From mathematics to programs: a verification journey

Sylvie Boldo

Inria

March 15th, 2016
Disclaimers

**Disclaimer 1**: this is joint work with

- François Clément,
- Jean-Christophe Filliâtre,
- Micaela Mayero,
- Guillaume Melquiond,
- Pierre Weis.
Disclaimer 1: this is joint work with

- François Clément,
- Jean-Christophe Filliâtre,
- Micaela Mayero,
- Guillaume Melquiond,
- Pierre Weis.

Disclaimer 2:
this is a computer science talk (with real pieces of mathematics inside).
Outline

1. Introduction

2. Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3. 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4. Conclusion
Mathematics

$\mathbb{R}, \int, \frac{\partial^2 u}{\partial t^2}$

theorems
Introduction

Mathematics

\[ \mathbb{R}, \int, \frac{\partial^2 u}{\partial t^2} \]
theorems

Applied Mathematics

numerical scheme, convergence algorithms + theorems
**Mathematics**

- $\mathbb{R}$, $\int$, $\frac{\partial^2 u}{\partial t^2}$
- theorems

**Applied Mathematics**

- numerical scheme, convergence algorithms + theorems

**Computer**

- floating-point numbers, implementation programs + ?
Motivations

PDE (Partial Differential Equation) ⇒ weather forecast
⇒ nuclear simulation
⇒ optimal control
⇒ ...
Motivations

PDE (Partial Differential Equation) ⇒ weather forecast
⇒ nuclear simulation
⇒ optimal control
⇒ ...

Usually too complex to solve by an exact mathematical formula
⇒ approximated by numerical scheme over discrete grids

⇒ mathematical proofs of the convergence of the numerical scheme
(we compute something close to the PDE solution if the grids size decreases)
Motivations

PDE (Partial Differential Equation) $\Rightarrow$ weather forecast
$\Rightarrow$ nuclear simulation
$\Rightarrow$ optimal control
$\Rightarrow$ ...

Usually too complex to solve by an exact mathematical formula
$\Rightarrow$ approximated by numerical scheme over discrete grids

$\Rightarrow$ mathematical proofs of the convergence of the numerical scheme
(we compute something close to the PDE solution if the grids size decreases)

$\Rightarrow$ C program implementing the scheme
Motivations

- **PDE** (Partial Differential Equation) ⇒ weather forecast
  ⇒ nuclear simulation
  ⇒ optimal control
  ⇒ ...

Usually too complex to solve by an exact mathematical formula
⇒ approximated by **numerical scheme over discrete grids**

⇒ mathematical proofs of the convergence of the numerical scheme
(we compute something close to the PDE solution if the grids size decreases)

⇒ **C** program implementing the scheme

**Let us machine-check this kind of programs!**
Outline

1. Introduction

2. Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3. 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4. Conclusion
Babylonian clay tablet (1800–1600 BC)
Babylonian clay tablet (1800–1600 BC)

= (1, 24, 51, 10)

= 1 + \frac{24}{60} + \frac{51}{60^2} + \frac{10}{60^3} = \frac{30547}{21600}

≈ 1,41421296 ≈ \sqrt{2}
Babylonian clay tablet (1800–1600 BC)

\[
\begin{align*}
\text{\(1\)} & \text{\(24\)} \text{\(51\)} \text{\(10\)} \\
= & \left(1, \frac{24}{60}, \frac{51}{60^2}, \frac{10}{60^3}\right) = \frac{30547}{21600} \\
\approx & \text{1,41421296} \approx \sqrt{2}
\end{align*}
\]

⇒ representation of a real number with a finite precision
Floating-Point Number

Using a finite number of bits (the precision $p$) and based on scientific notation, computers use floating-point (FP) numbers.

A FP number is only a string of bits.

11100011010010011110000111000000
Floating-Point Number

Using a finite number of bits (the precision $p$) and based on scientific notation, computers use floating-point (FP) numbers.

A FP number is only a string of bits.

$$11100011010010011110000111000000$$

We interpret it depending on the respective values of $s$ (sign), $e$ (exponent) and $f$ (fraction).

$$\begin{array}{c|c|c}
1 & 11000110 & 1001001111100001110000000 \\
1 & 11000110 & 1001001111100001110000000 \\
\hline
s & e & f
\end{array}$$
Floating-Point Number

We associate a real value:

\[
\begin{array}{cccc}
1 & 11000110 & 10010011110000111000000 \\
\downarrow & \downarrow & \downarrow \\
(-1)^s \times 2^{e-B} & \times & 1 \cdot f
\end{array}
\]

\[
(-1)^1 \times 2^{198-127} \times 1.100100111100001110000002 \\
-2^{54} \times 206727 \approx -3.724 \times 10^{21}
\]
Floating-Point Number

We associate a real value:

$$\begin{align*}
(−1)^s \times 2^{e−B} \times 1 \cdot f \\
\frac{1}{11000110} 10010011110000111000000
\end{align*}$$

$$(-1)^1 \times 2^{198−127} \times 1.1001001111000011100000002$$

$$-2^{54} \times 206727 \approx -3.724 \times 10^{21}$$

except for the special values of e: ±0, ±∞, NaN, subnormals.
Floating-Point Number Repartition
Floating-Point Number Repartition

0

\text{subnormals}

\mathbb{R}
Floating-Point Number Repartition

0

subnormals

binade (common exponent)
Floating-Point Number Repartition

subnormals

ulp(f)

binade (common exponent)
Floating-Point Computation

For the $+,-,\times,\div,\sqrt{}$, the result is the same as if the infinitely precise mathematical result was computed and then rounded to the nearest floating-point number.

$\Rightarrow$ guaranteed by the IEEE-754 standard (1985 & 2008).
Floating-Point Computation

For the $+, -, \times, \div, \sqrt{}$, the result is the same as if the infinitely precise mathematical result was computed and then rounded to the nearest floating-point number.

$\Rightarrow$ guaranteed by the IEEE-754 standard (1985 & 2008).

$\Rightarrow$ portability & accuracy
For the $\mathbf{+}$, $\mathbf{-}$, $\times$, $\div$, $\sqrt{}$, the result is the same as if the infinitely precise mathematical result was computed and then rounded to the nearest floating-point number.

$\Rightarrow$ guaranteed by the IEEE-754 standard (1985 & 2008).

$\Rightarrow$ portability & accuracy

$\Rightarrow$ if $x \in \mathbb{R}$ is not too small, $|x - \circ_{\text{double}}(x)| \leq 2^{-53}|x|$
Floating-Point Computations

More than one FP operation may lead to incorrect results.
Floating-Point Computations

More than one FP operation may lead to incorrect results.

Floating-point evaluations of \((x - 4)^4\) around 4.
Outline

1. Introduction

2. Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3. 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4. Conclusion
Formal proof

The proof is checked in its deep details until the computer agrees with it.

We often use formal proof checkers, meaning programs that only check a proof (they may also generate easy demonstrations).

Therefore the checker is a very short program (de Bruijn criteria: the correctness of the system as a whole depends on the correctness of a very small "kernel").
The Coq proof assistant (http://coq.inria.fr)

- Based on the Curry-Howard isomorphism. (equivalence between proofs and $\lambda$-terms)
- Few automations.
- Comprehensive libraries, including on $\mathbb{Z}$ and $\mathbb{R}$.
- **Coq kernel mechanically checks** each step of each proof.
- The method is to apply successively tactics (theorem application, rewriting, simplifications...) to transform or reduce the goal down to the hypotheses.
- The proof is handled starting from the conclusion.
Flocq: 16,000 lines of Coq, 700 theorems,

- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.
A Coq formalization of FP arithmetic: Flocq

Flocq: 16 000 lines of Coq, 700 theorems,
- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.

Applications:
- Frama-C/Jessie C code certifier
- CompCert certified C compiler

http://flocq.gforge.inria.fr/
A Coq formalization of FP arithmetic: Flocq

**Flocq**: 16 000 lines of Coq, 700 theorems,
- any radix, any format,
- both axiomatic and computable definitions of rounding,
- effective arithmetic operators,
- numerous theorems.

**Applications:**
- **Frama-C/Jessie**  \hspace{2cm} C code certifier
- **CompCert** \hspace{2cm} certified C compiler

http://flocq.gforge.inria.fr/
Example of Coq theorem

**Theorem (round\_NE\_abs)**

Let \( \varphi \) be a format, such that the rounding to nearest, ties to even (\( \circ \)) can be defined. For all \( x \in \mathbb{R} \), \( \circ(|x|) = |\circ(x)| \).
Example of Coq theorem

**Theorem (round\_NE\_abs)**

Let \( \varphi \) be a format, such that the rounding to nearest, ties to even (\( \circ \)) can be defined. For all \( x \in \mathbb{R} \), \( \circ(|x|) = |\circ(x)| \).

Lemma round\_NE\_abs: \( \forall x : \mathbb{R}, \)  
\[ \text{round beta fexp ZnearestE (Rabs x)} = \text{Rabs (round beta fexp ZnearestE x)}. \]
Example of Coq theorem

**Theorem (round_NE_abs)**

Let $\varphi$ be a format, such that the rounding to nearest, ties to even ($\circ$) can be defined. For all $x \in \mathbb{R}$, $\circ(|x|) = |\circ(x)|$.

Lemma round_NE_abs: forall x : R,
round beta fexp ZnearestE (Rabs x) = Rabs (round beta fexp ZnearestE x).
Proof with auto with typeclass_instances.
intros x; apply sym_eq.
unfold Rabs at 2.
destruct (Rcase_abs x) as [Hx|Hx].
rewrite round_NE_opp.
apply Rabs_left1.
rewrite <- (round_0 beta fexp ZnearestE).
apply round_le...
now apply Rlt_le.
apply Rabs_pos_eq.
rewrite <- (round_0 beta fexp ZnearestE).
apply round_le...
now apply Rge_le.
Qed.

With the stating of the theorem, the tactics, and the name of theorems.
Outline

1. Introduction

2. Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3. 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4. Conclusion
Annotation language: ACSL

- ANSI/ISO C Specification Language

⇒ For the programmer, the specification is easy to understand.
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions (and which variables are modified).
Annotation language: ACSL

- ANSI/ISO C Specification Language
- Behavioral specification language for C programs
- Pre-conditions and post-conditions to functions (and which variables are modified).
- Variants and invariants of the loops.
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions (and which variables are modified).
- variants and invariants of the loops.
- assertions
Annotation language: ACSL

- ANSI/ISO C Specification Language
- behavioral specification language for C programs
- pre-conditions and post-conditions to functions (and which variables are modified).
- variants and invariants of the loops.
- assertions
- In annotations, all computations are exact.
Annotation language: ACSL

- ANSI/ISO C Specification Language
- Behavioral specification language for C programs
- Pre-conditions and post-conditions to functions (and which variables are modified).
- Variants and invariants of the loops.
- Assertions
- In annotations, all computations are exact.

⇒ For the programmer, the specification is easy to understand.
A floating-point number is a triple:

- the floating-point number, really computed by the program, $x \rightarrow x_f$ floating-point part
A floating-point number is a triple:

- the **floating-point number**, really computed by the program,
  \[ x \rightarrow x_f \] floating-point part

- the **value that would have been obtained with exact computations**, 
  \[ x \rightarrow x_e \] exact part
A floating-point number is a triple:

- the \textbf{floating-point number}, really computed by the program, \( x \rightarrow x_f \) floating-point part
- the \textbf{value that would have been obtained with exact computations}, \( x \rightarrow x_e \) exact part
- the \textbf{value that we ideally wanted to compute}, \( x \rightarrow x_m \) model part
A floating-point number is a triple:

- the floating-point number, really computed by the program, 
  \[ x \rightarrow x_f \text{ floating-point part} \]
  \[ 1 + x + \frac{x^2}{2} \]

- the value that would have been obtained with exact computations, 
  \[ x \rightarrow x_e \text{ exact part} \]
  \[ 1 + x + \frac{x^2}{2} \]

- the value that we ideally wanted to compute 
  \[ x \rightarrow x_m \text{ model part} \]
  \[ \exp(x) \]
A floating-point number is a triple:

- the floating-point number, really computed by the program, 
  \[ x \rightarrow x_f \text{ floating-point part} \]
  \[ 1 + x + \frac{x^2}{2} \]

- the value that would have been obtained with exact computations, 
  \[ x \rightarrow x_e \text{ exact part} \]
  \[ 1 + x + \frac{x^2}{2} \]

- the value that we ideally wanted to compute 
  \[ x \rightarrow x_m \text{ model part} \]
  \[ \exp(x) \]

\[ \Rightarrow \text{ easy to split into method error and rounding error} \]
Methodology for the verification of C programs

C Program

The program is correct with respect to its specifications.
Methodology for the verification of C programs

Annotated C Program (specification, invariant)

Human → Theorem statements

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq → Proved Theorems

The program is correct with respect to its specifications
Methodology for the verification of C programs

Human

Annotated C Program (specification, invariant)

Frama-C

Theorem statements

Jessie

The program is correct with respect to its specifications.
Methodology for the verification of C programs

Annotated C Program (specification, invariant) → Frama-C → Theorem statements

Human → Automatic provers (Alt-Ergo, Gappa, Z3) → Coq → Human

The program is correct with respect to its specifications
Methodology for the verification of C programs

Annotated C Program (specification, invariant)

The program is correct with respect to its specifications

Sylvie Boldo (Inria)
From mathematics to programs: a verification journey
March 15th, 2016
Methodology for the verification of C programs

Human

Annotated C Program (specification, invariant)

Frama-C

Theorem statements

Jessie

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq ← Human

The program is correct with respect to its specifications
Methodology for the verification of C programs

Annotated C Program (specification, invariant) → Theorem statements

Human → Frama-C → Theorem statements

Frama-C → Jessie

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq ← Human

The program is correct with respect to its specifications
Methodology for the verification of C programs

Annotated C Program (specification, invariant) → Frama-C → Theorem statements

Human ↓ Jessie

Human ↫ Coq → Automatic provers (Alt-Ergo, Gappa, Z3)

The program is correct with respect to its specifications.

Sylvie Boldo (Inria)  From mathematics to programs: a verification March 15th, 2016 21 / 44
Methodology for the verification of C programs

Annotated C Program (specification, invariant) → Frama-C → Theorem statements

Human

Human

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq ← Human
Methodology for the verification of C programs

Human

Annotated C Program (specification, invariant)

Frama-C

Jessie

Theorem statements

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq

← Human
Methodology for the verification of C programs

Human

Annotated C Program (specification, invariant) → Frama-C

Theorem statements

Jessie

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq ← Human

The program is correct with respect to its specifications
Methodology for the verification of C programs

- Annotated C Program (specification, invariant)
- Theorem statements
- Proved Theorems

Human → Annotated C Program

Frama-C → Theorem statements

Jessie

Automatic provers (Alt-Ergo, Gappa, Z3)

Coq → Human

The program is correct with respect to its specifications.
Methodology for the verification of C programs

The program is correct with respect to its specifications

Human

Annotated C Program
(specification, invariant)

Frama-C

Theorem statements

Automatic provers
(Alt-Ergo, Gappa, Z3)

Coq

Human

Proved Theorems
Outline

1 Introduction

2 Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4 Conclusion
Looking for $u : \mathbb{R}^2 \rightarrow \mathbb{R}$ regular enough such that:

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

with given values for the initial position $u_0(x)$ and the initial velocity $u_1(x)$.

$\Rightarrow$ rope oscillation, sound, radar, oil prospection...
We want $u_j^k \approx u(j\Delta x, k\Delta t)$.

$$
\frac{u_j^k - 2u_j^{k-1} + u_j^{k-2}}{\Delta t^2} - c^2 \frac{u_{j+1}^{k-1} - 2u_j^{k-1} + u_{j-1}^{k-1}}{\Delta x^2} = s_j^{k-1}
$$

And other horrible formulas to initialize $u_j^0$ and $u_j^1$. 
We want $u_j^k \approx u(j \Delta x, k \Delta t)$.

$$
\frac{u_j^k - 2u_j^{k-1} + u_j^{k-2}}{\Delta t^2} - c^2 \frac{u_{j+1}^{k-1} - 2u_j^{k-1} + u_{j-1}^{k-1}}{\Delta x^2} = s_j^{k-1}
$$

And other horrible formulas to initialize $u_j^0$ and $u_j^1$.

Three-point scheme: $u_j^k$ depends on $u_{j-1}^{k-1}$, $u_j^{k-1}$, $u_{j+1}^{k-1}$ and $u_j^{k-2}$. 
Program

// initialization of p[i][0] and 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.;
}

Two different errors:
round-off errors
due to floating-point roundings
method errors
the scheme only approximates the exact solution
Program

// initialization of p[i][0] and 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.;
}

Two different errors:

- round-off errors
due to floating-point roundings
- method errors
  the scheme only approximates the exact solution
1 Introduction

2 Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4 Conclusion
Rounding error

Remainder:

\[ dp = p[i+1][k] - 2 \cdot p[i][k] + p[i-1][k]; \]
\[ p[i][k+1] = 2 \cdot p[i][k] - p[i][k-1] + a \cdot dp; \]

If we use a naive technique to bound the rounding errors, we get
Rounding error

Remainder:

$$dp = p[i+1][k] - 2 \cdot p[i][k] + p[i-1][k];$$
$$p[i][k+1] = 2 \cdot p[i][k] - p[i][k-1] + a \cdot dp;$$

If we use a naive technique to bound the rounding errors, we get

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

Remainder:

\[ dp = p[i+1][k] - 2 \times p[i][k] + p[i-1][k]; \]
\[ p[i][k+1] = 2 \times p[i][k] - p[i][k-1] + a \times dp; \]

If we use a naive technique to bound the rounding errors, we get

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

This is too much because the errors do compensate.
Definition of $\varepsilon_{i}^{k}$

Remainder:

$$dp = \ p[i+1][k] - 2 \times p[i][k] + p[i-1][k];$$
$$p[i][k+1] = 2 \times p[i][k] - p[i][k-1] + a \times dp;$$

Let $\varepsilon_{i}^{k+1}$ be the rounding error made during these two lines of computations.

We assume $a$, $p_{i-1}^{k}$, $p_{i}^{k}$, $p_{i+1}^{k}$ and $p_{i}^{k-1}$ are exact and we look into the rounding error of these two lines. It is called $\varepsilon_{i}^{k+1}$. 
Definition of $\varepsilon_i^k$

Remainder:

\[ dp = p[i+1][k] - 2 \cdot p[i][k] + p[i-1][k]; \]
\[ p[i][k+1] = 2 \cdot p[i][k] - p[i][k-1] + a \cdot dp; \]

Let $\varepsilon_i^{k+1}$ be the rounding error made during these two lines of computations.

We assume $a$, $p_{i-1}^k$, $p_i^k$, $p_{i+1}^k$ and $p_i^{k-1}$ are exact and we look into the rounding error of these two lines. It is called $\varepsilon_i^{k+1}$.

We know (from initializations) that the model values of the $|p_n^m|$ are bounded by 1. We assume that the floating-point values of the $|p_n^m|$ are bounded by 2.
Definition of $\varepsilon_{i}^{k}$

Remainder:

\[ dp = p[i+1][k] - 2 \times p[i][k] + p[i-1][k]; \]
\[ p[i][k+1] = 2 \times p[i][k] - p[i][k-1] + a \times dp; \]

Let $\varepsilon_{i}^{k+1}$ be the rounding error made during these two lines of computations.

We assume $a$, $p_{i-1}^{k}$, $p_{i}^{k}$, $p_{i+1}^{k}$ and $p_{i}^{k-1}$ are exact and we look into the rounding error of these two lines. It is called $\varepsilon_{i}^{k+1}$.

We know (from initializations) that the model values of the $|p_{n}^{m}|$ are bounded by 1. We assume that the floating-point values of the $|p_{n}^{m}|$ are bounded by 2.

\[ |\varepsilon_{m}^{n}| \leq 78 \times 2^{-52} \]
We have an **analytical expression** of the rounding error with known constants $\alpha_i^k$. 

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

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

1. We have an **analytical expression** of the rounding error with known constants \( \alpha_i^k \).
2. It is not that complicated!
   (we cannot get rid of the pyramidal double summation)
We have an analytical expression of the rounding error with known constants $\alpha_i^k$.

It is not that complicated!

(we cannot get rid of the pyramidal double summation)

The rounding error is bounded by $\bigO(k^2 \, 2^{-53})$:

$$\left| p_i^k - \text{exact}(p_i^k) \right| \leq 78 \times 2^{-53} \times (k + 1) \times (k + 2)$$
Outline

1 Introduction

2 Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4 Conclusion
We measure that $u$ and $u_j^k$ are close when $(\Delta x, \Delta t) \to 0$.

We define $e_j^k \overset{\text{def}}{=} \bar{u}_j^k - u_j^k$: convergence error

where $\bar{u}_j^k$ is the value of $u$ at the $(j, k)$ point of the grid.
We measure that $u$ and $u_j^k$ are close when $(\Delta x, \Delta t) \to 0$.

We define $e_j^k \overset{\text{def}}{=} \bar{u}_j^k - u_j^k$: convergence error
where $\bar{u}_j^k$ is the value of $u$ at the $(j, k)$ point of the grid.

We want to bound $\| e_h^{k\Delta t}(t) \| \Delta x$: the average of the convergence error on all points of the grid at a given time $k\Delta t(t) = \left\lfloor \frac{t}{\Delta t} \right\rfloor \Delta t$. 

\[ e_j^k \]
Method error

We measure that $u$ and $u_j^k$ are close when $(\Delta x, \Delta t) \to 0$.

We define $e_j^k \overset{\text{def}}{=} \bar{u}_j^k - u_j^k$: convergence error

where $\bar{u}_j^k$ is the value of $u$ at the $(j, k)$ point of the grid.

We want to bound $\left\| e_h^{k\Delta t(t)} \right\|_{\Delta x}$: the average of the convergence error on all points of the grid at a given time $k_{\Delta t}(t) = \left\lfloor \frac{t}{\Delta t} \right\rfloor \Delta t$.

We want to prove:

$$\left\| e_h^{k\Delta t(t)} \right\|_{\Delta x} = O_{[0,t_{\text{max}}]}(\Delta x^2 + \Delta t^2)$$
Big O = big pain

Usually, the big O uses one variable and \( f(x) = O_{\|x\| \to 0}(g(x)) \) means

\[
\exists \alpha, C > 0, \quad \forall x \in \mathbb{R}^n, \quad \|x\| \leq \alpha \Rightarrow |f(x)| \leq C \cdot |g(x)|.
\]
Usually, the big O uses one variable and $f(x) = O_{\|x\| \to 0}(g(x))$ means

$$\exists \alpha, C > 0, \quad \forall x \in \mathbb{R}^n, \quad \|x\| \leq \alpha \Rightarrow |f(x)| \leq C \cdot |g(x)|.$$  

Here 2 variables: $\Delta x$ (grid sizes, tends to 0), and $x$ (time and space). (Think about Taylor expansions)
Big O = big pain

Usually, the big O uses one variable and $f(x) = O_{\|x\| \to 0}(g(x))$ means

$$\exists \alpha, C > 0, \forall x \in \mathbb{R}^n, \|x\| \leq \alpha \Rightarrow |f(x)| \leq C \cdot |g(x)|.$$ 

Here 2 variables: $\Delta x$ (grid sizes, tends to 0), and $x$ (time and space). (Think about Taylor expansions)

$$\forall x, \exists \alpha, C > 0, \forall \Delta x \in \mathbb{R}^2, \|\Delta x\| \leq \alpha \Rightarrow |f(x, \Delta x)| \leq C \cdot |g(\Delta x)|$$

does not work.
We used a uniform big O:

$$\exists \alpha, C > 0, \quad \forall x, \Delta x, \quad \|\Delta x\| \leq \alpha \Rightarrow |f(x, \Delta x)| \leq C \cdot |g(\Delta x)|.$$ 

where variables $x$ and $\Delta x$ are restricted to subsets of $\mathbb{R}^2$. (for example such that $\Delta t > 0$) 
$\Rightarrow$ Taylor expansions
Proof idea 1/3: consistency

The truncation error is defined as how much the exact solution solves the numerical scheme:

\[ \varepsilon_j^{k-1} = \frac{\bar{u}_j^k - 2\bar{u}_j^{k-1} + \bar{u}_j^{k-2}}{\Delta t^2} - c^2 \frac{\bar{u}_{j+1}^{k-1} - 2\bar{u}_j^{k-1} + \bar{u}_{j-1}^{k-1}}{\Delta x^2} - s_j^{k-1} \]
Proof idea 1/3: consistency

The truncation error is defined as how much the exact solution solves the numerical scheme:

\[ \varepsilon^{k-1}_j = \frac{\bar{u}^k_j - 2\bar{u}^{k-1}_j + \bar{u}^{k-2}_j}{\Delta t^2} - c^2 \frac{\bar{u}^{k-1}_{j+1} - 2\bar{u}^{k-1}_j + \bar{u}^{k-1}_{j-1}}{\Delta x^2} - s^{k-1}_j \]

The consistency is the boundedness of the truncation error:

\[ \left\| \varepsilon^{k\Delta t(t)}_h \right\|_{\Delta x} = O_{[0,t_{\text{max}}]}(\Delta x^2 + \Delta t^2) \]

By Taylor series and many computations.
Proof idea 2/3: stability

We define a discrete energy by

\[
E_h(c)(u_h)^{k+\frac{1}{2}} \overset{\text{def}}{=} \frac{1}{2} \left\| \frac{u_h^{k+1} - u_h^k}{\Delta t} \right\|_{\Delta x}^2 + \frac{1}{2} \left\langle u_h^k, u_h^{k+1} \right\rangle_{A_h(c)}
\]

kinetic energy potential energy

\[
\left\langle v_h, w_h \right\rangle_{A_h(c)} \overset{\text{def}}{=} \left\langle A_h(c) v_h, w_h \right\rangle_{\Delta x} \quad \text{and} \quad (A_h(c) v_h)_j \overset{\text{def}}{=} -c^2 \frac{v_{j+1} - 2v_j + v_{j-1}}{\Delta x^2}.
\]
Proof idea 2/3: stability

We define a discrete energy by

\[ E_h(c)(u_h)^{k+\frac{1}{2}} \overset{\text{def}}{=} \frac{1}{2} \left\| \frac{u_{h}^{k+1} - u_{h}^{k}}{\Delta t} \right\|_{\Delta x}^2 + \frac{1}{2} \left\langle u_{h}^{k}, u_{h}^{k+1} \right\rangle_{A_h(c)} \]

\[ \text{kinetic energy} \quad \text{potential energy} \]

\[ \left\langle v_{h}, w_{h} \right\rangle_{A_h(c)} \overset{\text{def}}{=} \left\langle A_h(c) v_{h}, w_{h} \right\rangle_{\Delta x} \text{ and } (A_h(c) v_{h})_j \overset{\text{def}}{=} -c^2 \frac{v_{j+1} - 2v_{j} + v_{j-1}}{\Delta x^2}. \]

Note that this energy is constant if \( f = 0 \).

We prove an overestimation and an underestimation of this energy. \( \Rightarrow u_{h} \text{ does not diverge.} \)
Proof idea 3/3: convergence

The convergence error is solution of the same discrete scheme with inputs

\[ u_{0,j} = 0, \quad u_{1,j} = \frac{e_j^1}{\Delta t}, \quad \text{and} \quad s_j^k = \varepsilon_j^{k+1}. \]

+ proofs about the initializations.
Proof idea 3/3: convergence

The convergence error is solution of the same discrete scheme with inputs

\[ u_{0,j} = 0, \quad u_{1,j} = \frac{e^1_j}{\Delta t}, \quad \text{and} \quad s^k_j = \varepsilon^{k+1}_j. \]

+ proofs about the initializations.

All these proofs require the existence of \( \zeta \) and \( \xi \) in \( ]0,1[ \) with \( \zeta \leq 1 - \xi \) and we require that \( \zeta \leq \frac{c\Delta t}{\Delta x} \leq 1 - \xi \) (CFL conditions).
We proved that:

\[
\left\| e_h^{k_{\Delta t}(t)} \right\|_{\Delta x} = O \quad t \in [0, t_{\text{max}}] \quad (\Delta x^2 + \Delta t^2).
\]

\[
(\Delta x, \Delta t) \to 0 \\
0 < \Delta x \land 0 < \Delta t \land \\
\zeta \leq c \frac{\Delta t}{\Delta x} \leq 1 - \xi
\]
Extraction of the big O constants

The preceding result is a uniform big O defined by:

$$\exists \alpha, C > 0, \forall x, \Delta x, \quad \|\Delta x\| \leq \alpha \Rightarrow |f(x, \Delta x)| \leq C \cdot |g(\Delta x)|.$$
Extraction of the big O constants

The preceding result is a uniform big O defined by:

\[ \exists \alpha, C > 0, \ \forall x, \Delta x, \quad \| \Delta x \| \leq \alpha \Rightarrow |f(x, \Delta x)| \leq C \cdot |g(\Delta x)|. \]

Let \((\alpha_3, C_3)\) be the constants for the order-3 Taylor development of the exact solution and \((\alpha_4, C_4)\) for order-4. The initial support is \([\chi_1; \chi_2]\).
Extraction of the big O constants

The preceding result is a uniform big O defined by:

\[
\exists \alpha, C > 0, \quad \forall x, \Delta x, \quad \|\Delta x\| \leq \alpha \Rightarrow |f(x, \Delta x)| \leq C \cdot |g(\Delta x)|.
\]

Let \((\alpha_3, C_3)\) be the constants for the order-3 Taylor development of the exact solution and \((\alpha_4, C_4)\) for order-4. The initial support is \([\chi_1; \chi_2]\).

\[
\begin{align*}
\alpha &= \min(\alpha_3, \alpha_4, 1, t_{\text{max}}) \\
\quad s_1 &= \max(1, 2 \cdot C_4 \cdot (c^2 + 1), C_3 \cdot (1 + c^2/2) + 1) \\
\quad s_2 &= s_1^2 \left( |\chi_2| - |\chi_1| + 2 \cdot c \cdot t_{\text{max}} \cdot \left(1 + \frac{1}{\zeta}\right) + 3 \right) \\
\quad s_3 &= \frac{1}{\sqrt{2}} \left( C_3 \cdot (1 + c^2/2) + 1 \right) \cdot (\chi_2 - \chi_1 + 1 + (2 \cdot c + 4)) \\
&\quad + \frac{\sqrt{2}}{2\sqrt{2\xi - \xi^2}} (2 \cdot t_{\text{max}} \cdot s_2 + 2s) \\
\quad C &= \frac{\sqrt{2}}{\sqrt{2\xi - \xi^2}} \cdot 2 \cdot t_{\text{max}} \cdot s_3
\end{align*}
\]
Outline

1. Introduction

2. Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3. 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4. Conclusion
Program verification

- 154 lines of annotations for 32 lines of C
- **150 verification conditions:**
  - 44 about the behavior
  - 106 about the safety (runtime errors)
Program verification

- 154 lines of annotations for 32 lines of C
- 150 verification conditions:
  - 44 about the behavior
  - 106 about the safety (runtime errors)

<table>
<thead>
<tr>
<th>Prover</th>
<th>Behavior VC</th>
<th>Safety VC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt-Ergo</td>
<td>18</td>
<td>80</td>
<td>98</td>
</tr>
<tr>
<td>CVC3</td>
<td>18</td>
<td>89</td>
<td>107</td>
</tr>
<tr>
<td>Gappa</td>
<td>2</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>Z3</td>
<td>21</td>
<td>63</td>
<td>84</td>
</tr>
<tr>
<td>Automatically proved</td>
<td>23</td>
<td>94</td>
<td>117</td>
</tr>
<tr>
<td>Coq</td>
<td>21</td>
<td>12</td>
<td>33</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>106</td>
<td>150</td>
</tr>
</tbody>
</table>
Program verification

- About 90% of the safety goals (matrix access, Overflow, and so on) are proved automatically.
- 33 theorems are interactively proved using Coq for a total of about 15,000 lines of Coq and 30 minutes of compilation.

<table>
<thead>
<tr>
<th>Type of proofs</th>
<th>Nb spec lines</th>
<th>Nb lines</th>
<th>Compilation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence</td>
<td>991</td>
<td>5,275</td>
<td>42 s</td>
</tr>
<tr>
<td>Round-off + runtime errors</td>
<td>7,737</td>
<td>13,175</td>
<td>32 min</td>
</tr>
</tbody>
</table>
Outline

1 Introduction

2 Prerequisite
   - Floating-Point Arithmetic
   - Proof assistant: Coq
   - Deductive Program Verification

3 1-D Wave equation discretization
   - Presentation
   - Rounding Error
   - Method Error
   - Program Verification

4 Conclusion
Conclusion on the 1-D wave equation discretization

- Very high guarantee
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- interdisciplinary (formal methods / numerical analysis)
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- interdisciplinary (formal methods / numerical analysis)
- not only rounding errors:
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- interdisciplinary (formal methods / numerical analysis)
- not only rounding errors:
  - all other errors such as pointer dereferencing or division by zero
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- interdisciplinary (formal methods / numerical analysis)
- not only rounding errors:
  - all other errors such as pointer dereferencing or division by zero
  - link with mathematical properties
Conclusion on the 1-D wave equation discretization

- Very high guarantee

- interdisciplinary (formal methods / numerical analysis)

- not only rounding errors:
  - all other errors such as pointer dereferencing or division by zero
  - link with mathematical properties
  - any property can be checked
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- interdisciplinary (formal methods / numerical analysis)
- not only rounding errors:
  - all other errors such as pointer dereferencing or division by zero
  - link with mathematical properties
  - any property can be checked
- expressive annotation language (as expressive as Coq)
  ⇒ exactly the specification you want
Conclusion on the 1-D wave equation discretization

- Very high guarantee
- **interdisciplinary** (formal methods / numerical analysis)
- **not only rounding errors:**
  - all other errors such as pointer dereferencing or division by zero
  - link with mathematical properties
  - *any* property can be checked
- **expressive annotation language** (as expressive as Coq)
  \[ \Rightarrow \text{exactly the specification you want} \]
- an annotated C program to *convince* numerical analysts
Perspectives

- go deeper into numerical analysis
Perspectives

- go deeper into numerical analysis
  ⇒ proof of the finite element method
Perspectives

- go deeper into numerical analysis

⇒ proof of the finite element method

⇒ proof of the finite element method library
Perspectives

- go deeper into **numerical analysis**
  - proof of the finite element method
  - proof of the finite element method library
  - stability (floating-point stability / numerical analysis stability)
• go deeper into numerical analysis

⇒ proof of the finite element method

⇒ proof of the finite element method library

⇒ stability (floating-point stability / numerical analysis stability)

• prove and generalize well-known facts/algorithms/programs from the computer arithmetic community
Perspectives

- go deeper into **numerical analysis**
  
  ⇒ proof of the finite element method

  ⇒ proof of the finite element method library

  ⇒ stability (floating-point stability / numerical analysis stability)

- prove and generalize **well-known** facts/algorithms/programs from the computer arithmetic community

  ⇒ basic blocks to build upon