ANALYSE NUMÉRIQUE

COURS 1

NOMBRES ET ORDINATEUR

Ch. Baskiotis

LAPI – EISTI

8 février 2010

Éléments de cuisine I

Site du cours : *http ://sifoci.eisti.fr*

Éléments de cuisine I

Site du cours : *http ://sifoci.eisti.fr*

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Éléments de cuisine I

Site du cours : *http ://sifoci.eisti.fr*

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Vous y trouverez

Éléments de cuisine I

Site du cours : ***http ://sifoci.eisti.fr***

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Vous y trouverez

- Le planning du cours réactualisé chaque semaine.

Éléments de cuisine I

Site du cours : ***http ://sifoci.eisti.fr***

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Vous y trouverez

- Le planning du cours réactualisé chaque semaine.
- Le poly du cours et les transparents de Pau et de Cergy.

Éléments de cuisine I

Site du cours : ***http ://sifoci.eisti.fr***

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Vous y trouverez

- Le planning du cours réactualisé chaque semaine.
- Le poly du cours et les transparents de Pau et de Cergy.
- Des livres et des articles sous forme électronique dont vous êtes fortement sollicités à le consulter.

Éléments de cuisine I

Site du cours : ***http ://sifoci.eisti.fr***

sifoci ⇒ Systèmes Informatiques FOrmels et Intelligents

Vous y trouverez

- Le planning du cours réactualisé chaque semaine.
- Le poly du cours et les transparents de Pau et de Cergy.
- Des livres et des articles sous forme électronique dont vous êtes fortement sollicités à le consulter.
- Des informations de dernière minute, le cas échéant.

Éléments de cuisine II

13 séances : 10 cours 1h00, 10 TD 1h30, 3 TP 2h30

Examen : 3h00.

Éléments de cuisine II

13 séances : 10 cours 1h00, 10 TD 1h30, 3 TP 2h30

Examen : 3h00.

Pendant les TD on travaillera essentiellement sur les exercices du poly.

Éléments de cuisine II

13 séances : 10 cours 1h00, 10 TD 1h30, 3 TP 2h30

Examen : 3h00.

Pendant les TD on travaillera essentiellement sur les exercices du poly.
Les TP seront notés. Pour avoir une note >0 il faut que vous soyez présents.

Éléments de cuisine II

13 séances : 10 cours 1h00, 10 TD 1h30, 3 TP 2h30

Examen : 3h00.

Pendant les TD on travaillera essentiellement sur les exercices du poly.
Les TP seront notés. Pour avoir une note >0 il faut que vous soyez présents.
Il n'y aura pas de rattrapage du TP.

Éléments de cuisine II

13 séances : 10 cours 1h00, 10 TD 1h30, 3 TP 2h30

Examen : 3h00.

Pendant les TD on travaillera essentiellement sur les exercices du poly.
Les TP seront notés. Pour avoir une note >0 il faut que vous soyez présents.
Il n'y aura pas de rattrapage du TP.

CALCUL DE LA NOTE FINALE
XX% de la note du TP, ,(100-XX)% de la note de l'examen, avec 100 > 2*XX.

Pourquoi ce cours ?

Pourquoi ce cours ?

Faire des calculs, surtout avec un ordinateur, n'est pas simple.

Pourquoi ce cours ?

Faire des calculs, surtout avec un ordinateur, n'est pas simple.

L'analyse numérique est

- une branche des mathématiques,

Pourquoi ce cours ?

Faire des calculs, surtout avec un ordinateur, n'est pas simple.

L'analyse numérique est

- une branche des mathématiques, et
- un ensemble des techniques informatiques

Pourquoi ce cours ?

Faire des calculs, surtout avec un ordinateur, n'est pas simple.

L'analyse numérique est

- une branche des mathématiques, et
- un ensemble des techniques informatiques

qui vous permettront de mieux se débrouiller avec les calculs sur ordinateur.

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose...
et pour pas cher !

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose... et pour pas cher !
Maintenant, si vous multipliez trois fois rien par trois fois rien :

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose... et pour pas cher !
Maintenant, si vous multipliez trois fois rien par trois fois rien :
Rien multiplié par rien = rien.

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose... et pour pas cher !
Maintenant, si vous multipliez trois fois rien par trois fois rien :
Rien multiplié par rien = rien.
Trois multiplié par trois = neuf.

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose... et pour pas cher !
Maintenant, si vous multipliez trois fois rien par trois fois rien :
Rien multiplié par rien = rien.
Trois multiplié par trois = neuf.
Cela fait : rien de neuf.
D'après Raymond Devos

Une calcul ... issu presque d'un ordinateur

Une fois rien... c'est rien !
Deux fois rien... ce n'est pas beaucoup !
Mais trois fois rien !... Pour trois fois rien, on peut déjà acheter quelque chose... et pour pas cher !
Maintenant, si vous multipliez trois fois rien par trois fois rien :
Rien multiplié par rien = rien.
Trois multiplié par trois = neuf.
Cela fait : rien de neuf.
D'après Raymond Devos

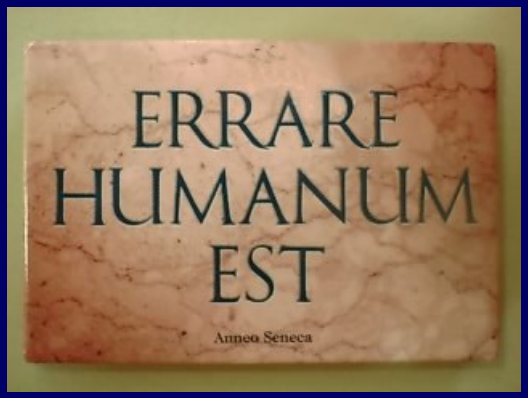
Un ordinateur ne se comporte pas mieux avec les calculs.

Au Commencement Était L'Erreur I

Tout calcul numérqiue est entaché d'erreur.

Au Commencement Était L'Erreur I

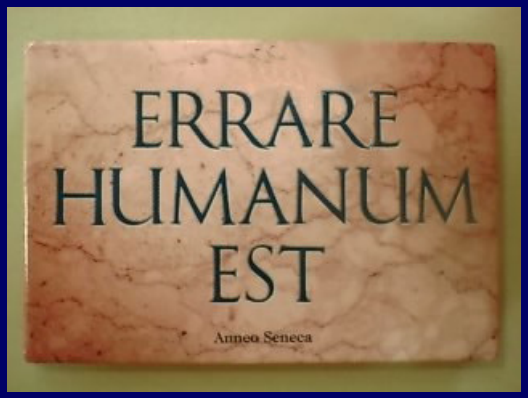
Tout calcul numérqiue est entaché d'erreur.



Errare humanum est, perseverare diabolicum

Au Commencement Était L'Erreur I

Tout calcul numérqiue est entaché d'erreur.



Errare humanum est, perseverare diabolicum

Donc l'analyse numérique cherche à extirper le diable qui se glisse dans les calculs !

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits
$\Rightarrow \mathcal{M}$ l'ensemble fini des nombres qu'un ordinateur peut stocker.

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits
$\Rightarrow \mathcal{M}$ l'ensemble fini des nombres qu'un ordinateur peut stocker.

$$\Rightarrow \mathcal{M} \subset \mathbb{R}$$

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits
$\Rightarrow \mathcal{M}$ l'ensemble fini des nombres qu'un ordinateur peut stocker.

$\Rightarrow \mathcal{M} \subset \mathbb{R}$
$\Rightarrow$ Pour un $x \in \mathbb{R}$ on n' a pas obligatoirement $x \in \mathcal{M}$.

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits
$\Rightarrow \mathcal{M}$ l'ensemble fini des nombres qu'un ordinateur peut stocker.

$\Rightarrow \mathcal{M} \subset \mathbb{R}$

$\Rightarrow$ Pour un $x \in \mathbb{R}$ on n' a pas obligatoirement $x \in \mathcal{M}$. Ainsi $x \in \mathbb{R}$ devient dans un ordinateur $fl(x) \in \mathcal{M}$, où $fl(x)$ sera appelé ***nombre-machine***, avec $x \neq fl(x)$.

Au Commencement Était L'Erreur II

L'erreur du calcul numérique est inévitable

Stockage d'un nombre réel dans un registre avec un nombre fini de bits
$\Rightarrow \mathcal{M}$ l'ensemble fini des nombres qu'un ordinateur peut stocker.

$\Rightarrow \mathcal{M} \subset \mathbb{R}$

$\Rightarrow$ Pour un $x \in \mathbb{R}$ on n' a pas obligatoirement $x \in \mathcal{M}$. Ainsi $x \in \mathbb{R}$ devient dans un ordinateur $fl(x) \in \mathcal{M}$, où $fl(x)$ sera appelé ***nombre-machine***, avec $x \neq fl(x)$.

Exemple : Si on rajoute 10 fois la valeur 0.1 le résultat qu'on obtient avec Scilab est $9.9999999999999890 0e-001$, c'est-à-dire on a une erreur de 1.11×10^{-16}.

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Preuve : Faisons des calculs à la main avec des nombres décimaux.

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Preuve : Faisons des calculs à la main avec des nombres décimaux.
Forme d'un nombre en notation scientifique normalisée : $0.a_1a_2a_3\ldots \times 10^n$, où $1 \leq a_1 \leq 9$ et $n \in \mathbb{Z}$.

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Preuve : Faisons des calculs à la main avec des nombres décimaux.
Forme d'un nombre en notation scientifique normalisée : $0.a_1a_2a_3\ldots \times 10^n$, où $1 \leq a_1 \leq 9$ et $n \in \mathbb{Z}$.

$\Rightarrow$ Problème : Le zéro $= 0.0 \times 10^0$ ne peut pas être représenté en écriture scientifique normalisée !

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Preuve : Faisons des calculs à la main avec des nombres décimaux.
Forme d'un nombre en notation scientifique normalisée : $0.a_1a_2a_3\ldots \times 10^n$, où $1 \leq a_1 \leq 9$ et $n \in \mathbb{Z}$.

$\Rightarrow$ Problème : Le zéro $= 0.0 \times 10^0$ ne peut pas être représenté en écriture scientifique normalisée !

$\Rightarrow$ Problème : Multiplication ou division avec deux digits décimaux :
$x_1 \times x_2 = (a_1 \times a_2) \times 10^{n_1+n_2}$ $\qquad$ $x_1/x_2 = (a_1/a_2) \times 10^{n_1-n_2}$

Au Commencement Était L'Erreur III

Ce n'est pas l'ordinateur qui est enquiquinant, c'est le calcul !

Preuve : Faisons des calculs à la main avec des nombres décimaux.
Forme d'un nombre en notation scientifique normalisée : $0.a_1a_2a_3\ldots \times 10^n$, où $1 \leq a_1 \leq 9$ et $n \in \mathbb{Z}$.

$\Rightarrow$ Problème : Le zéro $= 0.0 \times 10^0$ ne peut pas être représenté en écriture scientifique normalisée !

$\Rightarrow$ Problème : Multiplication ou division avec deux digits décimaux :
$x_1 \times x_2 = (a_1 \times a_2) \times 10^{n_1+n_2}$ $\qquad$ $x_1/x_2 = (a_1/a_2) \times 10^{n_1-n_2}$

Exemple : $\left(0.567 \times 10^{-4}\right) \times \left(0.234 \times 10^3\right) = 13.33 \times 10^{-3} = 0.133 \times 10^{-1}$
au lieu de 0.132678×10^{-1}.

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

$\Rightarrow$ Programmer un calcul n'est pas transcrire juste un algorithme en langage de programmation.

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

$\Rightarrow$ Programmer un calcul n'est pas transcrire juste un algorithme en langage de programmation.

Exemple : On a $(1.00 + 1.00 * 10^{20}) - 10^{20} = 0$

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

$\Rightarrow$ Programmer un calcul n'est pas transcrire juste un algorithme en langage de programmation.

Exemple : On a $(1.00 + 1.00 * 10^{20}) - 10^{20} = 0$
mais $1.00 + (1.00 * 10^{20} - 10^{20}) = 1.$

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

$\Rightarrow$ Programmer un calcul n'est pas transcrire juste un algorithme en langage de programmation.

Exemple : On a $(1.00 + 1.00 * 10^{20}) - 10^{20} = 0$
mais $1.00 + (1.00 * 10^{20} - 10^{20}) = 1.$

En Analyse Numérique on apprendra à organiser les calculs pour minimiser les erreurs.

Au Commencement Était L'Erreur IV

L'ordinateur fait pire !

Par exemple calculons `x = 1/9`. Réponse `x = 0.1111111`.
Continuons `y = x - 0.1111111`. Réponse `y = 1.111D-09` !

$\Rightarrow$ Programmer un calcul n'est pas transcrire juste un algorithme en langage de programmation.

Exemple : On a $(1.00 + 1.00 * 10^{20}) - 10^{20} = 0$
mais $1.00 + (1.00 * 10^{20} - 10^{20}) = 1.$

En Analyse Numérique on apprendra à organiser les calculs pour minimiser les erreurs.
En premier lieu, il faut examiner le stockage des nombres réels dans la mémoire de l'ordinateur.

L'ordinateur et les nombres I

- La base de numérotation est 2.

L'ordinateur et les nombres I

- La base de numérotation est 2.
- Les nombres sont stockés à l'aide des registres qui ont 16, 32 ou 64 bits.

L'ordinateur et les nombres I

- La base de numérotation est 2.
- Les nombres sont stockés à l'aide des registres qui ont 16, 32 ou 64 bits.
- Le stockage se fera selon la notation scientifique normalisée, à savoir $1.b_1b_2\ldots\times 10_E$, où $b_i \in \{0,1\}$ et E est donné en base binaire.

L'ordinateur et les nombres I

- La base de numérotation est 2.
- Les nombres sont stockés à l'aide des registres qui ont 16, 32 ou 64 bits.
- Le stockage se fera selon la notation scientifique normalisée, à savoir $1.b_1b_2\ldots\times 10_E$, où $b_i \in \{0,1\}$ et E est donné en base binaire.
- Le nombre de bits, noté p, après la décimale est fixé. p s'appelle la ***mantisse***.

L'ordinateur et les nombres I

- La base de numérotation est 2.
- Les nombres sont stockés à l'aide des registres qui ont 16, 32 ou 64 bits.
- Le stockage se fera selon la notation scientifique normalisée, à savoir $1.b_1b_2\ldots\times 10_E$, où $b_i \in \{0,1\}$ et E est donné en base binaire.
- Le nombre de bits, noté p, après la décimale est fixé. p s'appelle la ***mantisse***.
- Le bit avant la décimal est égal à 1. Donc on ne le stocke pas $\Rightarrow$ on gagne un bit ! Ce bit que l'on ne stocke pas, s'appelle le ***bit caché de la normalisation***.

L'ordinateur et les nombres I

- La base de numérotation est 2.
- Les nombres sont stockés à l'aide des registres qui ont 16, 32 ou 64 bits.
- Le stockage se fera selon la notation scientifique normalisée, à savoir $1.b_1b_2\ldots\times 10_E$, où $b_i \in \{0,1\}$ et E est donné en base binaire.
- Le nombre de bits, noté p, après la décimale est fixé. p s'appelle la ***mantisse***.
- Le bit avant la décimal est égal à 1. Donc on ne le stocke pas $\Rightarrow$ on gagne un bit ! Ce bit que l'on ne stocke pas, s'appelle le ***bit caché de la normalisation***.
- Le nombre de bits, noté q, pour l'exposant est fixé.

L'ordinateur et les nombres II

Il y a plusieurs normes pour le stockage des nombres.

L'ordinateur et les nombres II

Il y a plusieurs normes pour le stockage des nombres.

La plus répandue est la norme IEEE-754, qui, en simple précision (32 bits) est

Signe s	Exposant e	Mantisse m
$\mathbf{s = 1}$ bit	$\mathbf{q = 8}$ bits	$\mathbf{p = 23}$ bits
□	□□□□□□□□	□□□□□□□□□□□□□□□□□□□□□□□
$\pm$	$a_1a_2a_3a_4a_5a_6a_7a_8$	$b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}b_{11}b_{12}b_{13}b_{14}b_{15}b_{16}b_{17}b_{18}b_{19}b_{20}b_{21}b_{22}$

Exemple

Nombre décimal	465.40625_{10}
Conversion en binaire :	
partie entière	$465_{10} \Rightarrow 111010001_2$
partie décimale	$0.40625_{10} \Rightarrow 0.01101_2$
Nombre binaire	111010001.01101_2
sous forme normalisée	$1.1101000101101 \times 10^{1000}$
en norme IEEE-754	$0 \quad 1000 \quad 11010001011010000000000$

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle.

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle. La raison de cette exception est le nombre 0. Il sera stocké sous la forme

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle. La raison de cette exception est le nombre 0. Il sera stocké sous la forme

0	00000000	00000000000000000000000
ou		
1	00000000	00000000000000000000000

mais il n'y a pas de bit caché de normalisation.

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle. La raison de cette exception est le nombre 0. Il sera stocké sous la forme

0	00000000	00000000000000000000000
ou		
1	00000000	00000000000000000000000

mais il n'y a pas de bit caché de normalisation.

$\Rightarrow$ Par convention l'exposant 00000000 ne sera utiliser pour aucune autre représentation.

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle. La raison de cette exception est le nombre 0. Il sera stocké sous la forme

0	00000000	00000000000000000000000
ou		
1	00000000	00000000000000000000000

mais il n'y a pas de bit caché de normalisation.

$\Rightarrow$ Par convention l'exposant 00000000 ne sera utiliser pour aucune autre représentation.

N.B. Il y a deux zéros : un positif et un négatif.

Exceptions à la norme I

Une norme qui se respecte doit avoir des exceptions.

La norme IEEE-754 a deux exceptions :

1. La valeur d'un exposant doit être non nulle. La raison de cette exception est le nombre 0. Il sera stocké sous la forme

0	00000000	00000000000000000000000
ou		
1	00000000	00000000000000000000000

   mais il n'y a pas de bit caché de normalisation.

   $\Rightarrow$ Par convention l'exposant 00000000 ne sera utiliser pour aucune autre représentation.

   N.B. Il y a deux zéros : un positif et un négatif. ($\Rightarrow$ vos copies peuvent être notées avec zéro négatif).

Exceptions à la norme II

Conséquences de cette situation :

- Le plus petit nombre positif que nous pouvons stocker est

Exceptions à la norme II

Conséquences de cette situation :

- Le plus petit nombre positif que nous pouvons stocker est

  ce qui donne $1.0 \times 2^{1-127} = 2^{-126}$ et qui signifie que le domaine de variation de l'exposant E n'est pas entre -127 et 127 mais entre -126 et 127.

Exceptions à la norme II

Conséquences de cette situation :

- Le plus petit nombre positif que nous pouvons stocker est

  ce qui donne $1.0 \times 2^{1-127} = 2^{-126}$ et qui signifie que le domaine de variation de l'exposant E n'est pas entre -127 et 127 mais entre -126 et 127.

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus petits encore.

Exceptions à la norme II

Conséquences de cette situation :

- Le plus petit nombre positif que nous pouvons stocker est

height0	00000001	00000000000000000000000

  ce qui donne $1.0 \times 2^{1-127} = 2^{-126}$ et qui signifie que le domaine de variation de l'exposant E n'est pas entre -127 et 127 mais entre -126 et 127.

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus petits encore.
  $\Rightarrow$ Votre ordinateur est malin : lui il peut stocker et calculer avec des nombres à exposant nul, c'est-à-dire des nombres

0	00000000	$b_1 b_2 \cdots b_{23}$

  tel que il y a au moins un $i, 1 \leq i \leq 23$ avec $b_i \neq 0$.

Exceptions à la norme II

Conséquences de cette situation :

- Le plus petit nombre positif que nous pouvons stocker est

  ce qui donne $1.0 \times 2^{1-127} = 2^{-126}$ et qui signifie que le domaine de variation de l'exposant E n'est pas entre -127 et 127 mais entre -126 et 127.

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus petits encore.
  $\Rightarrow$ Votre ordinateur est malin : lui il peut stocker et calculer avec des nombres à exposant nul, c'est-à-dire des nombres

  tel que il y a au moins un $i, 1 \leq i \leq 23$ avec $b_i \neq 0$.
  Ce sont les ***nombres sous-normalisés*** que l'ordinateur utilise et qu'il vous interdit de les utiliser.

Exceptions à la norme III

2. Un ordinateur ne peut travailler qu'avec des objets concrets.

Exceptions à la norme III

2. Un ordinateur ne peut travailler qu'avec des objets concrets.
   $\Rightarrow$ On doit pouvoir représenter l'infini. Pour la norme IEEE-754, l'infini est

0 ou	11111111	00000000000000000000000
1	11111111	00000000000000000000000

Il y a bien sûr deux infinis : un positif et un négatif. $\Rightarrow$ Par convention l'exposant 11111111 ne sera utiliser pour aucune autre représentation.

Exceptions à la norme IV

Conséquences de cette situation :

- Le plus grand nombre positif que nous pouvons stocker est

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus grands encore.

Exceptions à la norme IV

Conséquences de cette situation :

- Le plus grand nombre positif que nous pouvons stocker est

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus grands encore.
  Ce sont les nombres qui s'appellent ...*Not a Number (NaN)*.

Exceptions à la norme IV

Conséquences de cette situation :

- Le plus grand nombre positif que nous pouvons stocker est

- Nous pouvons stocker ne signifie pas que l'ordinateur ne peut pas stocker des nombres positifs plus grands encore.
  Ce sont les nombres qui s'appellent ...*Not a Number (NaN)*.
  L'ordinateur refuse tout de même de faire des calculs avec des NaN.

La ménagerie des erreurs I

Entre un réel x et sa représentation machine $m(x)$, dite nombre-machine, on distingue plusieurs types d'erreurs :

La ménagerie des erreurs I

Entre un réel x et sa représentation machine $m(x)$, dite nombre-machine, on distingue plusieurs types d'erreurs :

- ***Erreur de représentation*** : notée Δx ou $\varepsilon(x)$:

$$\Delta x = m(x) - x$$

La ménagerie des erreurs I

Entre un réel x et sa représentation machine $m(x)$, dite nombre-machine, on distingue plusieurs types d'erreurs :

- *Erreur de représentation* : notée Δx ou $\varepsilon(x)$:

$$\Delta x = m(x) - x$$

- *Erreur absolue de représentation* :

$$|\Delta x| = |m(x) - x|$$

La ménagerie des erreurs I

Entre un réel x et sa représentation machine $m(x)$, dite nombre-machine, on distingue plusieurs types d'erreurs :

- *Erreur de représentation* : notée Δx ou $\varepsilon(x)$:

$$\Delta x = m(x) - x$$

- *Erreur absolue de représentation* :

$$|\Delta x| = |m(x) - x|$$

La ménagerie des erreurs II

- *Erreur relative de représentation*, notée $\imath\,(x)$:

$$\imath\,(x) = \frac{\Delta x}{m\,(x)} = \frac{m\,(x) - x}{m\,(x)}$$

La ménagerie des erreurs II

- *Erreur relative de représentation*, notée $\iota(x)$:

$$\iota(x) = \frac{\Delta x}{m(x)} = \frac{m(x) - x}{m(x)}$$

- *Erreur relative de précision*, notée $\eta(x)$:

$$\eta(x) = \frac{\Delta x}{x} = \frac{m(x) - x}{x} = \frac{m(x)}{x} - 1$$

La ménagerie des erreurs III

- *Erreur relative absolue de précision*

$$| \eta(x) | = \frac{| \Delta x |}{| x |} = \frac{| m(x) - x |}{| x |}$$

La ménagerie des erreurs III

- *Erreur relative absolue de précision*

$$| \eta(x) | = \frac{| \Delta x |}{| x |} = \frac{| m(x) - x |}{| x |}$$

L'erreur relative de précision permet d'exprimer le nombre-machine comme suit :

$$m(x) = x(1 + \eta(x))$$

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

– un bit de signe,

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse,

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

EXEMPLE Soit un ordinateur avec base de numérotation $\beta = 10$ et nombre de chiffres de la mantisse $p = 4$.

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

EXEMPLE Soit un ordinateur avec base de numérotation $\beta = 10$ et nombre de chiffres de la mantisse $p = 4$.

Pour cet ordinateur nous avons

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

EXEMPLE Soit un ordinateur avec base de numérotation $\beta = 10$ et nombre de chiffres de la mantisse $p = 4$.

Pour cet ordinateur nous avons

$$123.4567 = 0.1234567 \times 10^3 = \left(0.1234 + 0.567 \times 10^{-4}\right) \times 10^3$$

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

EXEMPLE Soit un ordinateur avec base de numérotation $\beta = 10$ et nombre de chiffres de la mantisse $p = 4$.

Pour cet ordinateur nous avons

$123.4567 = 0.1234567 \times 10^3 = \left(0.1234 + 0.567 \times 10^{-4}\right) \times 10^3$

Si l'ordinateur approche les valeurs réelles par troncature, alors $fl\,(x) = 0.1234 \times 10^3$.

Les nombres-machine I

Rappel : Pour représenter en machine un réel on utilise :

- un bit de signe,
- p bits de mantisse, et
- q bits d'exposant.

La base de numérotation de l'ordinateur est β.

EXEMPLE Soit un ordinateur avec base de numérotation $\beta = 10$ et nombre de chiffres de la mantisse $p = 4$.

Pour cet ordinateur nous avons

$123.4567 = 0.1234567 \times 10^3 = \left(0.1234 + 0.567 \times 10^{-4}\right) \times 10^3$

Si l'ordinateur approche les valeurs réelles par troncature, alors $fl\,(x) = 0.1234 \times 10^3$.

Si l'approximation se fait par arrondi, alors $fl\,(x) = 0.1235 \times 10^3$.

Les nombres-machine II

Un réel x s'écrit : $x = s \times w \times \beta^n$, où s est le signe.

Les nombres-machine II

Un réel x s'écrit : $x = s \times w \times \beta^n$, où s est le signe.
On peut, encore écrire :

$$x = s \times \left(r + t \times \beta^{-p}\right) \times \beta^n \text{ , avec } \beta^{-1} \leq \mid r \mid < 1 \text{ et } 0 \leq \mid t \mid < 1$$

Les nombres-machine II

Un réel x s'écrit : $x = s \times w \times \beta^n$, où s est le signe.
On peut, encore écrire :

$$x = s \times \left(r + t \times \beta^{-p}\right) \times \beta^n \text{ , avec } \beta^{-1} \leq \mid r \mid < 1 \text{ et } 0 \leq \mid t \mid < 1$$

Exemple : $\beta = 10, p = 4, w = 12.3456$ On a sous forme normalisée
$w = 0.123456 \times 10^2 = \left(0.1234 + 0.56 \times 10^{-4}\right) \times 10^2$

Les nombres-machine II

Un réel x s'écrit : $x = s \times w \times \beta^n$, où s est le signe.
On peut, encore écrire :

$$x = s \times \left(r + t \times \beta^{-p}\right) \times \beta^n \text{ , avec } \beta^{-1} \leq \mid r \mid < 1 \text{ et } 0 \leq \mid t \mid < 1$$

Exemple : $\beta = 10, p = 4, w = 12.3456$ On a sous forme normalisée
$w = 0.123456 \times 10^2 = \left(0.1234 + 0.56 \times 10^{-4}\right) \times 10^2$
Si le nombre-machine représentant x est obtenu par troncature, nous avons

$$m\left(x\right) = s \times r \times \beta^n$$

Les nombres-machine II

Un réel x s'écrit : $x = s \times w \times \beta^n$, où s est le signe.
On peut, encore écrire :

$$x = s \times \left(r + t \times \beta^{-p}\right) \times \beta^n \text{ , avec } \beta^{-1} \leq \mid r \mid < 1 \text{ et } 0 \leq \mid t \mid < 1$$

Exemple : $\beta = 10, p = 4, w = 12.3456$ On a sous forme normalisée
$w = 0.123456 \times 10^2 = \left(0.1234 + 0.56 \times 10^{-4}\right) \times 10^2$
Si le nombre-machine représentant x est obtenu par troncature, nous avons

$$m\left(x\right) = s \times r \times \beta^n$$

tandis que pour approximation par arrondi, nous avons

$$m\left(x\right) = \begin{cases} s \times r \times \beta^n, & \text{si } \mid x - r \times \beta^n \mid < \mid x - \left(r \times \beta^n + \beta^{-p}\right) \mid \\ s \times r \times \beta^n + \beta^{-p}, & \text{sinon} \end{cases}$$

Les nombres-machine III

THÉORÈME (de la précision relative).- Soit un calculateur avec base de numérotation β et nombre de chiffres de la mantisse p.

Les nombres-machine III

THÉORÈME (de la précision relative).- Soit un calculateur avec base de numérotation β et nombre de chiffres de la mantisse p.
Alors l'erreur relative de précision $\eta(x)$ est bornée comme suit :

Les nombres-machine III

THÉORÈME (de la précision relative).- Soit un calculateur avec base de numérotation β et nombre de chiffres de la mantisse p.
Alors l'erreur relative de précision $\eta(x)$ est bornée comme suit :

$$|\eta(x)| \leq \begin{cases} \beta^{1-p}, & \text{si approximation par troncature} \\ 0.5 \times \beta^{1-p}, & \text{si approximation par arrondi} \end{cases} \quad ; \; x \in \mathbb{R}$$

Précision de l'ordinateur

L'erreur de précision relative selon le standard IEEE-754 est :

$$\eta(x) \leq \begin{cases} \beta^{1-p}, & \text{si approximation par troncature} \\ 0.5 \times \beta^{1-p}, & \text{si approximation par arrondi} \end{cases}$$

Précision de l'ordinateur

L'erreur de précision relative selon le standard IEEE-754 est :

$$\eta(x) \leq \begin{cases} \beta^{1-p}, & \text{si approximation par troncature} \\ 0.5 \times \beta^{1-p}, & \text{si approximation par arrondi} \end{cases}$$

Dans les calculs on utilise la valeur absolue de l'erreur de précision relative $|\eta(x)|$.