Codage binaire des nombres

DIU Enseigner l'Informatique au Lycée

E. Jahier, L. Rieg, C. Parent-Vigouroux, B. Wack

UFR IM2AG, Université Grenoble Alpes

juilet 2020

Plan

Plan

Un programme « inoffensif »

En Python

Écrire une fonction qui donne le nombre de racines d'un trinôme du second degré.

```
def nb_racines_trinome(a, b, c):
    d = b*b - 4*a*c
    if d < 0:
        return 0
    elif d == 0:
        return 1
    else:
        return 2
```

Un programme « inoffensif »

En Python

Écrire une fonction qui donne le nombre de racines d'un trinôme du second degré.

```
def nb_racines_trinome(a, b, c):
    d = b*b - 4*a*c
    if d < 0:
        return 0
    elif d == 0:
        return 1
    else:
        return 2
```

```
>>> nb_racines_trinome(1, 6, 9)
1
>>> nb_racines_trinome(0.1, 0.6, 0.9)
0
```

- Qu'y a-t-il de si différent entre $X^2 + 6X + 9$ et $0,1X^2 + 0,6X + 0,9$?

Un cas d’école

Le 25 février 1991 (un mois après la première mise en service du système), un missile Patriot échoue (et de loin !) à intercepter un missile Scud à Dharan.

Ce missile automatisé :

- détecte une cible (un autre missile en mouvement)
- détermine sa vitesse et son cap
- calcule et corrige en temps réel la trajectoire à suivre pour intercepter la cible

Pas de problème sur le matériel, et le système avait fonctionné précédemment avec 50 à 80% de succès.
Mais la batterie ici mise en défaut était en fonction depuis longtemps (100 heures) : quelle différence ?

Plan

Quelques particularités intrinsèques

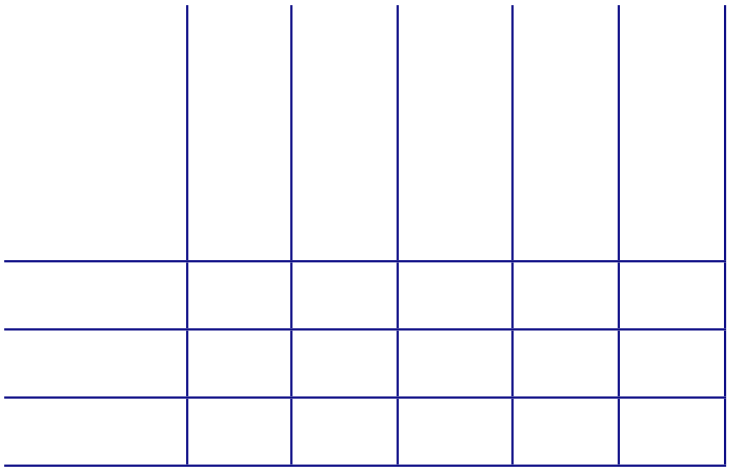
- Base 2 omniprésente
  - le *bit (binary digit)* : 0 ou 1, plus petite unité d'information
  - versatile (nombres, textes, images, sons...)
  - plus robuste qu'une mesure analogique
  - facile à construire (transistors)

Quelques particularités intrinsèques

- Base 2 omniprésente
  - le *bit (binary digit)* : 0 ou 1, plus petite unité d'information
  - versatile (nombres, textes, images, sons...)
  - plus robuste qu'une mesure analogique
  - facile à construire (transistors)
- La taille allouée à un nombre est bornée
  - la mémoire et les disques sont des supports physiques finis
  - le processeur doit pouvoir traiter un calcul efficacement

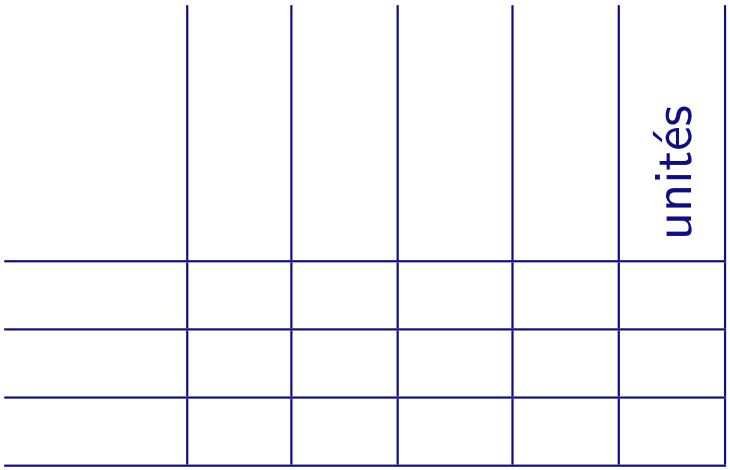
Les entiers

- numération à position



Les entiers

- numération à position



Les entiers

- numération à position

				deuxaines	unités

Les entiers

- numération à position

			quatraines	deuxaines	unités

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1
27 =					

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1
27 =	1	1	0	1	1

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1
27 =	1	1	0	1	1
	1	0	0	1	1

Les entiers

- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1
27 =	1	1	0	1	1
19 =	1	0	0	1	1

Les entiers

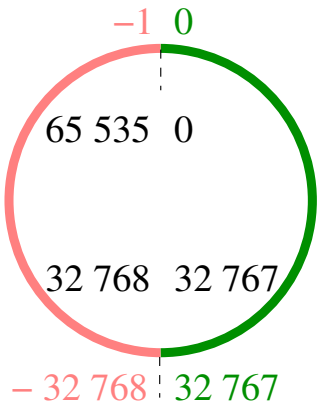
- numération à position

...	seizaines	huitaines	quatraines	deuxaines	unités
5 =	0	0	1	0	1
27 =	1	1	0	1	1
19 =	1	0	0	1	1

- Sur p bits on représente 2^p entiers différents (de 0 à $2^p - 1$)
- Si un calcul dépasse 2^p, on tronque les bits de poids fort (surpassement)
- en général $p = 32$ ou 64 selon la machine, le langage, la version du compilateur...

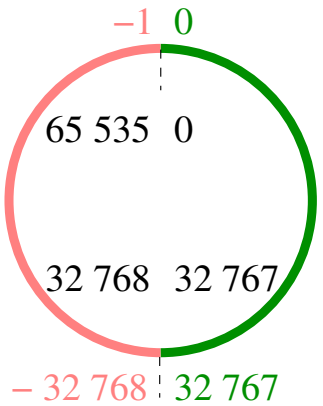
Et les autres ?

- Entiers relatifs
  - un bit de signe (peu pratique à l'usage)
  - plutôt représentés sur p bits par $n + 2^p$ (toujours modulo 2^p)



Et les autres ?

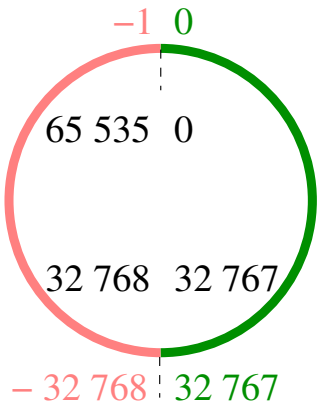
- Entiers relatifs
  - un bit de signe (peu pratique à l'usage)
  - plutôt représentés sur p bits par $n + 2^p$ (toujours modulo 2^p)



- En Python
  - Entiers *longs* (virtuellement illimités)
  - En interne : on écrit l'entier en base 2^{30} et on stocke les « chiffres » dans un tableau dont chaque cellule fait 4 octets.

Et les autres ?

- Entiers relatifs
  - un bit de signe (peu pratique à l'usage)
  - plutôt représentés sur p bits par $n + 2^p$ (toujours modulo 2^p)



- En Python
  - Entiers *longs* (virtuellement illimités)
  - En interne : on écrit l'entier en base 2^{30} et on stocke les « chiffres » dans un tableau dont chaque cellule fait 4 octets.
- Nombres à virgule
  - forcément limités en précision par la représentation machine finie
  - virgule fixe : ok pour la finance mais pas pour la science...

Les nombres en virgule flottante

Principe

Présentation du nombre similaire à la notation scientifique :

$$n = \pm\ m \times 2^e \qquad \text{avec } 1 \leqslant m < 2$$

	signe	exposant	mantisse	total
Simple précision	1 bit	8 bits	23 bits	32 bits
Double précision	1 bit	11 bits	52 bits	64 bits

Les nombres en virgule flottante

Principe

Présentation du nombre similaire à la notation scientifique :

$$n = \pm\ m \times 2^e \qquad \text{avec } 1 \leqslant m < 2$$

	signe	exposant	mantisse	total
Simple précision	1 bit	8 bits	23 bits	32 bits
Double précision	1 bit	11 bits	52 bits	64 bits

Interprétation

- 1 = négatif, 0 = positif
- exposant *biaisé* : représente en réalité $e + 127$ ou $e + 1023$
- la mantisse représente le développement en base 2, avec un premier 1 *implicite* avant la virgule

Exemples (en double précision)

Un exemple de nombre en virgule flottante

signe	exposant	mantisse
1	10010100110	1101000000000000000000...

Exemples (en double précision)

Un exemple de nombre en virgule flottante

signe	exposant	mantisse
1	10010100110	11010000000000000000...
< 0		

Exemples (en double précision)

Un exemple de nombre en virgule flottante

signe	exposant	mantisse
1	10010100110	1101000000000000000...
< 0	$= 1190$	

Exemples (en double précision)

Un exemple de nombre en virgule flottante

signe	exposant	mantisse
1	10010100110	1101000000000000000000...
< 0	$= 1190$	$= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16}$

Exemples (en double précision)

Un exemple de nombre en virgule flottante

signe	exposant	mantisse
1	10010100110	1101000000000000000000...
< 0	$= 1190$	$= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16}$

$$n = -2^{1190-1023} \cdot \frac{29}{16} = -29 \cdot 2^{163} \simeq -3,39068379860769477 \cdot 10^{50}$$

Exemples (en double précision)

Dans l'autre sens

$0,40625 =$

Exemples (en double précision)

Dans l'autre sens

$0,40625 = \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^5} = 0,01101_2$

Exemples (en double précision)

Dans l'autre sens

$0,40625 = \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^5} = 0,01101_2 = 1,101_2 \times 2^{-2}$

signe	exposant	mantisse
0	01111111101	101000...000

Exemples (en double précision)

Dans l'autre sens

$0,40625 = \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^5} = 0,01101_2 = 1,101_2 \times 2^{-2}$

signe	exposant	mantisse
0	01111111101	101000...000

Pour visualiser :

- avec le bout de code fourni :

```
>>> to_bits( '>d', 0.40625)
'0011111111011010000000000000000000000000000000000000000000000000
```

- ou à partir de Python 3.5 :

```
>>> 0.40625.hex()
'0x1.a000000000000p-2'
```

Les limites de ce format

- Plus grand nombre représentable en virgule flottante sur 64 bits :

Les limites de ce format

- Plus grand nombre représentable en virgule flottante sur 64 bits : $(1.11111\ldots1)\cdot 2^{1023} \simeq 1,7977\cdot 10^{308}$
- Plus petit nombre (en valeur absolue) représentable :

Les limites de ce format

- Plus grand nombre représentable en virgule flottante sur 64 bits : $(1.11111\ldots1)\cdot 2^{1023} \simeq 1,7977\cdot 10^{308}$
- Plus petit nombre (en valeur absolue) représentable : $(1.00000\ldots0)\cdot 2^{-1022} \simeq 2,2251\cdot 10^{-308}$

Les limites de ce format

- Plus grand nombre représentable en virgule flottante sur 64 bits : $(1.11111\ldots1)\cdot 2^{1023} \simeq 1,7977\cdot 10^{308}$
- Plus petit nombre (en valeur absolue) représentable : $(1.00000\ldots0)\cdot 2^{-1022} \simeq 2,2251\cdot 10^{-308}$
  (en réalité 2^{-1074} en trichant un peu)

Les limites de ce format

- Plus grand nombre représentable en virgule flottante sur 64 bits : $(1.11111\ldots1)\cdot 2^{1023} \simeq 1,7977\cdot 10^{308}$
- Plus petit nombre (en valeur absolue) représentable : $(1.00000\ldots0)\cdot 2^{-1022} \simeq 2,2251\cdot 10^{-308}$
  (en réalité 2^{-1074} en trichant un peu)
- À titre de comparaison :
  - 10^{80} protons dans l'univers
  - masse d'un électron : 10^{-30} kg ; masse de l'Univers : 10^{60} kg
  - rayon d'un électron : 10^{-15} m ; diamètre de la Voie lactée $= 10^{21}$ m

Connaître la précision de sa machine

- Algorithme de Malcolm :

```
def malcolm():
    x = 1.0
    y = x + 1.0
    i = 0
    while y - x == 1.0:
        x = x * 2.0
        y = x + 1.0
        i = i + 1
    return i
```

Connaître la précision de sa machine

- Algorithme de Malcolm :

```
def malcolm():
    x = 1.0
    y = x + 1.0
    i = 0
    while y - x == 1.0:
        x = x * 2.0
        y = x + 1.0
        i = i + 1
    return i
```

- boucle infinie...

Connaître la précision de sa machine

- Algorithme de Malcolm :

```
def malcolm():
    x = 1.0
    y = x + 1.0
    i = 0
    while y - x == 1.0:
        x = x * 2.0
        y = x + 1.0
        i = i + 1
    return i
```

- boucle infinie...
- et pourtant...

```
>>> malcolm()
53
```

Connaître la précision de sa machine

- Algorithme de Malcolm :

```
def malcolm():
    x = 1.0
    y = x + 1.0
    i = 0
    while y - x == 1.0:
        x = x * 2.0
        y = x + 1.0
        i = i + 1
    return i
```

- boucle infinie...
- et pourtant...

```
>>> malcolm()
53
```

Avec un algorithme similaire (à chercher en exercice) on peut trouver les valeurs possibles de l'exposant.

D'où des phénomènes de *cancellation* :

```
>>> x = 1. + 2**-52
>>> y = 1. - 2**-53
>>> x
1.0000000000000002
>>> y
0.9999999999999999
>>> x - 1
2.220446049250313e-16
>>> y - 1
-1.1102230246251565e-16
>>> x + y
2.0
>>> x - y
3.3306690738754696e-16
>>> x + y + x - y
2.0
>>> x + x + y - y
2.0000000000000004
```

Plan

La base

Plus surprenant :

```
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

La base

Plus surprenant :

```
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

Revenons au binaire

$0,1 =$

La base

Plus surprenant :

```
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

Revenons au binaire

$0,1 = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \ldots = 0,0001100110011\ldots$
qui est un développement infini périodique.

La base

Plus surprenant :

```
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

Revenons au binaire

$0,1 = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \ldots = 0,0001100110011\ldots$
qui est un développement infini périodique.

signe	exposant	mantisse	puis...
0	01111111011	100110011001...1001	10011001...

La base

Plus surprenant :

```
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

Revenons au binaire

$0,1 = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \ldots = 0,0001100110011\ldots$
qui est un développement infini périodique.

signe	exposant	mantisse	puis...
0	01111111011	100110011001...1001	10011001...

De plus on arrondit (en général) au plus proche et les derniers bits sont en réalité 010.

En machine $0,1 \simeq 0,1000000000000000055511115123126$ (sur 64 bits).

Compter en base 10 ?

- Sur calculatrice scientifique : $0,1 + 0,1 + 0,1 == 0,3$
- En Python : `import decimal`

  sur le principe du BCD (Binary Coded Decimal) :
  - 4 bits = 1 chiffre entre 0 et 9 (et on n'exploite pas les 6 autres valeurs possibles)

Plan

Ce qui marche bien

- Si x et y sont proches ($\frac{y}{2} \leqslant x \leqslant 2y$), alors $x - y$ est exact.

Ce qui marche bien

- Si x et y sont proches ($\frac{y}{2} \leqslant x \leqslant 2y$), alors $x - y$ est exact.
- Si $x \neq y$ alors $x - y \neq 0$

```
if (x != y) then z = 1/(x-y)
```

Ce qui marche bien

- Si x et y sont proches ($\frac{y}{2} \leqslant x \leqslant 2y$), alors $x - y$ est exact.
- Si $x \neq y$ alors $x - y \neq 0$

  ```
  if (x != y) then z = 1/(x-y)
  ```

- En l'absence de dépassement de capacité :

$$2 \times x = x + x = 2x$$

$$0.5 \times x = x/2 = \frac{x}{2}$$

Bonnes pratiques

- Privilégier les entiers

$$\frac{1}{1,0001} = \frac{0,9999}{1} \Leftrightarrow \frac{10000}{10001} = \frac{9999}{10000} \Leftrightarrow 9999 \cdot 10001 = 10000 \cdot 10000$$

Bonnes pratiques

- Privilégier les entiers

$$\frac{1}{1,0001} = \frac{0,9999}{1} \Leftrightarrow \frac{10000}{10001} = \frac{9999}{10000} \Leftrightarrow 9999 \cdot 10001 = 10000 \cdot 10000$$

- Se limiter dans les opérations
  - Addition et soustraction entre nombres de mêmes ordres de grandeur
  - Multiplication et division par 2
- Réorganiser les opérations
  - Factoriser : $\sqrt{x^2 + y^2} = |x| \sqrt{1 + \left(\frac{y}{x}\right)^2}$
  - Réorganiser les sommes dans l'ordre des valeurs absolues des termes

Bonnes pratiques

- Privilégier les entiers

$$\frac{1}{1,0001} = \frac{0,9999}{1} \Leftrightarrow \frac{10000}{10001} = \frac{9999}{10000} \Leftrightarrow 9999 \cdot 10001 = 10000 \cdot 10000$$

- Se limiter dans les opérations
  - Addition et soustraction entre nombres de mêmes ordres de grandeur
  - Multiplication et division par 2
- Réorganiser les opérations
  - Factoriser : $\sqrt{x^2 + y^2} = |x| \sqrt{1 + \left(\frac{y}{x}\right)^2}$
  - Réorganiser les sommes dans l'ordre des valeurs absolues des termes
- Bannir le test à zéro
  - `a == b` n'a en général pas de sens
  - Préférer `abs(a-b) < epsilon` (marge d'erreur)
    ou encore mieux `math.isclose(a, b)` (à partir de Python 3.5)

En résumé

- Pour la plupart des situations, ça se passe bien.

En résumé

- Pour la plupart des situations, ça se passe bien.
- Se méfier :
  - des itérations
  - et surtout des itérations instables

En résumé

- Pour la plupart des situations, ça se passe bien.
- Se méfier :
  - des itérations
  - et surtout des itérations instables
- Tester le même algorithme sur plusieurs systèmes distincts (base, précision) reste un bon critère.

Retour sur le missile Patriot

Dans la batterie de missile, afin de calculer la trajectoire d'interception, un compteur compte les 1/10 de seconde.
Puis ce nombre est multiplié par 0.1 codé sur 24 bits en virgule fixe : `0.00011001100110011001100`

Pour transposer ce choix en virgule flottante, il faut n'utiliser qu'une faible portion de la mantisse :
au lieu de `'0x1.99999999999ap-4'`
on considère `'0x1.999800000000p-4'`

- Que vaut le compteur au bout de 100 heures ?
- Sachant qu'un missile Scud se déplace à environ 1 676 mètres par seconde, quel écart de distance cela donne-t-il ?

Bibliographie

- William Kahan, *Why do we need a floating-point arithmetic standard ?*, 1981
- Jean-Michel Muller et al., *Handbook of Floating-Point Arithmetic*, 2009
- *The new IEEE 754-2008 standard for Floating-Point Arithmetic*