

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

Dario Liotta



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



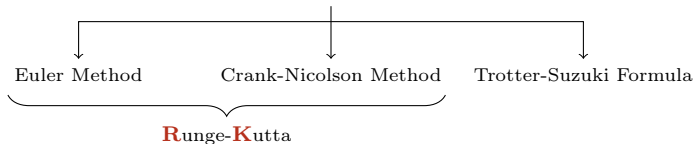
Dipartimento
di Fisica
e Astronomia
Galileo Galilei

September 6th 2025

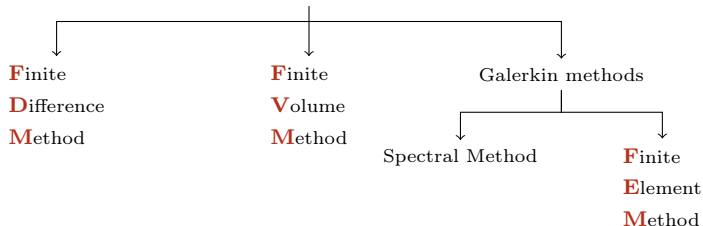
Course of **Quantum Information and Computing**
Academic Year 2024/2025

Numerical methods for differential equations

Ordinary Differential Equations

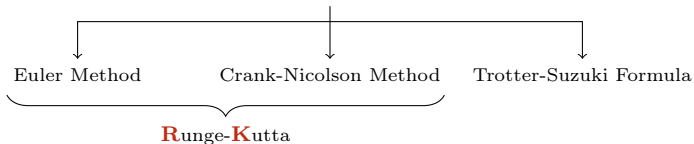


Partial Differential Equations

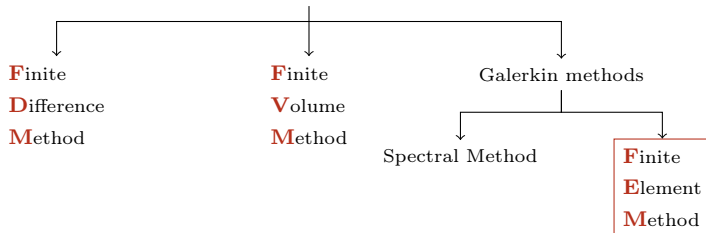


Numerical methods for differential equations

Ordinary Differential Equations



Partial Differential Equations



Introduction to the problem

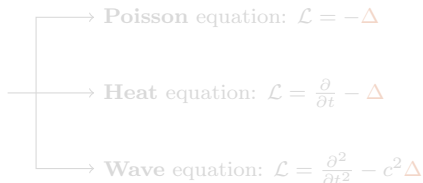
Solving a **PDE** means to find a function u such that

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

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

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

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



Introduction to the problem

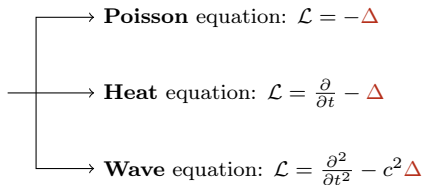
Solving a **PDE** means to find a function u such that

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

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



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

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

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

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

where

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

Benefits of the weak formulation

Strong formulation

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

Holds pointwise in Ω

Derivatives exist classically

Weak formulation

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

Holds on average on Ω

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 Ω

Derivatives exist classically

Weak formulation

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

Holds on average on Ω

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 $\frac{\partial u}{\partial n}$)

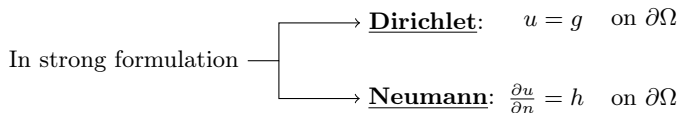
$u = g$ enforced on $\partial\Omega$ (final solution)

v free on $\partial\Omega$

$\frac{\partial u}{\partial n} = h$ 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 $\frac{\partial u}{\partial n}$)

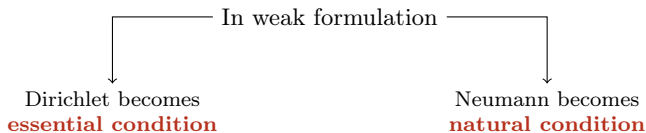
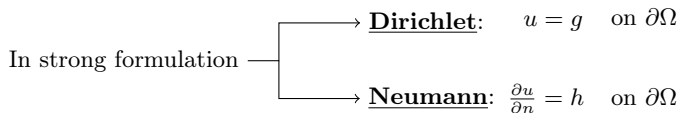
$u = g$ enforced on $\partial\Omega$ (final solution)

v free on $\partial\Omega$

$\frac{\partial u}{\partial n} = h$ 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 $\frac{\partial u}{\partial n}$)

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

v free on $\partial\Omega$

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

Shape functions

Galerkin methods allow to find an approximate solution

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

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

Shape functions

Galerkin methods allow to find an approximate solution

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

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

Shape functions

Galerkin methods allow to find an approximate solution

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

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

Final expression

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

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



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

where

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

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

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

form the **stiffness matrix**

is the **vector of unknowns**

is the **load vector**

Final expression

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

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



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

where

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

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

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

form the **stiffness matrix**

is the **vector of unknowns**

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:

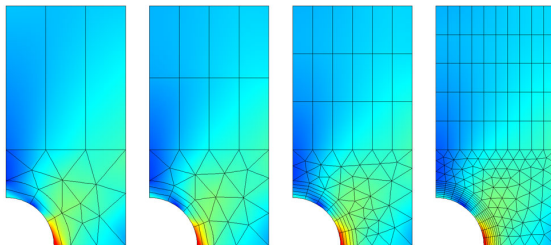
- Good approximation of **complex geometries**
- Better capture of **local effects**
- Possibility of **adaptive refinement**
- Natural construction of a **global solution**

Mesh discretization

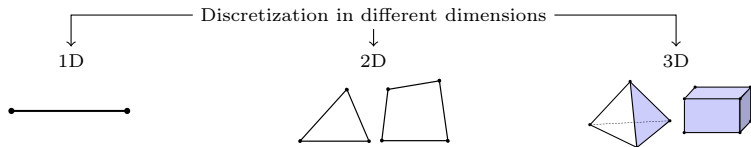
FEM approach consists in the subdivision of the domain in a so-called **mesh**

This choice brings several advantages:

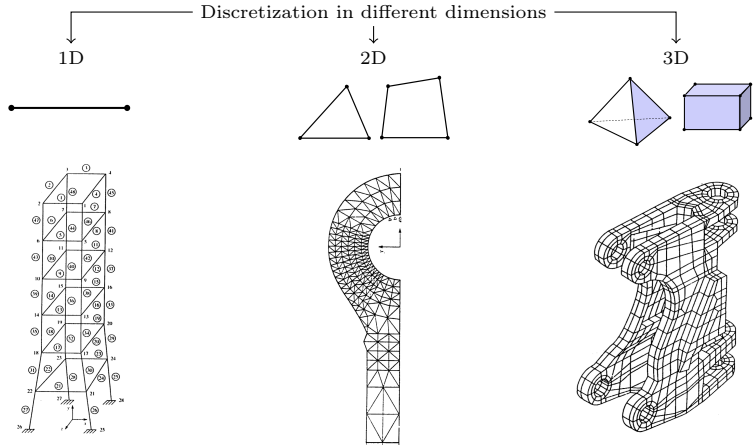
- Good approximation of **complex geometries**
- Better capture of **local effects**
- Possibility of **adaptive refinement**
- Natural construction of a **global solution**



Elements



Elements

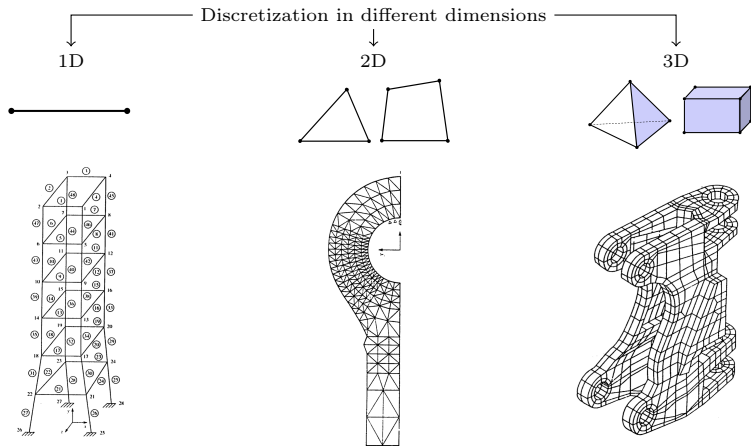


Frame elements

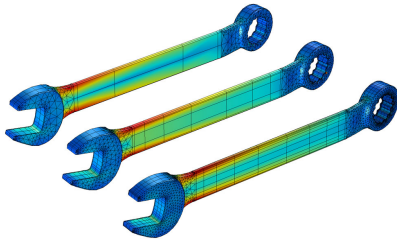
Triangular elements

Brick elements

Elements

Frame **elements**Triangular **elements**Brick **elements**Finite **Element** Method

Application examples

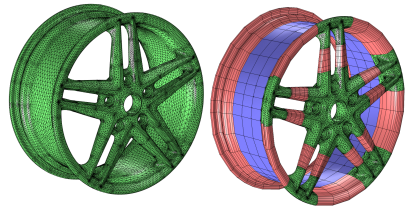


Manual mesh refinement of a wrench using different element types

Image from COMSOL Multiphysics 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 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

Choice of the base

Mesh division into sub-domains



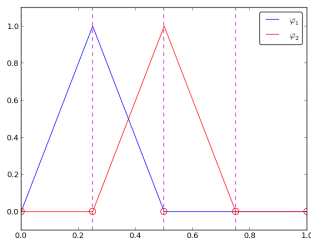
Choice of a **local basis functions**



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
PROJECT

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
PROJECT

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

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

Equation is separable

Spatial part

Temporal part

Finite Element Method

Finite Difference Method

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

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

Approach to the classical wave equation

Our first goal is to approximate the solution of

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

