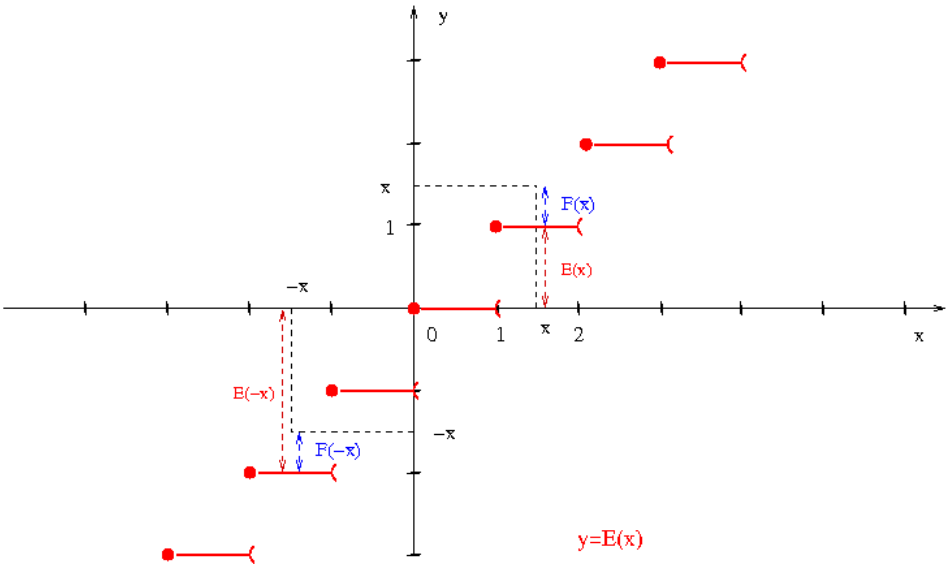
Représentation de nombres réels

Un réel $x \in \mathbb{R}$ se décompose toujours en
une **partie entière** $E(x)$ et une **partie fractionnaire** $F(x)$:

$$x = E(x) + F(x), \quad \text{où } E(x) \in \mathbb{Z} \text{ et } F(x) = x - E(x) \in [0, 1[.$$



Ne pas confondre $E(x)$ avec la troncature à l'unité d'un nombre,
i.e. la suppression des décimales.

- Souvent il n'est pas commode d'utiliser une
  représentation en virgule fixe :
  - La masse de la terre est de 5 973 600 000 000 000 000 000 000 kg ;
  - La masse du soleil est de 19 891 $\underbrace{000\cdots000}_{26 \text{ zéros}}$ kg ;
  - La masse d'un électron est de 0,$\underbrace{00\cdots00}_{27 \text{ zéros}}$91093822 grammes ;
  - La masse d'un proton est de 0,$\underbrace{00\cdots00}_{23 \text{ zéros}}$16726 grammes.
- On utilise plutôt la notation scientifique de la forme $a \times 10^e$, $e \in \mathbb{Z}$.
  Pour la notation scientifique **normalisée** on a $1 \leq |a| < 10$,
  tandis que pour la **notation ingénieur** $1 \leq |a| < 10^3$
  et l'exposant e est un multiple de 3.
  Exemples : Pour les masses de la terre et du soleil, on écrit
  $5{,}9736 \times 10^{24}$ kg et $1{,}9891 \times 10^{30}$ kg.
  Pour l'électron et le proton on a $9{,}1093822 \times 10^{-31}$ kg et
  $1{,}6726 \times 10^{-27}$ kg.

Réels en machine

Comment représenter des nombres réels en machine ?

- Le choix de la taille du mot mémoire influence la précision de la représentation des nombres.
  - Pour représenter exactement un rationnel $r = \frac{n}{d}$ il faut garder le numérateur $n \in \mathbb{Z}$ et le dénominateur $d \in \mathbb{N}^*$,
    sauf si dans la base choisie, r admet un développement fini ;
  - Un nombre irrationnel $x \in \mathbb{R} \setminus \mathbb{Q}$ ne peut jamais être représenté exactement.
- Sur un ordinateur, on utilise les nombres à virgule flottante de la forme $x = s \times m \times b^e$
  où b est la *base* ; $s \in \{-1, +1\}$ est le *signe* ;
  la *mantisse* m, ou *significande*, précise les chiffres significatifs ;
  l'*exposant* e donne l'ordre de grandeur.
  Exemple : en base 10

$$-37{,}5 = -37500 \times 10^{-3} = -0{,}000375 \times 10^{5} = -0{,}375 \times 10^{2}\,.$$

Réels en virgule flottante, base 10

Représentation en virgule flottante normalisée $x = s \times m \times 10^e$

Exemple : $x = -0{,}375 \times 10^{+2}$

→ le signe du nombre $s = (-1)^{s_m}$, avec $s_m \in \{0, 1\}$;

→ la mantisse m est un réel dans $]0, 1[$:
tous les chiffres significatifs sont à droite de la virgule ;

→ le digit de poids fort de la mantisse est différent de zéro,
le zéro est donc non représentable ;

→ l'exposant e est un entier relatif ;

→ la virgule et la base sont représentées de façon implicite ;

→ cette représentation du nombre est unique.

s_m	e	d_{-1}	d_{-2}	$\cdots$	d_{-p}

$$= (-1)^{s_m}\, 0, d_{-1} d_{-2} \ldots d_{-p} \times 10^{e} \quad \text{avec } d_{-1} \neq 0$$

Par **convention**, la représentation de 0 ne contient que des zéros.

Exemple en base dix (1)

- On considère une représentation avec
  1. une mantisse de 3 chiffres décimaux ;
  2. un exposant de 2 chiffres décimaux ;
  3. deux bits de signe.
- *Exemple :* $37{,}5 = 0{,}375 \times 10^2$ est représenté par

+	+	0	2	3	7	5

- Les nombres strictement positifs représentables vont de

	$+0{,}100 \times 10^{-99}$:	+	-	9	9	1	0	0
à	$+0{,}999 \times 10^{+99}$:	+	+	9	9	9	9	9

- Les nombres strictement négatifs représentables vont de

	$-0{,}999 \times 10^{+99}$:	-	+	9	9	9	9	9
à	$-0{,}100 \times 10^{-99}$:	-	-	9	9	1	0	0

- Tous les réels de l'intervalle $[-0{,}999 \times 10^{+99} ; 0{,}999 \times 10^{+99}]$ ne sont pas représentables (que $36 \cdot 10^4 + 1$).

Exemple en base dix (2)

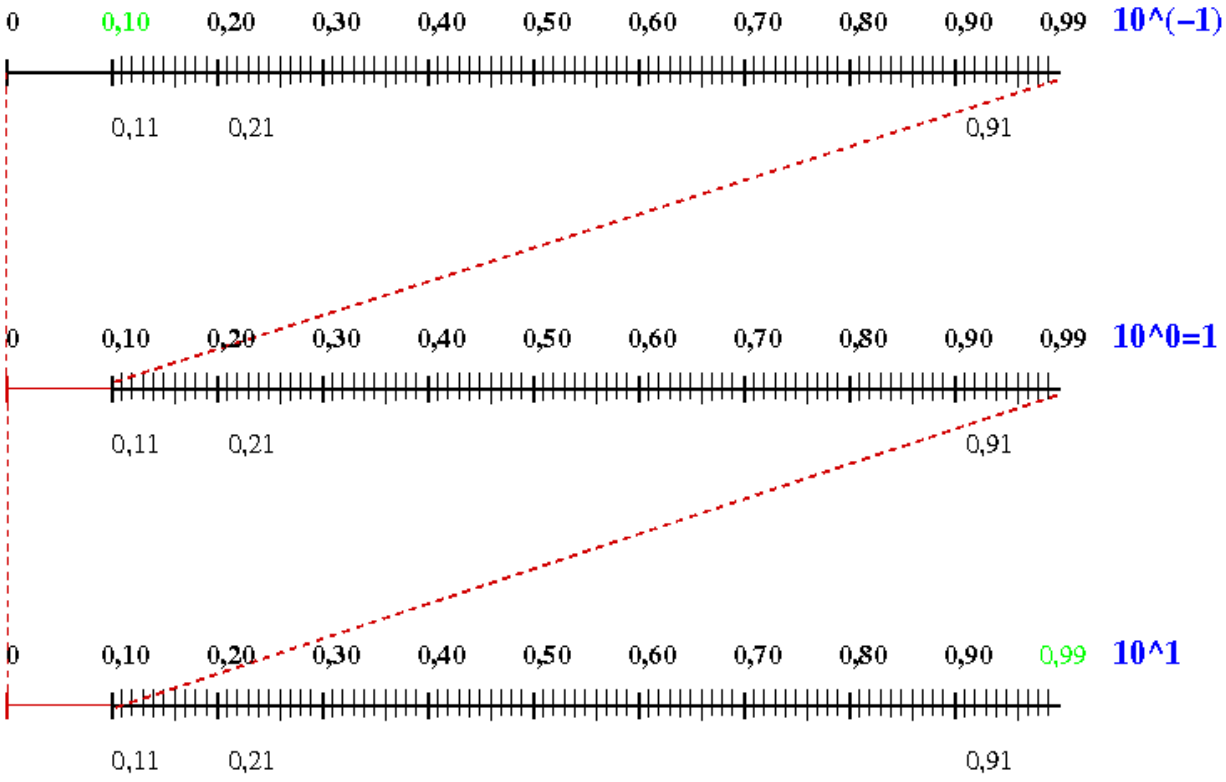
Représentation avec mantisse de 2 chiffres décimaux ($p = 2$) et l'exposant $e \in \{-1, 0, 1\}$. Nombres strict. positifs de

0,01 :

+	-	1	1	0

à 9,90 :

+	+	1	9	9



Problèmes de la représentation en virgule flottante

- On ne peut pas représenter des réels plus grands que 9,9 .
  Une opération ayant comme résultat un tel nombre engendre un *dépassement de capacité* ou **overflow**.
- On ne peut représenter des réels $x \in \mathbb{R}$ pour $0 < x < 0{,}010$.
  Une opération ayant comme résultat un tel nombre engendre un *soupassement de capacité* ou **underflow**.
- De l'intervalle $[0\,;\,9,9]$ on ne représente que 271 nombres réels.
  Ces valeurs représentées ne sont pas distribuées de façon uniforme.

  Une opération ayant comme résultat un nombre x non représentable engendre une **erreur d'arrondi** :
  on doit approximer le "vrai" résultat x par un réel $\tilde{x}$ représentable dans le système virgule flottante choisi.

Problèmes de la représentation en virgule flottante

Exemples : base 10, $p = 2$ et $e \in \{-1, 0, 1\}$.

- Les nombres 3 et 7 sont représentables :
  $3 = 0{,}30\ 10^1$; $7 = 0{,}70\ 10^1$.
  Mais le résultat de $3 + 7 = 10 = 0{,}10\ 10^2$ n'est plus représentable, d'où **overflow**.
- Les nombres $0{,}010 = 0{,}10\ 10^{-1}$ et $0{,}011 = 0{,}11\ 10^{-1}$ sont représentables, mais leur différence $0{,}011 - 0{,}010 = 0{,}001 = 0{,}10\ 10^{-2}$ ne l'est pas.
  De même $0{,}010/2 = 0{,}005 < 0{,}010$ n'est pas représentable.
  On a donc affaire à un **underflow**.
- Si on relâche la condition que le digit de plus fort poids soit non nul, on peut représenter ces nombres :
  $0{,}001 = 0{,}01\ 10^{-1}$ et $0{,}005 = 0{,}05\ 10^{-1}$
  Ces nombres «sous-normaux» (**subnormal**) sont utilisés pour représenter des quantités très petites mais non nulles.

Problèmes de la représentation en virgule flottante

Exemples **erreurs d'arrondi** *:* base 10, $p = 2$ et $e \in \{-1, 0, 1\}$.

- Le résultat de l'opération $1{,}0 - 0{,}011 = 0{,}989$ n'est pas représentable, et sera arrondi vers $0{,}99 = 0{,}99\ 10^0$
- De même, $5 + 0{,}09 = 0{,}509\ 10^1$ sera arrondi vers $0{,}51\ 10^1$.
- Soit $a = -9$, $b = 9$ et $c = 0{,}011$ et $d = a + b + c$:

$$d = (a + b) + c = 0{,}011 \neq a + (b + c) = 0$$

En effet, $b + c = 9{,}011 = 0{,}9011\ 10^1$ est non représentable et est arrondi vers $0{,}90\ 10^1 = b$.

L'addition des nombres en virgule flottante n'est pas associative !

- Prévoir le résultat de

$$\left(\left(\frac{1}{3}\right) * 3\right) - 1\ .$$

Bilan : représentation en virgule flottante

Soit la représentation en virgule flottante en base b :

$$(-1)^s\ 0, d_{-1} d_{-2} \ldots d_{-(p-1)} d_{-p} \cdot b^e$$

où $e_m \leq e \leq e_M$, $d_i \in \{0, 1, \ldots, b-1\}$ et $d_{-1} \neq 0$.

1. Les nombres à virgule flottante sont un sous-ensemble fini de $\mathbb{R}$ qui n'est pas stable pour les opérations arithmétiques.
2. Même si p et $e_M - e_m$ sont grands :
   $\rightarrow$ Les nombres en virgule flottante ne sont pas répartis de façon uniforme.
   $\rightarrow$ Il y aura toujours des overflows, underflows et erreurs d'arrondi.
3. Le nombre $\varepsilon = b^{1-p}$ est tel que entre 1 et $1 + \varepsilon$ aucun réel n'est représentable.
   Ce nombre (**précision machine**) sert à majorer les erreurs d'arrondi et d'approximation sur un système donné.

Exemple de code C

```
#include <stdio.h>
 int main( int argc, char **argv )
 {
    float machEps = 1.0f;

    do {
       machEps /= 2.0f;
    }
    while ((float)(1.0 + (machEps/2.0)) != 1.0);

    printf( "\n Epsilon  machine = %G\n", machEps );
    return 0;
 }
```

On trouve comme résultat `1.19209E-07`$= 1{,}192\ 10^{-7} \sim 2^{-23}$.

On verra que cette valeur «expérimentale» de la précision machine correspond bien au format `float` du langage `C`.

Réels en virgule flottante en base 2

- Pour le codage de nombres en virgule flottante en binaire, on peut apporter quelques améliorations, présentées dans la suite.
- Le standard **IEEE 754-2008** fixe, entre autres, comment représenter des nombres flottants en simple (4 octets) et double (8 octets) précision.

  En langage `C` ceci correspond aux formats `float` et `double`.

  IEEE = *Institute of Electrical and Electronics Engineers*
  (association professionnelle internationale, de droit américain)
- Ce standard ne fixe pas que les formats, mais aussi :
  les modes d'arrondi, les calculs avec les nombres flottants, des valeurs particulières, la détection de problèmes (overflow, . . .).
- On va s'intéresser à deux points concernant le format :
  les *exposants biaisés* et le *bit implicite*.

Réels en virgule flottante en base 2

On veut représenter les nombres en virgule flottante sur une machine suivant le format

signe mantisse 1 bit	exposant 4 bits	mantisse normalisée 8 bits

- Par exemple $(0{,}015)_8 = (0{,}000001101)_2 = (0{,}1101)_2 * (2^{-5})_{10}$;
- Mantisse positive, donc bit de signe égal à 0 ;
- Mantisse normalisée $(0,1101)_2$ avec exposant $(-5)_{10}$
- Codage en complément à deux sur 4 bits :
  $(5)_{10} = (0101)_2$, d'où le code CA2 : 1011
- Représentation du nombre :

0	1011	11010000

- Un comparateur logique, opérant bit à bit et de gauche à droite, ne peut pas facilement comparer des nombres :

0	1011	11010000
0	0011	11010000

Exposant biaisé

- Afin de simplifier les tests sur les mots mémoire, les exposants signés sont codés grâce à un *décalage*.
- En code *CA2*, sur k bits, on représente les entiers signés :

$$-2^{k-1} \leq n \leq 2^{k-1} - 1$$

- En ajoutant 2^{k-1}, on translate l'intervalle sur

$$0 \leq n + 2^{k-1} \leq 2^k - 1$$

- Le nombre 0 est codé par 2^{k-1} : $\boxed{1}\underbrace{\boxed{0}\cdots\boxed{0}}_{k-1 \text{ fois}}$

-2^{k-1} est codé par $\underbrace{\boxed{0}\boxed{0}\cdots\boxed{0}}_{k \text{ fois}}$ et $2^{k-1} - 1$ par $\underbrace{\boxed{1}\boxed{1}\cdots\boxed{1}}_{k \text{ fois}}$

- La valeur 2^{k-1} s'appelle le biais ou le décalage
  En anglais, on parle de **bias**, **offset** ou **excess**.

Exposant biaisé

- Si $0 \leq c(n) \leq 2^k - 1$, alors il code l'entier signé $n = c(n) - 2^{k-1}$.
  Si $-2^{k-1} \leq n \leq 2^{k-1} - 1$, alors son code est $c(n) = n + 2^{k-1}$.
- Ce codage est souvent utilisé dans des circuits qui ne peuvent pas traiter des nombres positifs et négatifs.
  Par exemple en traitement du signal (*DSP*).
- Pour un codage sur $k = 4$ bits et un biais de $2^{4-1} = 8$:

		CA2	biaisé			*CA2*	biaisé
-8	=	1000	0000	0	=	0000	1000
-7	=	1001	0001	1	=	0001	1001
-6	=	1010	0010	2	=	0010	1010
-5	=	1011	0011	3	=	0011	1011
-4	=	1100	0100	4	=	0100	1100
-3	=	1101	0101	5	=	0101	1101
-2	=	1110	0110	6	=	0110	1110
-1	=	1111	0111	7	=	0111	1111

Exposant biaisé : exemple de codage

On veut représenter les nombres en virgule flottante sur une machine suivant le format

signe mantisse	exposant	mantisse normalisée
1 bit	4 bits	8 bits

- Encore l'exemple
  $(0{,}015)_8 = (0{,}000001101)_2 = (0{,}1101)_2 * (2^{-5})_{10}$;
- Mantisse positive, donc bit de signe égal à 0 ;
- Mantisse normalisée $(0{,}1101)_2$ avec exposant $(-5)_{10}$
- Codage biaisé de l'exposant sur 4 bits :
  le *biais* est $2^{4-1} = 8$, l'*exposant biaisé* est $-5 + 8 = 3_{10}$,
  écrit en binaire sur 4 bits, l'exposant est codé par $(0011)_2$.
- Représentation du nombre :

0	0011	11010000

- Un comparateur logique, opérant bit à bit et de gauche à droite, peut facilement comparer les nombres :

0	0011	11010000
0	1011	11010000

Exposant biaisé : codage

- Sur k bits, pour coder un entier signé, on ajoute le biais 2^{k-1} et l'on représente le résultat en binaire naturel
- Inversement, étant donné un code sur k bits, on transforme le code binaire naturel non signé en décimal et l'on retranche le biais 2^{k-1}.
- Sur 3 bits, le biais est $2^{3-1} = +4$

-4	s'écrit 000	car	-4+4=0	et le code 000	représente 0-4=-4
-3	s'écrit 001	car	-3+4=1	et le code 001	représente 1-4=-3
-2	s'écrit 010	car	-2+4=2	et le code 010	représente 2-4=-2
-1	s'écrit 011	car	-1+4=3	et le code 011	représente 3-4=-1
0	s'écrit 100	car	0+4=4	et le code 100	représente 4-4= 0
1	s'écrit 101	car	1+4=5	et le code 101	représente 5-4= 1
2	s'écrit 110	car	2+4=6	et le code 110	représente 6-4= 2
3	s'écrit 111	car	3+4=7	et le code 111	représente 7-4= 3

Procédé du bit caché

- En base 2, la représentation normalisée de la mantisse commence toujours par 1.
- Le bit le plus significatif ne représente aucune information : on peut donc supposer sa présence de façon **implicite**, c'est-à-dire *sans le coder* !
- Sur un mot de p bits, on gagne ainsi un bit pour avoir une mantisse de $p+1$ **bits significatifs**.
  On **double** les mantisses représentables.
- Attention : on peut maintenant avoir une mantisse ne contenant que des zéros !
- Attention : en base $b > 2$, la mantisse normalisée commence par $d_{-1} \in \{1, 2, \ldots, b-1\}$.

  Le procédé du bit caché ne s'applique pas !

Procédé du bit caché

Exemple :

- Mantisse sur $p = 3$ bits
- Avec la contrainte du bit de poids fort non nul, nous n'avons que 4 mantisses distinctes :

  $100 = (0,5)_{10}, 101 = (0,625)_{10}, 110 = (0,75)_{10}, 111 = (0,875)_{10}$.
- Si l'on suppose qu'il y a un bit caché qui vaut 1, la mantisse contient 3 bits, mais l'on code une mantisse normalisée de 4 bits.
- Il y a maintenant 8 mantisses possibles :

1	000	soit $(0{,}5000)_{10}$;	1	100	soit $(0{,}7500)_{10}$;
1	001	soit $(0{,}5625)_{10}$;	1	101	soit $(0{,}8125)_{10}$;
1	010	soit $(0{,}6250)_{10}$;	1	110	soit $(0{,}8750)_{10}$;
1	011	soit $(0{,}6875)_{10}$;	1	111	soit $(0{,}9375)_{10}$.

- La précision augmente sans frais !

Bit caché et exposant biaisé. Exemple

Représentation comprenant :
un bit de signe, un *exposant biaisé* de 3 bits et une mantisse de 3 bits, avec utilisation du *bit caché*.

Code	Valeur binaire (notation sc.)	Valeur binaire (notation à virgule)	Valeur décimale
0 110 000	$0{,}1000.10^{10}$	10,00	2,00
0 110 001	$0{,}1001.10^{10}$	10,01	2,25
0 110 010	$0{,}1010.10^{10}$	10,10	2,50
0 110 011	$0{,}1011.10^{10}$	10,11	2,75
0 110 100	$0{,}1100.10^{10}$	11,00	3,00
0 110 101	$0{,}1101.10^{10}$	11,01	3,25
0 110 100	$0{,}1110.10^{10}$	11,10	3,50
0 110 111	$0{,}1111.10^{10}$	11,11	3,75
0 111 000	$0{,}1000.10^{11}$	100,0	4,00
0 111 001	$0{,}1001.10^{11}$	100,1	4,50
0 111 010	$0{,}1010.10^{11}$	101,0	5,00

Bilan : bit caché et exposant biaisé

Avantages du codage utilisant un bit caché et un exposant biaisé :

1. on peut facilement comparer deux nombres ;
2. on obtient le successeur d'un nombre en ajoutant simplement 1 au code ;
3. on augmente la précision de la mantisse sans coût matériel.

Format flottant IEEE

1. Le standard IEEE utilise le bit caché pour la mantisse : il est de poids 2^0 et la mantisse est donc $1, d_{-1} \cdots d_{-p}$;
2. Le décalage/biais de l'exposant sur k bits est de $2^{k-1} - 1$ et on ne représente que des exposants signés de $-2^{k-1} + 2$ à $2^{k-1} - 1$. Pour $k = 8$, biais égal à 127 et les exposants vont de -126 à +127.
3. Les exposants biaisés 0 et $2^k - 1$ sont utilisés pour représenter des nombres sous-normaux, l'infini `Inf` et un nombre non définie `NaN`
4. Résumé format IEEE 754 $\boxed{s}\boxed{e}\boxed{d_{-1}}\boxed{d_{-2}}\cdots\boxed{d_{-p}}$

Nom	Taille	Signe s	Exposant e	Biais	Mantisse	Précision $p+1$	Chiffres significatifs
(float)	32 bits	1 bit	8 bits	127	23 bits	24	7
(double)	64 bits	1 bit	11 bits	1023	52 bits	54	16

Exemple de code C

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

main()
{
  float somme= 0.0f, inc= 0.1f;  int i;

  printf("\n Taille de  float : %d octet(s) \n",sizeof(float));

  for ( i=1; i<=10; ++i ) somme+= inc ;
  if( somme == 1 )
    printf("\n    10*0.1 = 1  \n");
  else
    printf("\n    10*0.1 <> 1 \n");

  printf("\n somme = %f\n",somme);      /* Format par défaut */
  printf("\n somme = %.15g,  inc = %.15g \n\n",somme,inc);
                                       /* Format étendu     */
}
```


Exemple de code C : affichage des résultats

```
 Taille de  float     : 4 octet(s)

   10*0.1 <> 1

  somme = 1.000000

  somme = 1.00000011920929,  inc = 0.100000001490116
```

- Ce n'est donc pas une bonne idée de faire des test d'égalité entre des nombres en virgule flottante.
- Explication : on a $(0{,}1)_{10} = (0.0001100110\underline{0011}\ldots)_2$
  Pour coder $(0{,}1)_{10}$ en virgule flottante, avec une mantisse en binaire, on est obligé de faire une approximation.
- Il vaut mieux faire un test d'inégalité :

  `if( fabs(somme-1)< epsilon )`

  où `epsilon` est la précision, par exemple 10^{-6}.

Conclusion

- La représentation des réels en machine nécessite de choisir la taille mémoire :
  souvent 4 octets ou 8 octets, des fois 16 octets.
- Les nombres réels représentables en machine sont en nombre fini, ils constituent un ensemble discret.
- Les calculs en flottant peuvent provoquer des **underflow**, **overflow** et **erreurs d'arrondi**.
  Certains standards permettent de gérer des **exceptions** : `Inf`, `NaN`, nombres sous-normaux,...
- On mesure la puissance d'une unité de calcul en virgule flottante en **FLOPS** (en anglais, FLoating point Operations Per Second).
  En juin 2013, l'ordinateur Tianhe-2 de la *NUDT*, Chine, a effectué $33,86$ petaFLOPS $= 33,86 \times 10^{15}$ FLOPS contre 10^{10} FLOPS pour un processeur «normal» à $2.5 - GHz$.

Représenter de l'information

- Afin de représenter et/ou échanger une information, on a besoin d'un code :
  1. pas compliqué et facile à manipuler ;
  2. compréhensible par d'autres que l'émetteur ;
  3. économe en espace ou temps.
- La représentation de quantités par des symboles ou *chiffres* est un premier exemple de code.
- Mais on peut avoir d'autres types d'informations à représenter :
  1. La notation mathématique pour représenter des résultats mathématiques ;
  2. La notation musicale pour représenter des morceaux de musique ;
  3. La notation phonétique pour représenter la prononciation d'un mot ;
  4. Le code braille qui permet aux aveugles de lire grâce au toucher ;

  :
- Ces codes sont utilisables directement par un être humain, ayant plus ou moins d'expérience.