A Short Introduction to Finite Element Analysis*

Hanfeng Zhai†

Department of Mechanical Engineering,
Stanford University

April 5, 2025

1 Synopsis

Finite Element Analysis (FEA) is a numerical technique for finding approximate solutions to boundary-value problems for partial differential equations (PDEs). It is one of the mostly used techniques in computational engineering (especially in computational solid mechanics), used to simulate physical phenomena across many fields. By breaking down complex structures or continua into smaller *finite elements* and constructing approximate solutions piecewise, FEA enables engineers to analyze stress distributions in mechanical parts, heat transfer in solids, fluid flow, electromagnetic fields, and more. It is especially useful for problems with complex geometries or heterogeneous materials, where analytical solutions are intractable. FEA allows for the mesh to vary in resolution to capture important details and helps reduce the need for physical prototypes through predictive simulation.

Contents

2 Mathematical Formulation of FEA

The FEA procedure starts from a well-posed **strong form** of a boundary-value problem (BVP). It progresses to a **weak form**[1], which is then discretized via the **Galerkin** finite element method. This general approach applies in one, two, or three spatial dimensions [1].

*This is a short note (summary) accompanying the graduate-level course `ME335A` being taught at Stanford University by Prof. Adrian J. Lew. The note is constantly being updated. Please e-mail the author if you find any mistakes or questions.

†E-mail: hzhai@stanford.edu

[1]variational formulation

For time-dependent partial differential equations, the temporal domain is discretized separately using finite difference schemes, implicit or explicit time-stepping methods, or other appropriate techniques. This temporal discretization converts the continuous time derivative into a difference quotient, resulting in a sequence of spatial problems that can be solved using the finite element method at each discrete time level.

2.1 Strong Form

In the strong form[2], one specifies the governing PDE throughout the domain and the boundary conditions that the solution u must satisfy on the domain boundaries. These conditions impose "strong" requirements on u (for example, requiring u to be sufficiently smooth to have the necessary derivatives).

For example, consider a simple second-order elliptic PDE (Poisson's equation) for an unknown field u:

$$\text{(1D)} \quad -\frac{d^2u}{dx^2} = f(x)$$

$$\text{(2D)} \quad -\nabla^2 u(x,y) = f(x,y)$$

$$\text{(3D)} \quad -\nabla^2 u(x,y,z) = f(x,y,z)$$

defined on a domain Ω (for example, an interval in 1D, or a region in $\mathbb{R}^2$ or $\mathbb{R}^3$). Here f is a given source term. To complete the strong form, boundary conditions are specified: for instance, one may prescribe u on a portion of the boundary (Dirichlet conditions) and prescribe the normal derivative on the remaining portion (Neumann conditions). For the Poisson example, the strong form can be stated as

$$-\Delta u(x) = f(x) \quad \text{for } x \in \Omega;$$

$$u(x) = u_D(x) \quad \text{on } \Gamma_D;$$

$$\nabla u(x) \cdot n = t_N(x) \quad \text{on } \Gamma_N,$$

where Δ is the Laplace operator, n is the outward normal on the boundary, u_D is a specified value on the Dirichlet boundary, and t_N is a specified flux on the Neumann boundary. This differential statement (PDE and boundary conditions) constitutes the strong form of the problem.

2.2 Weak Form

Solving the strong form directly requires finding a function $u(x)$ that satisfies the PDE and all boundary conditions exactly. Instead, we reformulate the problem in its *weak form*, which relaxes the strict differentiability requirements on u. The idea is to multiply the governing differential equation by an arbitrary *test function* $v(x)$ that vanishes on the Dirichlet boundary and integrate over Ω. Through integration by parts (using the divergence theorem or Green's identity), the second-order derivatives on u are transferred onto v, reducing the required differentiability of u. The boundary terms produced by integration by parts naturally incorporate the Neumann boundary conditions.

For the Poisson model, the weak (variational) form is: find u in an appropriate function space (typically $H^1(\Omega)$, with u satisfying the Dirichlet boundary values) such that

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_N} t_N \, v \, ds,$$

for all test functions v with $v|_{\Gamma_D} = 0$.

Here, the left side defines a bilinear form

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx,$$

and the right side a linear functional

$$L(v) = \int_\Omega f \, v \, dx + \int_{\Gamma_N} t_N \, v \, ds.$$

In the weak form, u is only required to have first derivatives, which is a weaker requirement than in the strong form.

[2] the differential equation

2.3 Numerical Discretization

The weak formulation above is posed in an infinite-dimensional function space. The Galerkin finite element method discretizes the problem by restricting u and v to finite-dimensional subspaces spanned by piecewise-polynomial basis functions.

First, the domain Ω is partitioned into a mesh of subdomains (finite elements). In 1D, the elements are intervals; in 2D, they are typically triangles or quadrilaterals; in 3D, they may be tetrahedra or hexahedra. A finer mesh allows for a higher resolution of the solution. Each mesh node corresponds to a degree of freedom in the discrete solution.

Next, we select a set of basis functions $\{\phi_j(x)\}$ defined on the mesh (often constructed as low-order polynomials, such as linear "hat" functions). We then approximate the solution $u(x)$ by a linear combination of these basis functions:

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

where N is the total number of basis functions and U_j are the unknown coefficients.

Substituting the approximate solution $u_h(x)$ into the weak form and choosing the test functions as the basis functions (the Galerkin method) leads to a system of N linear equations. In particular, for each basis function ϕ_i[3] (with $i = 1, \ldots, N$) we obtain

$$\sum_{j=1}^{N} U_j \int_{\Omega} \nabla\phi_j \cdot \nabla\phi_i \, dx = \int_{\Omega} f \, \phi_i \, dx + \int_{\Gamma_N} t_N \, \phi_i \, ds.$$

This system can be written in matrix form as

$$K \, \mathbf{U} = \mathbf{F},$$

where the stiffness matrix K has entries

$$K_{ij} = \int_{\Omega} \nabla\phi_j \cdot \nabla\phi_i \, dx,$$

and the load vector F has components

$$F_i = \int_{\Omega} f \, \phi_i \, dx + \int_{\Gamma_N} t_N \, \phi_i \, ds.$$

Essential (Dirichlet) boundary conditions are enforced by setting the corresponding entries of $\mathbf{U}$ to the prescribed values.

3 Solution Procedure of FEA

The finite element method follows a standard sequence of steps applicable to 1D, 2D, or 3D problems. A high-level summary is as follows:

1. **Define the strong form:** Clearly state the governing PDE and boundary conditions (*Neumann, Dirichlet, & Robin*) on the domain Ω.

2. **Derive the weak form:** Multiply the PDE by an arbitrary test function v and integrate over Ω. Use integration by parts to transfer derivatives from u to v and incorporate natural boundary conditions.

3. **Discretize the domain (mesh generation):** Partition the domain into finite elements. In 1D, use intervals; in 2D, use triangles or quadrilaterals; in 3D, use tetrahedra or hexahedra.

4. **Choose shape functions:** Select appropriate polynomial basis functions on each element. Assemble a global basis from these local shape functions.

[3] In the course notes and HWs, we would like to use the notation N_i as it aligns with shape functions.

5. **Formulate the discrete system:** Express the approximate solution $u_h(x)$ as a linear combination of the basis functions and substitute into the weak form. This yields a linear system for the unknown coefficients.

6. **Apply boundary conditions:** Impose essential boundary conditions[4] by directly setting the corresponding nodal values.

7. **Solve the linear system:** Solve the assembled matrix equation $K\mathbf{U} = \mathbf{F}$ using appropriate linear algebra techniques.

Post-processing steps include computing derived quantities and visualizing the solution over the domain.

4 Domain Discretization and Boundary Conditions

Below are simple examples for 1D, 2D, and 3D domains.

1D Domain

Figure 1 illustrates a one-dimensional domain subdivided into four finite elements (five nodes). A Dirichlet boundary condition (for example, $u = 0$) is applied at the left end, and a Neumann boundary condition is applied at the right end.

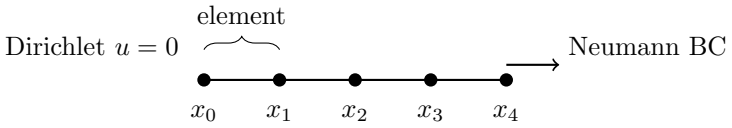

Figure 1: 1D domain discretization with four finite elements (nodes x_0 to x_4).

2D Domain

Figure 2 shows a square two-dimensional domain discretized with a simple mesh of four square elements. The left edge is subject to a fixed value (Dirichlet condition), while the top edge has a Neumann condition (indicated by an arrow).

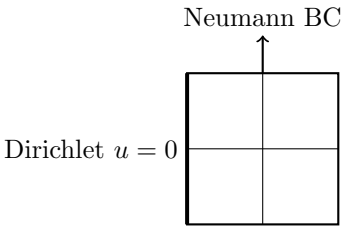

Figure 2: 2D square domain with a simple mesh (four square elements).

3D Domain

Figure 3 depicts a cubic domain in three dimensions. The cube is conceptually divided into smaller subcubes. One face of the cube is fixed (Dirichlet condition) and an arrow on the opposite face indicates a Neumann boundary condition.

[4] in the general cases (e.g., 2^{nd} order PDE) are the DIrichlet B.C.s

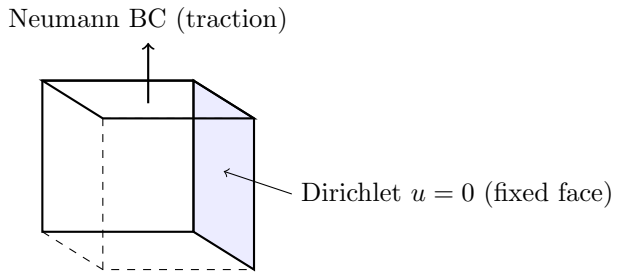

Figure 3: 3D cubic domain (illustrated in perspective).

5 Computational Tools and Practical Notes

Modern computational tools automate the finite element process. Open-source libraries like **FEniCS** provide a high-level platform for defining the weak form of a partial differential equation and solving it using the finite element method[5]. For instance, FEniCS (implemented in Python) allows one to specify the variational formulation, including the bilinear form $a(u, v)$ and the linear functional $L(v)$, together with the boundary conditions. The software then manages mesh generation, the assembly of the stiffness matrix K and the load vector F, and calls appropriate linear solvers to compute the solution. Other popular finite element software packages include commercial products such as ANSYS, Abaqus, and COMSOL Multiphysics, which offer graphical user interfaces and specialized capabilities for a wide range of engineering applications. These tools typically encompass three main stages: pre-processing (which covers geometry and mesh generation and the assignment of material properties and boundary conditions), the solver stage (where the matrix equations are assembled and solved), and post-processing (which involves visualizing results and analyzing data).

Acknowledgement

Special thanks go to Professor Adrian Lew for offering me the chance to be a TA for this course. The vivid discussions with the students (during and outside problem sessions) in the course motivated this note, including José Hasbani, Enzo Andreacchio, Reese Dunne, Dhruv Biswas, Timothy Lee, Rachele Russo, William Cai, Kai Jun Chen, Annika Yong, Lujia Liu, Myung Chul Kim, Ben Alessio, Beverley Yeo, Shufan Xia, Jeff Li, and many others who asked great questions. Also, thanks to the previous TAs, including Philip DePond and Yi Shu, for providing great examples.

References

[1] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2003.

[5] See FEniCS documentation for more details.