Finite Element Methods

The finite element method is based on a “weak” form of any partial differential equation, using a corresponding integral version of the governing PDE (called “strong” form). This weak form is derived (from the strong form) by a variational principle and by discretization of the solution domain into a spatial mesh spanned by elements with unknowns allocated to their vertices (called “nodes”).

The application of finite elements goes back to an engineer’s type analysis of mechanical stability problems, the methodology is therefore useful to solve problem sets with very complex domains and matches object-oriented programming techniques. Furthermore, the retention of the integral form can be beneficial for problems with difficult boundary conditions or discontinuities which can be integrated.

The flip-side of this is that the construction of the numerical equations that need to be solved may take considerably longer than the actual solution process itself. Variational methods are also particularly difficult to constrain when iterative, implicit solution methods are used.

1. Solution algorithm

Usually, only spatial derivatives are discretized with the finite element method, whereas finite differences are used to discretize the time derivatives. The spatial discretization is carried out locally over small regions of simple but arbitrarily shaped elements (the finite elements).

The discretization process itself results in a matrix equation relating the loads (input) at specified points in the element (called nodes) to the displacements (output) at these same points. In order to solve the equations over large regions, one sums node-by-node the matrix equations for smaller sub-regions (elements), finally ending up with the assembly of a global matrix equation.

This system of equations can then be solved by standard linear algebra techniques to yield all nodal displacements, which completes the numerical solution algorithm.

2. Philosophy

Consider a boundary value problem given on a domain Ω with a boundary $\Gamma = \partial\Omega$ such that a solution $u(x)$ satisfies the PDE:

$$\mathbf{F}\{u(x)\} = s(x) \tag{1}$$

where $\mathbf{F}$ is some differential operator and $s(x)$ a source term:

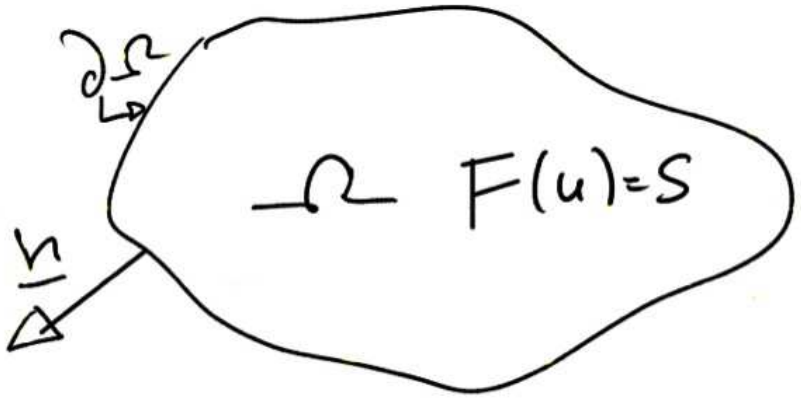


Figure 1: solution domain and its boundary

As boundary conditions, we can have

Dirichlet (fixed value, “essential” BC)

$$u|_{\partial\Omega} = g \tag{2}$$

type constraints, where the value of $u(x)$ is given on $\partial\Omega$, and/or

Neumann (flux, “natural” BC)

$$n_i \frac{\partial u}{\partial x_i} = \vec{n} \cdot \nabla \mathbf{u} = h \tag{3}$$

conditions, where we specify the derivatives at the boundary.

If the PDE is, for example, an elastic deformation problem, then $u(x)$ would be the displacements, and Dirichlet conditions of $g = 0$ correspond to “no-slip” conditions at the boundary $\partial\Omega$.

The finite element analysis then proceeds by two steps:

1. Converting the governing PDE from the regular, “strong” form (which we used for FD) to the “weak” integral form.

2. Discretizing the domain Ω into “elements” on which an approximate, numerical solution for $u(x)$ is to be obtained using simplified polynomials, so called basis resp. “shape” functions.

In this brief introduction, we only provide a highly abbreviated treatment, lacking any mathematical rigor. In addition, we will omit any detailed discussion of different element types, or shape functions, as well as implementation issues such as order of integration.

However, these issues may become important in practice, as choices in shape functions and element type may strongly affect solution robustness and accuracy.

To compare finite elements (FE) with finite difference (FD) methods, their main differences are summarized in the following table:

Finite Differences	Finite Elements
approximates the PDE	approximates the solution of the PDE
mainly restricted to simple, rectangular domains	complex geometries fairly easily implemented
regional, or adaptive mesh refinement hard to implement	regional mesh refinement easy, adaptive refinement fairly straightforward
simple implementation (special case of FE)	involved first implementation
requires programming from scratch if solving new equations	for well-written existing code, only minor changes are needed to solve different equations

3. Example: 1D heat conduction with finite elements

The various steps involved in performing the finite element method are best illustrated with a simple example.
Consider the partial differential equation

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} + Q \tag{4}$$

which governs transient heat conduction in one-dimension with a constant source term Q. The dependent variable in this equation is the temperature T, the independent variables are time t and distance x, and κ is the thermal diffusivity. We are interested in computing the temperature function $T(x,t)$ which satisfies equation (4).

The first step of the finite element method involves choosing an element-type which defines where and how the discretization is carried out. The simplest element for one dimensional problems is a 2-node element (see Figure 2).

One could use more nodes per element which would have the effect of increasing accuracy, but it would also increase the amount of equations and thus the cost of numerical solution.

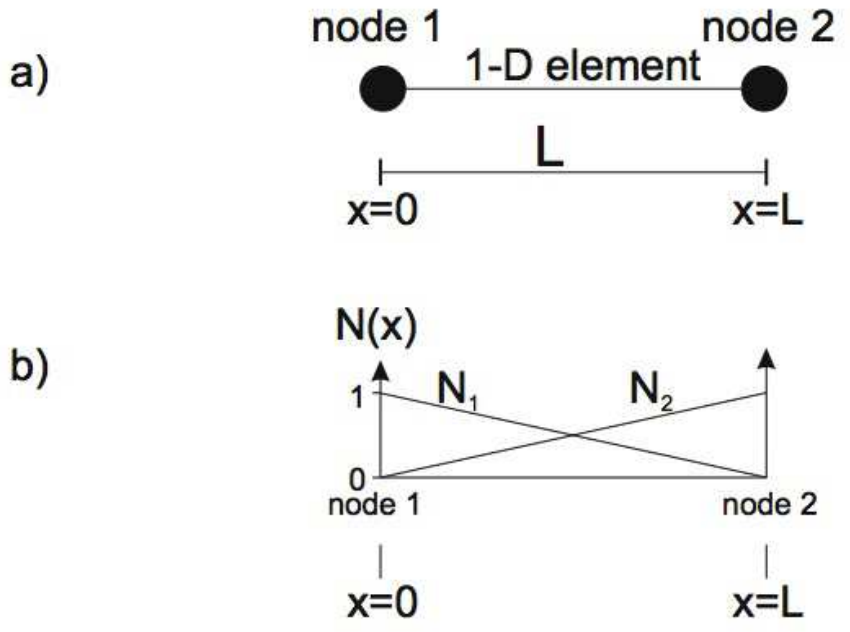


Figure 2: (a) a 2-node, one dimensional finite element with (b) linear shape functions

The second step of the finite element method involves approximating the continuous variable $T(x)$ in terms of nodal variables T_i using simple functions $N_i(x)$ called shape functions. If one focuses on one element (which contains 2 nodes), and one assumes that temperature varies linearly between two nodes, one can write

$$T(x) \approx N_1(x) \cdot T_1 + N_2(x) \cdot T_2 \tag{5}$$

or, using matrix notation,

$$T(x) \approx \begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix} \left\{ \begin{array}{c} T_1 \\ T_2 \end{array} \right\} = \vec{N}(x)^T \cdot \vec{T} \quad . \tag{6}$$

In these equations, $T(x)$ is the unknown continuous variable at time t that needs to be approximated (within any given element) in terms of the two temperatures T_1 and T_2 at the nodes “1” and “2”.

Since we made the choice that temperature varies linearly between two nodes, we have to use the following type of shape functions

$$N_1(x) = 1 - \frac{x}{L}, \qquad N_2(x) = \frac{x}{L}, \tag{7}$$

where L is the length of the element and x is the spatial variable which varies from 0 at node 1 to L at node 2 (Figure 2b).

The shape functions have the following important properties:

- N_1 = 1 at node “1” while N_1 = 0 at node “2”
- N_2 = 0 at node “1” while N_2 = 1 at node “2”
- N_1 + N_2 = 1 (over the entire element)
- the N_i are only locally defined (i.e., they only connect adjacent nodes)

Note that the shape functions $N_i(x)$ are simply interpolating functions (i.e., they are used to interpolate the solution over a finite element). Also, the special choice of the shape functions is directly related to the special choice of an element type.

For example in one-dimension, variation within a 2-node element cannot be uniquely described by a function with an order greater than linear (2 parameter model), and variation within a 3-node element cannot be uniquely described by a function with an order greater than quadratic (3 parameter model), and so on.

The next step is to substitute our approximation for the continuous variable into the governing differential equation. Thus, substituting eq. (6) into eq. (4) leads to

$$\frac{\partial}{\partial t}\left(\begin{bmatrix} N_1 & N_2 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix}\right) - \kappa \frac{\partial^2}{\partial x^2}\left(\begin{bmatrix} N_1 & N_2 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix}\right) - Q = R \qquad (8)$$

where the residual "R" is a measure of the error introduced during discretization. Note that the original partial differential equation is now replaced by an equation in the discretized (nodal) variables T_1 and T_2. Thus, we now have only one equation for two unknowns, which obviously cannot be solved.

We can, however, re-formulate our problem into the problem of finding values for T_1 and T_2 such that the residual gets minimized (ideally R is zero as in the original equation). For this approach, we should "somehow" generate a system of equations where the number of equations equals the number of unknowns. In the finite element method, this is achieved by requiring that the integral of the weighted residual is zero on an element basis.

To perform this step practically, one must multiply (resp. "weight") the residual R in eq. (8) by a set of weighting functions (each in turn), integrate over the element and equate to zero.

Many methods (e.g., collocation, subdomain, least squares and Galerkin) may be used to achieve this procedure, their main difference being a suitable choice of appropriate weighting functions.

Here, we will only consider the Galerkin method: The weighting functions are chosen to be identical to the shape functions N_i.

By carrying out the steps just described one obtains

$$\int_0^L \left\{ \begin{array}{c} N_1 \\ N_2 \end{array} \right\} \frac{\partial}{\partial t} \Big(\left[N_1 \;\; N_2 \right] \left\{ \begin{array}{c} T_1 \\ T_2 \end{array} \right\} \Big) dx -$$

$$\int_0^L \left\{ \begin{array}{c} N_1 \\ N_2 \end{array} \right\} \kappa \, \frac{\partial^2}{\partial x^2} \Big(\left[N_1 \;\; N_2 \right] \left\{ \begin{array}{c} T_1 \\ T_2 \end{array} \right\} \Big) dx - \int_0^L \left\{ \begin{array}{c} N_1 \\ N_2 \end{array} \right\} Q \, dx = \left\{ \begin{array}{c} 0 \\ 0 \end{array} \right\} \quad (9)$$

However, in our special example with linear shape functions, double differentiation of these functions would cause them to vanish.

This difficulty can be resolved by applying Green's theorem (integration by parts) to yield typically

$$\int N_i \frac{\partial^2 N_j}{\partial x^2} dx = -\int \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} dx + \left\{ \begin{matrix} \textit{boundary} \\ \textit{terms} \end{matrix} \right\} \tag{10}$$

where the boundary terms can usually be ignored.

Assuming that κ and Q are not functions of x (and that the $N_i(x)$ do not depend on t), we can rewrite the last expressions as

$$\int_0^L \left[\begin{matrix} N_1 N_1 & N_1 N_2 \\ N_2 N_1 & N_2 N_2 \end{matrix} \right] dx \, \frac{\partial}{\partial t} \left\{ \begin{matrix} T_1 \\ T_2 \end{matrix} \right\} +$$

$$\kappa \int_0^L \left[\begin{matrix} \frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial x}\frac{\partial N_2}{\partial x} \\ \frac{\partial N_2}{\partial x}\frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x}\frac{\partial N_2}{\partial x} \end{matrix} \right] dx \left\{ \begin{matrix} T_1 \\ T_2 \end{matrix} \right\} - Q \int_0^L \left\{ \begin{matrix} N_1 \\ N_2 \end{matrix} \right\} dx = \left\{ \begin{matrix} 0 \\ 0 \end{matrix} \right\} \tag{11}$$

Now we are arrived at two equations for the two unknowns T_1 and T_2, as required.

By substitution of the shape functions $N_i(x)$ according to equation (7), the three integrals can be evaluated, and equation set (11) yields

$$\begin{bmatrix} \frac{L}{3} & \frac{L}{6} \\ \frac{L}{6} & \frac{L}{3} \end{bmatrix} \frac{\partial}{\partial t} \begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix} + \kappa \begin{bmatrix} \frac{1}{L} & -\frac{1}{L} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix} - Q \begin{Bmatrix} \frac{L}{2} \\ \frac{L}{2} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \qquad (12)$$

which can be simplified using matrix notation to

$$\hat{\mathbf{A}} \cdot \frac{\partial}{\partial t} \vec{T} + \hat{\mathbf{B}} \cdot \vec{T} = \vec{F} \qquad (13)$$

where

$$\hat{\mathbf{A}} = \begin{bmatrix} \frac{L}{3} & \frac{L}{6} \\ \frac{L}{6} & \frac{L}{3} \end{bmatrix}, \quad \hat{\mathbf{B}} = \kappa \begin{bmatrix} \frac{1}{L} & -\frac{1}{L} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix}, \quad \vec{F} = Q \begin{Bmatrix} \frac{L}{2} \\ \frac{L}{2} \end{Bmatrix}, \qquad (14)$$

and

$$\vec{T}(t) = \begin{Bmatrix} T_1(t) \\ T_2(t) \end{Bmatrix}. \qquad (15)$$

The next step to perform is the discretization of the time derivative, which is usually achieved with a finite difference approximation.

Assuming an implicit time discretization, we can rewrite eq. (13) as

$$\hat{\mathbf{A}} \cdot \frac{\vec{T}^{n+1} - \vec{T}^{n}}{\Delta t} + \hat{\mathbf{B}} \cdot \vec{T}^{n+1} = \vec{F} \tag{16}$$

where $\vec{T}^{n+1}$ is the future temperature at the nodes (i.e., the unknowns) and $\vec{T}^{n}$ is the vector of old (known) temperatures. Rearranging terms, we obtain

$$\left[\frac{1}{\Delta t}\hat{\mathbf{A}} + \hat{\mathbf{B}}\right] \cdot \vec{T}^{n+1} = \frac{1}{\Delta t}\hat{\mathbf{A}} \cdot \vec{T}^{n} + \vec{F} \tag{17}$$

or, more compactly,

$$\hat{\mathbf{K}} \cdot \vec{T}^{n+1} = \hat{\mathbf{C}} \cdot \vec{T}^{n} + \vec{F} \tag{18}$$

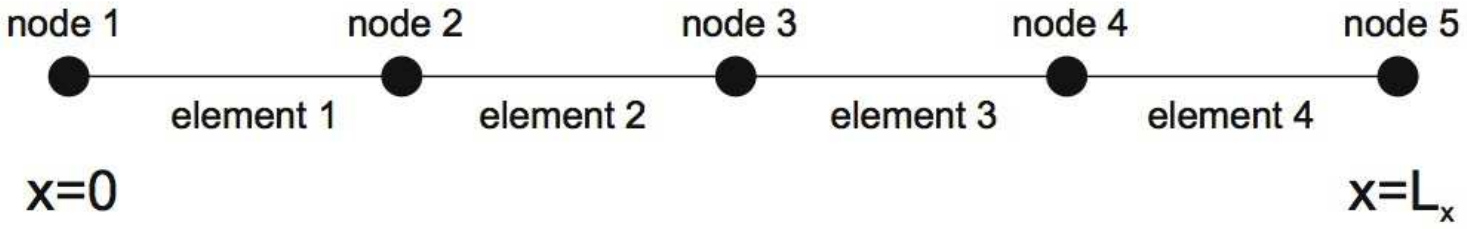
In equation (18), everything appearing on the right hand side combines to form a (known) vector, where we have substituted

$$\hat{\mathbf{C}} = \frac{1}{\Delta t}\hat{\mathbf{A}}.$$

On the left hand side, matrix $\hat{\mathbf{K}}$ is referred to as the element stiffness matrix, and $\vec{T}(t)$ is the unknown element vector.

Remember that so far we have only carried out the discretization of a single element. In general, however, we want to obtain a solution for the whole domain. We divide therefore the solution domain into many elements in order to obtain an global solution.

Let us consider a small one-dimensional mesh, consisting of 4 elements (once you get the idea you can easily consider more elements).



Now, instead of having just 2 unknowns, we have 5 unknowns, related to the five nodes in the (now global) mesh.

The generation of the global matrix equation is done by adding up node-by-node the matrix equations derived for a single element (i.e., eq. (18)). Note that whereas node “1” contains only a contribution from element 1, node “2” has contributions from both elements 1 and 2.

Performing this process (using the notation introduced above and assuming that each element matrix is the same) leads to

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 \\ K_{21} & K_{22}+K_{11} & K_{12} & 0 & 0 \\ 0 & K_{21} & K_{22}+K_{11} & K_{12} & 0 \\ 0 & 0 & K_{21} & K_{22}+K_{11} & K_{12} \\ 0 & 0 & 0 & K_{21} & K_{22} \end{bmatrix} \cdot \vec{T}^{n+1} =$$

$$\begin{bmatrix} C_{11} & C_{12} & 0 & 0 & 0 \\ C_{21} & C_{22}+C_{11} & C_{12} & 0 & 0 \\ 0 & C_{21} & C_{22}+C_{11} & C_{12} & 0 \\ 0 & 0 & C_{21} & C_{22}+C_{11} & C_{12} \\ 0 & 0 & 0 & C_{21} & C_{22} \end{bmatrix} \cdot \vec{T}^n + Q \begin{Bmatrix} \text{L/2} \\ 1 \\ 1 \\ 1 \\ \text{L/2} \end{Bmatrix}$$

which in matrix notation becomes

$$\hat{\mathbf{K}}_g \cdot \vec{T}^{n+1} = \hat{\mathbf{C}}_g \cdot \vec{T}^n + \vec{F}_g, \tag{19}$$

where the subscript "g" indicates that matrices and all vectors refer now to the entire "global" problem and not simply to a single element.
Note that both matrices $\hat{\mathbf{K}}_g$ and $\hat{\mathbf{C}}_g$ are symmetrical, which is an important (though not necessary) property when it comes to solving the equation.

Finally, note that most non-zero terms are clustered near the main diagonal (called diagonal dominance) which also helps when the equations are solved.

Equation system (19) can be rewritten in the form

$$\hat{\mathbf{K}}_g \cdot \vec{T}^{n+1} = \vec{b} \tag{20}$$

which is the classical form of a linear algebraic equation system.

Here, the matrix $\hat{\mathbf{K}}_g$ is referred to as the stiffness (or coefficient) matrix, $\vec{b}$ is referred to as the right hand side (or “load”) vector, and $\vec{T}^{n+1}$ is the unknown solution (or “reaction”) vector.

The final step needed to solve the system is a choice of appropriate boundary conditions.

There are two main possibilities, fixed temperature, fixed temperature gradient, or some combination of both. Fixed temperature boundaries are implemented by performing the following steps:

(1) zeroing the entries in the relevant equation
(2) placing a “1” on the diagonal entry of the stiffness matrix and
(3) setting the value at the correct position of the right hand side vector equal to the desired value

Alternatively, if one wishes to implement zero-flux boundary conditions, one does not have to do anything explicitly (i.e., it is the default boundary condition created when one ignored the boundary terms; non-zero-flux boundary conditions would be slightly more complex).

We are now ready to compute the solution to the 1D heat flow problem:

(1) Define all physical (e.g., diffusivity, source term, length of spatial domain) and numerical (e.g., number of elements and nodes) parameters

(2) Define the spatial coordinate x and the time domain for your wanted solution

(3) Within an element loop, define the element matrices $\hat{A}$ and $\hat{B}$ and the element load vector $\vec{F}$ (see eqs. (13) and (14)). Use these to compute $\hat{K}$ and $\hat{C}$ (eqs. (17) and (18)). Sum these matrices node-by-node to form the global matrices $\hat{K}_g$ and $\hat{C}_g$ and the global vector $\vec{F}_g$ (see eq. (19)). Since the element properties do not depend on time, these global matrices only need to be calculated once and can be saved for later use.

(4) Within a time loop, perform the operations on the right hand side of equation (19) (i.e., multiply $\hat{C}_g$ with the old temperature vector $\vec{T}^n$ and then add the resulting vector to $\vec{F}_g$) to form the right-hand-side vector $\vec{b}$

(5) Apply boundary conditions

(6) Solve equation (20) for the new temperature, and continue to the next time step