

Preuves formelles et équation des ondes

Sylvie Boldo – Équipe-projet ProVal

2 février 2009

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



INRIA

centre de recherche **SACLAY - ÎLE-DE-FRANCE**

Preuves formelles et équation des ondes

Le Coq et la Corde

Sylvie Boldo – Équipe-projet ProVal

2 février 2009

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



INRIA

centre de recherche **SACLAY - ÎLE-DE-FRANCE**

Motivations

Les ordinateurs calculent tout et n'importe quoi :

- prévisions météo,
- BTP,
- trajectoire d'avions,
- ...

Et pourtant, qu'est-ce qui **garantit** que le résultat calculé est correct ?

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

475 millions de \$.

Ariane 5 : explosion consécutive à un overflow

500 millions de \$.

Motivations

But : sécurité des codes numériques

Motivations

But : sécurité des codes numériques

Outils : méthodes formelles

Motivations

But : **sécurité des codes numériques**

Outils : **méthodes formelles**

⇒ on va vérifier ce qui est **vraiment écrit** par les programmeurs !

Motivations

- On commence à pouvoir **spécifier et prouver formellement** des programmes avec des nombres flottants.

Motivations

- On commence à pouvoir **spécifier et prouver formellement** des programmes avec des nombres flottants.
- Regardons de **vrais** programmes d'analyse numérique

Motivations

- On commence à pouvoir **spécifier et prouver formellement** des programmes avec des nombres flottants.
- Regardons de **vrais** programmes d'analyse numérique
- Bon d'accord, on en regarde un facile. . .

Motivations

- On commence à pouvoir **spécifier et prouver formellement** des programmes avec des nombres flottants.
- Regardons de **vrais** programmes d'analyse numérique
- Bon d'accord, on en regarde un facile. . .

- Ce travail est
 - ▶ financé par l'ANR blanche CerPAN (2005–2009)

Motivations

- On commence à pouvoir **spécifier et prouver formellement** des programmes avec des nombres flottants.
- Regardons de **vrais** programmes d'analyse numérique
- Bon d'accord, on en regarde un facile. . .

- Ce travail est
 - ▶ financé par l'ANR blanche CerPAN (2005–2009)
 - ▶ en collaboration avec **François Clément** (INRIA Paris - Rocquencourt), **Jean-Christophe Filliâtre** (CNRS, LRI) et **Micaela Mayero** (Université Paris 13).

Plan

- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Plan

- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Nombre flottant

Ce n'est qu'une **suite de bits**,

11100011010010011110000111000000

Nombre flottant

Ce n'est qu'une **suite de bits**,

11100011010010011110000111000000

qui sera interprétée en fonction des des valeurs de s (signe), e (exposant) et f (fraction).

1	11000110	10010011110000111000000
s	e	f

Nombre flottant

On lui associe une **valeur réelle** :

$$\begin{array}{ccc} \boxed{1} & \boxed{11000110} & \boxed{10010011110000111000000} \\ s & e & f \\ \downarrow & \downarrow & \downarrow \\ (-1)^s \times & 2^{e-B} \times & 1 \bullet f \\ \\ (-1)^1 \times & 2^{198-127} \times & 1.10010011110000111000000_2 \\ & & -2^{54} \times 206727 \approx -3.724 \times 10^{21} \end{array}$$

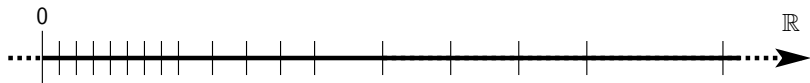
Nombre flottant

On lui associe une **valeur réelle** :

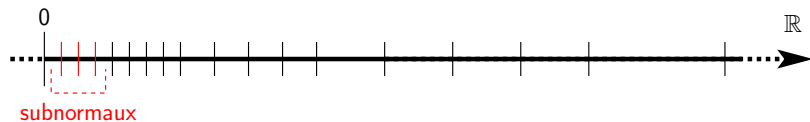
$$\begin{array}{ccc} \boxed{1} & \boxed{11000110} & \boxed{10010011110000111000000} \\ s & e & f \\ \downarrow & \downarrow & \downarrow \\ (-1)^s \times & 2^{e-B} \times & 1 \bullet f \\ \\ (-1)^1 \times & 2^{198-127} \times & 1.10010011110000111000000_2 \\ & -2^{54} \times 206727 \approx & -3.724 \times 10^{21} \end{array}$$

sauf pour des valeurs spéciales de e : ± 0 , $\pm \infty$, NaN, dénormalisés.

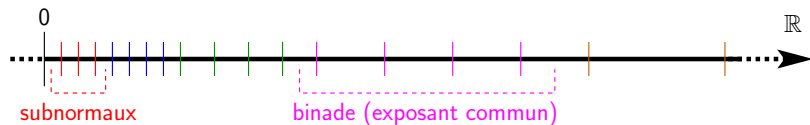
Répartition des nombres à virgule flottante



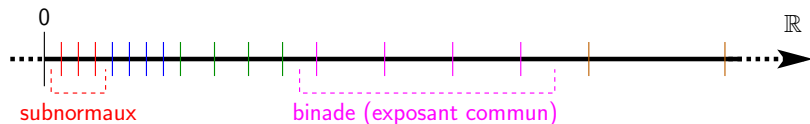
Répartition des nombres à virgule flottante



Répartition des nombres à virgule flottante

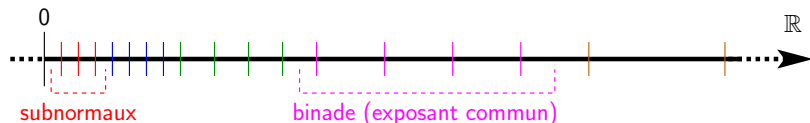


Répartition des nombres à virgule flottante



On utilise des propriétés triviales sur l'erreur absolue d'un calcul :

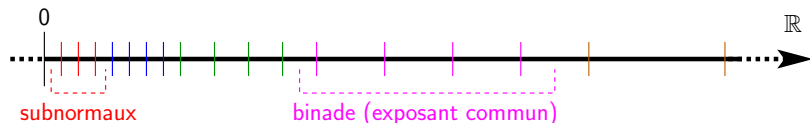
Répartition des nombres à virgule flottante



On utilise des propriétés triviales sur l'erreur absolue d'un calcul :

Si $|x| \leq 1$ et $|y| \leq 2$,

Répartition des nombres à virgule flottante



On utilise des propriétés triviales sur l'erreur absolue d'un calcul :

Si $|x| \leq 1$ et $|y| \leq 2$,

alors $|x \ominus y| \leq 3$ et $\varepsilon_{x \ominus y} \leq 3 \cdot 2^{-53} + \varepsilon_x + \varepsilon_y$.

Plan

- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Preuve formelle

La preuve est vérifiée dans ses moindres détails, jusqu'à ce que l'ordinateur l'accepte.

Nous utilisons des assistants de preuves (*formal proof checkers*), c'est-à-dire des programmes qui ne font que **vérifier** une preuve (et parfois générer une preuve triviale).

En conséquence, le vérificateur est un programme très court (critère de de Bruijn : la correction d'un système entier ne dépend que de la correction d'un très **petit** « **noyau** »).

L'assistant de preuves Coq (<http://coq.inria.fr>)

- basé sur l'isomorphisme de Curry-Howard (équivalence entre preuve et λ -terme)
⇒ **garantie théorique**
- **peu d'automatisations**
- des **bibliothèques** fournies, notamment sur \mathbb{Z} et \mathbb{R}
- **nombres flottants** grâce à L. Théry, M. Daumas et L. Rideau
- Coq vérifie **mécaniquement** chaque étape de chaque preuve.
- Le but est d'appliquer successivement des **tactiques** (application de théorème, réécriture, calcul. . .) pour modifier ou résoudre un but.
- La preuve est faite en partant de la conclusion.

1 Outils

- Les nombres flottants
- Le coq : les preuves formelles
- Annotations flottantes

2 Récurrence linéaire d'ordre 2

3 La corde : l'équation des ondes

4 Maîtrise des erreurs d'arrondi

5 Erreur de méthode

Méthode

On **annote** le code C/Java.

Méthode

On **annote le code C/Java**.

On ajoute en commentaires des pré et post-conditions aux fonctions (et les variables modifiées).

Méthode

On **annote le code C/Java**.

On ajoute en commentaires des pré et post-conditions aux fonctions (et les variables modifiées).

On donne les variants et invariants de boucle.

On ajoute des assertions.

...

Méthode

On **annote le code C/Java**.

On ajoute en commentaires des pré et post-conditions aux fonctions (et les variables modifiées).

On donne les variants et invariants de boucle.

On ajoute des assertions.

...

L'outil génère des obligations de preuve correspondant aux annotations de l'utilisateur.

Il ne reste qu'à prouver ces obligations.

La plate-forme Why

Java

C

La plate-forme Why

Java



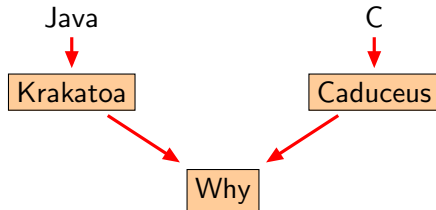
Krakatoa

C

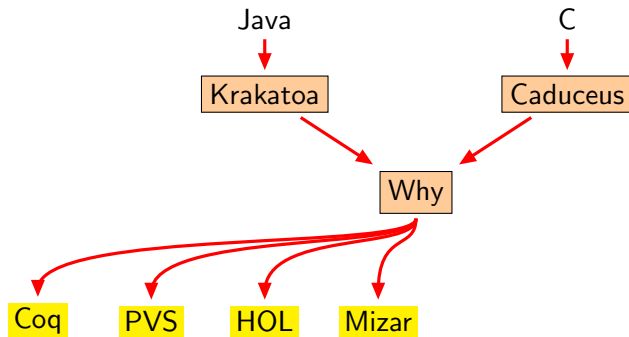


Caduceus

La plate-forme Why

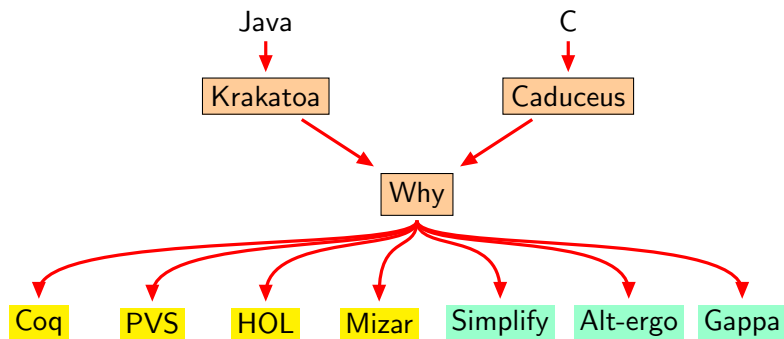


La plate-forme Why



Obligations de preuve

La plate-forme Why



Obligations de preuve

Le modèle Why des flottants

Un flottant dans un programme est un triplet :

- le **nombre flottant**, calculé effectivement par le programme,
 $x \rightarrow x_f$ partie flottante

Le modèle Why des flottants

Un flottant dans un programme est un triplet :

- le **nombre flottant**, calculé effectivement par le programme,
 $x \rightarrow x_f$ partie flottante
- la **valeur obtenue avec des calculs exacts**,
 $x \rightarrow x_e$ partie exacte

Le modèle Why des flottants

Un flottant dans un programme est un triplet :

- le **nombre flottant**, calculé effectivement par le programme,
 $x \rightarrow x_f$ partie flottante
- la **valeur obtenue avec des calculs exacts**,
 $x \rightarrow x_e$ partie exacte
- la **valeur idéalement calculée**
 $x \rightarrow x_m$ partie modèle

Le modèle Why des flottants

Un flottant dans un programme est un triplet :

- le **nombre flottant**, calculé effectivement par le programme,
 $x \rightarrow x_f$ partie flottante $1+x+x*x/2$
- la **valeur obtenue avec des calculs exacts**,
 $x \rightarrow x_e$ partie exacte $1 + x + \frac{x^2}{2}$
- la **valeur idéalement calculée**
 $x \rightarrow x_m$ partie modèle $\exp(x)$

Le modèle Why des flottants

Un flottant dans un programme est un triplet :

- le **nombre flottant**, calculé effectivement par le programme,
 $x \rightarrow x_f$ partie flottante $1+x+x*x/2$
- la **valeur obtenue avec des calculs exacts**,
 $x \rightarrow x_e$ partie exacte $1 + x + \frac{x^2}{2}$
- la **valeur idéalement calculée**
 $x \rightarrow x_m$ partie modèle $\exp(x)$

⇒ séparer facilement erreur de méthode et erreur d'arrondi

Exemple 1 : Sterbenz

```
float Sterbenz(float x, float y){  
  return x-y;  
}
```

Exemple 1 : Sterbenz

```
/*@ requires y/2 <= x <= 2*y  
   @ ensures \result == x-y  
   @*/
```

```
float Sterbenz(float x, float y){  
    return x-y;  
}
```

Exemple 2 : Malcolm

```
double malcolm() {  
  double A, B;  
  A=2;  
  while (A != (A+1))  
    A*=2;  
  
  B=1;  
  while ((A+B)-A != B)  
    B++;  
  return B; }
```

Exemple 2 : Malcolm

```
/*@ ensures \result == 2 */
double malcolm() {
    double A, B;
    A=2;  /*@ assert A==2 */
    /*@ invariant A == 2 ^^ my_log(A)
       @    && 1 <= my_log(A) <= 53
       @ variant (53-my_log(A)) */
    while (A != (A+1))
        A*=2;
    /*@ assert A == 2 ^^ (53) */

    B=1;  /*@ assert B==1 */
    /*@ invariant B == IRNDD(B) && 1 <= B <= 2
       @ variant (2-IRNDD(B)) */
    while ((A+B)-A != B)
        B++;
    return B; }

```

Exemple 3 : exponentielle naïve en simple précision

```
/*@ requires |x| <= 1./32
   @ ensures |\result - cos(x)| <= 2-23
   @ */
```

```
float moncos(float x) {
    return 1.f-x*x*.5f;
}
```

Exemple 3 : exponentielle naïve en simple précision

```
/*@ requires |x| <= 1./32
   @ ensures |\result - cos(x)| <= 2-23
   @ */
```

```
float moncos(float x) {
  return 1.f-x*x*.5f;
}
```

4 lignes de Coq,
par les tactiques `interval` et `gappa` (G. Melquiond, S. Boldo, J.-C. Filliâtre).

Plan

- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Récurrance linéaire d'ordre 2

On veut calculer

$$u_{n+1} = 2 \times u_n - u_{n-1}$$

Récurrance linéaire d'ordre 2

On veut calculer

$$u_{n+1} = 2 \times u_n - u_{n-1}$$

Une majoration naïve donne une borne exponentielle.

Récurrance linéaire d'ordre 2

On veut calculer

$$u_{n+1} = 2 \times u_n - u_{n-1}$$

Une majoration naïve donne une borne exponentielle.

⇒ l'erreur peut être de la forme $\sum_0^n \varepsilon_i$ avec $|\varepsilon_i|$ raisonnable

⇒ erreur en n^2

Récurrance linéaire d'ordre 2

On veut calculer

$$u_{n+1} = 2 \times u_n - u_{n-1}$$

Une majoration naïve donne une borne exponentielle.

⇒ l'erreur peut être de la forme $\sum_0^n \varepsilon_i$ avec $|\varepsilon_i|$ raisonnable

⇒ erreur en n^2 (probablement n en fait)

Récurrance linéaire d'ordre 2 : code

```
double comput_seq(double u0, double u1, int N) {  
    int i;  
    double uprev, ucur, tmp;  
    uprev=u0;  
    ucur=u1;  
  
    for (i=2; i<=N; i++) {  
        tmp=2*ucur;  
        tmp-=uprev;  
        uprev=ucur;  
        ucur=tmp;  
    }  
    return ucur;  
}
```

Récurrance linéaire d'ordre 2 : code annoté I

```
/*@ requires 2 <= N <= 2^^26 &&  
  @   \exact(u0)==u0 && \exact(u1)==u1 &&  
  @   \forall int k; 0<=k<=N => |u0+k*(u1-u0)| <=1  
  @ ensures \exact(\result)==u0+N*(u1-u0) &&  
  @   \round_error(\result) <= N*(N+1)/2.*2^^(-53)  
  @*/  
  
/*@ predicate mkp(double uc, double up, int n) */
```

Récurrance linéaire d'ordre 2 : code annoté

```
double comput_seq(double u0, double u1, int N) {  
    int i;  
    double uprev, ucur, tmp;  
    uprev=u0; ucur=u1;  
  
    /*@ invariant 2 <= i && i <= N+1 &&  
       @   \exact(ucur) ==u0+(i-1)*(u1-u0) &&  
       @   \exact(uprev)==u0+(i-2)*(u1-u0) &&  
       @   mkp(ucur,uprev,i-2)  
       @ variant N-i*/  
    for (i=2; i<=N; i++) {  
        tmp=2*ucur;  
        /*@ assert tmp==2*ucur */  
        tmp--uprev;  
        uprev=ucur; ucur=tmp;  
    }  
    return ucur; }
```

Récurrance linéaire d'ordre 2 : code annoté

où $\text{mkp}(uc, up, n) =$

$$\begin{aligned} \exists \varepsilon : \mathbb{N} \rightarrow \mathbb{R}, \quad \forall i \leq n, |\varepsilon(i)| \leq 2^{-53} & \quad \wedge \\ up - \text{exact}(up) = \sum_{i=0}^{n-1} (n - i) \times \varepsilon(i) & \quad \wedge \\ uc - \text{exact}(uc) = \sum_{i=0}^n (n + 1 - i) \times \varepsilon(i) & \end{aligned}$$

- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

La corde mathématique

Je cherche u de \mathbb{R}^2 dans \mathbb{R} solution de l'équation différentielle suivante, connaissant la valeur de u et de sa dérivée pour $t = 0$:

$$\frac{\partial^2 u(x, t)}{\partial t^2} - c^2 \frac{\partial^2 u(x, t)}{\partial x^2} = 0.$$

La corde discrétisée

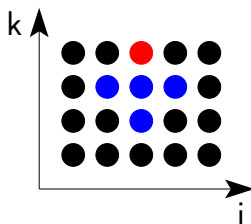
```
[...] // initialisations de p[i][0] et p[i][1]

for (k=1; k<nk; k++) {
  p[0][k+1] = 0.;
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
  }
  p[ni][k+1] = 0.;
}
```

La corde discrétisée

```
[...] // initialisations de p[i][0] et p[i][1]

for (k=1; k<nk; k++) {
  p[0][k+1] = 0.;
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
  }
  p[ni][k+1] = 0.;
}
```



- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Erreur d'arrondi

Si on utilise la méthode naïve pour borner les erreurs d'arrondis, on obtient

Erreur d'arrondi

Si on utilise la méthode naïve pour borner les erreurs d'arrondis, on obtient

$$|p_i^k - \text{exact}(p_i^k)| \leq O\left(2^k 2^{-53}\right)$$

Erreur d'arrondi

Si on utilise la méthode naïve pour borner les erreurs d'arrondis, on obtient

$$|p_i^k - \text{exact}(p_i^k)| \leq O\left(2^k 2^{-53}\right)$$

C'est beaucoup (trop) car **les erreurs se compensent**.

Définition de ε_i^k

Rappel :

$$\begin{aligned} dp &= p[i+1][k] - 2.*p[i][k] + p[i-1][k]; \\ p[i][k+1] &= 2.*p[i][k] - p[i][k-1] + a*dp; \end{aligned}$$

Soit ε_i^{k+1} l'erreur commise lors de ces deux lignes de calculs.

On considère a , p_{i-1}^k , p_i^k , p_{i+1}^k et p_i^{k-1} exacts et on regarde l'erreur flottante finale de ces 2 lignes. C'est ε_i^{k+1} .

Définition de ε_i^k

Rappel :

$$\begin{aligned} dp &= p[i+1][k] - 2.*p[i][k] + p[i-1][k]; \\ p[i][k+1] &= 2.*p[i][k] - p[i][k-1] + a*dp; \end{aligned}$$

Soit ε_i^{k+1} l'erreur commise lors de ces deux lignes de calculs.

On considère a , p_{i-1}^k , p_i^k , p_{i+1}^k et p_i^{k-1} exacts et on regarde l'erreur flottante finale de ces 2 lignes. C'est ε_i^{k+1} .

On sait que les valeurs modèles de $|p_n^m|$ sont majorées par 1. On suppose que les valeurs flottantes des $|p_n^m|$ sont majorées par 2.

Définition de ε_i^k

Rappel :

$$\begin{aligned} dp &= p[i+1][k] - 2.*p[i][k] + p[i-1][k]; \\ p[i][k+1] &= 2.*p[i][k] - p[i][k-1] + a*dp; \end{aligned}$$

Soit ε_i^{k+1} l'erreur commise lors de ces deux lignes de calculs.

On considère a , p_{i-1}^k , p_i^k , p_{i+1}^k et p_i^{k-1} exacts et on regarde l'erreur flottante finale de ces 2 lignes. C'est ε_i^{k+1} .

On sait que les valeurs modèles de $|p_n^m|$ sont majorées par 1. On suppose que les valeurs flottantes des $|p_n^m|$ sont majorées par 2.

$$|\varepsilon_n^m| \leq 85 \times 2^{-52}$$

Pyramide

L'expression analytique de l'erreur de p_i^k va donc dépendre (et en fait dépendre **uniquement**) des valeurs de :

$$\begin{array}{ccccccc}
 & & & & \varepsilon_i^k & & \\
 & & & & \varepsilon_i^{k-1} & & \varepsilon_{i+1}^{k-1} \\
 & & & \varepsilon_{i-1}^{k-1} & \varepsilon_i^{k-1} & \varepsilon_{i+1}^{k-1} & \\
 & & \varepsilon_{i-2}^{k-2} & \varepsilon_{i-1}^{k-2} & \varepsilon_i^{k-2} & \varepsilon_{i+1}^{k-2} & \varepsilon_{i+2}^{k-2} \\
 & \dots & & & \vdots & & \dots \\
 \varepsilon_{i-k}^0 & & \dots & & \varepsilon_i^0 & & \dots & \varepsilon_{i+k}^0
 \end{array}$$

Définition de α_i^k

Étant donné $a \in \mathbb{R}$, je définis $\alpha : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{R}$ telle que

$$\alpha_0^0 = 1 \quad \forall i \neq 0, \alpha_i^0 = 0$$

$$\alpha_{-1}^1 = \alpha_1^1 = (1 - a) \quad \alpha_0^1 = 2a \quad \forall i \notin \{-1, 0, 1\}, \alpha_i^1 = 0$$

$$\alpha_i^k = a \times (\alpha_{i-1}^{k-1} + \alpha_{i+1}^{k-1}) + 2(1 - a) \times \alpha_i^{k-1} - \alpha_i^{k-2}$$

Valeurs des α_i^k

Pour $a = 0.9$, on a :

				1			
			0.9	0.2	0.9		
	0.81	0.36	0.66	0.36	0.81		
0.729	0.486	0.495	0.58	0.495	0.486	0.729	
...			
0.9^k							0.9^k

Valeurs des α_i^k

Pour $a = 0.9$, on a :

									$\Sigma =$
				1					1
			0.9	0.2	0.9				2
		0.81	0.36	0.66	0.36	0.81			3
	0.729	0.486	0.495	0.58	0.495	0.486	0.729		4
		
0.9^k				...				0.9^k	$k + 1$

Valeurs des α_i^k

Pour $a = 0.9$, on a :

			1				
		0.9	0.2	0.9			
	0.81	0.36	0.66	0.36	0.81		
0.729	0.486	0.495	0.58	0.495	0.486	0.729	
...			
0.9^k							0.9^k

Pour des raisons techniques, j'ai besoin de $\alpha_i^k \geq 0$.

Valeurs des α_i^k

Pour $a = 0.9$, on a :

			1				
			0.9	0.2	0.9		
		0.81	0.36	0.66	0.36	0.81	
	0.729	0.486	0.495	0.58	0.495	0.486	0.729
0.9^k
							0.9^k

Pour des raisons techniques, j'ai besoin de $\alpha_i^k \geq 0$.

François Clément \leftrightarrow Bruno Salvy \leftrightarrow Manuel Kauers \leftrightarrow Veronika Pillwein

Valeurs des α_j^k

Pour $a = 0.9$, on a :

			1				
			0.9	0.2	0.9		
		0.81	0.36	0.66	0.36	0.81	
	0.729	0.486	0.495	0.58	0.495	0.486	0.729
0.9 ^k	⋮		⋮	⋮		⋮	0.9 ^k
			⋮				

Pour des raisons techniques, j'ai besoin de $\alpha_j^k \geq 0$.

François Clément \leftrightarrow Bruno Salvy \leftrightarrow Manuel Kauers \leftrightarrow Veronika Pillwein

$$\alpha_n^j = \sum_{k=j}^n \binom{2k}{j+k} \binom{n+k+1}{2k+1} (-1)^{j+k} a^k = a^j \sum_{k=0}^{n-j} P_k^{(2j,0)}(1-2a)$$

Le résultat suit grâce au théorème de Féjer, étendu par Askey et Gaspe.

Expression analytique

En fait, l'erreur de p_i^k est la somme de tous ces gens :

$$\begin{array}{ccccccc} & & & \varepsilon_i^k & & & \\ & & & 0.2\varepsilon_i^{k-1} & & & \\ & & 0.9\varepsilon_{i-1}^{k-1} & & 0.9\varepsilon_{i+1}^{k-1} & & \\ 0.81\varepsilon_{i-2}^{k-2} & & 0.36\varepsilon_{i-1}^{k-2} & & 0.36\varepsilon_{i+1}^{k-2} & & 0.81\varepsilon_{i+2}^{k-2} \\ & \dots & & \vdots & & & \dots \\ 0.9^k \varepsilon_{i-k}^0 & & & \dots & & & 0.9^k \varepsilon_{i+k}^0 \end{array}$$

$$p_i^k - \text{exact}(p_i^k) = \sum_{l=0}^k \sum_{j=-l}^l \alpha_j^l \varepsilon_{i+j}^{k-l}$$

Expression analytique : conséquences

- 1 On a une **expression analytique** de l'erreur d'arrondi.

Expression analytique : conséquences

- 1 On a une **expression analytique** de l'erreur d'arrondi.
- 2 C'est pas si compliqué!
(on pouvait pas éviter la double sommation pyramidale)

Expression analytique : conséquences

- 1 On a une **expression analytique** de l'erreur d'arrondi.
- 2 C'est pas si compliqué!
(on pouvait pas éviter la double sommation pyramidale)
- 3 Ça fait une borne d'erreur en $\mathcal{O}(k^2 2^{-53})$:

$$\left| p_i^k - \text{exact} \left(p_i^k \right) \right| \leq 85 \times 2^{-53} \times (k + 1) \times (k + 2)$$

Expression analytique : conséquences

- 1 On a une **expression analytique** de l'erreur d'arrondi.
- 2 C'est pas si compliqué!
(on pouvait pas éviter la double sommation pyramidale)
- 3 Ça fait une borne d'erreur en $\mathcal{O}(k^2 2^{-53})$:

$$\left| p_i^k - \text{exact} \left(p_i^k \right) \right| \leq 85 \times 2^{-53} \times (k + 1) \times (k + 2)$$

- 4 C'est pas drôle à prouver formellement.
Par exemple, $\sum_{l=1}^k \sum_{j=-l+1}^{l+1} \alpha_{j-1}^l \varepsilon_{i+j}^{k-l}$ devient

```
(sum_f_z (fun l : Z => sum_f_z (fun j : Z => alpha a (j
- 1) (Zabs_nat l) * eps (i + j) (k - l)) (- 1 + 1) (1 +
1)) 1 k)%R.
```

ε_i^k : les bords

On a la propriété de **récurrence** :

Si l'erreur est de la forme $\sum \sum \dots$ aux étapes $(i-1, k-1)$, $(i, k-1)$, $(i+1, k-1)$ et $(i, k-2)$, alors l'erreur est de la forme $\sum \sum \dots$ à l'étape (i, k) .

ε_i^k : les bords

On a la propriété de **réurrence** :

Si l'erreur est de la forme $\sum \sum \dots$ aux étapes $(i-1, k-1)$, $(i, k-1)$, $(i+1, k-1)$ et $(i, k-2)$, alors l'erreur est de la forme $\sum \sum \dots$ à l'étape (i, k) .

Problème : **les bords!** : $i = 0$ et $i = n_i$.

Là, l'erreur vaut 0, donc n'est pas la somme compliquée qu'on souhaite...

ε_i^k : les bords

On a la propriété de **réurrence** :

Si l'erreur est de la forme $\sum \sum \dots$ aux étapes $(i-1, k-1)$, $(i, k-1)$, $(i+1, k-1)$ et $(i, k-2)$, alors l'erreur est de la forme $\sum \sum \dots$ à l'étape (i, k) .

Problème : **les bords!** : $i = 0$ et $i = n_i$.

Là, l'erreur vaut 0, donc n'est pas la somme compliquée qu'on souhaite. . .

sauf si. . .

ε_i^k : les bords

On a la propriété de **réurrence** :

Si l'erreur est de la forme $\sum \sum \dots$ aux étapes $(i-1, k-1)$, $(i, k-1)$, $(i+1, k-1)$ et $(i, k-2)$, alors l'erreur est de la forme $\sum \sum \dots$ à l'étape (i, k) .

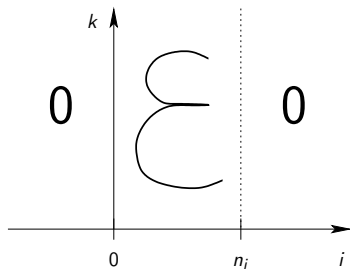
Problème : **les bords!** : $i = 0$ et $i = n_i$.

Là, l'erreur vaut 0, donc n'est pas la somme compliquée qu'on souhaite. . .

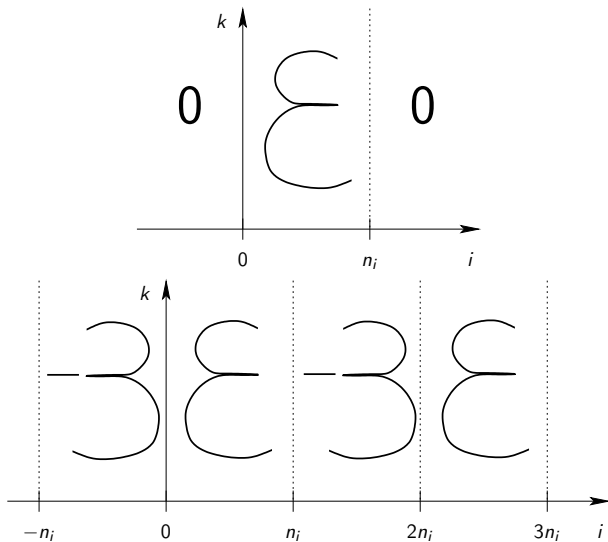
sauf si. . .

On étend artificiellement les ε_i^k pour $i < 0$ et $i > n_i$ avec les bonnes valeurs (négatives inversées).

Je prends le ε



Je prends le ε et je le retourne



- 1 Outils
 - Les nombres flottants
 - Le coq : les preuves formelles
 - Annotations flottantes
- 2 Récurrence linéaire d'ordre 2
- 3 La corde : l'équation des ondes
- 4 Maîtrise des erreurs d'arrondi
- 5 Erreur de méthode

Erreur de méthode (travail en cours !)

Erreur de méthode (travail en cours !)

On va prouver d'une part la **consistance** (les $|p_i^k|$ sont bornés) :

Erreur de méthode (travail en cours !)

On va prouver d'une part la **consistance** (les $|p_i^k|$ sont bornés) :

⇒ on doit redéfinir le produit scalaire, la norme
... et les dizaines de lemmes les concernant !

Erreur de méthode (travail en cours !)

On va prouver d'une part la **consistance** (les $|p_i^k|$ sont bornés) :

⇒ on doit redéfinir le produit scalaire, la norme
... et les dizaines de lemmes les concernant !

⇒ on devrait utiliser $\langle u, v \rangle_{L_h^2} = \sum_{-\infty}^{+\infty} u_j v_j$.
On se ramène à $\langle u, v \rangle_n = \sum_0^n u_j v_j$.

Erreur de méthode (travail en cours !)

On va prouver d'autre part la **convergence**
(les $|p_i^k|$ tendent vers la solution exacte).

Erreur de méthode (travail en cours !)

On va prouver d'autre part la **convergence**
(les $|p_i^k|$ tendent vers la solution exacte).

⇒ on a besoin de définitions mathématiques “propres” ($f = O(g)$, développements limités, $O(dx^2 + dt^2)$...)

Erreur de méthode (travail en cours !)

On va prouver d'autre part la **convergence**
(les $|p_i^k|$ tendent vers la solution exacte).

⇒ on a besoin de définitions mathématiques “propres” ($f = O(g)$, développements limités, $O(dx^2 + dt^2)$...)

Attention aux échanges implicites entre les quantificateurs existentiels et universels :

$$\forall x, \exists C, P(x, C) \neq \exists C, \forall x, P(x, C)$$

Erreur de méthode (travail en cours !)

On va prouver d'autre part la **convergence**
(les $|p_i^k|$ tendent vers la solution exacte).

⇒ on a besoin de définitions mathématiques “propres” ($f = O(g)$, développements limités, $O(dx^2 + dt^2)$...)

Attention aux échanges implicites entre les quantificateurs existentiels et universels :

$$\forall x, \exists C, P(x, C) \neq \exists C, \forall x, P(x, C)$$

⇒ il semble que la fonction solution doivent être C^n avec ses dérivées n -ièmes bornées pour n “assez grand” .
⇒ développement limité uniforme ?

Erreur de méthode (travail en cours !)

On va prouver d'autre part la **convergence**
(les $|p_i^k|$ tendent vers la solution exacte).

⇒ on a besoin de définitions mathématiques “propres” ($f = O(g)$, développements limités, $O(dx^2 + dt^2)$...)

Attention aux échanges implicites entre les quantificateurs existentiels et universels :

$$\forall x, \exists C, P(x, C) \neq \exists C, \forall x, P(x, C)$$

⇒ il semble que la fonction solution doivent être C^n avec ses dérivées n -ièmes bornées pour n “assez grand” .
⇒ développement limité uniforme ?

Bref, c'est plus compliqué et plus long que prévu !

Conclusion (après 2 500 lignes de Coq pour 6 lignes de C)

Il reste à

- faire une preuve **irréprochable** de l'erreur de méthode,
- la prouver formellement.

Conclusion (après 2 500 lignes de Coq pour 6 lignes de C)

Il reste à

- faire une preuve **irréprochable** de l'erreur de méthode,
- la prouver formellement.

Ça nous a appris que

- Les preuves formelles, c'est bien, car ça augmente la confiance et ça trouve les erreurs ou les flous dans les démonstrations.
- Entre les preuves des numériciens et les preuves formelles, il y a un gouffre **très** profond.