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 **D**ifferential **E**quations

- Euler Method
- Crank-Nicolson Method
- Trotter-Suzuki Formula

$\underbrace{\text{Euler Method} \quad \text{Crank-Nicolson Method}}_{\text{Runge-Kutta}}$

Partial **D**ifferential **E**quations

- **F**inite **D**ifference **M**ethod
- **F**inite **V**olume **M**ethod
- Galerkin methods
  - Spectral Method
  - **F**inite **E**lement **M**ethod

Numerical methods for differential equations

Ordinary Differential Equations

- Euler Method
- Crank-Nicolson Method
- Trotter-Suzuki Formula

$\underbrace{\text{Euler Method} \quad \text{Crank-Nicolson Method}}_{\text{Runge-Kutta}}$

Partial Differential Equations

- Finite Difference Method
- Finite Volume Method
- Galerkin methods
  - Spectral Method
  - Finite Element Method

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 Ω 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 Ω 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

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

where

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

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

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

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

where

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

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

Benefits of the weak formulation

Strong formulation	Weak formulation
$u \in C^2(\Omega)$	$u, v \in H^1(\Omega)^*$
Holds pointwise in Ω	Holds on average on Ω
Derivatives exist classically	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	**Weak formulation**
$u \in C^2(\Omega)$	$u, v \in H^1(\Omega)^*$
Holds pointwise in Ω	Holds on average on Ω
Derivatives exist classically	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.

In strong formulation

- **Dirichlet**: $u = g$ on $\partial\Omega$
- **Neumann**: $\frac{\partial u}{\partial n} = h$ on $\partial\Omega$

In weak formulation

Dirichlet becomes **essential condition**	Neumann becomes **natural condition**
$\boldsymbol{v} = \mathbf{0}$ on $\partial\Omega \Rightarrow$ cancels boundary term (no information available on $\frac{\partial u}{\partial n}$)	$\boldsymbol{v}$ **free** on $\partial\Omega$
$\boldsymbol{u} = \boldsymbol{g}$ enforced on $\partial\Omega$ (final solution)	$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \boldsymbol{h}$ naturally enters weak form

On boundary conditions

Another difference lies in the boundary condition prescription.

In strong formulation

- **Dirichlet**: $u = g \quad \text{on } \partial\Omega$
- **Neumann**: $\frac{\partial u}{\partial n} = h \quad \text{on } \partial\Omega$

In weak formulation

Dirichlet becomes **essential condition**	Neumann becomes **natural condition**
$\boldsymbol{v} = \mathbf{0}$ on $\partial\Omega \Rightarrow$ cancels boundary term (no information available on $\frac{\partial u}{\partial n}$)	$\boldsymbol{v}$ **free** on $\partial\Omega$
$\boldsymbol{u} = \boldsymbol{g}$ enforced on $\partial\Omega$ (final solution)	$\frac{\partial u}{\partial n} = \boldsymbol{h}$ naturally enters weak form

On boundary conditions

Another difference lies in the boundary condition prescription.

In strong formulation

- **Dirichlet**: $u = g \quad \text{on } \partial\Omega$
- **Neumann**: $\frac{\partial u}{\partial n} = h \quad \text{on } \partial\Omega$

In weak formulation

Dirichlet becomes **essential condition**	Neumann becomes **natural condition**
$\boldsymbol{v} = \mathbf{0}$ on $\partial\Omega \Rightarrow$ cancels boundary term (no information available on $\frac{\partial u}{\partial n}$)	$\boldsymbol{v}$ **free** on $\partial\Omega$
$\boldsymbol{u} = \boldsymbol{g}$ enforced on $\partial\Omega$ (final solution)	$\boldsymbol{\frac{\partial u}{\partial n}} = \boldsymbol{h}$ naturally enters weak form

Shape functions

Galerkin methods allow to find an approximate solution

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

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

$$\boxed{a\left(u_h, v_h\right) = F\left(v_h\right) \qquad \forall v_h \in V_h}$$

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

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

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

Shape functions

Galerkin methods allow to find an approximate solution

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

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

$$\boxed{a\left(u_h, v_h\right) = F\left(v_h\right) \qquad \forall v_h \in V_h}$$

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

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

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

Shape functions

Galerkin methods allow to find an approximate solution

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

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

$$\boxed{a\left(u_h, v_h\right) = F\left(v_h\right) \qquad \forall v_h \in V_h}$$

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

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

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

Final expression

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

$$\sum_{j=1}^{N} u_j a\left(\phi_j, \phi_i\right) = F\left(\phi_i\right) \qquad \forall i = 1, \ldots, N$$

$$\Downarrow$$

$$\boxed{A\boldsymbol{u} = \boldsymbol{F}}$$

where

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

$\boldsymbol{u} = \left(u_1, \ldots, u_N\right)^T$ is the **vector of unknowns**

$\boldsymbol{F} = \left(F\left(\phi_1\right), \ldots, F\left(\phi_N\right)\right)^T$ is the **load vector**

Final expression

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

$$\sum_{j=1}^{N} u_j a\left(\phi_j, \phi_i\right) = F\left(\phi_i\right) \qquad \forall i = 1, \ldots, N$$

$$\Downarrow$$

$$\boxed{A\boldsymbol{u} = \boldsymbol{F}}$$

where

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

$\boldsymbol{u} = \left(u_1, \ldots, u_N\right)^T$ is the **vector of unknowns**

$\boldsymbol{F} = \left(F\left(\phi_1\right), \ldots, F\left(\phi_N\right)\right)^T$ 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:

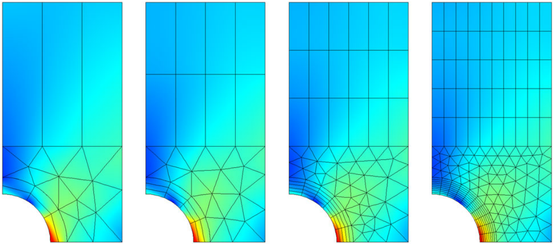
- Good approximation of **complex geometries**
- Better capture of **local effects**
- Possibility of **adaptive refinement**
- Natural construction of a **global solution**

Mesh discretization

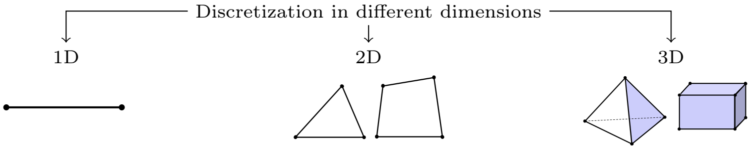
FEM approach consists in the subdivision of the domain in a so-called **mesh**

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**



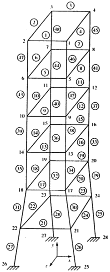
Elements



Elements

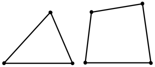
Discretization in different dimensions

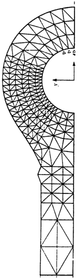
1D



Frame elements

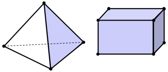
2D

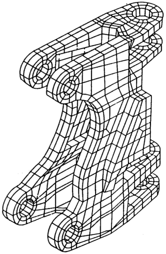




Triangular elements

3D

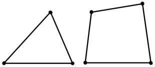
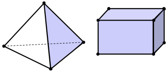
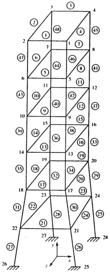
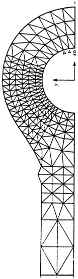
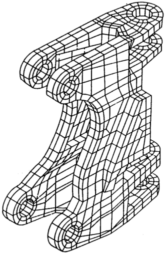




Brick elements

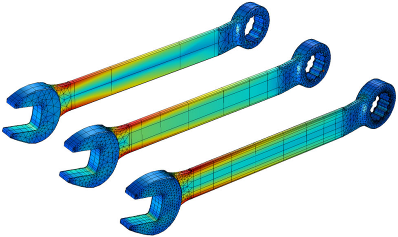
Elements

Discretization in different dimensions

1D	2D	3D
		
		
Frame elements	**Triangular elements**	**Brick elements**

Finite Element Method

Application examples

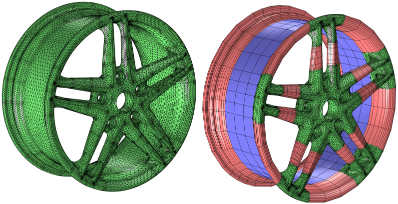


Manual mesh refinement of a wrench using different element types

Image from COMSOL Multiplysics Cyclopedia, ‘‘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 Multiplysics 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

$\downarrow$

Choice of a **local basis functions**

$\downarrow$

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

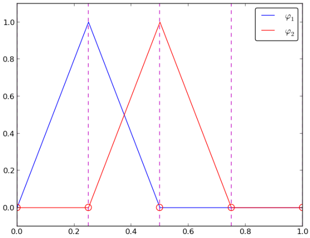
↓

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:

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:

`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:

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

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

$$\boxed{\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0} \longleftarrow \textbf{d'Alembert equation}$$

Separate discretization

Spatial part

Finite Element Method

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

Temporal part

Finite Difference Method

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

Approach to the classical wave equation

Our first goal is to approximate the solution of

$$\boxed{\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0} \longleftarrow \textbf{d'Alembert equation}$$

Separate discretization

Spatial part

Finite Element Method

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

Temporal part

Finite Difference Method

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

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

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

$$\Downarrow$$

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

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

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

$$\Downarrow$$

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

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

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

$$\Downarrow$$

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

Matrix formulation

Let's define

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

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

$$\Downarrow$$

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

Matrix formulation

Let's define

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

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

$$\Downarrow$$

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

Time discretization

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

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

$$\Downarrow$$

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

Time discretization

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

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

$$\Downarrow$$

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

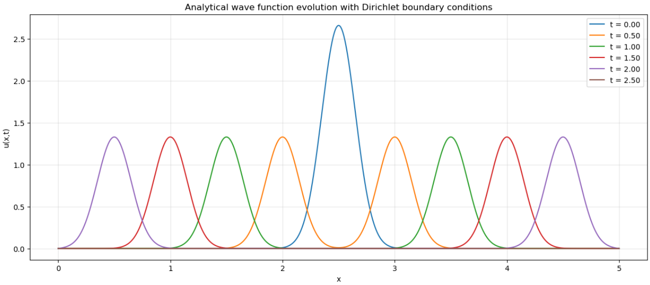
Analytical solutions

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

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

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

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



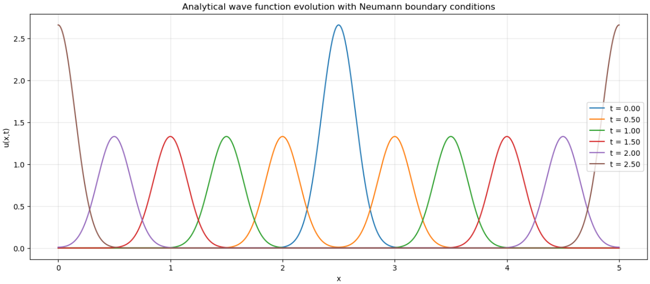
Analytical solutions

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

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

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

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