Lemme (Monotonie)

$$\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \square(x) \leq \square(y).$$

Démonstration.

- Si $\bigtriangledown(x) < \bigtriangledown(y)$,
  1. $x < \bigtriangledown(y)$ par définition de $\bigtriangledown(x)$,
  2. $\square(x) \leq \bigtriangleup(x) \leq \bigtriangledown(y) \leq \square(y)$.

Propriétés des modes d'arrondi

Lemme (Monotonie)

$$\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \square(x) \leq \square(y).$$

Démonstration.

- Si $\bigtriangledown(x) < \bigtriangledown(y)$,
  1. $x < \bigtriangledown(y)$ par définition de $\bigtriangledown(x)$,
  2. $\square(x) \leq \bigtriangleup(x) \leq \bigtriangledown(y) \leq \square(y)$.
- Si $\bigtriangledown(x) \geq \bigtriangledown(y)$,
  1. $\bigtriangledown(x) = \bigtriangledown(y)$ par définition de $\bigtriangledown(y)$,
  2. $\bigtriangleup(x) = \bigtriangleup(y)$ par idempotence ou successeur,
  3. si $\square(y) = \bigtriangleup(y)$, alors $\square(x) \leq \square(y)$,
  4. sinon $\square(x) = \square(y)$ par monotonie locale.

Propriétés des modes d'arrondi

Lemme (Monotonie)

$$\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \square(x) \leq \square(y).$$

Corollaire (Comparaison avec un nombre représentable)

$$\forall x \in \mathbb{F},\ \forall y \in \mathbb{R},\ x \leq y \Rightarrow x \leq \square(y).$$

Bornes d'erreur

Lemme (Erreur d'arrondi en arrondi au plus près – modèle standard)

Pour tout $x \in \mathbb{R}$, il existe ε et δ tel que

$$\circ(x) = x \cdot (1 + \varepsilon) + \delta \quad et \quad |\varepsilon| \leq 2^{-p} = u \quad et \quad |\delta| \leq 2^{e_{\min}-1}.$$

Qui plus est, $\delta = 0$ ou $\varepsilon = 0$.

Bornes d'erreur

Lemme (Erreur d'arrondi en arrondi au plus près – modèle standard)

Pour tout $x \in \mathbb{R}$, il existe ε et δ tel que

$$\circ(x) = x \cdot (1+\varepsilon) + \delta \quad \textit{et} \quad |\varepsilon| \leq 2^{-p} = u \quad \textit{et} \quad |\delta| \leq 2^{e_{\min}-1}.$$

Qui plus est, $\delta = 0$ ou $\varepsilon = 0$.

Démonstration.

1. Supposition : $0 < x \notin \mathbb{F}$.
2. $\bigtriangledown(x) = m \cdot 2^e$ et $\bigtriangleup(x) = (m+1) \cdot 2^{e+1}$.
3. $|\circ(x) - x| \leq (\bigtriangleup(x) - \bigtriangledown(x))/2 = 2^{e-1}$.
   - Si $\bigtriangledown(x)$ est dénormal, $e = e_{\min}$.
     $\varepsilon = 0$ et $\delta = \circ(x) - x$ d'où $|\delta| \leq 2^{e_{\min}-1}$.
   - Si $\bigtriangledown(x)$ est normal, $2^{p-1} \leq m$.
     $\delta = 0$ et $\varepsilon = (\circ(x) - x)/x$ d'où $|\varepsilon| \leq 2^{e-1}/(2^{p-1} \cdot 2^e) = 2^{-p}$.

Bornes d'erreur

Lemme (Erreur d'arrondi en arrondi au plus près – modèle standard)

Pour tout $x \in \mathbb{R}$, il existe ε et δ tel que

$$\circ(x) = x \cdot (1 + \varepsilon) + \delta \quad et \quad |\varepsilon| \leq 2^{-p} = u \quad et \quad |\delta| \leq 2^{e_{\min}-1}.$$

Qui plus est, $\delta = 0$ ou $\varepsilon = 0$.

Lemme (Erreur d'arrondi en arrondi dirigé)

Pour tout $x \in \mathbb{R}$, il existe ε et δ tel que

$$\square(x) = x \cdot (1 + \varepsilon) + \delta \quad et \quad |\varepsilon| < 2^{-p+1} \quad et \quad |\delta| < 2^{e_{\min}}.$$

Qui plus est, $\delta = 0$ ou $\varepsilon = 0$.

Addition dénormalisée

Lemme (Exactitude de l'addition dénormalisée)

$$\forall x, y \in \mathbb{F},\ |x + y| \leq 2^{e_{\min}+p} \Rightarrow x + y \in \mathbb{F}.$$

Addition dénormalisée

Lemme (Exactitude de l'addition dénormalisée)

$$\forall x, y \in \mathbb{F},\ |x+y| \leq 2^{e_{\min}+p} \Rightarrow x+y \in \mathbb{F}.$$

Démonstration.

1. $x = m_x \cdot 2^{e_x}$ et $y = m_y \cdot 2^{e_y}$.
2. $m = m_x \cdot 2^{e_x - e_{\min}} + m_y \cdot 2^{e_y - e_{\min}}$ et $x + y = m \cdot 2^{e_{\min}}$.
3. $|m| \leq 2^p$ donc $x + y$ est représentable.

□

Addition dénormalisée

Lemme (Exactitude de l'addition dénormalisée)

$$\forall x, y \in \mathbb{F},\ |x+y| \leq 2^{e_{\min}+p} \Rightarrow x+y \in \mathbb{F}.$$

Démonstration.

1. $x = m_x \cdot 2^{e_x}$ et $y = m_y \cdot 2^{e_y}$.
2. $m = m_x \cdot 2^{e_x - e_{\min}} + m_y \cdot 2^{e_y - e_{\min}}$ et $x + y = m \cdot 2^{e_{\min}}$.
3. $|m| \leq 2^p$ donc $x + y$ est représentable.

□

Corollaire (Erreur d'arrondi pour l'addition)

$$\forall x, y \in \mathbb{F},\ \exists \varepsilon,\ \circ(x+y) = (x+y)\cdot(1+\varepsilon) \quad et \quad |\varepsilon| \leq 2^{-p}.$$

Plan

Une norme, à quoi bon ?

Protagonistes :

- langage de programmation,
- compilateur,
- système d'exploitation,
- processeur.

Chacun peut perturber le comportement d'un algorithme.

Parenthésage : le cas Fortran

Norme Fortran 77 (6.6.4)

Once the interpretation has been established in accordance with those rules, the processor may evaluate any mathematically equivalent expression, provided that the integrity of parentheses is not violated.

Parenthésage : le cas Fortran

Norme Fortran 77 (6.6.4)

Once the interpretation has been established in accordance with those rules, the processor may evaluate any mathematically equivalent expression, provided that the integrity of parentheses is not violated.

Exemple (Addition)

L'expression `a+b+c+d` peut être compilée en

- `((a+b)+c)+d` : interprétation naturelle,
- `(a+b)+(c+d)` : parallélisme,
- `a+(b+(c+d))` : pourquoi pas ?

Non-associativité des calculs flottants

Exemple

Évaluation de `delta + 1. + (-1.)` avec `delta` $= 2^{-p}$.

- $\circ(\circ(\delta + 1) - 1) = \circ(1 - 1) = 0$.
- $\circ(\delta + \circ(1 - 1)) = \circ(\delta + 0) = \delta$.

Précision intermédiaire : le cas C

Norme C99 (5.2.4.2.2 §8)

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.

Précision intermédiaire : le cas C

Norme C99 (5.2.4.2.2 §8)

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.

Exemple (Précision étendue)

```
double x = 1.0;
double y = 0x1p-53 + 0x1p-64;
double z = x + y;
```

Précision intermédiaire : le cas C

Norme C99 (5.2.4.2.2 §8)

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.

Exemple (Précision étendue)

```
double x = 1.0;
double y = 0x1p-53 + 0x1p-64;
double z = x + y;
```

peut produire les valeurs suivantes :

- $z = 1 + 2^{-52}$: addition en `binary64`,
- $z = 1$: addition en `binary80` puis stockage en `binary64`.

Précision intermédiaire : le cas C

Norme C99 (5.2.4.2.2 §8)

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.

Exemple (Précision étendue)

```
double x = 1.0;
double y = 0x1p-53 + 0x1p-64;
double z = x + y;
```

peut produire les valeurs suivantes :

- $z = 1 + 2^{-52}$: addition en `binary64`,
- $z = 1$: addition en `binary80` puis stockage en `binary64`.
  Précision supérieure $\not\Rightarrow$ résultat plus précis !

Bornes d'erreur modulo langages et compilateurs

Objectif : obtenir une borne d'erreur valable pour n'importe quel parenthésage et/ou précision intermédiaire potentiellement étendue de n additions en effectuant une seule analyse d'erreur.

Bornes d'erreur modulo langages et compilateurs

Objectif : obtenir une borne d'erreur valable pour n'importe quel parenthésage et/ou précision intermédiaire potentiellement étendue de n additions en effectuant une seule analyse d'erreur.

Nouvelle borne pour l'addition :

$$|\circ(x+y)-(x+y)| \leq \varepsilon(|x|+|y|)+\delta$$

avec $\varepsilon \gtrsim n2^{-p}$ et $\delta = n2^{e_{\min}}$.

Le cas du FMA

Définition (Fused-multiply-add)

Multiplication puis addition sans arrondi intermédiaire :

$$\mathrm{fma}(a, b, c) = \square(a \times b + c).$$

Disponible sur les architectures modernes.

Le cas du FMA

Définition (Fused-multiply-add)

Multiplication puis addition sans arrondi intermédiaire :

$$\mathrm{fma}(a, b, c) = \square(a \times b + c).$$

Disponible sur les architectures modernes.

Exemple (Optimisation à l'aide du FMA)

L'expression `a*b+c*d` peut se compiler vers

- `(a*b)+(c*d)`,
- `fma(a,b,c*d)`,
- `fma(c,d,a*b)`.

Le cas du FMA

Exemple (Optimisation à l'aide du FMA)

```
double f(double a, double b) {
  return a >= b ? sqrt(a * a - b * b) : 0;
}
```

Le cas du FMA

Exemple (Optimisation à l'aide du FMA)

```
double f(double a, double b) {
  return a >= b ? sqrt(a * a - b * b) : 0;
}
```

- $\circ(\circ(a^2) - \circ(b^2)) \geq 0$: tout va bien.
- `sqrt(fma(a,a,-b*b))` : potentiellement NaN quand $a = b$.

Plan

Plan

Sterbenz

Théorème (Sterbenz)

Soient a et b dans $\mathbb{F}$ tels que

$$\frac{b}{2} \leq a \leq 2b.$$

Alors le réel $a - b$ est représentable.
En particulier, il est calculé sans erreur par la soustraction flottante.

Erreur de l'addition I

Théorème (FastTwoSum)

Soient a et b deux flottants. On calcule

$$\begin{aligned} r_1 &= \circ(a+b) \\ b' &= \circ(r_1 - a) \\ r_2 &= \circ(b - b') \end{aligned}$$

Alors, si $|a| \geq |b|$, on a l'égalité mathématique $a + b = r_1 + r_2$.

Erreur de l'addition II

Théorème (TwoSum)

Soient a et b deux flottants. On calcule

$$\begin{aligned} r_1 &= \circ(a+b) \\ b' &= \circ(r_1 - a) \\ a' &= \circ(r_1 - b') \\ \epsilon_b &= \circ(b - b') \\ \epsilon_a &= \circ(a - a') \\ r_2 &= \circ(\epsilon_a + \epsilon_b) \end{aligned}$$

Alors on a l'égalité mathématique $a + b = r_1 + r_2$.

Veltkamp/Dekker

Théorème (Veltkamp/Dekker)

S'il n'y a ni underflow, ni overflow, il y a un algorithme n'utilisant que des calculs flottants qui calcule l'erreur exacte de la multiplication.

Veltkamp/Dekker

Théorème (Veltkamp/Dekker)

S'il n'y a ni underflow, ni overflow, il y a un algorithme n'utilisant que des calculs flottants qui calcule l'erreur exacte de la multiplication.

Idée :

couper les flottants en 2, multiplier les morceaux, ajouter dans le bon ordre

Veltkamp/Dekker – programme

```
/*@ requires xy == \round_double(\NearestEven,x*y) &&
  @          \abs(x) <= 0x1.p995 &&
  @          \abs(y) <= 0x1.p995 &&
  @          \abs(x*y) <= 0x1.p1021;
  @ ensures  ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Veltkamp/Dekker – programme

$xy = \circ(x \times y)$

```
/*@ requires  xy == \round_double(\NearestEven,x*y) &&
  @           \abs(x) <= 0x1.p995 &&
  @           \abs(y) <= 0x1.p995 &&
  @           \abs(x*y) <= 0x1.p1021;
  @ ensures   ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                 ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert  C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Veltkamp/Dekker – programme

```
/*@ requires  xy == \round_double(\NearestEven,x*y) &&
  @           \abs(x) <= 0x1.p995 &&
  @           \abs(y) <= 0x1.p995 &&
  @           \abs(x*y) <= 0x1.p1021;
  @ ensures   ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                   ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert  C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Overflow

Veltkamp/Dekker – programme

```
/*@ requires xy == \round_double(\NearestEven,x*y) &&
  @          \abs(x) <= 0x1.p995 &&
  @          \abs(y) <= 0x1.p995 &&
  @          \abs(x*y) <= 0x1.p1021;
  @ ensures  ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Si pas d'underflow,

Veltkamp/Dekker – programme

```
/*@ requires xy == \round_double(\NearestEven,x*y) &&
  @          \abs(x) <= 0x1.p995 &&
  @          \abs(y) <= 0x1.p995 &&
  @          \abs(x*y) <= 0x1.p1021;
  @ ensures  ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                  ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

on a l'erreur exacte

Veltkamp/Dekker – programme

```
/*@ requires xy == \round_double(\NearestEven,x*y) &&
  @          \abs(x) <= 0x1.p995 &&
  @          \abs(y) <= 0x1.p995 &&
  @          \abs(x*y) <= 0x1.p1021;
  @ ensures  ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Couper x et y

Veltkamp/Dekker – programme

```
/*@ requires xy == \round_double(\NearestEven,x*y) &&
  @          \abs(x) <= 0x1.p995 &&
  @          \abs(y) <= 0x1.p995 &&
  @          \abs(x*y) <= 0x1.p1021;
  @ ensures  ((x*y == 0 || 0x1.p-969 <= \abs(x*y))
  @                ==> x*y == xy+\result);
  @*/
double Dekker(double x, double y, double xy) {

  double C,px,qx,hx,py,qy,hy,tx,ty,r2;
  C=0x8000001p0;
  /*@ assert C == 0x1p27+1; */

  px=x*C; qx=x-px; hx=px+qx; tx=x-hx;

  py=y*C; qy=y-py; hy=py+qy; ty=y-hy;

  r2=-xy+hx*hy;
  r2+=hx*ty;
  r2+=hy*tx;
  r2+=tx*ty;
  return r2;
}
```

Multiplier les moitiés
et ajouter les résultats

Erreur de la multiplication avec un FMA

Avec un FMA, calculer l'erreur de la multiplication devient trivial :

Théorème (FastTwoMult)

Soient a et b deux flottants. On calcule :

$$\begin{aligned} r_1 &= \circ(a \times b) \\ r_2 &= \circ(a \times b - r_1) \end{aligned}$$

Alors $a \times b = r_1 + r_2$.

Erreur de la multiplication avec un FMA

Avec un FMA, calculer l'erreur de la multiplication devient trivial :

Théorème (FastTwoMult)

Soient a et b deux flottants. On calcule :

$$\begin{aligned} r_1 &= \circ(a \times b) \\ r_2 &= \circ(a \times b - r_1) \end{aligned}$$

Alors $a \times b = r_1 + r_2$.

L'erreur est parfois trop petite pour être représentable.
$2^{e_{\min}} \times 2^{-1}$ est arrondi en 0 et l'erreur est $2^{e_{\min}-1}$ pas représentable.

Erreur du FMA

Théorème (FmaErr)

Soient a, x et y des flottants. On calcule

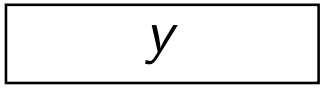
$$\begin{aligned} r_1 &= \circ(ax+y) \\ (u_1,u_2) &= \mathrm{FastTwoMult}(a,x) \\ (\alpha_1,\alpha_2) &= \mathrm{TwoSum}(y,u_2) \\ (\beta_1,\beta_2) &= \mathrm{TwoSum}(u_1,\alpha_1) \\ \gamma &= \circ(\circ(\beta_1-r_1)+\beta_2) \\ (r_2,r_3) &= \mathrm{FastTwoSum}(\gamma,\alpha_2) \end{aligned}$$

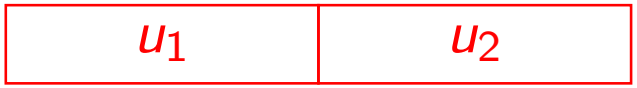
Alors, si l'erreur de la multiplication du FMA est représentable, on a

$$ax+y = r_1+r_2+r_3.$$

De plus, r_1, r_2 et r_3 vérifient $|r_2+r_3| \leq \frac{1}{2}ulp(r_1)$ et $|r_3| \leq \frac{1}{2}ulp(r_2)$.

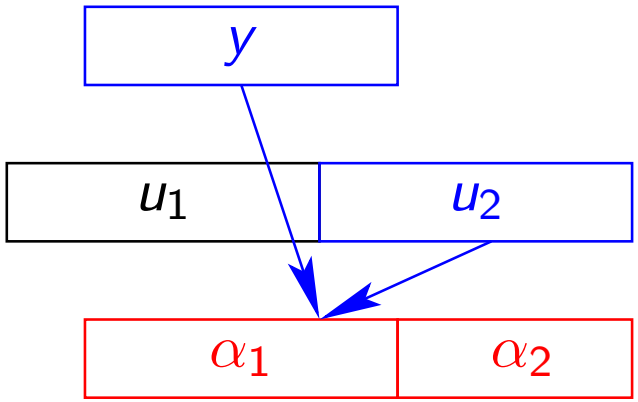
Algorithme ErrFmac : comment fonctionne-t-il ?





$$u_1 + u_2 = ax$$

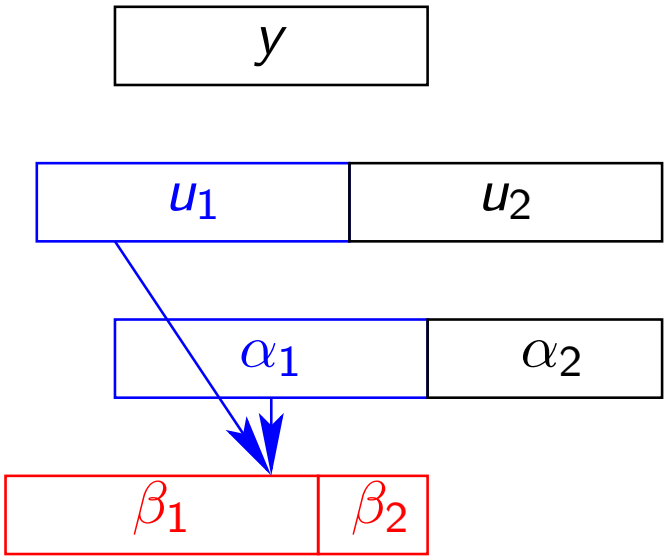
Algorithme ErrFmac : comment fonctionne-t-il ?



$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

Algorithme ErrFmac : comment fonctionne-t-il ?

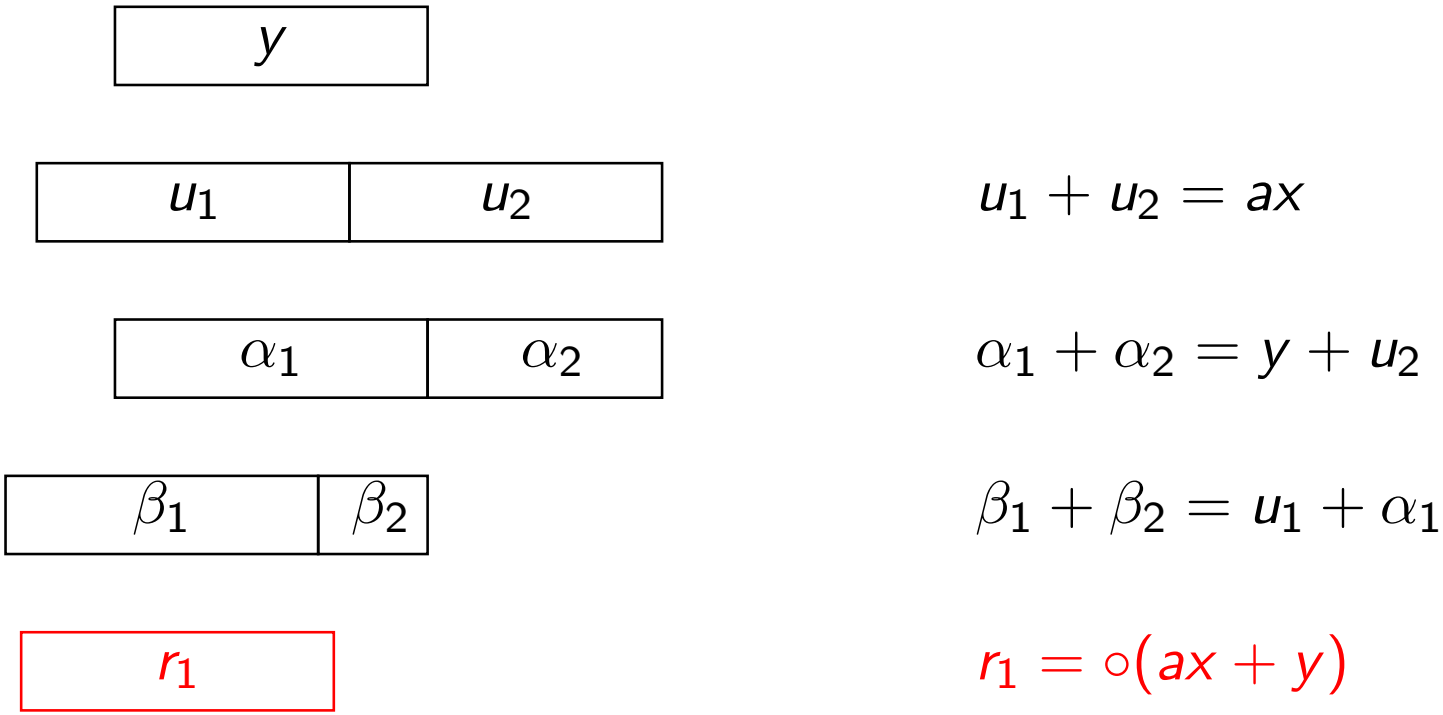


$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

Algorithme ErrFmac : comment fonctionne-t-il ?



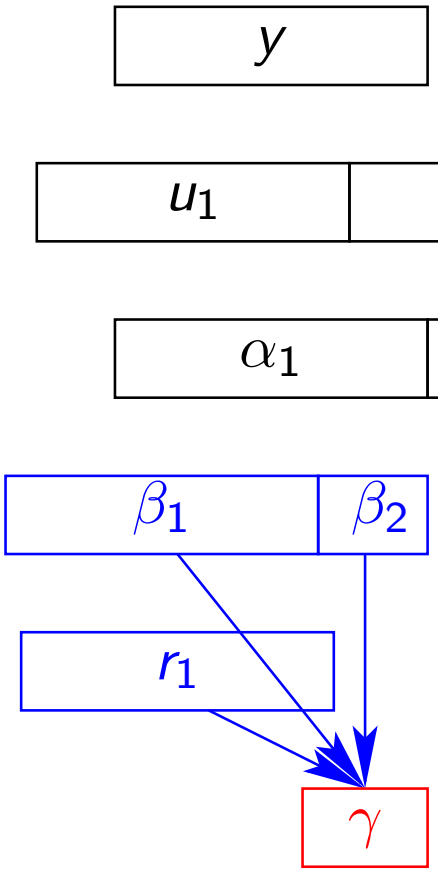
$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

$$r_1 = \circ(ax + y)$$

Algorithme ErrFmac : comment fonctionne-t-il ?



$$u_1 + u_2 = ax$$

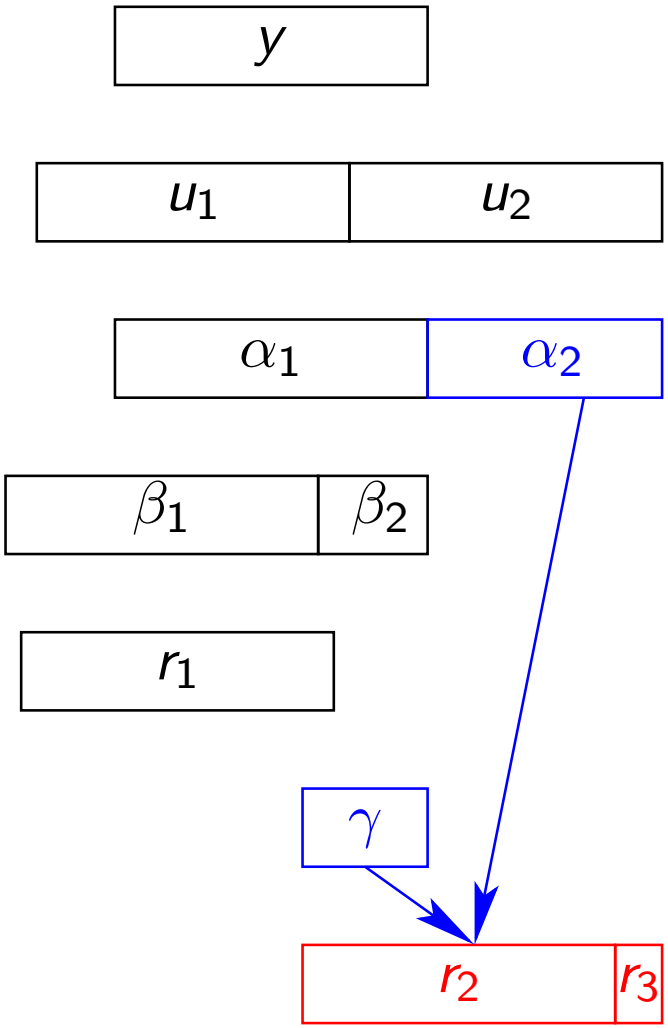
$$\alpha_1 + \alpha_2 = y + u_2$$

$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

$$r_1 = \circ(ax + y)$$

$$\gamma = \circ(\circ(\beta_1 - r_1) + \beta_2)$$

Algorithme ErrFmac : comment fonctionne-t-il ?



$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

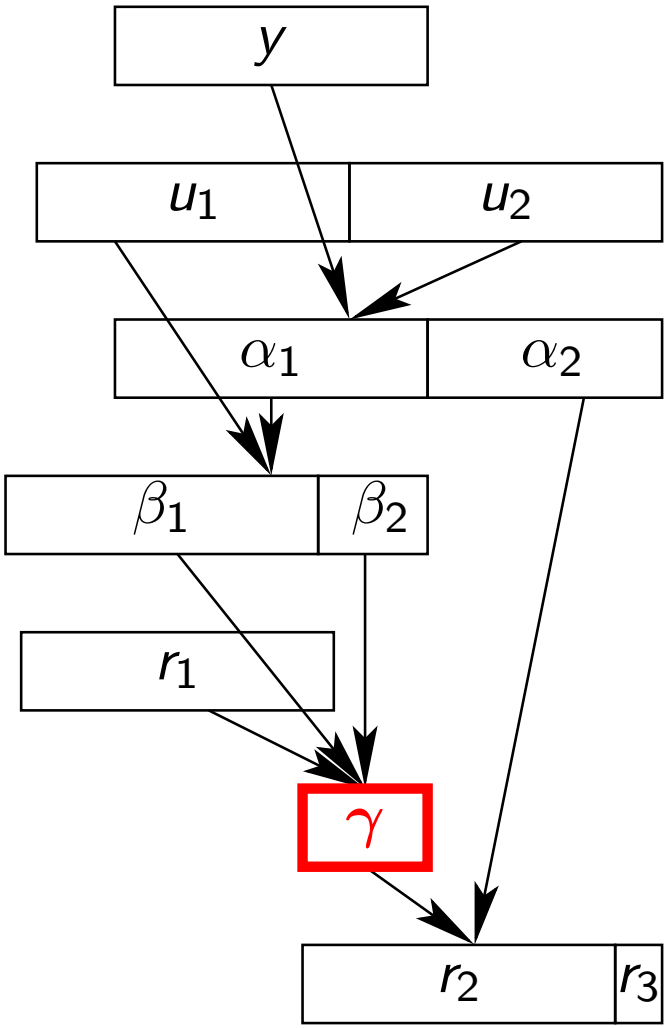
$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

$$r_1 = \circ(ax + y)$$

$$\gamma = \circ(\circ(\beta_1 - r_1) + \beta_2)$$

$$r_2 + r_3 = \gamma + \alpha_2$$

Algorithme ErrFmac : γ n'a pas d'erreur d'arrondi



$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

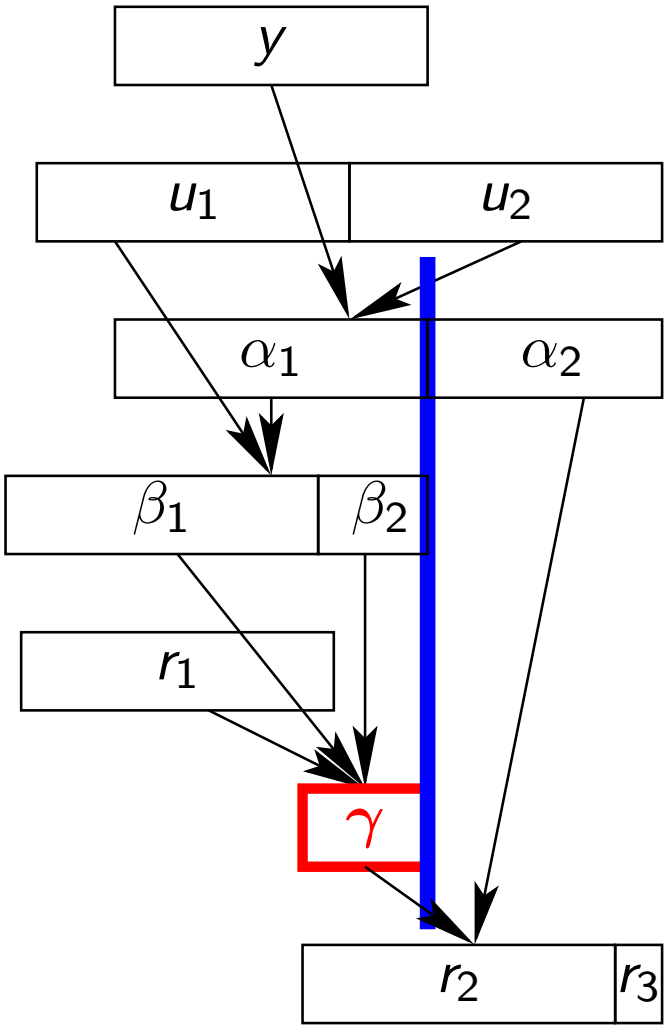
$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

$$r_1 = \circ(ax + y)$$

$$\gamma = \circ(\circ(\beta_1 - r_1) + \beta_2)$$

$$r_2 + r_3 = \gamma + \alpha_2$$

Algorithme ErrFmac : γ n'a pas d'erreur d'arrondi



$$u_1 + u_2 = ax$$

$$\alpha_1 + \alpha_2 = y + u_2$$

$$\beta_1 + \beta_2 = u_1 + \alpha_1$$

$$r_1 = \circ(ax + y)$$

$$\gamma = \circ(\circ(\beta_1 - r_1) + \beta_2)$$

$$r_2 + r_3 = \gamma + \alpha_2$$

Plan

Discriminant précis

Il est très difficile de calculer précisément $b^2 - ac$.

Discriminant précis

Il est très difficile de calculer précisément $b^2 - ac$.

Théorème (Kahan)

S'il n'y a ni overflow, ni underflow, il y a un algorithme qui calcule $b^2 - a \times c$ à 2 ulps près.

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Underflow

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Overflow

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

2 ulps

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discriminant(double a, double b, double c) {
  double p,q,d,dp,dq;
  p=b*b;
  q=a*c;
  i
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Appels de fonctions

⇒ pré-conditions à prouver

⇒ post-conditions garanties

Discriminant précis – programme

```
/*@ requires (b==0.   || 0x1.p-916 <= \abs(b*b)) &&
  @          (a*c==0. || 0x1.p-916 <= \abs(a*c)) &&
  @          \abs(b) <= 0x1.p510 &&
  @          \abs(a) <= 0x1.p995 && \abs(c) <= 0x1.p995 &&
  @          \abs(a*c) <= 0x1.p1021;
  @ ensures  \result ==0. || \abs(\result -(b*b-a*c)) <= 2.*ulp(\result);
  @ */
double discrimi                                          {
  double p,q,d;
  p=b*b;
  q=a*c;

  if (p+q <= 3*fabs(p-q))
    d=p-q;
  else {
    dp=Dekker(b,b,p);
    dq=Dekker(a,c,q);
    d=(p-q)+(dp-dq);
  }
  return d;
}
```

Dans la preuve initiale, test supposé correct

⇒ Preuve supplémentaire quand le test est incorrect

Cas particulier de la méthode de Horner

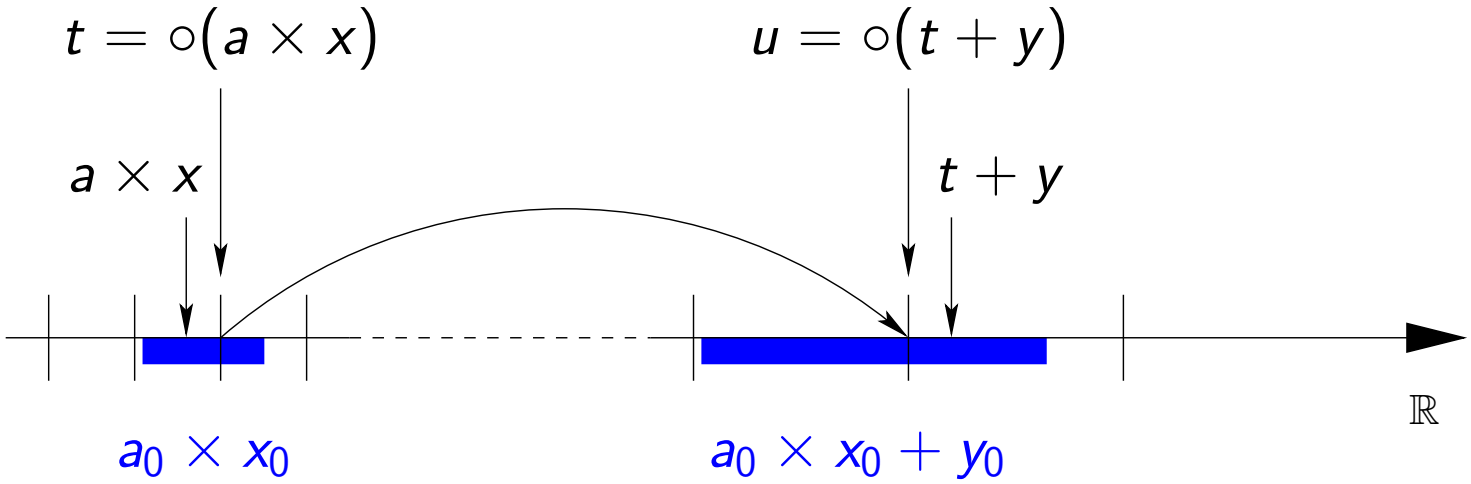
Évaluons par la méthode de Horner un polynôme tel que `1+x+x*x/2+...` pour x petit.

Cas particulier de la méthode de Horner

Évaluons par la méthode de Horner un polynôme tel que `1+x+x*x/2+...` pour x petit.

$\Rightarrow$ les précédentes erreurs sont négligeables car multipliées par x
$\Rightarrow$ l'erreur n'est (presque) que celle de la dernière étape
$\Rightarrow$ le calcul est presque juste.

Cas particulier de la méthode de Horner



Précision de la méthode de Horner

Théorème (Axpy)

Soient les réels a_0, x_0 et y_0. Soient les nombres flottants représentables a, x et y avec une précision $p \geq 6$. Les nombres flottants représentables t et u sont définis par $t = \circ(a \times x)$ et $u = \circ(t + y)$. Si

$$\left(3 + 2^{4-p}\right) \times |a \times x| \leq |y|, \text{ et}$$

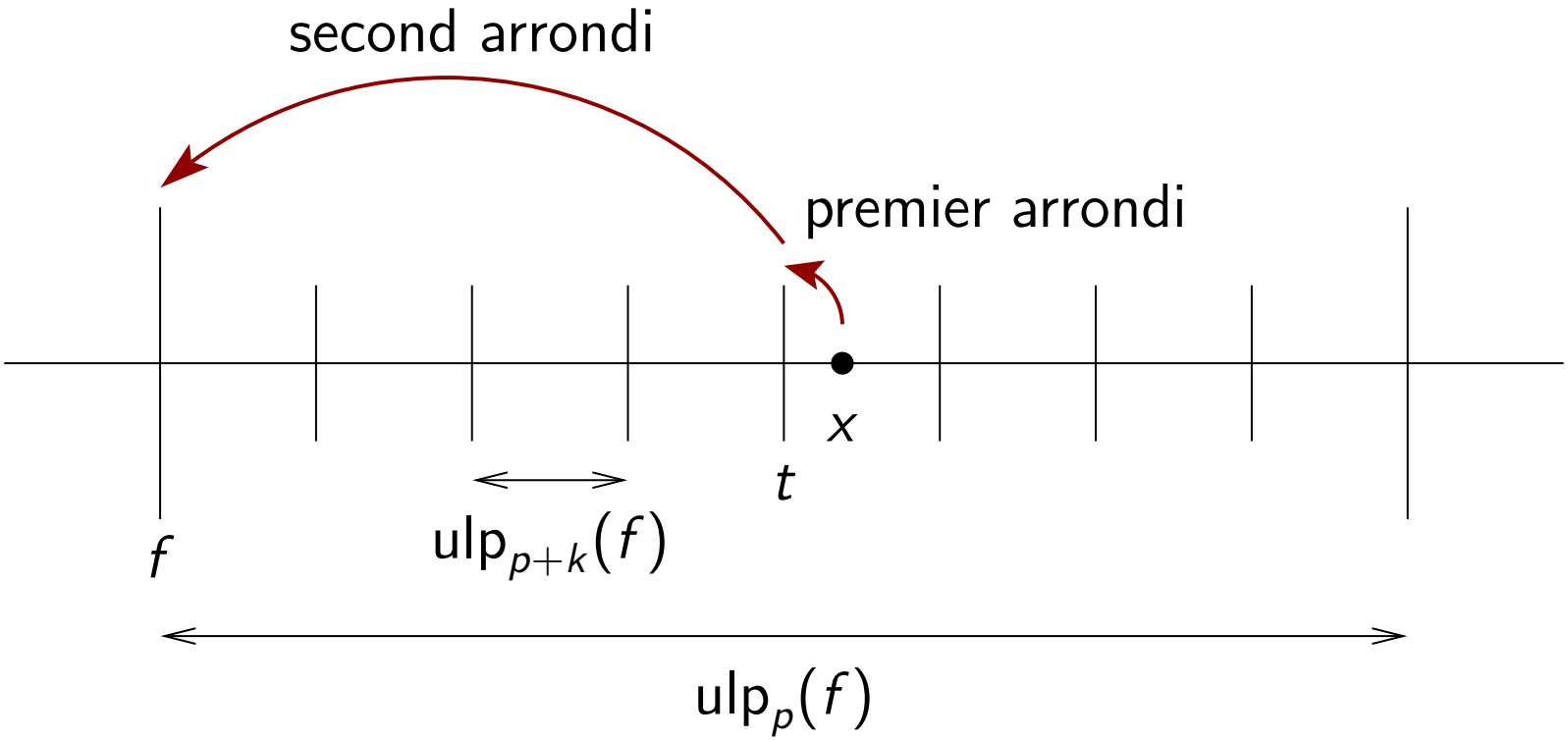
$$|y_0 - y| + |a_0 \times x_0 - a \times x| < \frac{2^{1-p}}{12} \times |y|,$$

alors $u = \square(a_0 \times x_0 + y_0)$.

Sous conditions, on calcule l'arrondi fidèle du réel exact voulu.

Double arrondi

Rappel : un double arrondi peut conduire au « mauvais » résultat, pour $\circ_p(\circ_{p+k}(x))$:



Arrondi impair

$$\square_{\mathrm{odd}}(x) = \begin{cases} x & \text{si } x \in \mathbb{F}, \\ \triangle(x) & \text{si sa mantisse est impaire,} \\ \triangledown(x) & \text{sinon.} \end{cases}$$

Arrondi impair

$$\square_{\mathrm{odd}}(x) = \begin{cases} x & \text{si } x \in \mathbb{F}, \\ \triangle(x) & \text{si sa mantisse est impaire,} \\ \triangledown(x) & \text{sinon.} \end{cases}$$

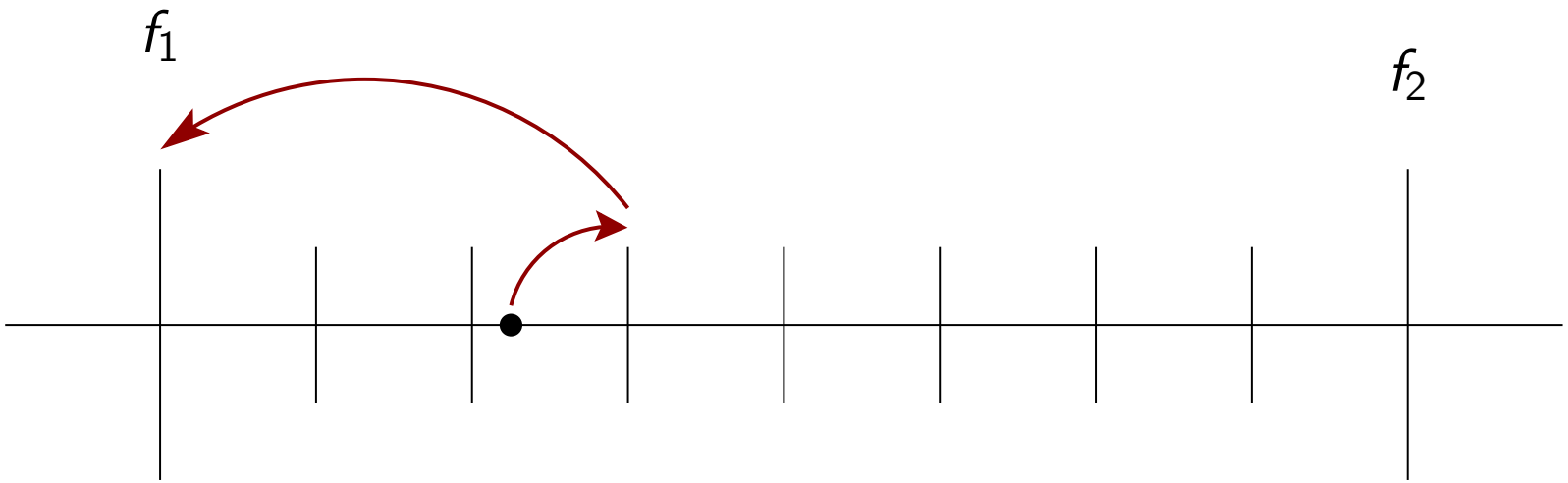
$\Rightarrow$ arrondi fidèle

$\Rightarrow$ le résultat est impair sauf s'il est exact

$\Rightarrow$ facile à calculer en matériel (avec test en logiciel)

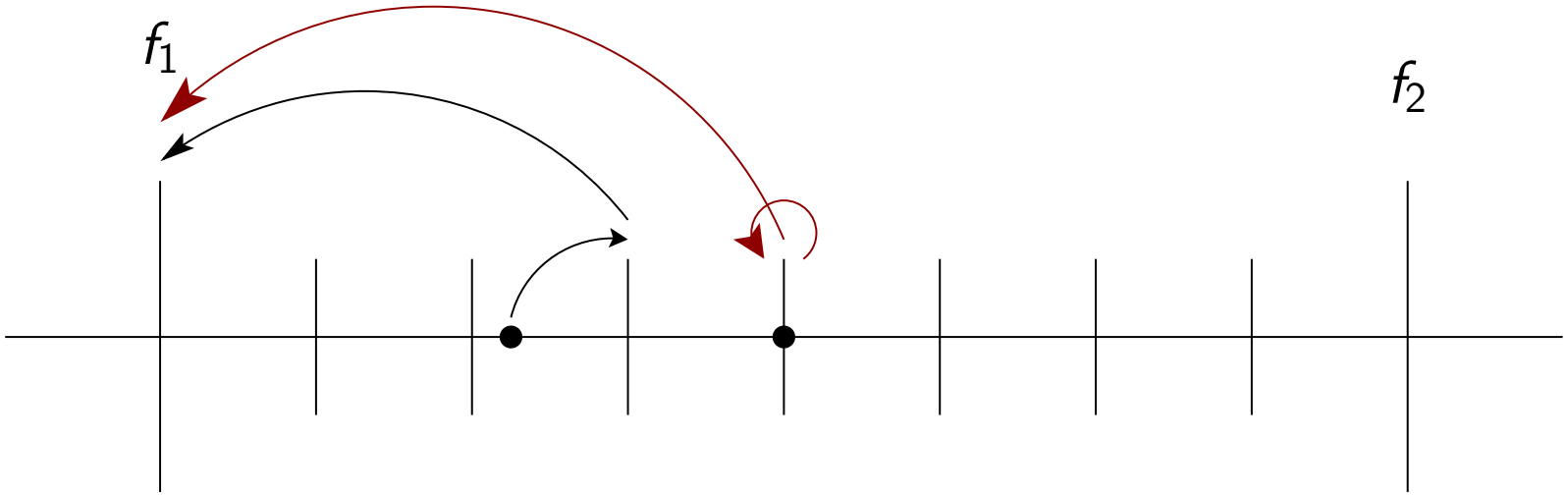
Double arrondi avec un premier arrondi impair

On calcule $\circ^p\left(\square^{p+k}_{\text{odd}}(x)\right)$:



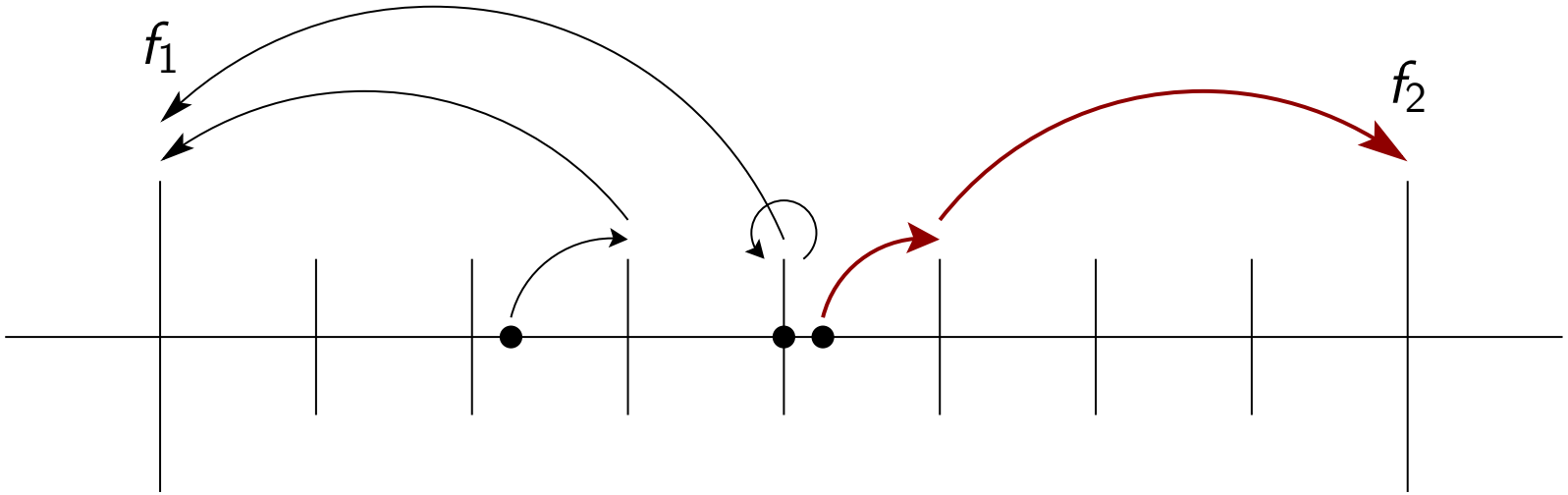
Double arrondi avec un premier arrondi impair

On calcule $\circ^p\left(\square^{p+k}_{\text{odd}}(x)\right)$:



Double arrondi avec un premier arrondi impair

On calcule $\circ^p\left(\square^{p+k}_{\text{odd}}(x)\right)$:



Double arrondi avec un premier arrondi impair

Théorème (To_Odd_Even_is_Even)

Si $p \geq 2$, $k \geq 2$ *et* $e_{\min}^{\text{ext}} \leq e_{\min} - 2$, *alors*

$$\forall x \in \mathbb{R},\ \circ^{p}\left(\square_{\text{odd}}^{p+k}(x)\right) = \circ^{p}(x).$$

Double arrondi avec un premier arrondi impair

Théorème (To_Odd_Even_is_Even)

Si $p \geq 2$, $k \geq 2$ *et* $e_{\min}^{\text{ext}} \leq e_{\min} - 2$, *alors*

$$\forall x \in \mathbb{R},\ \circ^p \left(\square_{\text{odd}}^{p+k}(x) \right) = \circ^p(x).$$

On s'en sert pour calculer $\circ(x + y + z)$ et $\circ(a \times x + y)$ et dans des conversions base 10/base 2 en compilation.

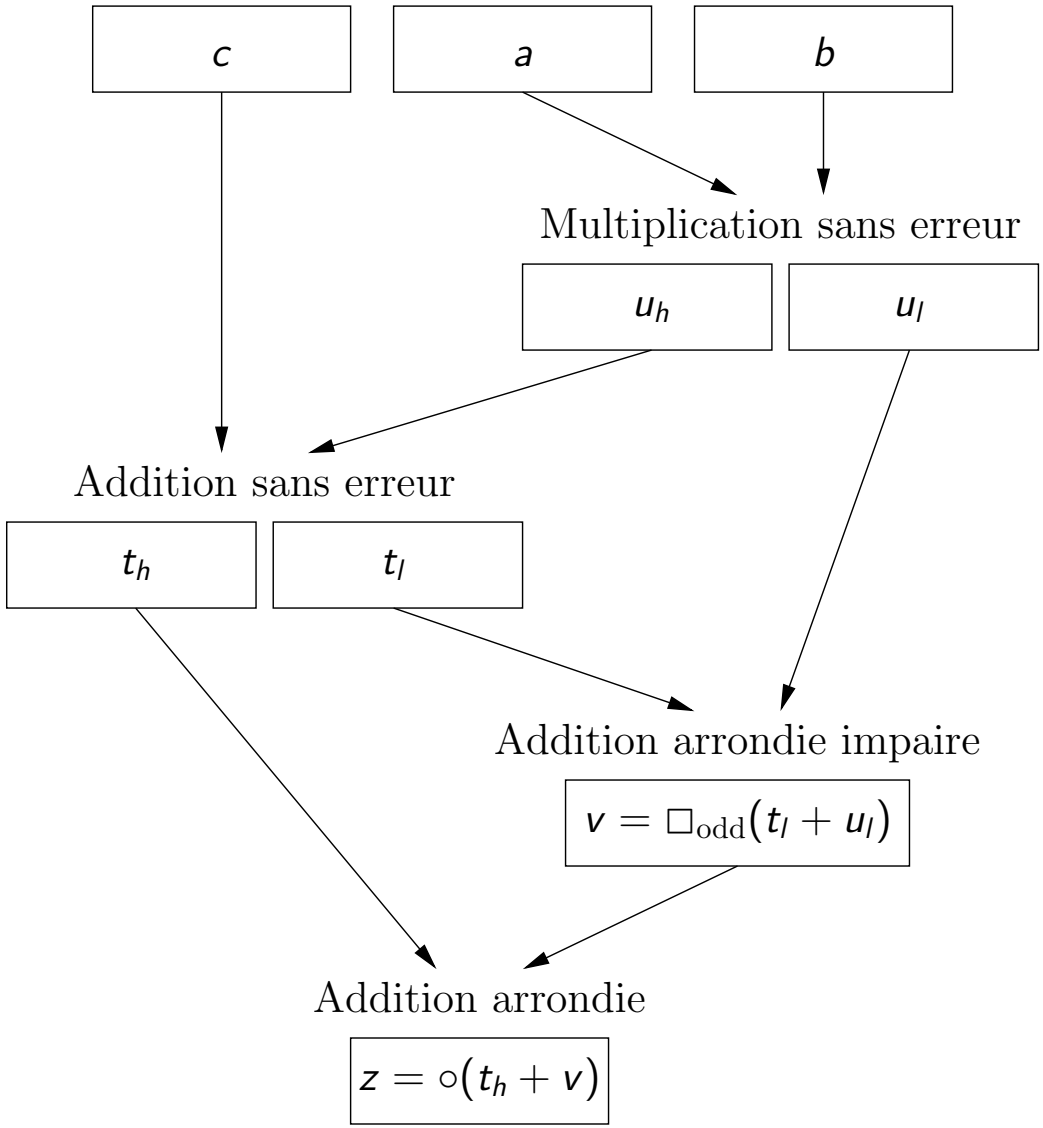
Émulation du FMA

Théorème (FmaErr)

Soient a, x et y des flottants.

$$\begin{aligned}(u_h, u_l) &= \mathrm{ExactMult}(a, x)\\ (t_h, t_l) &= \mathrm{TwoSum}(y, u_h)\\ v &= \square_{\mathrm{odd}}(t_l + u_l)\\ z &= \circ(t_h + v)\end{aligned}$$

Alors, si l'erreur de la multiplication du FMA est représentable et que la précision $p \geq 5$, on a $z = \circ(a \times x + y)$.



Division de l'Itanium

Sur l'IA-64, les opérateurs flottants aident aux opérations entières.
$\Rightarrow$ pour a/b, on utilise une table qui donne $1/b$ approché à 2^{-8}.
$\Rightarrow$ itération pour doubler la précision.

Division de l'Itanium

Sur l'IA-64, les opérateurs flottants aident aux opérations entières.
$\Rightarrow$ pour a/b, on utilise une table qui donne $1/b$ approché à 2^{-8}.
$\Rightarrow$ itération pour doubler la précision.

Itération usuelle peut être inférieure au résultat exact $\hookrightarrow$ mal arrondie
$\Rightarrow$ ajout d'une constante magique

Division de l'Itanium

Théorème

Soient a et b des entiers 16 bits. On calcule en précision $p = 64$:

$$\begin{aligned} y_0 &= \mathrm{ApproxTable}(1/b) \\ q_0 &= \circ(a \times y_0) \\ e_0 &= \circ(1 + 2^{-17} - b \times y_0) \\ q_1 &= \circ(q_0 + e_0 \times q_0) \\ q &= \lfloor q_1 \rfloor \end{aligned}$$

Alors, $q = \lfloor a/b \rfloor$.

Plan

Conclusion

Les calculs sur ordinateur,

Conclusion

Les calculs sur ordinateur,

- c'est rapide,

Conclusion

Les calculs sur ordinateur,

- c'est rapide,
- mais ce n'est pas exact.

Conclusion

Les calculs sur ordinateur,

- c'est rapide,
- mais ce n'est pas exact.

Il faut donc

Conclusion

Les calculs sur ordinateur,

- c'est rapide,
- mais ce n'est pas exact.

Il faut donc

- garder un œil critique sur les réponses fournies,

Conclusion

Les calculs sur ordinateur,

- c'est rapide,
- mais ce n'est pas exact.

Il faut donc

- garder un œil critique sur les réponses fournies,
- ou attendre que votre programme favori soit prouvé.

Ce dont je ne vous ai pas parlé.

Arithmétique à virgule flottante

- Arithmétique décimale
- Implantation en matériel
- Implantation en logiciel

Méthodes formelles

- Preuve formelle
  - Formalisation
  - Vérification de circuits
  - Vérification d'algorithmes
  - Automatisation
- Preuve de programmes
  - Spécification
  - Approche déductive
  - Interprétation abstraite

Évaluation de fonctions élémentaires

- Réduction d'arguments
- Évaluation polynomiale
- Calcul multi-précision
- Dilemme du fabricant de tables

Dernier exemple. . .

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,
- l'année suivante, je prends 1€ de frais et je multiplie par 3,

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,
- l'année suivante, je prends 1€ de frais et je multiplie par 3,
- ...
- après n ans, je prends 1€ de frais et je multiplie par n,

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,
- l'année suivante, je prends 1€ de frais et je multiplie par 3,
- ...
- après n ans, je prends 1€ de frais et je multiplie par n,
- Pour récupérer mon argent, il y a 1€ de frais.

Il était une fois, dans une galaxie très très lointaine

Mon banquier m'a proposé cet investissement:

- vous me donnez $e \approx 2{,}718\ 28\ldots$ €,
- l'année suivante, je prends 1€ de frais et je multiplie par 1,
- l'année suivante, je prends 1€ de frais et je multiplie par 2,
- l'année suivante, je prends 1€ de frais et je multiplie par 3,
- ...
- après n ans, je prends 1€ de frais et je multiplie par n,
- Pour récupérer mon argent, il y a 1€ de frais.

Dans 50 ans, pour ma retraite, combien d'argent aurai-je?

Combien?

Machine	Valeur

Combien?

Machine	**Valeur**
HP-48S	$+2{,}903\ 83 \quad 10^{52}$ €

$\Rightarrow$ Super!

Combien?

Machine	**Valeur**	
HP-48S	$+2{,}903\ 83$	10^{52} €
C (format `double`)	$-4{,}396\ 80$	10^{48} €

$\Rightarrow$ Super!

$\Rightarrow$ Oups!

Combien?

Machine	**Valeur**	
HP-48S	$+2{,}903\ 83$	10^{52} €
C (format `double`)	$-4{,}396\ 80$	10^{48} €
C (format `float`)	$-\infty$	

$\Rightarrow$ Super!

$\Rightarrow$ Oups!

$\Rightarrow$ Oups!!!!

Combien?

Machine	**Valeur**		
HP-48S	$+2{,}903\ 83$	10^{52} €	$\Rightarrow$ Super!
C (format `double`)	$-4{,}396\ 80$	10^{48} €	$\Rightarrow$ Oups!
C (format `float`)	$-\infty$		$\Rightarrow$ Oups!!!!
Maple (10 chiffres)	$-1{,}396\ 14$	10^{55} €	
Maple (20 chiffres)	$+1{,}207\ 82$	10^{45} €	$\Rightarrow$ Hein?

Combien?

Machine	**Valeur**	
HP-48S	$+2{,}903\ 83$	10^{52} €
C (format `double`)	$-4{,}396\ 80$	10^{48} €
C (format `float`)	$-\infty$	
Maple (10 chiffres)	$-1{,}396\ 14$	10^{55} €
Maple (20 chiffres)	$+1{,}207\ 82$	10^{45} €
Valeur exacte	$\approx 0{,}02$€	

$\Rightarrow$ Super!

$\Rightarrow$ Oups!

$\Rightarrow$ Oups!!!!

$\Rightarrow$ Hein?

Merci de votre attention.