IN5270/IN9270 Summary

Xing Cai

Course content (from the official course description)

The course provides a thorough introduction to design, analysis (both theoretical and empirical), and programming of difference and elemental methods to solve differential equations.

In addition, the subject also includes verification and software testing for these numerical methods.

Curriculum

- Stand-alone module: **Finite Difference Computing with Exponential Decay Models**
- Testbook 1: **Finite Difference Computing with PDEs**
  - Chapters 1, 2 & 5 (full overlap with Chapter 10 in Textbook 2)
- Textbook 2: **Introduction to Numerical Methods for Variational Problems**
  - Chapters 3, 4, 5, 6, 7 & 10
    (**Note:** The numbering of the chapters follows the bookmanuscript available online)

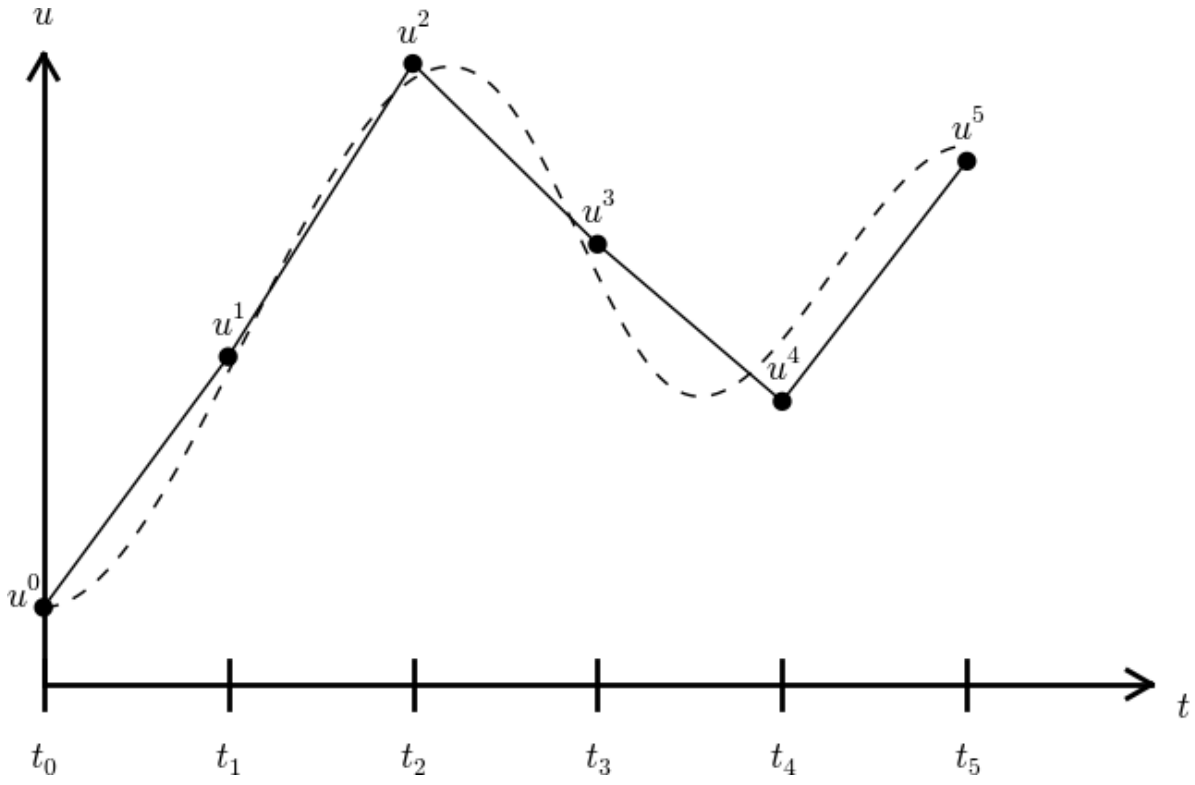
Please read the semester webpage for more info about the most important parts in the different chapters.

Expected learning outcome

- You'll know some of the most common differential equations.
- You'll have mastered the basic steps in constructing and applying the difference and element methods to simple representative examples of differential equations.
- You'll have good knowledge of programming techniques for implementing difference and element methods in simple 1D cases and for the use of selected software in simple 2D and 3D cases.
- You'll have basic knowledge of theoretical and empirical analysis of the difference and elemental methods for accuracy and stability.
- You'll have good knowledge of verification and software testing of the difference and elemental methods.

Finite difference methods

- The finite difference method is the simplest method for solving differential equations
- A very useful tool to know, even if you mainly aim at using the finite element



Finite difference methods

Key concepts:

- mesh
- mesh function
- finite difference approximations

The steps in the finite difference method:

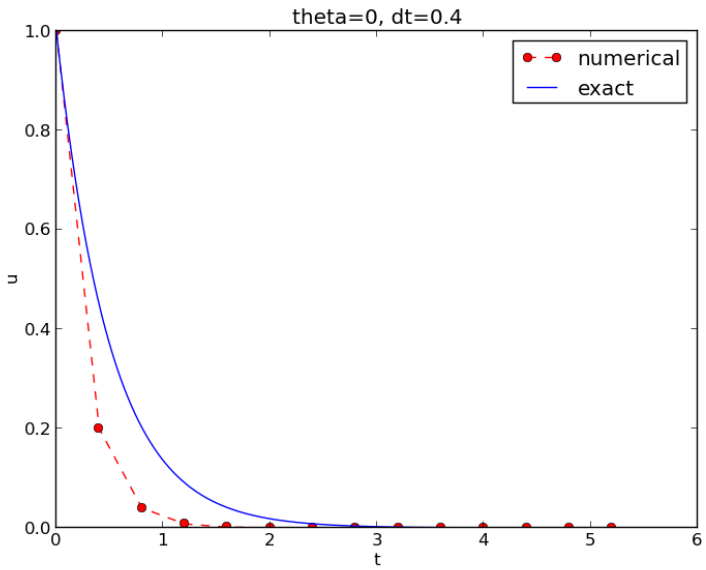
1. discretizing the domain,
2. fulfilling the equation at discrete time points,
3. replacing derivatives by finite differences,
4. formulating a recursive algorithm ***or a system of (non)linear algebraic equations***

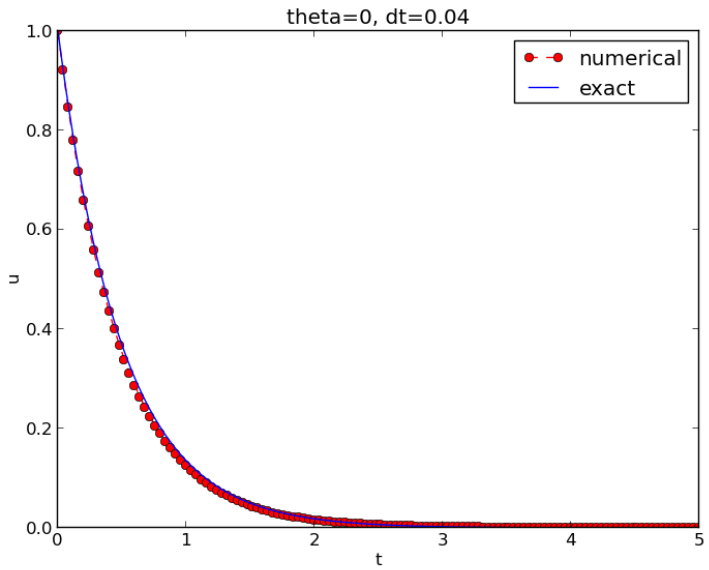
FDM for a basic model for exponential decay

The world's simplest ordinary differential equation (ODE):

$$u'(t) = -au(t), \quad u(0) = I, \ t \in (0, T]$$

Although we know the exact solution of this mathematical model, it serves as an illustrating example of FDM.





The unifying θ-rule (for the finite differencing schemes)

The Forward Euler, Backward Euler, and Crank-Nicolson schemes can be formulated as one scheme with a varying parameter θ:

$$\frac{u^{n+1} - u^n}{t_{n+1} - t_n} = -a(\theta u^{n+1} + (1-\theta)u^n) \tag{1}$$

- $\theta = 0$: Forward Euler
- $\theta = 1$: Backward Euler
- $\theta = 1/2$: Crank-Nicolson

For this ODE, we can recursively solve for u^{n+1} when u^n is known:

$$u^{n+1} = \frac{1 - (1-\theta)a(t_{n+1} - t_n)}{1 + \theta a(t_{n+1} - t_n)} u^n \tag{2}$$

Verifying an implementation

How can you ensure that an implementation of a numerical method is correct?

- Verification = bring evidence that the program works
- Find suitable test problems
  - Often very useful: *The method of manufactured solutions*

Computing the norm of the error

- The numerical error e^n (known at different time levels t_n) is a mesh function
- Usually we want one number to quantify the error
- Compute approximately a norm of the numerical error

Recall: Norms of a function $f(t)$ defined over the entire domain $t \in [0, T]$:

$$||f||_{L^2} = \left(\int_0^T f(t)^2 dt \right)^{1/2} \tag{3}$$

$$||f||_{L^1} = \int_0^T |f(t)| dt \tag{4}$$

$$||f||_{L^\infty} = \max_{t \in [0,T]} |f(t)| \tag{5}$$

Norms of mesh functions

- Problem: $e^n = e(t_n)$ is a mesh function and hence not defined for all t. How to integrate e which is only known at discrete time levels?
- Idea: Apply a numerical integration rule, using only the mesh points of the mesh function.

The trapezoidal rule for approximately computing L_2 norm of e:

$$||e||_{L^2} \approx \left(\Delta t \left(\frac{1}{2}(e^0)^2 + \frac{1}{2}(e^{N_t})^2 + \sum_{n=1}^{N_t-1} (e^n)^2 \right) \right)^{1/2}$$

Common simplification yields the L^2 norm of a mesh function:

$$||e||_{L^2} \approx \left(\Delta t \sum_{n=0}^{N_t} (e^n)^2 \right)^{1/2}$$

Even a correct implementation sometimes may not work!

For the ODE model of exponential decay, we can observe that the Forward Euler and Crank-Nicolson schemes may produce (growing) oscillating solutions, which don't agree with the monotonly decaying exact solution.

We ask the question (about stability):

- Under what circumstances, i.e., values of the input data I, a, and Δt, will the Forward Euler and Crank-Nicolson schemes result in undesired oscillatory solutions?

Another question (about accuracy) to be raised is

- How does Δt impact the error in the numerical solution?

Details can be found in the stand-alone module.

FDM for the simple ODE model of vibration

$$u''(t) + \omega^2 u = 0, \quad u(0) = I, \ u'(0) = 0, \ t \in (0, T]$$

Again, although we know the exact solution $u_{\mathrm{e}}(t) = I\cos(\omega t)$, this ODE model is used as another illustrating example of FDM.

The resulting discrete equation at $t = t_n$:

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + \omega^2 u^n = 0$$

This will give a recursive algorithm for computing u^{n+1} when u^n and u^{n-1} are known.

Computing the first step

Discretize $u'(0) = 0$ by a centered difference

$$\frac{u^1 - u^{-1}}{2\Delta t} = 0 \quad \Rightarrow \quad u^{-1} = u^1$$

Inserted in the scheme for $n = 0$ gives

$$u^1 = u^0 - \frac{1}{2}\Delta t^2 \omega^2 u^0$$

Analysis: deriving an exact solution of the discrete equations

- We have a linear, homogeneous, difference equation for u^n.
- Has solutions $u^n \sim IA^n$, where A is unknown (number).
- Trick for simplifying the algebra: $u^n = IA^n$, with $A = \exp(i\tilde{\omega}\Delta t)$, then find $\tilde{\omega}$
- $\tilde{\omega}$: unknown *numerical frequency* (easier to calculate than A)
- $\omega - \tilde{\omega}$ is the angular *frequency error*
- Use the real part as the physical relevant part of a complex expression

Expected exact solution of the discrete equations

$$u^n = IA^n = I\exp\left(i\tilde{\omega}\Delta t\, n\right) = I\exp\left(i\tilde{\omega}t\right) = I\cos(\tilde{\omega}t) + iI\sin(\tilde{\omega}t)$$

$$\begin{aligned}
\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} &= I\frac{A^{n+1} - 2A^n + A^{n-1}}{\Delta t^2} \\
&= I\frac{\exp\left(i\tilde{\omega}(t+\Delta t)\right) - 2\exp\left(i\tilde{\omega}t\right) + \exp\left(i\tilde{\omega}(t-\Delta t)\right)}{\Delta t^2} \\
&= I\exp\left(i\tilde{\omega}t\right)\frac{1}{\Delta t^2}\left(\exp\left(i\tilde{\omega}(\Delta t)\right) + \exp\left(i\tilde{\omega}(-\Delta t)\right) - 2\right) \\
&= I\exp\left(i\tilde{\omega}t\right)\frac{2}{\Delta t^2}\left(\cos(\tilde{\omega}\Delta t) - 1\right) \\
&= -I\exp\left(i\tilde{\omega}t\right)\frac{4}{\Delta t^2}\sin^2\left(\frac{\tilde{\omega}\Delta t}{2}\right)
\end{aligned}$$

Solving for the numerical frequency

The discrete equation with $u^n = I\exp\left(i\tilde{\omega}\Delta t\, n\right) = I\exp\left(i\tilde{\omega}t\right)$ inserted gives

$$-I\exp\left(i\tilde{\omega}t\right)\frac{4}{\Delta t^2}\sin^2\left(\frac{\tilde{\omega}\Delta t}{2}\right) + \omega^2 I\exp\left(i\tilde{\omega}t\right) = 0$$

which after dividing by $I\exp\left(i\tilde{\omega}t\right)$ results in

$$\frac{4}{\Delta t^2}\sin^2\left(\frac{\tilde{\omega}\Delta t}{2}\right) = \omega^2$$

Solve for $\tilde{\omega}$:

$$\tilde{\omega} = \pm\frac{2}{\Delta t}\sin^{-1}\left(\frac{\omega\Delta t}{2}\right)$$

We can thus find the accuracy of $\tilde{\omega}$ and the stability requirement, both dependent on Δt. (Read Section 1.4 of Textbook 1 for details.)

FDM for a simple wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \qquad x \in (0, L),\ t \in (0, T] \tag{6}$$

$$u(x, 0) = I(x), \qquad x \in [0, L] \tag{7}$$

$$\frac{\partial}{\partial t} u(x, 0) = 0, \qquad x \in [0, L] \tag{8}$$

$$u(0, t) = 0, \qquad t \in (0, T] \tag{9}$$

$$u(L, t) = 0, \qquad t \in (0, T] \tag{10}$$

Standard finite differencing:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}, \tag{11}$$

also need to handle initial and boundary conditions

Analysis of the difference equations

Mathematical analysis can be used to study both the **accuracy** and **stability** of the finite difference scheme.

Details can be found in Section 2.10 of Textbook 1.

Combining prescribed basis functions for approximation

General idea of finding an approximation $u(x)$ to some given $f(x)$ on a domain Ω:

$$u(x) = \sum_{i=0}^{N} c_i \psi_i(x)$$

where

- $\psi_i(x)$ are prescribed functions that span *function space* V
- c_i, $i = 0, \ldots, N$, are unknown coefficients to be determined

There are three approaches:

- The least squares method
- The projection (or Galerkin) method (**main focus**)
- The interpolation (or collocation) method

The projection (or Galerkin) method

Minimizing the error $e(x) = f(x) - u(x)$, which is known on every point in the domain Ω, is achieved by requiring

$$(e, \psi_i) \equiv \int_\Omega e(x)\psi_i(x)dx = 0, \quad i \in \mathcal{I}_s$$

which is equivalent to

$$(e, v) = 0, \quad \forall v \in V$$

Using $v = \psi_i$, for $i = 0, 1, \ldots, N$, we will get a linear system

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \ i \in \mathcal{I}_s, \quad A_{i,j} = (\psi_i, \psi_j), \ b_i = (f, \psi_i)$$

The collocation or interpolation method

Another idea for approximating $f(x)$ by $u(x) = \sum_j c_j \psi_j$:

- Force $u(x_i) = f(x_i)$ at some selected *collocation* points $\{x_i\}_{i \in \mathcal{I}_s}$
- Then u is said to *interpolate* f
- The method is known as *interpolation* or *collocation*

$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x_i) = f(x_i) \quad i \in \mathcal{I}_s$$

This will result in a linear system with no need for integration:

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i \in \mathcal{I}_s \tag{12}$$

$$A_{i,j} = \psi_j(x_i) \tag{13}$$

$$b_i = f(x_i) \tag{14}$$

Lagrange polynomials

The *Lagrange interpolating polynomials* ψ_j are defined as

$$\psi_i(x) = \prod_{j=0, j\neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N}$$

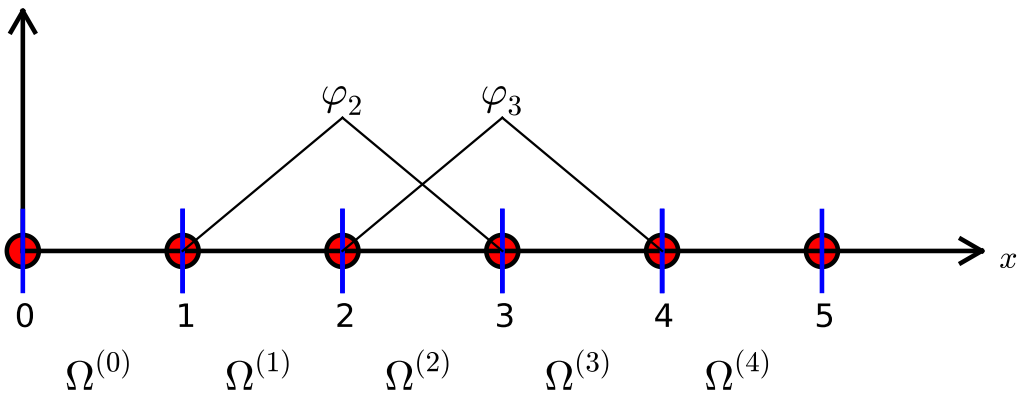
They have the property that

$$\psi_i(x_j) = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

- Lagrange polynomials are thus very convenient for the interpolation method.
- Moreover, Lagrange polynomials are heavily used to define finite element basis functions.

FEM uses basis functions with local support

- *Local support*: $\psi_i(x) \neq 0$ for x only in a small subdomain of Ω
- $u(x)$ constructed by such "locally-supported" basis functions is a piecewise polynomial defined over many (small) subdomains
- We introduce φ_i as the name of these finite element functions



Elements and nodes

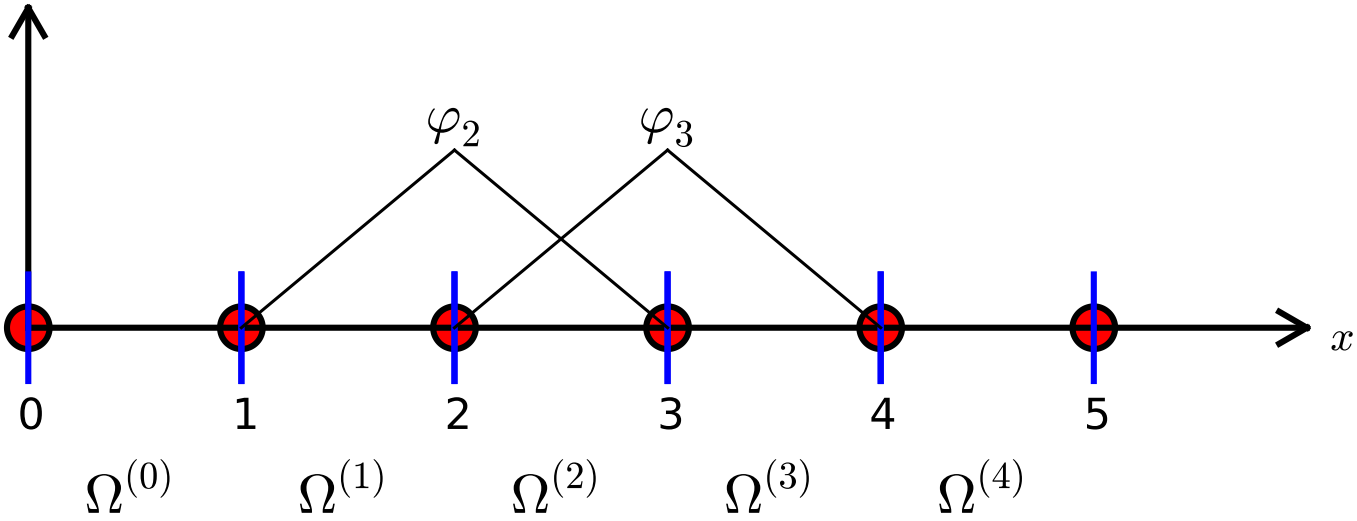
Split Ω into N_e non-overlapping subdomains called *elements*:

$$\Omega = \Omega^{(0)} \cup \cdots \cup \Omega^{(N_e-1)}$$

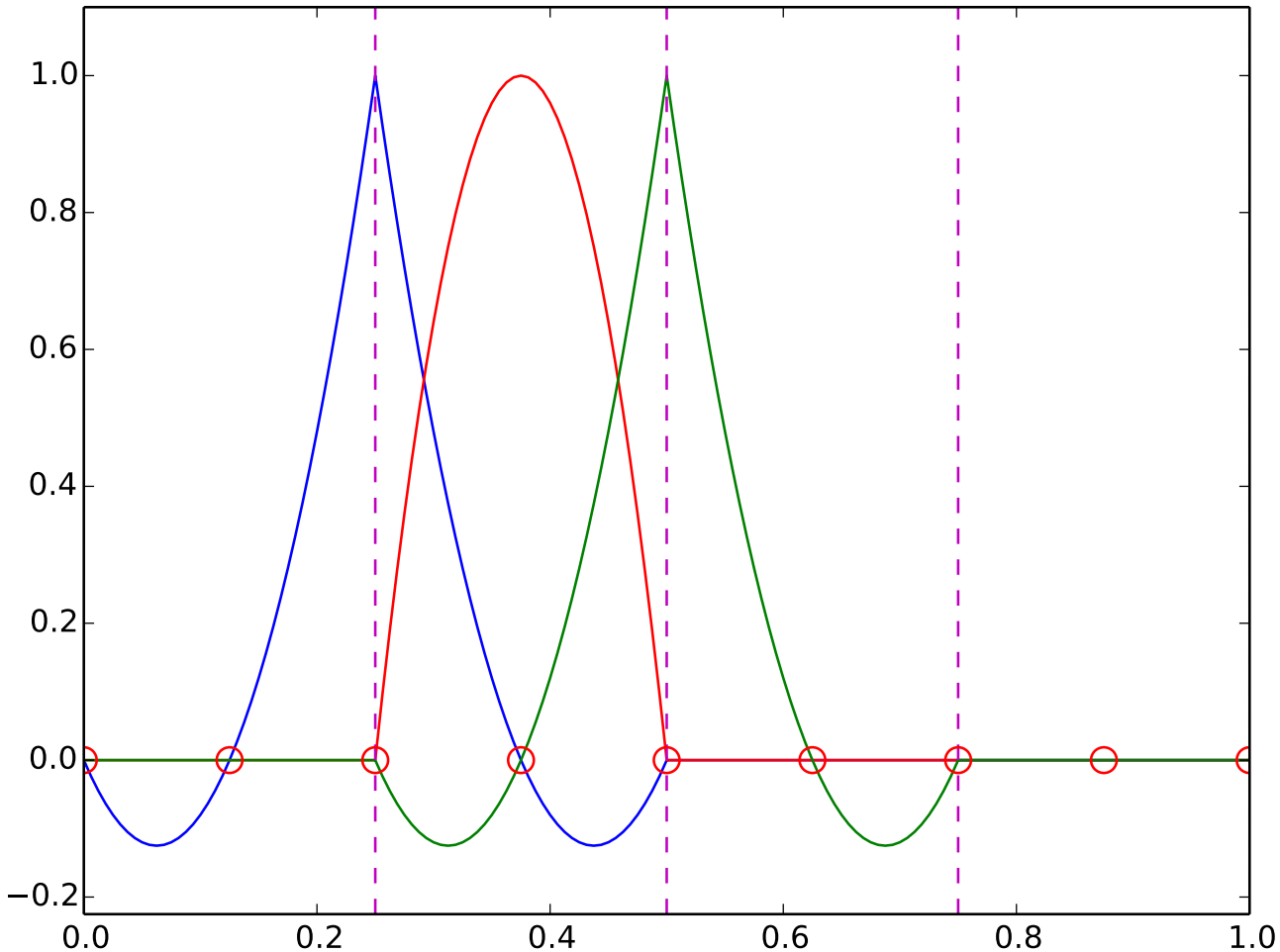
On each element, use a certain number of points called *nodes*:

- The finite element basis functions are named $\varphi_i(x)$
- $\varphi_i = 1$ at node i and 0 at all other nodes
- φ_i is a Lagrange polynomial on each element
- For nodes at the boundary between two elements, φ_i is made up of a Lagrange polynomial over each element

Example on elements with two nodes (P1 elements)



Example on elements with three nodes (P2 elements)



Interpretation of the coefficients c_i

Important property: c_i is the value of $u(x)$ at node i, with coordinate x_i:

$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x_i) = c_i \varphi_i(x_i) = c_i$$

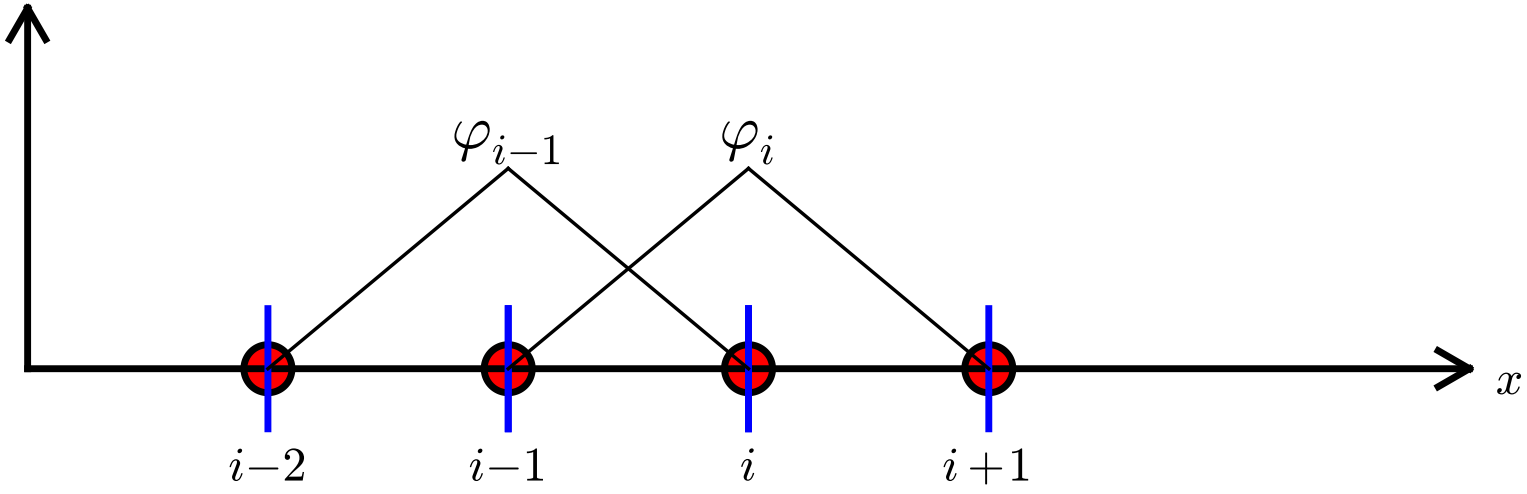
This is because $\varphi_j(x_i) = 0$ if $i \neq j$ and $\varphi_i(x_i) = 1$.

Note: It is possible to evaluate $u(x) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x)$ for any x inside the domain.

Properties of the basis functions

- $\varphi_i(x) \neq 0$ only on those elements that contain global node i
- $\varphi_i(x)\varphi_j(x) \neq 0$ if and only if i and j are global node numbers in the same element

Since $A_{i,j} = \int \varphi_i(x)\varphi_j(x)\,\mathrm{d}x$, *most of the elements in the coefficient matrix will be zero*



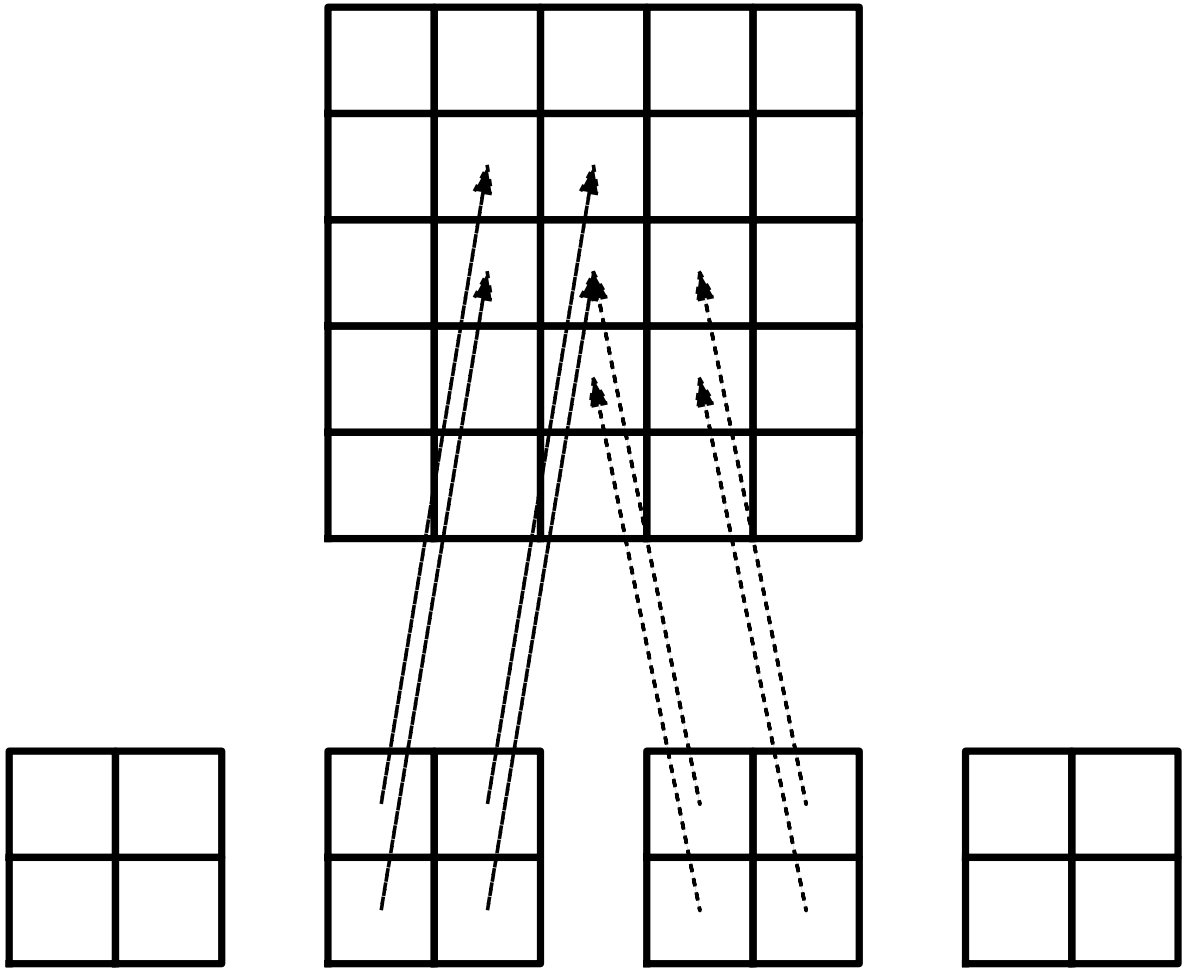
Split the integrals into elementwise integrals

$$A_{i,j} = \int_\Omega \varphi_i \varphi_j dx = \sum_e \int_{\Omega^{(e)}} \varphi_i \varphi_j dx, \quad A^{(e)}_{i,j} = \int_{\Omega^{(e)}} \varphi_i \varphi_j dx$$

Important observations:

- $A^{(e)}_{i,j} \neq 0$ if and only if i and j are nodes in element e (otherwise no overlap between the basis functions)
- All the nonzero entries in $A^{(e)}_{i,j}$ can be collected in a "condensed" *element matrix*, $\tilde{A}^{(e)}_{r,s}$, whose number of rows and columns equals the number of nodes in element e
- The element matrix has contributions from the φ_i functions associated with the nodes in element
- It is convenient to introduce a *local numbering* of the nodes in an element: $r, s = 0, 1, \ldots, d$

Illustration of the matrix assembly: regularly numbered P1 elements



Mapping to a reference element

Instead of computing

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx$$

we now map $\Omega^{(e)} = [x_L, x_R]$ to a standardized reference element domain $[-1, 1]$ with local coordinate X

$$x = \frac{1}{2}(x_L + x_R) + \frac{1}{2}(x_R - x_L)X$$

Integral transformation

Reference element integration: just change integration variable from x to X. Introduce local basis function

$$\tilde{\varphi}_r(X) = \varphi_{q(e,r)}(x(X))$$

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X) \underbrace{\frac{dx}{dX}}_{\det J = h/2} dX = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X) \det J \, dX$$

$$\tilde{b}^{(e)}_r = \int_{\Omega^{(e)}} f(x)\varphi_{q(e,r)}(x)dx = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X) \det J \, dX$$

Advantages of the reference element

- Always the same domain for integration: $[-1, 1]$ in 1D cases
- We only need formulas for $\tilde{\varphi}_r(X)$ over one element (no need for *piecewise* polynomial definition)
- $\tilde{\varphi}_r(X)$ is the same for all elements: no dependence on element length and location, which is "factored out" in the mapping and $\det J$

Standardized basis functions for P1 elements

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X) \tag{15}$$

$$\tilde{\varphi}_1(X) = \frac{1}{2}(1 + X) \tag{16}$$

Note: simple polynomial expressions (no need to consider piecewisely defined functions)

Standardized basis functions for P2 elements

$$\tilde{\varphi}_0(X) = \frac{1}{2}(X-1)X \tag{17}$$

$$\tilde{\varphi}_1(X) = 1 - X^2 \tag{18}$$

$$\tilde{\varphi}_2(X) = \frac{1}{2}(X+1)X \tag{19}$$

Easy to generalize to arbitrary order!

Variational form of PDE; notation

- $u_e(x)$ is the *exact* solution of $\mathcal{L}(u_e) = 0$ and $\mathcal{B}_i(u_e) = 0$
- $u(x)$ denotes an *approximate* solution
- $V = \text{span}\{\psi_0(x), \ldots, \psi_N(x)\}$, V has basis $\{\psi_i\}_{i \in \mathcal{I}_s}$
- We seek $u \in V$
- $\mathcal{I}_s = \{0, \ldots, N\}$ is an index set
- $u(x) = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$
- Inner product: $(u, v) = \int_\Omega uv \, \mathrm{d}x$
- Norm: $||u|| = \sqrt{(u, u)}$

Residual-minimizing principles

- When solving a PDE of the form $\mathcal{L}(u_e) = 0$, we do not know u_e and thus cannot work directly with the error $e = u_e - u$
- We know, however, the *error in the equation*: the residual R

Inserting $u = \sum_j c_j \psi_j(x)$ in $\mathcal{L} = 0$ gives a residual R, which is a function of x inside Ω

$$\mathcal{L}(u) = \mathcal{L}\left(\sum_j c_j \psi_j(x)\right) = R(x) \neq 0$$

Goal: minimize R with respect to $\{c_i\}_{i \in \mathcal{I}_s}$ (and hope it makes a small e too)

$$R = R(c_0, \ldots, c_N; x)$$

The Galerkin method

Idea: make R orthogonal to V,

$$(R, v) = 0, \quad \forall v \in V$$

This implies

$$(R, \psi_i) = 0, \quad i \in \mathcal{I}_s$$

$N + 1$ equations for $N + 1$ unknowns $\{c_i\}_{i \in \mathcal{I}_s}$

Terminology: Test and trial functions

- ψ_j used in $\sum_j c_j \psi_j$ is called *trial function*
- ψ_i that is used as weight in Galerkin's method is called *test function*

Boundary conditions

- u known: Dirichlet boundary condition
- u' known: Neumann boundary condition
- Must have $\psi_i = 0$ where Dirichlet conditions apply

Integration by parts has many advantages

Second-order derivatives should be integrated by parts, for example,

$$\int_0^L u''(x)v(x)dx = -\int_0^L u'(x)v'(x)dx + [vu']_0^L$$
$$= -\int_0^L u'(x)v'(x)dx + u'(L)v(L) - u'(0)v(0)$$

Motivation:

- Lowers the order of derivatives
- Gives more symmetric forms (incl. matrices)
- Enables easy handling of Neumann boundary conditions
- Finite element basis functions φ_i have discontinuous derivatives (at cell boundaries) and are not suited for terms with φ_i''

We use a boundary function to deal with non-zero Dirichlet boundary conditions

- What about nonzero Dirichlet conditions? Say $u(L) = D$
- We always require $\psi_i(L) = 0$ (i.e., $\psi_i = 0$ where Dirichlet conditions applies)
- Problem: $u(L) = \sum_j c_j \psi_j(L) = \sum_j c_j \cdot 0 = 0 \neq D$ - always!
- Solution: $u(x) = B(x) + \sum_j c_j \psi_j(x)$
- $B(x)$: user-constructed boundary function that fulfills the Dirichlet conditions
- If $u(L) = D$, make sure $B(L) = D$
- No restrictions of how $B(x)$ varies in the interior of Ω

The resulting linear system

Example:

$$-u'' = f, \quad u'(0) = C, \; u(L) = D, \quad u = D + \sum_j c_j \psi_j$$

Variational formulation:

$$\int_\Omega u' v' dx = \int_\Omega f v dx - v(0)C \quad \forall v \in V$$

Resulting linear system $\mathbf{Ac} = \mathbf{b}$

$$\mathbf{A} = (A_{ij}), \quad A_{ij} = \int_\Omega \psi_j'(x)\psi_i'(x)dx$$

$$\mathbf{b} = (b_i), \quad b_i = \int_\Omega f(x)\psi_i(x)dx - \psi_i(0)C$$

Computing with finite elements

Finite element basis function φ_j replaces ψ_j.

How to deal with the boundary conditions? We must enforce $\varphi_i = 0$ at the Dirichlet part of the boundary.

Example: $u(0) = 0$ and $u(L) = 0$. We just exclude φ_0 and φ_{N_n-1} and work with

$$u = \sum_{j \in \mathcal{I}_s} c_j \varphi_{\nu(j)}, \quad \nu(j) = j + 1$$

General construction of a boundary function

In case of *nonzero Dirichlet conditions*, with finite element basis functions, φ_i, $B(x)$ can be constructed in a completely general way.

Define

- I_b: set of indices with nodes where u is known
- U_i: Dirichlet value of u at node i, $i \in I_b$

The general formula for B is now

$$B(x) = \sum_{j \in I_b} U_j \varphi_j(x)$$

The approximation $u(x)$ is thus of the form:

$$u = \sum_{j \in \mathcal{I}_s} c_j \varphi_{\nu(j)} + \sum_{j \in I_b} U_j \varphi_j$$

Cellwise computation & reference element

Cellwise computation: Each elements computes an element matrix and vector. These are later assembled to the global linear system.

Use of a reference element ($X \in [-1, 1]$) is based on a mapping from the physical coordinate x to the reference local coordinate X.

Please refer to Section 6.1.4 in the manuscript of Textbook 2.

Variational form for time-dependent PDEs

- First, use finite differencing to discretize the time derivative(s).
- Then, treat the resulting time discrete equation as a stationary PDE at each time level, to be solved by the finite element method in space

Important details can be found in Chapter 7 in the manuscript of Textbook 2.

Solving nonlinear ODE and PDE problems

This is the last topic taught in this course, so the students are expected to have a rather fresh memory of the important contents.

- Nonlinear ODE (system) or time-dependent nonlinear PDE
  - Explicit time integration – one way of linearization
  - Implicit time integration applied to nonlinear ODE (system) – a (system of) nonlinear algebraic equation(s) per time step
  - Implicit time integration applied to time-dependent nonlinear PDE – a stationary nonlinear PDE per time step
- To solve a stationary nonlinear PDE
  - FDM or FEM for spatial discretization – giving rise to a system of nonlinear algebraic equations
  - Another approach: linearization at the differential equation level
- Newton's method is the most general approach to solving nonlinear algebraic equations
- Picard iterations are in some cases easier to apply

Important details can be found in Chapter 10 in the manuscript of Textbook 2.