Mathématiques assistées par ordinateur

Chapitre 1 : Introduction

Michael Eisermann

Mat249, DLST L2S4, Année 2008-2009
www-fourier.ujf-grenoble.fr/~eiserm/cours#mao
Document mis à jour le 6 juillet 2009

Objectifs de ce chapitre

Le principal but de ce premier chapitre est de donner une introduction rapide au calcul sur ordinateur, afin de poser les fondements et de consolider les premières expériences pratiques en TD et TP.

On commence par la représentation des nombres entiers, rationnels, réels, complexes et les opérations de calcul qui leur sont propres.

De coté mathématique, je suppose acquis la définition/construction des entiers $\mathbb{Z}$, des rationnels $\mathbb{Q}$, des réels $\mathbb{R}$, et des complexes $\mathbb{C}$.

De coté ordinateur, il faut distinguer le calcul exact dans $\mathbb{Z}$ et $\mathbb{Q}$ et le calcul approché dans $\mathbb{R}$ et $\mathbb{C}$. Aucun ordinateur ne peut implémenter les nombres réels ou complexes, tant aimés des mathématiciens !

La dichotomie « exact vs approché » nous occupera tout le long du semestre, et il vaut mieux se familiariser dès le début avec ses particularités, notamment du calcul approché. Concrètement, dans ce premier chapitre, il faut bien assimiler toutes les règles de bon sens.

Sommaire

Avant de calculer il faut représenter les nombres

D'abord nous cherchons une numération qui se prête bien au calcul.

DXXXVII	537	MMVIII	2008	XLIII	43
$+$LXXIX	$+79$	$-$LXXIX	-79	$\times$LXXIX	$\times 79$
DMXVI	616	MCMXXIX	1929	MMMXXXIIIX	3397

La numération romaine ne facilite pas le calcul. Voici un extrait d'une lettre écrite au moyen âge par un savant à un riche marchand. Ce dernier cherchait conseil pour savoir où envoyer son enfant étudier la science des nombres et du calcul.

> *Si tu désires l'initier à la pratique de l'addition ou de la soustraction, n'importe quelle université française, allemande ou anglaise fera l'affaire. Mais, pour la maîtrise de l'art de multiplier ou de diviser, je te recommande plutôt de l'envoyer dans une université italienne.*

La numération positionnelle a depuis révolutionné le calcul !

Ainsi elle a contribué à démocratiser les mathématiques.

Numération positionnelle des nombres entiers

Rappelons ce merveilleux outil qui est la numération positionnelle.

En base 10 : $\mathbf{2008}_{\text{dec}} = \mathbf{2} \cdot 10^3 + \mathbf{0} \cdot 10^2 + \mathbf{0} \cdot 10^1 + \mathbf{8} \cdot 10^0$

En base 2 : $\mathbf{1011}_{\text{bin}} = \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0 = 11_{\text{dec}}$

Comment obtenir le développement en base 2 ?

$$\mathbf{1011}_{\text{bin}} = \underbrace{(((\mathbf{1} \cdot 2 + \mathbf{0}) \cdot 2 + \mathbf{1}) \cdot 2}_{q \cdot 2} + \underbrace{\mathbf{1}}_{+r}$$

Algorithme de conversion : On itère la division euclidienne.

$$\begin{array}{rcl} a & = & 2 \cdot \text{quo} + \text{rem} \\ \hline 11 & = & 2 \cdot 5 + \mathbf{1} \\ 5 & = & 2 \cdot 2 + \mathbf{1} \\ 2 & = & 2 \cdot 1 + \mathbf{0} \\ 1 & = & 2 \cdot 0 + \mathbf{1} \end{array}$$

Cet algorithme marche pour tout entier, en n'importe quelle base.
Le même algorithme nous servira plus tard pour les polynômes.

La division euclidienne

Proposition (division euclidienne des entiers)

Pour tout $a, b \in \mathbb{Z}$ où $b \neq 0$ il existe un unique couple $q, r \in \mathbb{Z}$ tel que

$$a = qb + r \quad \textit{et} \quad 0 \leq r < |b|.$$

*On appelle $q =: a \operatorname{quo} b$ le **quotient** et $r =: a \operatorname{rem} b$ le **reste** de la division euclidienne de a par b.*

Démonstration. Exercice de révision !

Corollaire

Soit $B \in \mathbb{Z}$ tel que $B \geq 2$. Tout entier $a \in \mathbb{Z}$ s'écrit de manière unique comme $a = \pm \sum_{k=0}^{\ell-1} a_k B^k$ tel que $a_k \in \{0, 1, \ldots, B-1\}$ et $a_{\ell-1} \neq 0$.

Démonstration. Exercice de révision !

On appelle $\ell \in \mathbb{N}$ la ***longueur*** du développement de a en base B.
On la note $\operatorname{len}_B(a) := \ell$; quand $B = 2$ on écrit $\operatorname{len}(a) := \operatorname{len}_2(a)$.

Les ordinateurs calculent en binaire

On peut effectuer les opérations arithmétiques en base 2. Exemple :

```
  01101011        01101011
+ 01001111      - 01001111
----------      ----------
  10111010        00011100
```

Les microprocesseurs utilisent de tels entiers « machine » :

taille en mémoire	plage des entiers qui peuvent être représentées			
		min		max
1 octet = 8 bits		0	$2^8 - 1 =$	255
variante signée	$-2^7 =$	-128	$2^7 - 1 =$	127
2 octets = 16 bits		0	$2^{16} - 1 =$	65535
variante signée	$-2^{15} =$	-32768	$2^{15} - 1 =$	32767
4 octets = 32 bits		0	$2^{32} - 1 =$	4294967295
variante signée	$-2^{31} =$	-2147483648	$2^{31} - 1 =$	2147483647
8 octets = 64 bits		0	$2^{64} - 1 =$	18446744073709551615
variante signée	$-2^{63} =$	-9223372036854775808	$2^{63} - 1 =$	9223372036854775807

Avantage : les calculs sont très rapides.
Inconvénient : la taille est limitée et fixée d'avance.

Implémentation des grands entiers

Le processeur ne fournit que des entiers à 64 bits (au plus).
Tout calcul dépassant $0, \ldots, 2^{64}-1$ sera tronqué. Exemple :

$$2^{63} + 2^{63} = 0 \qquad \text{avec des entiers machine à 64 bits !}$$

Comment calculer donc avec des grands entiers ? Solution :

$$a = \sum_{k=0}^{\ell-1} a_k B^k \quad \text{où} \quad B = 2^{32}.$$

➤ Un grand entier est un tableau de petits entiers (chiffres).
➤ Les opérations $+$, $-$, $*$, quo, rem s'implémente comme à l'école.

Tout logiciel de calcul formel implémente ainsi les grands entiers :

$$2^{63} + 2^{63} = 18446744073709551616$$

$$50! = 30414093201713378043612608166064768844377641568960512000000000000$$

⚠ Plus les entiers sont longs, plus les calculs sont coûteux.

- Addition et soustraction sont de complexité linéaire : $O(\ell)$.
- Multiplication et division sont de complexité quadratique : $O(\ell^2)$.

Pour les grands entiers il existe des algorithmes sous-quadratiques (Karatsuba, FFT) donc plus efficaces que l'algorithme scolaire.

L'algorithme d'Euclide

Tous les calculs numériques, aussi complexes qu'ils soient, sont basés sur les opérations élémentaires $+$, $-$, $*$, quo, rem des entiers.

Voici un premier exemple célèbre :

Algorithme 1 pgcd de deux entiers selon Euclide

Entrée: deux entiers $a, b \in \mathbb{Z}$

Sortie: le pgcd positif de a et b

---

tant que $b \neq 0$ **faire**
  $r \leftarrow a \operatorname{rem} b, \quad a \leftarrow b, \quad b \leftarrow r$
fin tant que
si $a < 0$ **alors** $a \leftarrow -a$
retourner a

Proposition

Pour tout $a, b \in \mathbb{Z}$ l'algorithme ci-dessus se termine et renvoie le pgcd positif de a et b. Il nécessite au plus $2\operatorname{len}(a)$ itérations.

Démonstration. Exercice de révision !

Nombres rationnels

On sait représenter les entiers $a, b \in \mathbb{Z}$ et effectuer leurs opérations.

Tout nombre rationnel $x \in \mathbb{Q}$ s'écrit comme $x = \frac{a}{b}$ avec $a, b \in \mathbb{Z}$, $b \neq 0$.

On réduit chaque fraction de sorte que $\mathrm{pgcd}(a, b) = 1$ et $b > 0$.

➤ Pour représenter x on stocke simplement a et b.

➤ On implémente les opérations arithmétiques comme suit :

$$\frac{a}{b} \pm \frac{c}{d} = \frac{ad \pm bc}{bd}, \qquad \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd}, \qquad \frac{a}{b} / \frac{c}{d} = \frac{ad}{bc},$$

bien sûr suivi de la réduction des fractions (calcul du pgcd dans $\mathbb{Z}$).

☞ Tous les calculs numériques, aussi complexes qu'ils soient, sont basés sur les opérations élémentaires $+$, $-$, $*$, quo, rem des entiers.

Conclusion

Les calculs avec les rationnels sont exacts et a priori sans limitation.
Les seuls facteurs limitant sont le temps et la mémoire disponibles.

Un défaut des nombres rationnels

Proposition

Il n'existe pas de nombre rationnel $r \in \mathbb{Q}$ tel que $r^2 = 2$.

Démonstration. Soit $r = \frac{a}{b}$ tel que $a, b \in \mathbb{Z}$ et $b \neq 0$. Quitte à réduire la fraction $\frac{a}{b}$, on peut supposer que $\mathrm{pgcd}(a, b) = 1$. Supposons que $r^2 = \frac{a^2}{b^2} = 2$. Alors $a^2 = 2b^2$. Comme 2 divise a^2, l'entier a doit être pair, donc $a = 2\bar{a}$ avec $\bar{a} \in \mathbb{Z}$. On a alors $a^2 = 4\bar{a}^2 = 2b^2$ et ainsi $b^2 = 2a^2$. Maintenant 2 divise b^2, l'entier b aussi doit donc être pair. Ceci contredit notre hypothèse que $\mathrm{pgcd}(a, b) = 1$. Il n'existe donc pas de nombre rationnel r tel que $r^2 = 2$. □

⚠ Les nombres rationnels ne sont pas « complets ».
Ainsi ils ne suffisent pas pour l'analyse (limites, dérivées, intégrales).

Les nombres réels

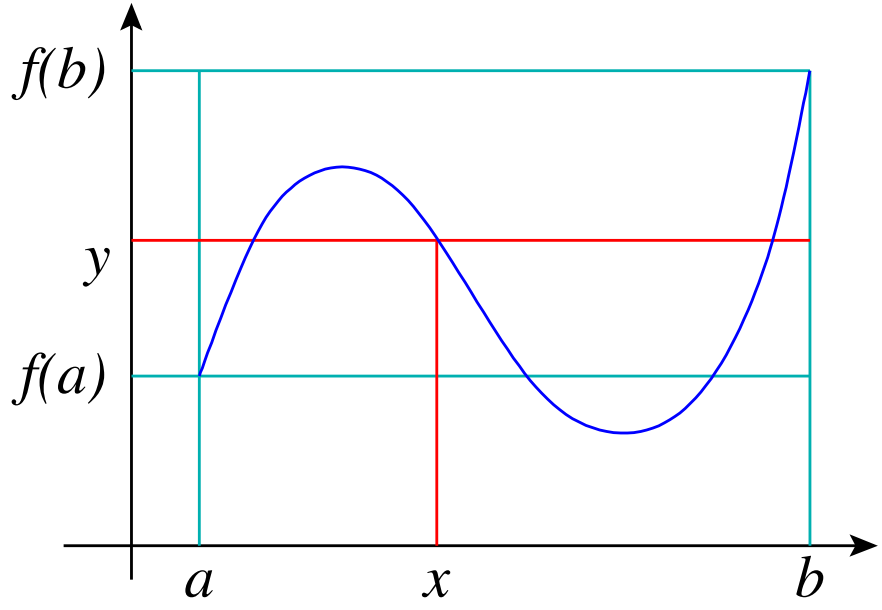
Existe-t-il des racines, en fait ?

La réponse dépend d'une construction subtile : les nombres réels $\mathbb{R}$!

On note $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ l'intervalle réel entre a et b.

Théorème (des valeurs intermédiaires, TVI)

Soit $f : [a, b] \to \mathbb{R}$ une fonction continue. Pour tout réel y vérifiant $f(a) < y < f(b)$ il existe (au moins) un $x \in [a, b]$ tel que $f(x) = y$. □

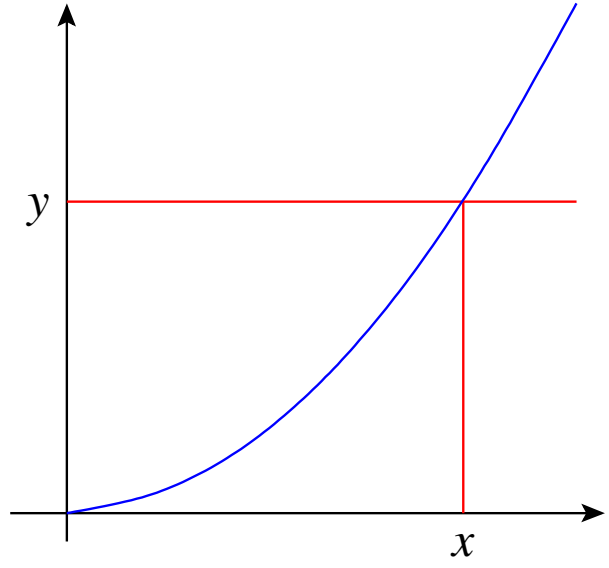


Ce résultat est caractéristique pour les nombres réels $\mathbb{R}$.

Il serait faux pour les rationnels, par exemple $f : \mathbb{Q} \to \mathbb{Q}$, $f(x) = x^2$.

Nombres réels et racine $n$ième

Considérons la fonction $f : \mathbb{R}_+ \to \mathbb{R}_+$ définie par $f(x) = x^n$.



Elle est continue et strictement croissante.
Elle vérifie $f(0) = 0$ et $f(x) \to +\infty$ pour $x \to +\infty$.

Corollaire (existence des racines réelles)

Pour tout réel $y \in \mathbb{R}_{\geq 0}$ il existe un unique réel $x \in \mathbb{R}_{\geq 0}$ tel que $x^n = y$.
On le note $x =: \sqrt[n]{y}$, ou bien $x =: \sqrt{y}$ dans le cas $n = 2$.

On a vu que $\sqrt{2}$ n'est pas rationnel. Donc $\sqrt{2} \in \mathbb{R}$ mais $\sqrt{2} \notin \mathbb{Q}$.

Comment calculer avec des réels ?

L'ordinateur peut effectuer des calculs exacts dans $\mathbb{Z}$ et dans $\mathbb{Q}$.

Problème : certaines valeurs comme $\sqrt{2}$ ne sont pas rationnelles.

Comment effectuer de tels calculs sur ordinateur ?

Deux méthodes sont possibles :

- **Calcul formel :** on manipule formellement des symboles et des formules, toujours **exacts**. Exemple :

  entrée `r:= sqrt(2)` $\Longrightarrow$ sortie $\sqrt{2}$
  entrée `r^2-2` $\Longrightarrow$ sortie `0`

- **Calcul approché :** on approche tout nombre réel par un développement (binaire) fini, typiquement **inexact**. Exemple :

  entrée `a:= evalf(r)` $\Longrightarrow$ sortie `1.41421356237`
  entrée `a^2-2` $\Longrightarrow$ sortie `4.4408920985e-16`

 Chaque méthode a ses avantages et ses inconvénients.
Selon l'application que vous envisagez, c'est à vous de choisir.

Pour faire un choix informé, il faut comprendre les calculs !

Bonnes et mauvaises notations

Comment écrire une approximation de $r := \sqrt{2}$?

Faux — N'écrivez jamais $\sqrt{2} = 1.41421356$: c'est faux parce que

$$1.41421356^2 = 1.99999999932878736 < 2$$

Très flou — $r \approx 1.41421356$ n'est pas faux, mais $r \approx 5$ n'est pas faux non plus. Problème : Ce qui compte est la marge d'erreur.

Moins flou — $r = 1.41421356\ldots$ suggère que tous les chiffres indiqués sont corrects. (Vraiment ?)

La bonne notation — Encadrement $1.41421356 < r < 1.41421357$ ou approximation avec marge d'erreur : $|r - 1.41421356| \leq 10^{-8}$.

Cette écriture est honnête et lisible. En particulier c'est un énoncé précis que l'on peut vérifier. (En l'occurrence il s'avère correct.)

Règle de bon sens

Avec n décimales on ne peut approcher qu'à 10^{-n} près.

Réciproquement, si la précision est d'ordre de 10^{-n}, il est inutile, voire trompeur, de rajouter des chiffres non significatifs.

Nombres à virgule fixe

Tout réel admet un développement décimal ou binaire :
$\frac{1}{3} = 0.3333333333\ldots_{\text{dec}} = \quad 0.0101010101010101\ldots_{\text{bin}}$ (périodique)
$\pi = 3.1415926535\ldots_{\text{dec}} = 11.0010010000111111\ldots_{\text{bin}}$ (non périodique)

Développement en base B

Soit $B \in \mathbb{Z}$ tel que $B \geq 2$. Tout nombre réel $x \in \mathbb{R}$ peut être développé en une série $x = \pm \sum_{k=-n}^{\infty} a_k B^{-k}$ de sorte que $a_k \in \{0, 1, \ldots, B-1\}$.


Aucun ordinateur ne peut implémenter les nombres réels $\mathbb{R}$!

Sur ordinateur on ne peut stocker que le début du développement :

$$\hat{x} = a_{-n}B^n + \cdots + a_0 B^0 + a_1 B^{-1} + \cdots + a_m B^{-m}$$

Évidemment, une certaine erreur d'arrondi ($\leq B^{-m}$) est inévitable.

Réconfort : on peut très bien travailler avec des approximations...
mais il faut les manipuler intelligemment puis interpréter les résultats.

Rendons la virgule flottante !

Les nombres à **virgule fixe** ont un grave défaut pratique :
ils sont inefficaces pour des nombres très grands comme

$$N = 602200000000000000000000.0$$

ou des nombres très petits comme

$$h = 0.00000000000000000000000000000006626.$$

Mieux adaptés sont les nombres à **virgule flottante** :

$$N = 6.022 \cdot 10^{23}, \qquad h = 6.626 \cdot 10^{-34}.$$

Le principe est de décaler la virgule en compensant par l'exposant :

$$123.456 = 1234.56 \cdot 10^{-1} = 12345.6 \cdot 10^{-2} = 123456 \cdot 10^{-3}$$

Notation dite « scientifique » utilisée sur ordinateur

En base 10 nous écrivons `1.2345e-67` pour $1.2345 \cdot 10^{-67}$.

⚠ À ne pas confondre avec $1.2345e^{-67} = 1.2345 \cdot \exp(-67)$.

Nombres à virgule flottante

Exemples de nombres flottants : $N = 6.022 \cdot 10^{23}$, $h = 6.626 \cdot 10^{-34}$.

En général, les nombres à virgule flottante sont de la forme $m \cdot B^e$ où

- $m \in \mathbb{Z}$ est appelé la ***mantisse***,
- $B = 2$ ou $B = 10$ est la ***base***, et
- $e \in \mathbb{Z}$ est ***l'exposant***.

Calcul arrondi : on arrondit la mantisse à une longueur fixée ℓ.
On veut donc restreindre la mantisse à $m \in \{1 - B^\ell, \ldots, B^\ell - 1\}$.
Pour ce faire on supprime les chiffres les moins significatifs :
Si $\mathrm{len}_B(m) = \ell + s$, alors $m \leftarrow m \operatorname{quo} B^s(\pm 1)$ et $e \leftarrow e + s$.

Exemple : le type `double` calcule en binaire ($B = 2$). Il prévoit $\ell = 53$ bits pour la mantisse, 11 bits pour l'exposant, 1 bit pour le signe.

Erreur d'arrondi : avec mantisse de longueur ℓ on peut approcher tout nombre réel à une erreur relative de $B^{1-\ell}$ près. ($2^{-53} \approx 10^{-16}$)

Opérations sur les nombres à virgule flottante

Multiplions $N = 6.022 \cdot 10^{23}$ et $h = 6.626 \cdot 10^{-34}$!

Résultat exact : $Nh = 39.901772 \cdot 10^{-11} = 3.9901772 \cdot 10^{-10}$.

On l'arrondit à $Nh \approx 3.990 \cdot 10^{-10}$ (mantisse à 4 décimales).

Principe du calcul arrondi

Pour les nombres à virgule flottante $x = m_1 B^{e_1}$ et $y = m_2 B^{e_2}$
les opérations $x + y$, $x - y$, $x * y$, x/y sont définies comme suit :

Partant du résultat exact $z = mB^e$ on arrondit à $\hat{z} = \hat{m}B^{\hat{e}}$ en réduisant la mantisse à la longueur prescrite (de ℓ chiffres).

Avantage : la mantisse reste petite, les calculs sont rapides.

Inconvénient : chaque opération introduit une erreur d'arrondi.

Perspective : calcul arrondi fiable

On a le choix d'arrondir vers le plus proche (le standard),
vers $-\infty$, vers $+\infty$, vers 0, ou vers $\pm\infty$ (arrondis dirigés).
Au lieu d'une approximation $\hat{z} \approx z$ sans contrôle d'erreur on garantit ainsi un encadrement $\underline{z} \leq z \leq \overline{z}$. Cette ***arithmétique d'intervalles*** permet de faire des calculs rigoureux avec des nombres flottants.

Les flottants exigent des précautions particulières !

Un exemple perturbant

```
s:= 0.0;
for( k:=1; k<=10; k:=k+1) { s:= s + 0.1; };
if ( s = 1.0 ) print("Le compte est bon."); else
print("Vous êtes accusé de détournement de fonds !");
```

Quel en sera le résultat ? Ça dépend !

- de la représentation des nombres : décimal ou binaire ?
- des arrondis effectués lors des calculs approchés.
- des aléas de la propagation des erreurs.

Analogie en base 10 : $\frac{1}{3} \approx 0.33333$. On a $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$ (exact) mais $0.33333 + 0.33333 + 0.33333 = 0.99999 < 1$ (erreur d'arrondi).

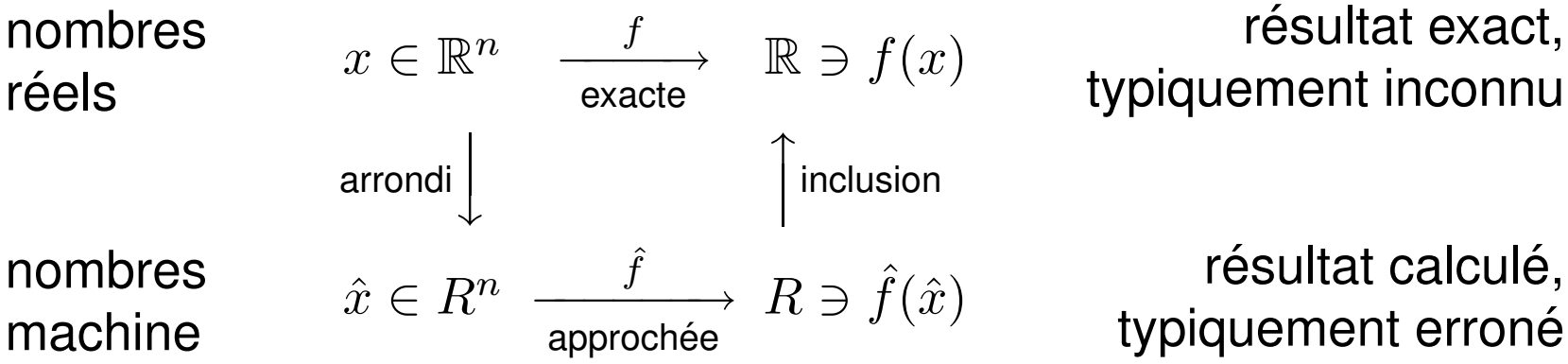
Exercice pratique : tester ces calculs sous Xcas puis sous C/C++ avec les types `float`, `double`, ou `long double`. (Surprise !)

Exercice mathématique : développer 0.1_{dec} en binaire, arrondir (à 24 ou 53 bits), puis effectuer les calculs (arrondis à 24 ou 53 bits).

Le problème fondamental du calcul approché

Deux sources d'erreurs / d'imprécisions :

1. erreurs dans les données initiales (mesurées, approchées)
2. erreurs d'arrondi lors des calculs (nombres à virgule flottante)



Problème fondamental du calcul approché

En général ce diagramme ne commute pas, c'est-à-dire $f(x) \neq \hat{f}(\hat{x})$.
Seule la marge d'erreur donne autorité au résultat calculé $\hat{f}(\hat{x})$.

Il est indispensable de majorer l'erreur totale $|f(x) - \hat{f}(\hat{x})|$.
Sinon il est inutile, voire nuisible, de se lancer dans le calcul.

Erreurs absolue et relative

Notons x, y, z les valeurs exactes et $\hat{x}, \hat{y}, \hat{z}$ des valeurs approchées.

Les deux mesures d'erreurs

Erreur absolue $\Delta x := |\hat{x} - x|$, ***erreur relative*** $\delta x := \frac{|\hat{x}-x|}{|x|}$.

Rappel : les flottants mB^e où la mantisses a ℓ chiffres approchent tout réel à une erreur relative de $B^{1-\ell}$ près.

Comment les erreurs se propagent-elles dans les calculs ?

Comparons somme exacte $z = x + y$ et somme approchée $\hat{z} = \hat{x} + \hat{y}$:

$$\begin{aligned}\Delta z = |\hat{z} - z| &= |(\hat{x} + \hat{y}) - (x + y)| = |(\hat{x} - x) + (\hat{y} - y)| \\ &\leq |\hat{x} - x| + |\hat{y} - y| = \Delta x + \Delta y\end{aligned}$$

Observation

Lors d'une addition, les erreurs ***absolues*** s'additionnent (au pire).

Propagation d'erreurs

Comparons le produit exact $z = xy$ au produit approché $\hat{z} = \hat{x}\hat{y}$:

$$\begin{aligned}\delta z = \frac{|\hat{z} - z|}{|z|} &= \frac{|\hat{x}\hat{y} - xy|}{|xy|} = \frac{|\hat{x}\hat{y} - x\hat{y} + x\hat{y} - xy|}{|xy|} \\ &\leq \frac{|\hat{x}\hat{y} - x\hat{y}| + |x\hat{y} - xy|}{|xy|} = \frac{|\hat{x}\hat{y} - x\hat{y}|}{|xy|} + \frac{|x\hat{y} - xy|}{|xy|} \\ &= \frac{|\hat{x} - x|}{|x|} \cdot \frac{|\hat{y}|}{|y|} + \frac{|\hat{y} - y|}{|y|} = \delta x \cdot \frac{|\hat{y}|}{|y|} + \delta y \approx \delta x + \delta y\end{aligned}$$

Observation

Lors d'une multiplication, les erreurs ***relatives*** s'additionnent.

Regardons finalement une fonction $f : \mathbb{R} \to \mathbb{R}$, supposée dérivable :

$$|f(\hat{x}) - f(x)| = \frac{|f(\hat{x}) - f(x)|}{\Delta x} \cdot \Delta x \approx |f'(x)| \cdot \Delta x$$

Observation

Une application $x \mapsto f(x)$ multiplie l'erreur absolue par $|f'(x)|$.

Nombres complexes

Tout nombre complexe $z \in \mathbb{C}$ s'écrit comme $z = x + iy$ avec $x, y \in \mathbb{R}$.

Ici $i^2 = -1$, donc $(x + iy)(x' + iy') = (xx' - yy') + i(xy' + yx')$.

On appelle x la partie réelle et y la partie imaginaire.

Représentations sur ordinateur :

- calcul formel : x, y sont exacts (entiers ou rationnels).
- calcul approché : x, y sont approchés (nombres flottants).

Le comportement diffère drastiquement !

`solve(z^3-1=0,z)` donne les trois racines exactes

$$1, \qquad -\frac{1}{2} + i\frac{\sqrt{3}}{2}, \qquad -\frac{1}{2} - i\frac{\sqrt{3}}{2}.$$

`solve(z^3-1.0=0.0,z)` donne seulement une approximation

$$-0.500000000004 - 0.866025403782i.$$

Bien entendu ce dernier résultat n'est pas exact :

$$(-0.500000000004 - 0.866025403782i)^3 - 1 \approx -3.36 \cdot 10^{-13}.$$

Polynômes et fractions rationnelles

Après les nombres entiers, rationnelles, réels, complexes,
les objets les plus utilisés sont les polynômes et leur fractions :

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$$

$$\frac{P(X)}{Q(X)} = \frac{a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0}{b_m X^m + b_{m-1} X^{m-1} + \cdots + b_1 X + b_0}$$

➤ On peut stocker un polynôme par la liste de ces coefficients.
➤ On peut stocker une fraction par numérateur & dénominateur.
(Bien sûr, il existe aussi des polynômes à plusieurs variables.)

Comme toujours, deux approches sont possibles :

- calcul formel : les coefficients sont exacts.
- calcul approché : les coefficients sont des flottants.

Opérations de base (voir le TD/TP) :

- opérations arithmétiques, notamment la division euclidienne
- algorithme d'Euclide pour le pgcd, comme vu pour les entiers
- évaluation $P(z)$ en un point z par la méthode de Horner
- développement en $(X - z)$, interpolation en plusieurs points

Expressions et fonctions

À noter : on distingue l'expression $\sin(x) + x^2$
et la fonction $f : \mathbb{R} \to \mathbb{R}$ définie par $f(x) = \sin(x) + x^2$.

C'est déjà le cas en mathématiques, et plus encore en calcul formel !

Raison : différentes expressions peuvent définir la même fonction !

Exemple : Les expressions $\sin(x)^2 + \cos(x)^2$ et 1 sont différentes, tandis que les fonctions associées $g, f : \mathbb{R} \to \mathbb{R}$ définies par $g(x) = \sin(x)^2 + \cos(x)^2$ et $f(x) = 1$ sont identiques.

Calcul formel : l'ordinateur (comme nous d'ailleurs) travaille sur des expressions formelles comme ci-dessus. Dans des cas favorables il arrive à les simplifier, ou à montrer leur équivalence : voir le TP !

Exemple : `simplify( sin(Pi/4)^2 + cos(Pi/4)^2 - 1 )` donne `0.`

Calcul approché : on évalue les expressions en nombres flottants.

`evalf( sin(Pi/4)^2 + cos(Pi/4)^2 - 1 )` donne `2.22044e-16.`

Conteneurs, vecteurs et matrices

Un ***conteneur*** regroupe des objets plus simples.

On accède à ces objets par un indice (adapté au conteneur).

Exemples : liste, séquence, tableau, ensemble, . . . (toujours finis !)

Vecteurs et matrices sont fréquemment utilisés (algèbre linéaire).

Comme toujours il faut distinguer calcul exact et calcul approché :

```
P:=matrix(3,3,[8,-5,4,-5,-1,-5,-6,4,-3]); Q:= P^-1; P*Q;
```

$$P = \begin{pmatrix} 8 & -5 & -4 \\ -5 & -1 & 5 \\ -6 & 4 & 3 \end{pmatrix},\ Q = \begin{pmatrix} \frac{23}{5} & \frac{1}{5} & \frac{29}{5} \\ 3 & 0 & 4 \\ \frac{26}{5} & \frac{2}{5} & \frac{33}{5} \end{pmatrix},\ PQ = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
P:=matrix(3,3,[8.0,-5,-4,-5,-1,5,-6,4,3]); Q:= P^-1; P*Q;
```

$$P = \begin{pmatrix} 8.0 & -5 & -4 \\ -5 & -1 & 5 \\ -6 & 4 & 3 \end{pmatrix},\ Q = \begin{pmatrix} 4.6 & 0.2 & 5.8 \\ 3.0 & 0.0 & 4.0 \\ 5.2 & 0.4 & 6.6 \end{pmatrix},\ PQ = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 3 \cdot 10^{-15} & 0.0 & 1.0 \end{pmatrix}$$

Résumé : représentation des données

Types élémentaires :

- exacts : nombres entiers et rationnels, variables formelles, ...
- approchés : nombres à virgule flottante (réels, complexes)

Types composés

- polynômes $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, fractions, ...
- expressions x^2, $\sin(x + x^2)$, $\frac{\exp(x)-1}{x}$, ...
- conteneurs, listes, vecteurs, matrices, ...

Un objet composé est en ***mode exact*** si tous ses sous-objets le sont. Sinon il est en ***mode approché***. Les algorithmes en dépendent !

La commande `evalf` passe du mode ***exact*** au mode ***approché***. Elle est « sens unique » ! (Pourquoi ?)

Passons à la pratique !

Lisez l'introduction à Xcas, puis expérimentez abondamment !
Familiarisez-vous avec votre calculatrice, elle vous servira partout !