

# Simulation and Analysis of 1D Wave Propagation under Various Physical Models

Dario Liotta



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



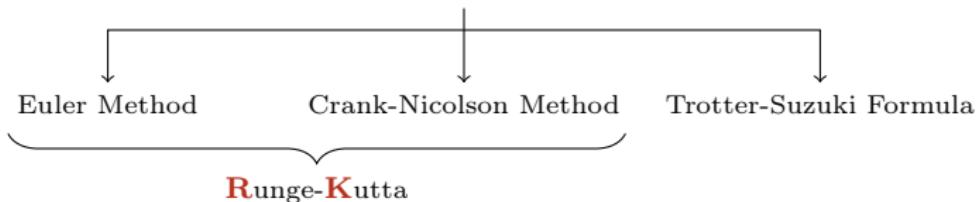
Dipartimento  
di Fisica  
e Astronomia  
Galileo Galilei

September 6th 2025

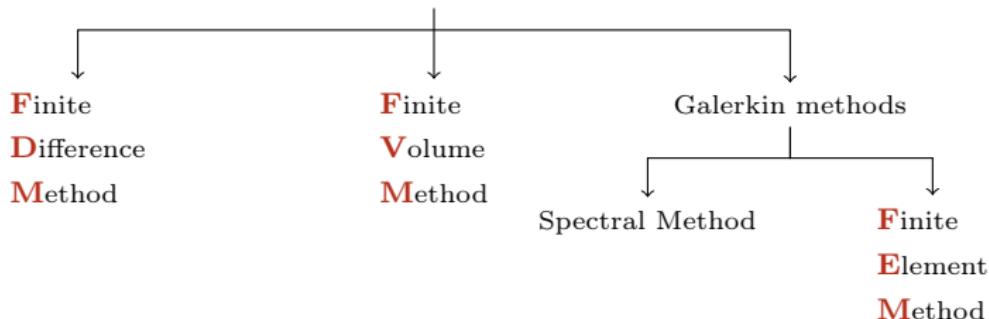
Course of **Quantum Information and Computing**  
Academic Year 2024/2025

# Numerical methods for differential equations

## Ordinary Differential Equations

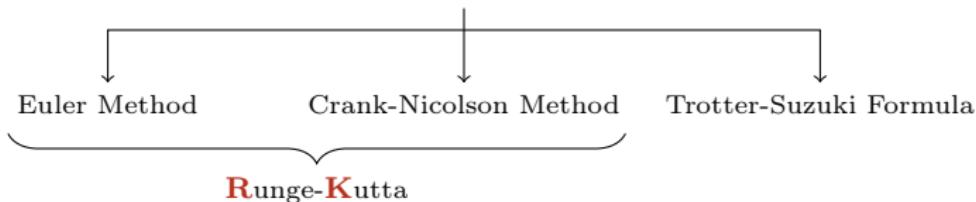


## Partial Differential Equations

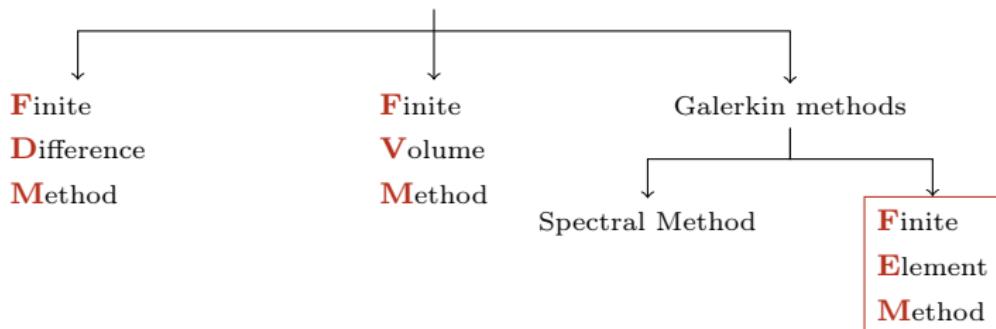


# Numerical methods for differential equations

## Ordinary Differential Equations



## Partial Differential Equations



# Introduction to the problem

Solving a **PDE** means to find a function  $u$  such that

$$\mathcal{L}u = f$$

where  $\mathcal{L}$  is a differential operator and  $f$  is a source term.

The equation holds in a domain  $\Omega$  and is completed by prescribing **boundary conditions** on  $\partial\Omega$ .

In most physical applications  $\mathcal{L}$  is a second-order operator

- Poisson equation:  $\mathcal{L} = -\Delta$
- Heat equation:  $\mathcal{L} = \frac{\partial}{\partial t} - \Delta$
- Wave equation:  $\mathcal{L} = \frac{\partial^2}{\partial t^2} - c^2 \Delta$

# Introduction to the problem

Solving a **PDE** means to find a function  $u$  such that

$$\mathcal{L}u = f$$

where  $\mathcal{L}$  is a differential operator and  $f$  is a source term.

The equation holds in a domain  $\Omega$  and is completed by prescribing **boundary conditions** on  $\partial\Omega$ .

In most physical applications  $\mathcal{L}$  is a second-order operator

- **Poisson** equation:  $\mathcal{L} = -\Delta$
- **Heat** equation:  $\mathcal{L} = \frac{\partial}{\partial t} - \Delta$
- **Wave** equation:  $\mathcal{L} = \frac{\partial^2}{\partial t^2} - c^2 \Delta$

# Weak formulation

Galerkin methods rely on a **weak formulation**

- Multiply by a test function  $v$  and integrate over the entire domain

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} f v d\Omega$$

- Integrate by parts the left hand side

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} \nabla u \cdot \nabla v d\Omega - \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

- Substitute and get the new expression

$$\int_{\Omega} \nabla u \cdot \nabla v d\Omega = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

# Weak formulation

Galerkin methods rely on a **weak formulation**

- Multiply by a test function  $v$  and integrate over the entire domain

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} f v d\Omega$$

- Integrate by parts the left hand side

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} \nabla u \cdot \nabla v d\Omega - \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

- Substitute and get the new expression

$$\int_{\Omega} \nabla u \cdot \nabla v d\Omega = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

# Weak formulation

Galerkin methods rely on a **weak formulation**

- Multiply by a test function  $v$  and integrate over the entire domain

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} f v d\Omega$$

- Integrate by parts the left hand side

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} \nabla u \cdot \nabla v d\Omega - \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

- Substitute and get the new expression

$$\int_{\Omega} \nabla u \cdot \nabla v d\Omega = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

# Weak formulation

Galerkin methods rely on a **weak formulation**

- Multiply by a test function  $v$  and integrate over the entire domain

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} f v d\Omega$$

- Integrate by parts the left hand side

$$-\int_{\Omega} (\Delta u) v d\Omega = \int_{\Omega} \nabla u \cdot \nabla v d\Omega - \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

- Substitute and get the new expression

$$\int_{\Omega} \nabla u \cdot \nabla v d\Omega = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds$$

## About the test function

The test function  $v$  is introduced to check whether the PDE is satisfied on average throughout the domain.

The problem becomes to find  $u$  such that

$$a(u, v) = F(v) \quad \forall v \in V$$

where

$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v d\Omega \quad \text{is a } \underline{\text{bilinear form}}$$

$$F(v) = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds \quad \text{is a } \underline{\text{linear functional}}$$

## About the test function

The test function  $v$  is introduced to check whether the PDE is satisfied on average throughout the domain.

The problem becomes to find  $u$  such that

$$a(u, v) = F(v) \quad \forall v \in V$$

where

$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v d\Omega \quad \text{is a } \underline{\text{bilinear form}}$$

$$F(v) = \int_{\Omega} f v d\Omega + \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds \quad \text{is a } \underline{\text{linear functional}}$$

# Benefits of the weak formulation

## Strong formulation

$$u \in C^2(\Omega)$$

Holds pointwise in  $\Omega$

Derivatives exist classically

## Weak formulation

$$u, v \in H^1(\Omega)^*$$

Holds on average on  $\Omega$

Derivatives exist in the distributional sense

In short: weak formulation requires **less regularity**

\*  $H^1(\Omega)$  is a **Sobolev space** of functions with square-integrable first derivatives:

$$w \in H^1(\Omega) = \left\{ w \in L^2(\Omega) \mid \nabla w \in L^2(\Omega)^d \right\}$$

# Benefits of the weak formulation

## Strong formulation

$$u \in C^2(\Omega)$$

Holds pointwise in  $\Omega$

Derivatives exist classically

## Weak formulation

$$u, v \in H^1(\Omega)^*$$

Holds on average on  $\Omega$

Derivatives exist in the distributional sense

In short: weak formulation requires **less regularity**

\*  $H^1(\Omega)$  is a **Sobolev space** of functions with square-integrable first derivatives:

$$w \in H^1(\Omega) = \left\{ w \in L^2(\Omega) \mid \nabla w \in L^2(\Omega)^d \right\}$$

# On boundary conditions

Another difference lies in the boundary condition prescription.



$v = 0$  on  $\partial\Omega \Rightarrow$  cancels boundary term  
 (no information available on  $\frac{\partial u}{\partial n}$ )

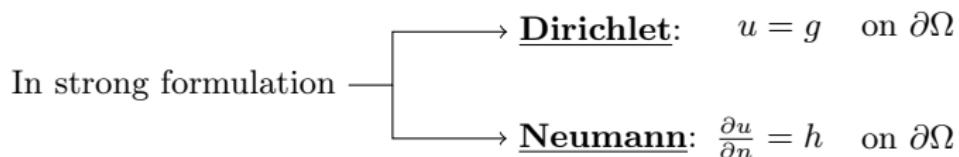
$v$  free on  $\partial\Omega$

$u = g$  enforced on  $\partial\Omega$  (final solution)

$\frac{\partial u}{\partial n} = h$  naturally enters weak form

# On boundary conditions

Another difference lies in the boundary condition prescription.



$v = 0$  on  $\partial\Omega \Rightarrow$  cancels boundary term  
 (no information available on  $\frac{\partial u}{\partial n}$ )

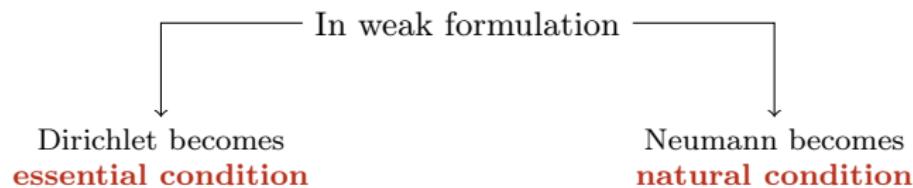
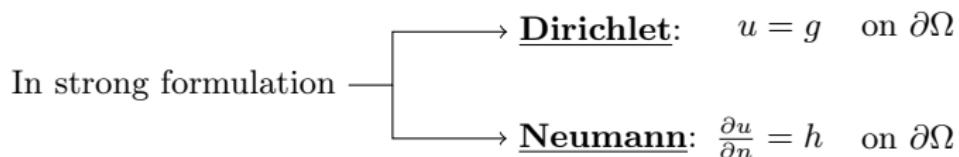
$v$  free on  $\partial\Omega$

$u = g$  enforced on  $\partial\Omega$  (final solution)

$\frac{\partial u}{\partial n} = h$  naturally enters weak form

# On boundary conditions

Another difference lies in the boundary condition prescription.



$v = \mathbf{0}$  on  $\partial\Omega \Rightarrow$  cancels boundary term  
 (no information available on  $\frac{\partial u}{\partial n}$ )

$v$  free on  $\partial\Omega$

$u = g$  enforced on  $\partial\Omega$  (final solution)

$\frac{\partial u}{\partial n} = h$  naturally enters weak form

# Shape functions

Galerkin methods allow to find an approximate solution

$u_h \in V_h \subset H^1(\Omega)$  where  $V_h$  is a **finite-dimensional** space

In this framework, the goal is to find  $u_h$  such that

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h$$

A basis of function  $\{\phi_i\}$  is chosen to express  $u_h$  and to use it as test:

$$u_h = \sum_{j=1}^N u_j \phi_j \implies a\left(\sum_{j=1}^N u_j \phi_j, \phi_i\right) = F(\phi_i) \quad \forall i = 1, \dots, N$$

Functions  $\phi_i$  model the solution  $\longrightarrow$  **shape functions**

# Shape functions

Galerkin methods allow to find an approximate solution

$$u_h \in V_h \subset H^1(\Omega) \quad \text{where } V_h \text{ is a **finite-dimensional** space}$$

In this framework, the goal is to find  $u_h$  such that

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h$$

A **basis of function**  $\{\phi_i\}$  is chosen to express  $u_h$  and to use it as test:

$$u_h = \sum_{j=1}^N u_j \phi_j \implies a\left(\sum_{j=1}^N u_j \phi_j, \phi_i\right) = F(\phi_i) \quad \forall i = 1, \dots, N$$

Functions  $\phi_i$  model the solution  $\longrightarrow$  **shape functions**

# Shape functions

Galerkin methods allow to find an approximate solution

$$u_h \in V_h \subset H^1(\Omega) \quad \text{where } V_h \text{ is a **finite-dimensional** space}$$

In this framework, the goal is to find  $u_h$  such that

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h$$

A **basis of function**  $\{\phi_i\}$  is chosen to express  $u_h$  and to use it as test:

$$u_h = \sum_{j=1}^N u_j \phi_j \implies a\left(\sum_{j=1}^N u_j \phi_j, \phi_i\right) = F(\phi_i) \quad \forall i = 1, \dots, N$$

Functions  $\phi_i$  model the solution → **shape functions**

# Final expression

By linearity of  $a(\cdot, \cdot)$ , the problem reduces to a **finite linear system**:

$$\sum_{j=1}^N u_j a(\phi_j, \phi_i) = F(\phi_i) \quad \forall i = 1, \dots, N$$

$\Downarrow$

$$A\mathbf{u} = \mathbf{F}$$

where

$$A_{i,j} = a(\phi_j, \phi_i)$$

form the **stiffness matrix**

$$\mathbf{u} = (u_1, \dots, u_N)^T$$

is the **vector of unknowns**

$$\mathbf{F} = (F(\phi_1), \dots, F(\phi_N))^T$$

is the **load vector**

# Final expression

By linearity of  $a(\cdot, \cdot)$ , the problem reduces to a **finite linear system**:

$$\sum_{j=1}^N u_j a(\phi_j, \phi_i) = F(\phi_i) \quad \forall i = 1, \dots, N$$

$\Downarrow$

$$A\mathbf{u} = \mathbf{F}$$

where

$A_{i,j} = a(\phi_j, \phi_i)$  form the **stiffness matrix**

$\mathbf{u} = (u_1, \dots, u_N)^T$  is the **vector of unknowns**

$\mathbf{F} = (F(\phi_1), \dots, F(\phi_N))^T$  is the **load vector**

# Mesh discretization

**FEM** approach consists in the subdivision of the domain in a so-called **mesh**

This choice brings several advantages:

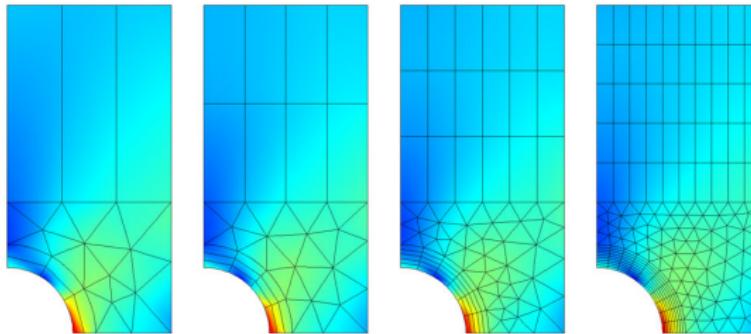
- Good approximation of **complex geometries**
- Better capture of **local effects**
- Possibility of **adaptive refinement**
- Natural construction of a **global solution**

# Mesh discretization

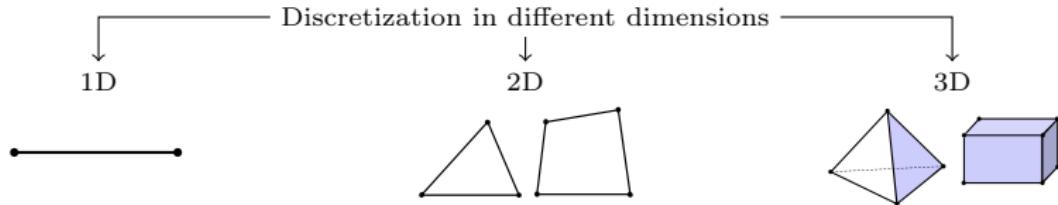
**FEM** approach consists in the subdivision of the domain in a so-called **mesh**

This choice brings several advantages:

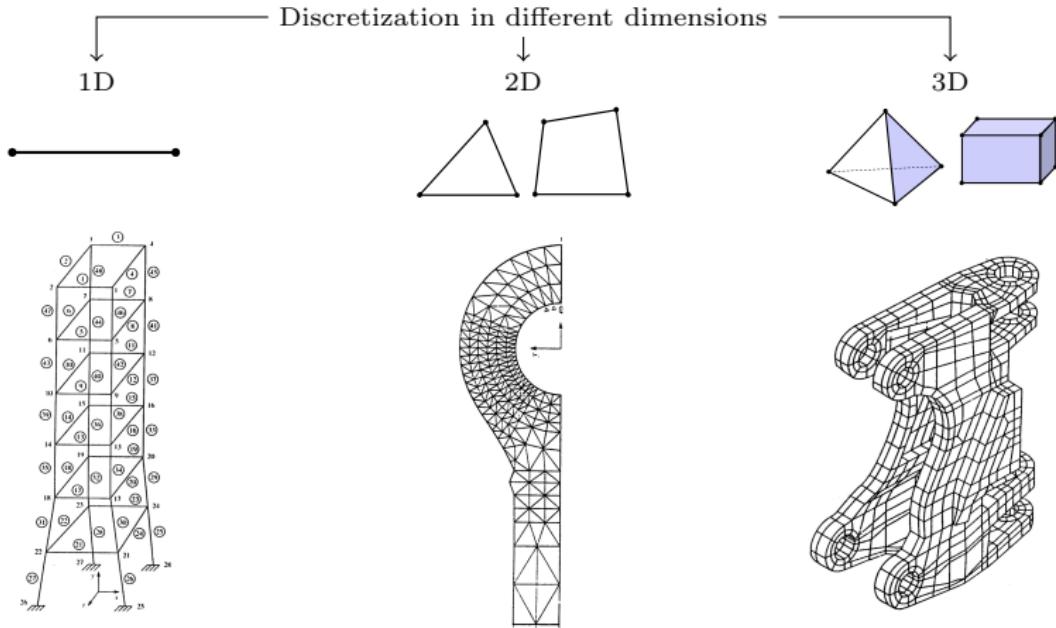
- Good approximation of **complex geometries**
- Better capture of **local effects**
- Possibility of **adaptive refinement**
- Natural construction of a **global solution**



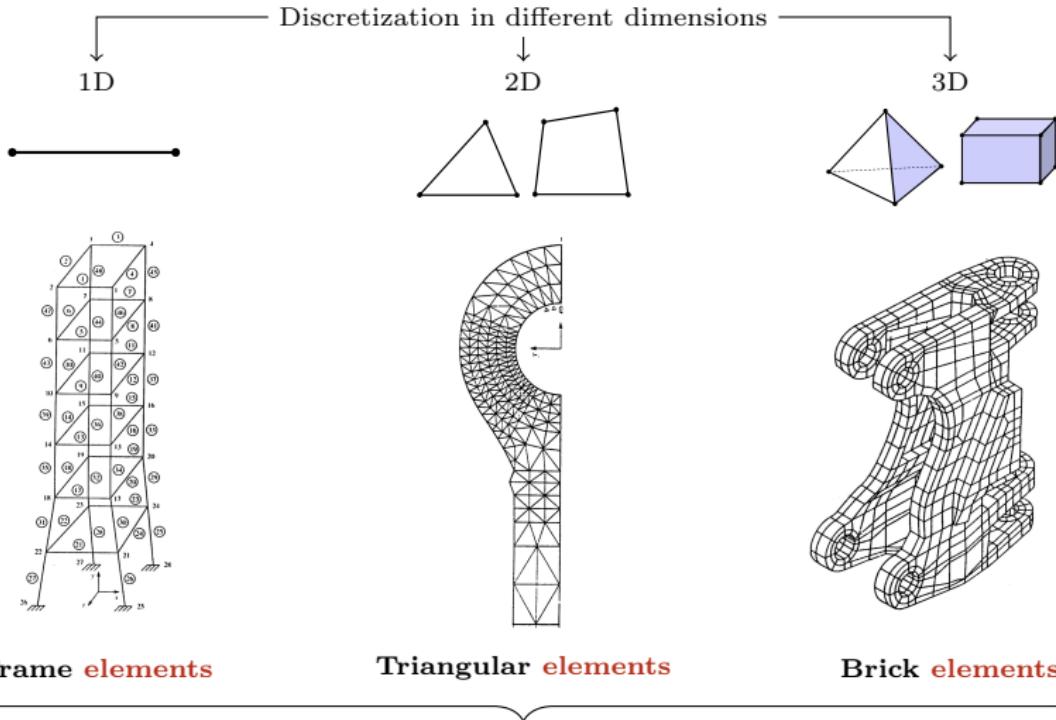
# Elements



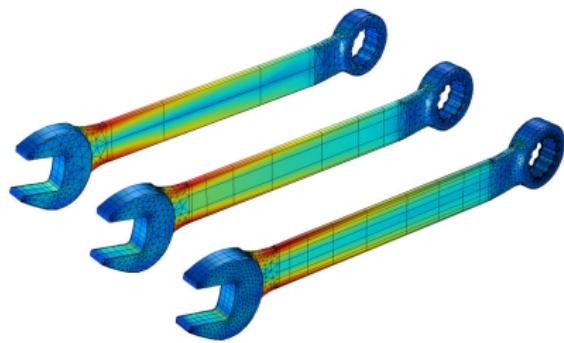
# Elements



# Elements



# Application examples

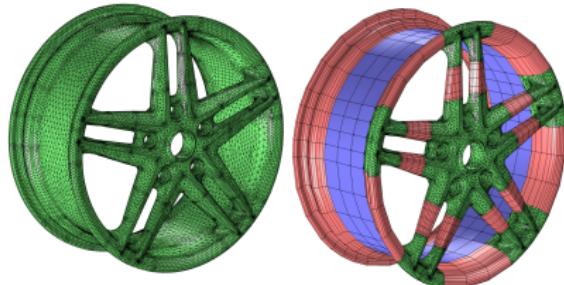


*Manual mesh refinement of a wrench using different element types*

Image from COMSOL Multiphysics Encyclopedia, "Finite Element Mesh Refinement", 21st of February 2017

*Mesh of a wheel rim composed of tetrahedrons in green, bricks in blue and prisms in pink*

Image from COMSOL Multiphysics Blog, "Meshing Your Geometry: When to Use the Various Element Types", Walter Frei, 4th of November 2013



# Choise of the base

Mesh division into sub-domains



Choice of a **local basis functions**



Exploitation of compact support



- Leads to sparse matrices
- Allows local interpolation
- Enhances numerical stability
- Enables efficient parallelization

# Choise of the base

Mesh division into sub-domains



Choice of a **local basis functions**



Exploitation of compact support



- Leads to sparse matrices
- Allows local interpolation
- Enhances numerical stability
- Enables efficient parallelization

# Choise of the base

Mesh division into sub-domains



Choice of a **local basis functions**



Exploitation of compact support



- Leads to sparse matrices
- Allows local interpolation
- Enhances numerical stability
- Enables efficient parallelization

## Choise of the base

### Mesh division into sub-domains



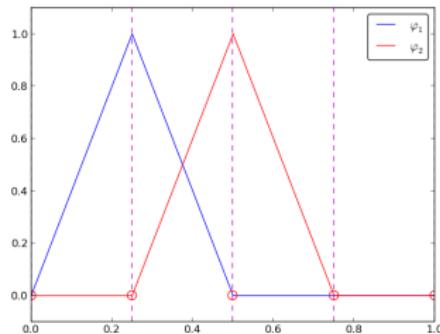
## Choice of a local basis functions



#### - Exploitation of **compact support**



- Leads to **sparse matrices**
  - Allows **local interpolation**
  - Enhances **numerical stability**
  - Enables **efficient parallelization**



# FEniCS library

A leading software platform for finite element computations is **FEniCS**.

- Open-source and freely available
- Multi-language support (C++ and Python APIs)
- Parallel computing with MPI support

FEniCS package:  $\left\{ \begin{array}{ll} \text{DOLFIN} & (\text{backend core engine and PETSc interface}) \\ \text{UFL} & (\text{symbolic language}) \\ \text{FIAT} & (\text{shape functions tabulator}) \\ \text{FFC} & (\text{C++ compiler for efficient local assembly}) \\ \text{MSHR} & (\text{mesh generator}) \end{array} \right.$

# FEniCS library

A leading software platform for finite element computations is **FEniCS**.

- **Open-source** and freely available
- **Multi-language support** (C++ and Python APIs)
- **Parallel computing** with MPI support



FEniCS package:

DOLFIN	(backend core engine and PETSc interface)
UFL	(symbolic language)
FIAT	(shape functions tabulator)
FFC	(C++ compiler for efficient local assembly)
MSHR	(mesh generator)

# FEniCS library

A leading software platform for finite element computations is **FEniCS**.

- **Open-source** and freely available
- **Multi-language support** (C++ and Python APIs)
- **Parallel computing** with MPI support



FEniCS package:  $\left\{ \begin{array}{ll} \text{DOLFIN} & (\text{backend core engine and PETSc interface}) \\ \text{UFL} & (\text{symbolic language}) \\ \text{FIAT} & (\text{shape functions tabulator}) \\ \text{FFC} & (\text{C++ compiler for efficient local assembly}) \\ \text{MSHR} & (\text{mesh generator}) \end{array} \right.$

# A minimal FEniCS example: setup

Setup of a Poisson equation with Neumann boundary conditions in FEniCS:

- Generation of the mesh

```
domain = mesh.create_interval(MPI.COMM_WORLD, nx, [0.0, L])
```

- Definition of the finite element function space

```
V = functionspace(domain, ("Lagrange", 1))
```

- Definition of trial function and test function

```
u = ufl.TrialFunction(V)
v = ufl.TestFunction(V)
```

- Definition of the source term

```
f = fem.Constant(domain, default_scalar_type(-6))
```

# A minimal FEniCS example: solution

Solving Poisson equation with Neumann boundary conditions in FEniCS:

- Weak formulation

```
a = ufl.dot(ufl.grad(u), ufl.grad(v)) * ufl.dx
F = f * v * ufl.dx
```

- Solution of the linear system

```
problem = LinearProblem(a, F,
                        petsc_option = {"ksp_type": "preonly",
                                      "pc_type" : "lu"
                                    })
u_h = problem.solve()
```

# Approach to the classical wave equation

Our first goal is to approximate the solution of

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad \leftarrow \text{d'Alembert equation}$$



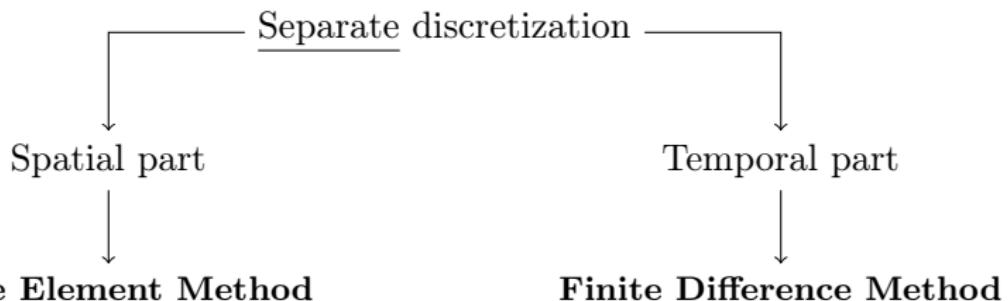
$$\int_0^L \frac{\partial^2 u}{\partial x^2} v dx = \frac{\partial u}{\partial x} v \Big|_0^L - \int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} dx$$

$$\frac{\partial^2 u}{\partial t^2} \simeq \frac{u^{(n+1)} - 2u^{(n)} + u^{(n-1)}}{\Delta t^2}$$

# Approach to the classical wave equation

Our first goal is to approximate the solution of

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad \leftarrow \text{d'Alembert equation}$$



$$\int_0^L \frac{\partial^2 u}{\partial x^2} v dx = \frac{\partial u}{\partial x} v \Big|_0^L - \int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} dx$$

$$\frac{\partial^2 u}{\partial t^2} \simeq \frac{u^{(n+1)} - 2u^{(n)} + u^{(n-1)}}{\Delta t^2}$$

# From PDE to ODEs

To do so, a **separable base** must be chosen:

$$u_h(x, t) = \sum_{j=1}^N u_j(t) \phi_j(x)$$

Managing the boundary term separately, the weak formulation goes as

$$\int_0^L \frac{\partial^2 u}{\partial t^2} v dx + c^2 \int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} dx = 0 \quad \forall v \in H^1([0, L])$$



$$\sum_{j=1}^N \frac{d^2 u_j}{dt^2} \int_0^L \phi_j \phi_i dx + c^2 \sum_{j=1}^N u_j \int_0^L \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx = 0 \quad \forall i = 1, \dots, N$$

# From PDE to ODEs

To do so, a **separable base** must be chosen:

$$u_h(x, t) = \sum_{j=1}^N u_j(t) \phi_j(x)$$

Managing the boundary term separately, the weak formulation goes as

$$\int_0^L \frac{\partial^2 \textcolor{blue}{u}}{\partial t^2} \textcolor{green}{v} dx + c^2 \int_0^L \frac{\partial \textcolor{blue}{u}}{\partial x} \frac{\partial \textcolor{green}{v}}{\partial x} dx = 0 \quad \forall v \in H^1([0, L])$$



$$\sum_{j=1}^N \frac{d^2 u_j}{dt^2} \int_0^L \phi_j \phi_i dx + c^2 \sum_{j=1}^N u_j \int_0^L \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx = 0 \quad \forall i = 1, \dots, N$$

# From PDE to ODEs

To do so, a **separable base** must be chosen:

$$u_h(x, t) = \sum_{j=1}^N u_j(t) \phi_j(x)$$

Managing the boundary term separately, the weak formulation goes as

$$\int_0^L \frac{\partial^2 \textcolor{blue}{u}}{\partial t^2} \textcolor{green}{v} dx + c^2 \int_0^L \frac{\partial \textcolor{blue}{u}}{\partial x} \frac{\partial \textcolor{green}{v}}{\partial x} dx = 0 \quad \forall v \in H^1([0, L])$$



$$\sum_{j=1}^N \frac{d^2 \textcolor{blue}{u}_j}{dt^2} \int_0^L \phi_j \phi_i dx + c^2 \sum_{j=1}^N u_j \int_0^L \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx = 0 \quad \forall i = 1, \dots, N$$

# Matrix formulation

Let's define

$$M : M_{i,j} = \int_0^L \phi_j \phi_i dx \quad \leftarrow \text{Mass matrix}$$

$$A : A_{i,j} = \int_0^L \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx \quad \leftarrow \text{Stiffness matrix}$$



$$M \frac{d^2 u}{dt^2} + c^2 A u = 0$$

# Matrix formulation

Let's define

$$M : M_{i,j} = \int_0^L \phi_j \phi_i dx \quad \leftarrow \text{Mass matrix}$$

$$A : A_{i,j} = \int_0^L \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx \quad \leftarrow \text{Stiffness matrix}$$



$$M \frac{d^2 \mathbf{u}}{dt^2} + c^2 A \mathbf{u} = 0$$

# Time discretization

Let's apply **implicit central difference scheme**:

$$M \frac{\mathbf{u}^{(n+1)} - 2\mathbf{u}^{(n)} + \mathbf{u}^{(n-1)}}{\Delta t^2} + c^2 A \mathbf{u}^{(n+1)} = 0$$



$$\left( \frac{1}{\Delta t^2} M + c^2 A \right) \mathbf{u}^{(n+1)} = \frac{2}{\Delta t^2} M \mathbf{u}^{(n)} - \frac{1}{\Delta t^2} M \mathbf{u}^{(n-1)}$$

# Time discretization

Let's apply **implicit central difference scheme**:

$$M \frac{\mathbf{u}^{(n+1)} - 2\mathbf{u}^{(n)} + \mathbf{u}^{(n-1)}}{\Delta t^2} + c^2 A \mathbf{u}^{(n+1)} = 0$$



$$\left( \frac{1}{\Delta t^2} M + c^2 A \right) \mathbf{u}^{(n+1)} = \frac{2}{\Delta t^2} M \mathbf{u}^{(n)} - \frac{1}{\Delta t^2} M \mathbf{u}^{(n-1)}$$

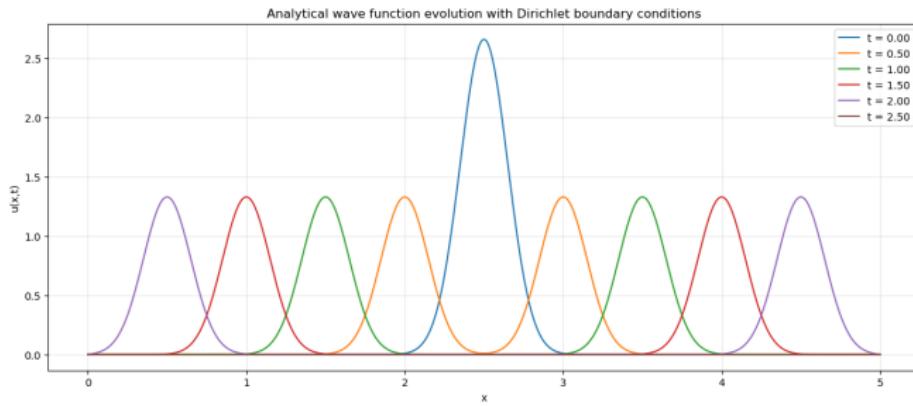
# Analytical solutions

Solutions are known since 1747 due to d'Alembert himself.

- Dirichlet boundary conditions:  $u(0, t) = u(L, t) = 0$

$$u(x, t) = \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi}{L}ct\right) \sin\left(\frac{n\pi}{L}x\right)$$

with  $A_n = \frac{2}{L} \int_0^L f(x) \sin\left(\frac{n\pi}{L}x\right) dx$ .



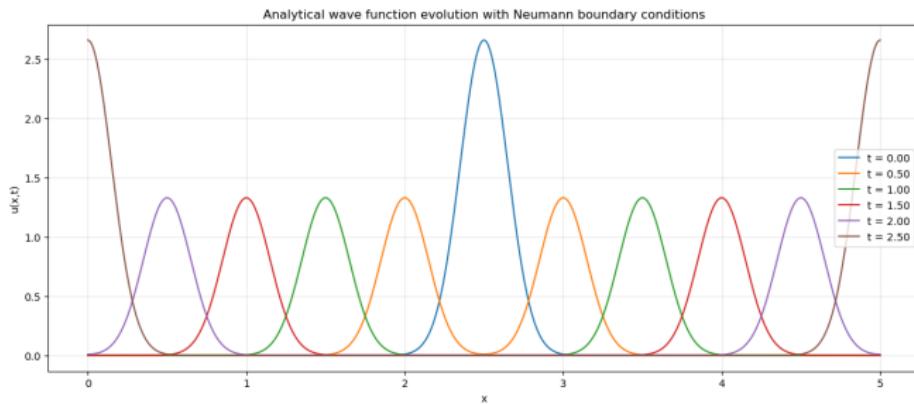
# Analytical solutions

Solutions are known since 1747 due to d'Alembert himself.

- **Neumann** boundary conditions:  $\frac{\partial u}{\partial x}\Big|_{x=0} = \frac{\partial u}{\partial x}\Big|_{x=L} = 0$

$$u(x, t) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi}{L}ct\right) \cos\left(\frac{n\pi}{L}x\right)$$

with  $A_0 = \frac{1}{L} \int_0^L f(x)dx$ ,  $A_n = \frac{2}{L} \int_0^L f(x) \cos\left(\frac{n\pi}{L}x\right) dx$ .



# Approximate solutions

The selection of discretization steps should\* take into account

$$\text{CFL stability condition : } \frac{c\Delta t}{\Delta x} \lesssim 1$$

Choosing first-order Lagrange polynomials as  $\{\phi_i\}$ :

\*While for implicit schemes it is only a recommendation, for explicit ones it is mandatory.

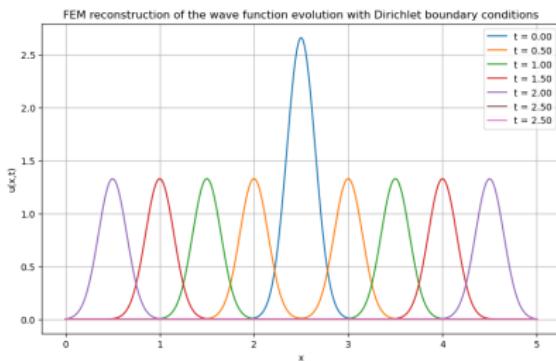
# Approximate solutions

The selection of discretization steps should\* take into account

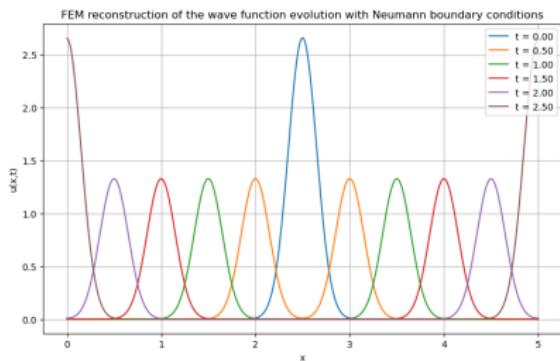
$$\text{CFL stability condition : } \frac{c\Delta t}{\Delta x} \lesssim 1$$

Choosing first-order Lagrange polynomials as  $\{\phi_i\}$ :

**Dirichlet**



**Neumann**



\* While for implicit schemes it is only a recommendation, for explicit ones it is mandatory.

# Energy loss

A strong indicator of numerical correctness is **energy conservation**.

$$\text{Energy: } E = \underbrace{\frac{T}{2c^2} \int_0^L \left( \frac{\partial u}{\partial t} \right)^2 dx}_{\text{Kinetic}} + \underbrace{\frac{T}{2} \int_0^L \left( \frac{\partial u}{\partial x} \right)^2 dx}_{\text{Potential}}$$

$T$  is the *tension*, which can be arbitrarily set to 1.

Numerical energy decay is *independent* of the CFL number, being entirely caused by the **implicit scheme** itself.

# Energy loss

A strong indicator of numerical correctness is **energy conservation**.

$$\text{Energy: } E = \underbrace{\frac{T}{2c^2} \int_0^L \left( \frac{\partial u}{\partial t} \right)^2 dx}_{\text{Kinetic}} + \underbrace{\frac{T}{2} \int_0^L \left( \frac{\partial u}{\partial x} \right)^2 dx}_{\text{Potential}}$$

$T$  is the *tension*, which can be arbitrarily set to 1.

Numerical energy decay is *independent* of the CFL number, being entirely caused by the **implicit scheme** itself.

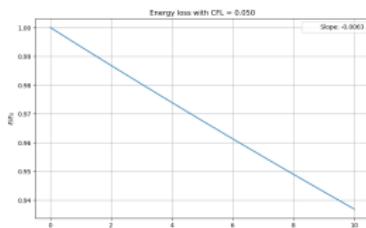
# Energy loss

A strong indicator of numerical correctness is **energy conservation**.

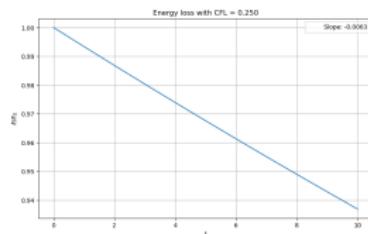
$$\text{Energy: } E = \underbrace{\frac{T}{2c^2} \int_0^L \left( \frac{\partial u}{\partial t} \right)^2 dx}_{\text{Kinetic}} + \underbrace{\frac{T}{2} \int_0^L \left( \frac{\partial u}{\partial x} \right)^2 dx}_{\text{Potential}}$$

$T$  is the *tension*, which can be arbitrarily set to 1.

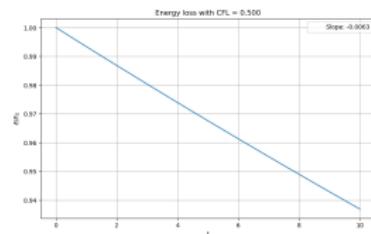
CFL = 0.05



CFL = 0.25



CFL = 0.50



Numerical energy decay is *independent* of the CFL number, being entirely caused by the **implicit scheme** itself.

# What if $c = c(x)$ ?

If  $c$  is not constant, its square must be interpolated as the wave function:

$$c(2x) \longrightarrow c_h^2(x) = \sum_{k=1}^N c_k^{(2)} \psi_k(x)$$

Velocity must be included in the stiffness matrix  $A$ :

$$A_{i,j} = \int_0^L \left( \sum_{k=1}^N c_k^{(2)} \psi_k(x) \right) \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx$$



$$M \frac{d^2 u}{dt^2} + A u = 0$$

# What if $c = c(x)$ ?

If  $c$  is not constant, its square must be interpolated as the wave function:

$$c(2x) \longrightarrow c_h^2(x) = \sum_{k=1}^N c_k^{(2)} \psi_k(x)$$

Velocity must be included in the stiffness matrix  $A$ :

$$A_{i,j} = \int_0^L \left( \sum_{k=1}^N c_k^{(2)} \psi_k(x) \right) \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx$$



$$M \frac{d^2 u}{dt^2} + A u = 0$$

# What if $c = c(x)$ ?

If  $c$  is not constant, its square must be interpolated as the wave function:

$$c(2x) \longrightarrow c_h^2(x) = \sum_{k=1}^N c_k^{(2)} \psi_k(x)$$

Velocity must be included in the stiffness matrix  $A$ :

$$A_{i,j} = \int_0^L \left( \sum_{k=1}^N c_k^{(2)} \psi_k(x) \right) \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx$$



$$M \frac{d^2 \mathbf{u}}{dt^2} + A \mathbf{u} = 0$$

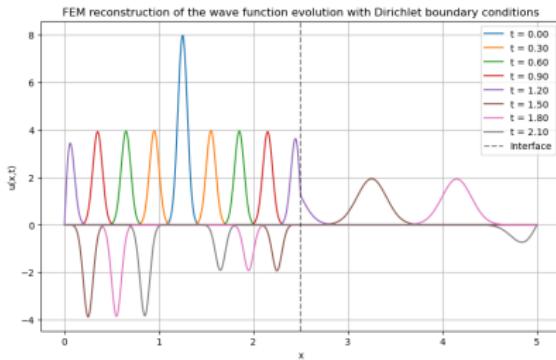
### Piecewise constant velocity

One simple example is the **piecewise constant function**:

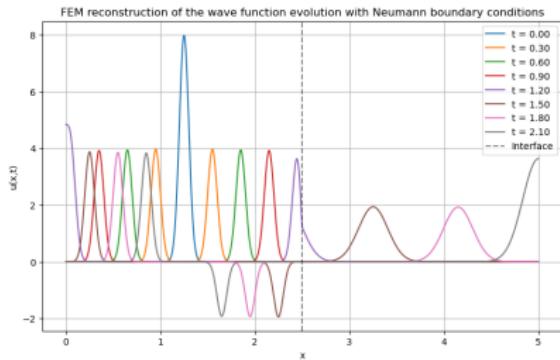
$$c(x) = \begin{cases} c_1 & x \leq x_0 \\ c_2 & x > x_0 \end{cases}$$

Placing initial pulse on the left with  $c_1 = 1, c_2 = 3$ :

## Dirichlet



Neumann



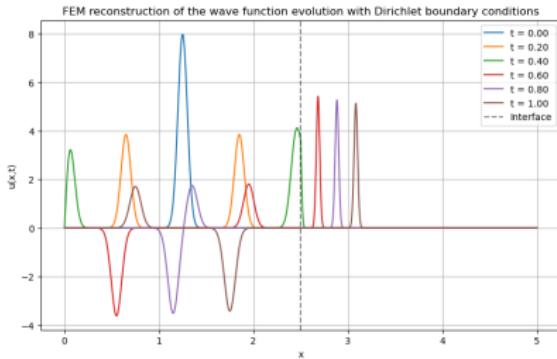
## Piecewise constant velocity

One simple example is the **piecewise constant function**:

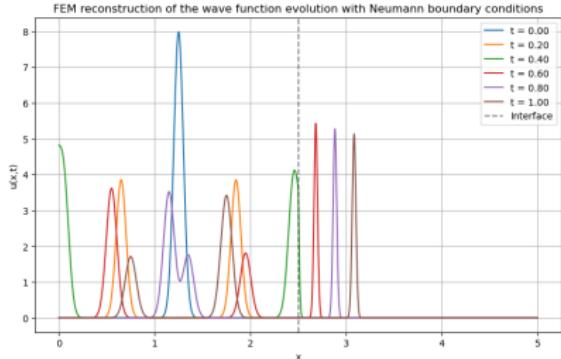
$$c(x) = \begin{cases} c_1 & x \leq x_0 \\ c_2 & x > x_0 \end{cases}$$

Placing initial pulse on the left with  $c_1 = 3, c_2 = 1$ :

### Dirichlet



Neumann



# Fresnel coefficients

In this case, the analytical reference values are the **Fresnel coefficients**.

$$T = \frac{2c_1}{c_1 + c_2} \quad R = \frac{c_2 - c_1}{c_1 + c_2}$$

$T$  and  $R$  are numerically evaluated as the ratio of the transmitted and reflected waves to the incident one.

Smaller velocity  $\rightarrow$  Smaller coefficient  $\rightarrow$  Higher sensitivity to numerical errors

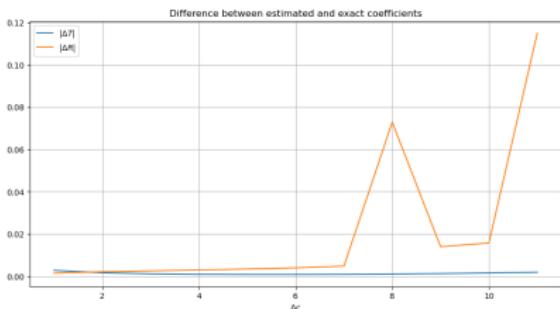
# Fresnel coefficients

In this case, the analytical reference values are the **Fresnel coefficients**.

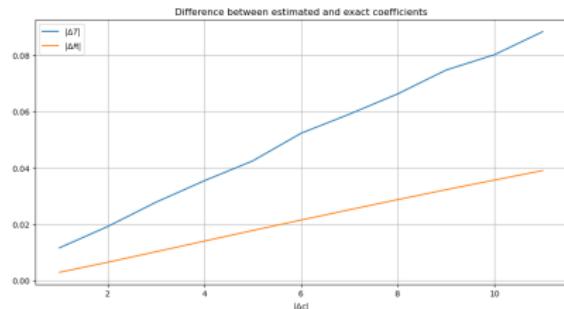
$$T = \frac{2c_1}{c_1 + c_2} \quad R = \frac{c_2 - c_1}{c_1 + c_2}$$

$T$  and  $R$  are numerically evaluated as the ratio of the **transmitted** and **reflected** waves to the **incident** one.

$$c_1 < c_2$$



$$c_1 > c_2$$

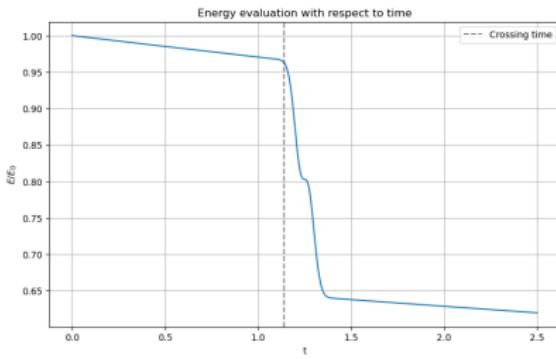


Smaller velocity  $\rightarrow$  Smaller coefficient  $\rightarrow$  Higher sensitivity to numerical errors

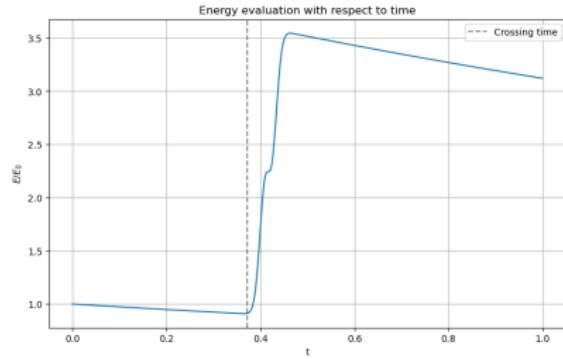
# Energy variation

Right after the wave crosses the interface, a clear **non-conservation of energy** emerges.

$$c_1 = 1 \quad c_2 = 3$$



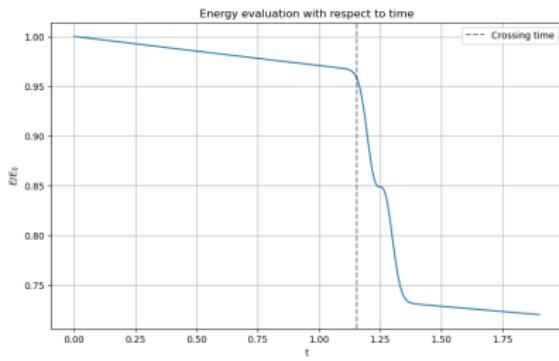
$$c_1 = 3 \quad c_2 = 1$$



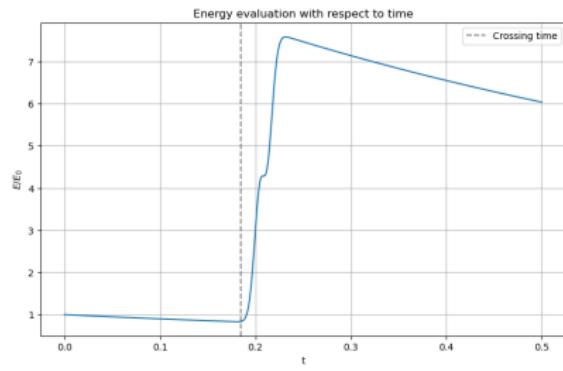
# Energy variation

Right after the wave crosses the interface, a clear **non-conservation of energy** emerges.

$$c_1 = 1 \quad c_2 = 6$$



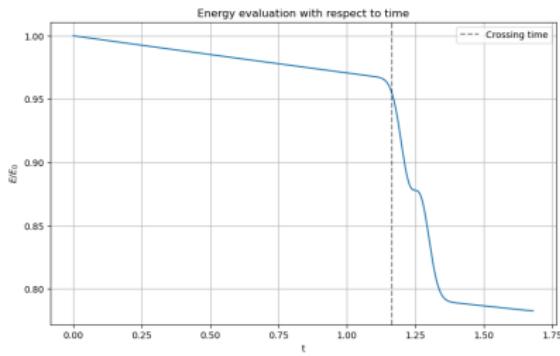
$$c_1 = 6 \quad c_2 = 1$$



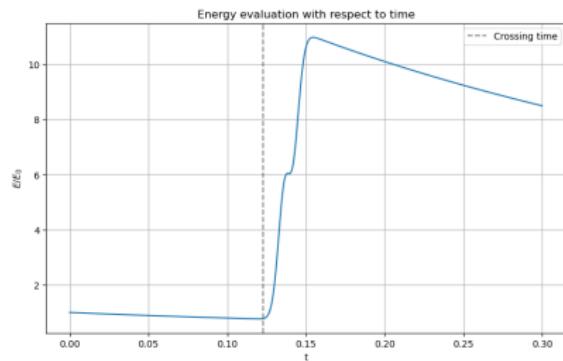
# Energy variation

Right after the wave crosses the interface, a clear **non-conservation of energy** emerges.

$$c_1 = 1 \quad c_2 = 9$$

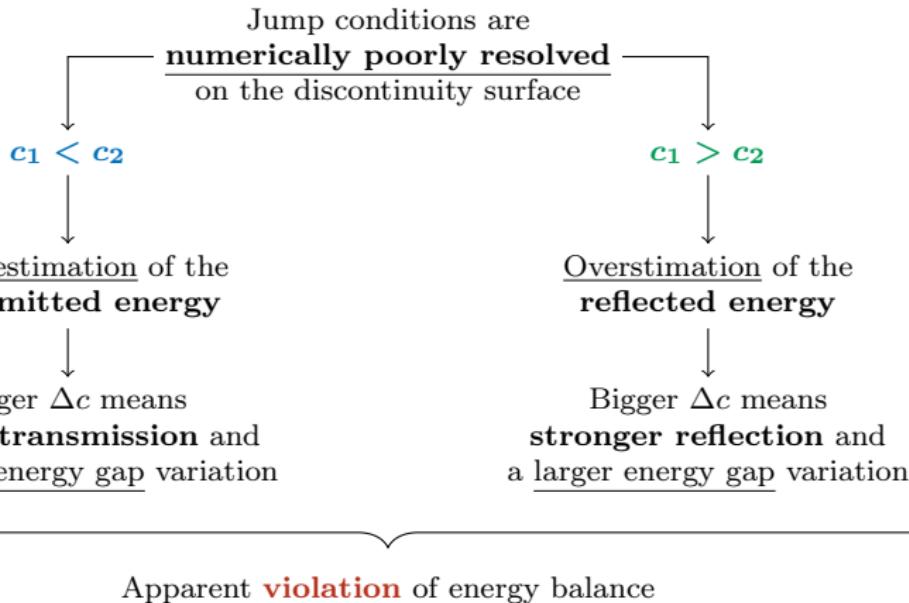


$$c_1 = 9 \quad c_2 = 1$$



# Energy variation

Right after the wave crosses the interface, a clear **non-conservation of energy** emerges.



# Approach to the Schrödinger equation

Moving in the quantum realm, our next goal is to approximate the solution of a **1D time-dependent Schrödinger equation**:

$$i\hbar \frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \psi}{\partial x^2} + V(x)\psi(x, t)$$

Three main differences from what we saw before:

- ① It appears a linear term.

We must introduce a **weighted mass matrix**:  $V : V_{i,j} = \int_{\Omega} V(x)\phi_j(x)dx$

- ② The PDE is of first order in time.

Crank-Nicholson method: