R. M. Feenstra
5/19/25

Finite Element Computations

These lecture notes provide an introduction to the "finite element" method of solving differential equations, particularly equations such as Laplace's equation or Poisson's equation as encountered in electrostatics problems. The finite-element method is closely related to what is often call the "finite difference" method, with the two names sometimes used interchangeably. In some ways the latter method can be viewed as being just a simple version of the former. However, in practice (and for nonlinear problems in particular) there are some substantial distinctions between the two methods, as will be discussed here.

The basis of the finite-element method is easily seen by considering Laplace's equation for the electrostatic potential V in 1 dimension, $d^2V/dx^2 = 0$. This differential equation is written in a form for which the second derivative is evaluated numerically on a grid, $x_i = (i-1)\Delta x$ with $i = 1,2,\ldots,n$, where Δx is a finite (nonzero) difference that the derivative is evaluated over. Denoting $V(x_i)$ by V_i, we have

$$\frac{V_{i+1} - 2V_i + V_{i-1}}{(\Delta x)^2} = 0 \ . \tag{1}$$

From this equation, it is clear that

$$V_i = \frac{V_{i+1} + V_{i-1}}{2} \tag{2}$$

so that the potential at any given point is the average of the potential at surrounding points.

This same property of a solution to Laplace's equation – that the potential is an average of the surrounding values – holds for any spatial dimension. The 1-dimensional case is trivial, since if we have a potential of, say, 0 V at some point and 1 V at another point, then the solution of Eq. (2) yields simply a linear variation in potential between those two points. However, for 2 or more dimensions then the solution is not so trivial. Let's consider 2 dimensions in particular, with a square region in space having specified potential energies along its boundaries, say, 0, 1, 2, and 3 V as we go around the square. Then we would like to solve for the potential values in the interior of this square region (this type of finite-difference problem is considered in many electrostatics textbooks, e.g. [1]). Generalizing Eqs. (1) and (2) to this case, and assuming an equal grid spacing in both dimensions of the grid, it is easily shown that the potential $V_{i,j}$ is given by

$$V_{i,j} = \frac{V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}}{4} \quad . \tag{3}$$

Now, the solution of the electrostatics problem (on the grid, or on a continuous set of x, y values) is nontrivial. However, an iterative solution can easily be written down: denoting the potential at the k^{th} iteration by $V_{i,j}^{(k)}$, a possible solution is given by

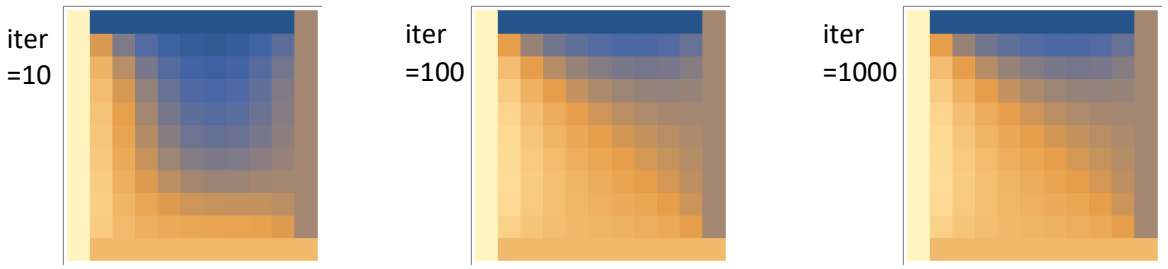
$$V_{i,j}^{(k+1)} = \frac{V_{i+1,j}^{(k)} + V_{i-1,j}^{(k)} + V_{i,j+1}^{(k)} + V_{i,j-1}^{(k)}}{4} \quad . \tag{4}$$

In general, an iterative solution in which we write $V_{i,j}^{(k+1)}$ to be a function of the $V_{i',j'}^{(k)}$ values (for all i' and j') is not guaranteed to converge. The requirement for convergence is closely related to the well-known convergence properties of Newton's method for finding roots of an equation [2]. For Laplace's equation as illustrated here, convergence does in fact occur, although for Poisson's equation as discussed below the situation is more complicated.

Returning to our problem with potential values on the boundaries of 0, 1, 2, and 3 V, the computation starts by initializing the potential matrix accordingly. That is, we assign the boundary values, which remain fixed for all iterations. (The corner values of the potential do not enter the computation in any way, although for plotting purposes it is convenient to have these initialized to be one or the other of the neighboring values on the boundary). The matrix of potential values in the interior of the grid is then initialized to some value, say, $V_{i,j}^{(0)} = 0$ for i and j both ranging from 2 to $n - 1$. Actually, a better guess would be the average of the values on the boundaries, which would be 1.5 V for the problem we're considering, or even better would be to use some sort of bilinear interpolation of the values at the boundaries. However, in the examples below we will use an initialization of 0 since in that case it's easier to judge (just using inspection by eye) how well the solution is converged.

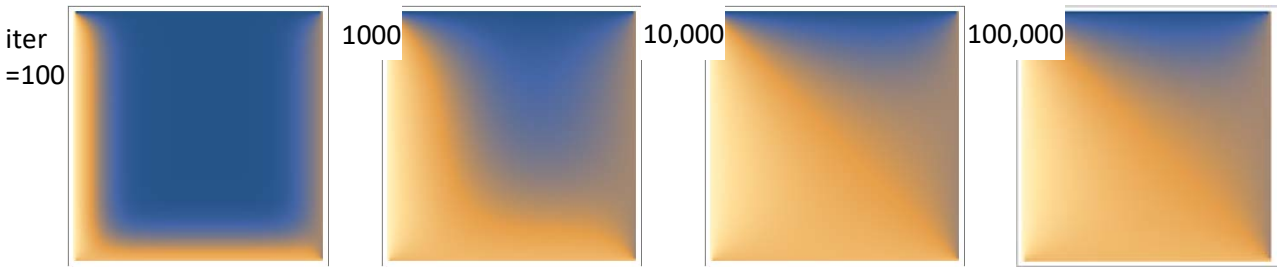
Below are some results obtained using Mathematica for this electrostatic problem, with an 11×11 grid (that is, boundary values on all sides and an interior matrix of 9×9 values):

```
Clear["Global`*"];  n=11;
pot = Table[0, {i,n}, {j,n}];
Do [ pot[[i]] [[n]] = 1, {i,n} ];
Do [ pot[[n]] [[j]] = 2, {j,n} ];
Do [ pot[[i]] [[1]] = 3, {i,n} ];
For [ iter=1, iter<=1000, iter++,
   For [ i=2, i<=n-1, i++,
      For [ j=2, j<=n-1, j++, pot[[i]] [[j]] = N[ pot[[i-1]] [[j]] + pot[[i+1]] [[j]] + pot[[i]] [[j-1]]
                                                  + pot[[i]] [[j+1]] ] / 4 ]]];
ArrayPlot[pot, ColorFunction->"M10DefaultDensityGradient", PlotRange->{0,3}]
```



In the color plots above, high values of the potential are shown with light colors and low values with dark colors (using a plotting range of 0 to 3). The upper left-hand corner of the plot shows

the result for the first row and first column of the matrix. Good convergence is apparent in these results after about 100 iterations (with the results for that number of iterations appearing very similar to what is obtained with 1000 iterations). It is interesting to also examine the same sort of computation on a denser, 101×101 grid, as shown below:



These results are similar to what is obtained on the 11×11 grid, although for the denser grid many more iterations are required; it appears that about 10,000 iteration are required for good convergence. Thus, for an $n \times n$ grid, the number of iterations needed to achieve a given level of convergence appears to scale like n^2. (Offhand, this result seems to make sense, with one factor of n needed in order to have an iterative correction that spans over an entire row or column of the matrix, and a second factor needed since the iterative corrections get smaller as n increases in accordance with n^{-1}; however, a much more detailed analysis is needed in order to rigorously understand the convergence). In terms of run-time needed to obtain a converged solution, that scales like n^4, i.e., n^2 iterations needed for convergence, and n^2 matrix elements which must be updated at each iteration step.

With the results above for the simple electrostatics problem on a square grid, we can now make a few comments concerning the distinction between "finite difference" and "finite element" computations. The above computation, with results computed on a simple, uniformly spaced grid of x and y values, is certainly a finite-difference one. Let us consider what would happen if we were solving a different sort of differential equation, such as analyzing elasticity equations to determine stress and strain that occurs in a structure (such as a bridge) under the effects of an applied load. In such applications, there can be much greater stress at certain points in the structure (e.g. near a bolt) than elsewhere; to understand potential failure of the structure, it is imperative to understand in detail the situation near these high-stress points. In principle one could use a very dense grid to handle that problem, but if this dense grid is used *everywhere* in the structure then the run-time for the computation can become unmanageably large. Hence, more sophisticated solutions are required, in which a *variable* sized grid is used throughout the structure. Additionally, the grid itself should not be a simple square or rectangular one, since for such grids as applied to elasticity problems there is a certain type of instability (associated with shear stress) that arises. Rather, grids containing all sorts of polygonal shapes (often with triangular faces) are employed. The techniques involved in forming these grids are an integral part of the "finite element" method.

Due to the fact that the grid is very much more complicated in a finite-element computation than a finite-difference one, the method of solution of the problem differs from what was described above in our example of the electrostatics problem on a square. Actually, there are *two* types of distinctions between what was described above and what is generally used in finite-element

computations. The first distinction is that, due to the more complicated grids used in finite-element computations, the set of equations that must be solved are correspondingly more complicated than for finite-difference analysis. An entire formalism exists for working out the values of the coefficients in these linear equations. The complications associated with complex grids are so large that, generally speaking, software code for performing finite-element computations using complicated polygonal elements are rarely developed by individual users; rather, large software packages such as COMSOL or ANSYS are generally used (as applied to many types of engineering analysis). In contrast, it is very easy to write code for a simple finite-difference computation, and such codes are routinely developed for specialized applications.

The second distinction between finite-difference and finite-element computations can be explained by returning to the examples above, which we solved using an iterative method. As an alternate method of solving the problem, we could have written a set of linear equations that relate the potential at one point to the potential at other points (including boundary points). Expressing those equations in matrix form, then in principle we could have performed a *single* matrix inversion operation (applied to the 11×11 or 101×101 respective grids) in order to *exactly* solve the problem. Now, for the 101×101 matrix, this sort of inversion might be somewhat time consuming. However, this matrix is rather sparse, and specialized methods exist for inverting such matrices. Of course, in all cases it is desirable to utilize a method of solution that requires the least amount of run time. Comparing an iterative solution of a finite-difference problem with what can be achieved using matrix inversion, it turns out that the latter method is faster whenever the matrix size is sufficiently large. This same sort of situation would also occur (in principle) for finite-element computations, but since the matrices in that case are nearly always quite large, then the matrix-inversion type of solution is always used (i.e. in commercial packages). In contrast, for finite-difference code, especially as developed for specialized applications, the iterative method of solution is sometimes used. This distinction in the method of solution has significant consequences when we furthermore consider solutions of differential equations that contain a *nonlinear* term, as we will now discuss.

Let us turn to consider solutions of Poisson's equation, in particular as might occur in semiconductor materials in which the occupation of electrons in the conduction band (or holes in the valence band), as well as occupation of electrons or holes on dopant atoms, depends sensitively on the electrostatic potential in the material (e.g. in a *pn*-junction). In writing the relevant equations it is convenient to switch to using a potential *energy*, U, rather than just the potential V, with the energy (for an electron) given by $U = -eV$ with $e = 1.602 \times 10^{-19}$ C. Poisson's equation is then given by

$$\nabla^2 U = \frac{e}{\kappa \varepsilon_0} \rho(U) \tag{5}$$

with ρ being the charge density in the material, κ being the dielectric constant, and ε_0 being a constant (the permittivity of free space). As just mentioned, in semiconductors we have a situation where ρ depends strongly on U, so that the term on the right-hand side of Eq. (5) introduces a substantial nonlinearity into the problem.

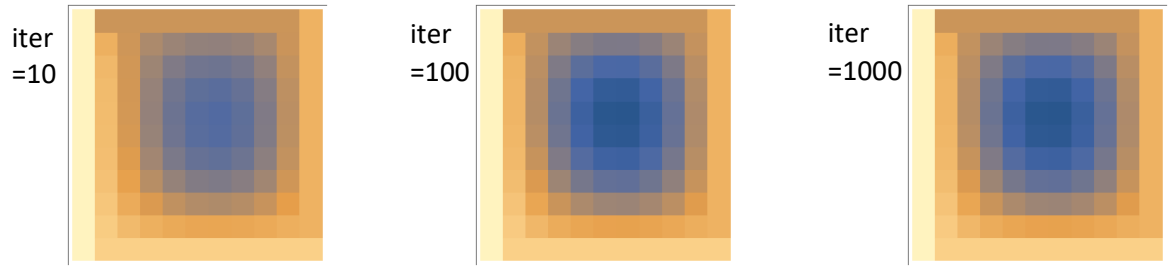
There are two methods for dealing with this sort of nonlinearity in a finite-difference or finite-element computation. First, if we are employing an iterative solution of the form shown in Eq. (4)

(which in practice would only occur for a finite-difference computation), then it is straightforward to include a term such as on the right-hand side of Eq. (5), as we will illustrate below. Second, if we employ a matrix-inversion type of solution (which in practice is always used for finite-element computations), then the nonlinearity must be treated in an approximate way: The nonlinear term, which for matrix of a $U_{i,j}$ values is itself a matrix of charge-density values, is generally attenuated down to zero, and then it is gradually turned in an iterative manner as a sequence of matrix-inversion solutions are computed. That is, starting with the nonlinear term attenuated to zero, a matrix inversion is used to solve for U, then the nonlinear term is updated using these new U values (but still using substantial attenuation). A further matrix inversion is then performed, a new evaluation of the nonlinear term is made (using less attenuation than previously), etc. Hence, we end up again with an iterative solution, but of a different sort than the one which would be employed in an iterative finite-difference computation which uses a local iterative solution (as in Eq. (4)) for every element of the matrix. This distinction in the type of solution utilized for nonlinear problems is another characteristic that distinguishes finite-element and finite-difference computations (i.e., the latter might utilize a local iterative method of solution, while in practice the former never does). With nonlinearity, determining which sort of approach minimizes the run time for a given type of problem now becomes a complicated issue. In any case, for semiconductor-type problems, the finite-difference method using a local iterative solution throughout is found to achieve good performance at least in certain, specialized situations [3].

Let us consider a simple example of an iterative solution to Eq. (5). Writing that equation in a 2-dimensional finite-difference form, we have for the $(k+1)$ iteration,

$$U_{i,j}^{(k+1)} = \frac{1}{4}\left[U_{i+1,j}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k)} - \frac{(\Delta x)^2 e\rho_{i,j}}{\kappa\varepsilon_0}\right] \tag{6}$$

where in general the charge density $\rho_{i,j}$ will depend on the local value of the potential energy, $\rho_{i,j} = \rho\left(U_{i,j}^{(k)}\right)$. In the example we consider here, for the purpose of illustration, we will assume just a constant value of the charge density; we take this to be $0.5\kappa\varepsilon_0/e(\Delta x)^2$ (the scale factors here ensure that our result is invariant with the grid size employed for the solution). We also use the same boundary conditions for the potential (but now measured in eV) as assumed in the examples above. A solution of Eq. (6) for this situation, using a 11×11 grid, appears as



(with a plotting range of -2.3 to $+3$ used). We see that our assumed charge density leads to a parabolic variation in the potential, superimposed on what arises from the boundary values themselves. Most importantly, the method of solution is no more complicated than that for

Laplace's equation discussed above (and similarly even if the charge density is taken to be a function of the potential). As mentioned above, ρ is often a strong function of U in semiconductor problems, so that convergence of the iterative solution is not guaranteed. However, in practice, instabilities and/or nonconvergence are found not to be issues, at least for a certain class of problems that has been studied in detail [3].

Summarizing the distinctions between finite-difference and finite-element computations, the main one has to do with the grid that is utilized: the former employ a simple square or rectangular grid whereas the latter uses a grid containing complicated polygonal shapes (often with triangular faces, with the hypotenuses of the triangles which connect "nodes" in the grid structure being essential for ensuring stability in elasticity problems). Due to this difference in grids, the set of resulting linear equations that must be solved is very much more complicated for finite-element computations than for finite-difference ones. For linear problems, then in principle, both types of computations can be solved either by inversion of an entire matrix describing the problem or by a local iterative method in which the value (of potential, or strain, or whatever is being investigated) being sought is updated based upon surrounding values. However, in practice, matrix inversion is always used in finite-element computations (as in large commercial packages). For finite-difference computations, again, either method of solution can be used, but the local iterative one offers some advantages for nonlinear problems.

Nonlinear problems can be handled using the finite-difference method in just the same manner as for linear problems, by updating a value being sought based upon surrounding values. In contrast, to solve a nonlinear problem by the finite-element method, a sequence of matrix-inversion solutions is employed, with the nonlinearity of the problem gradually turned on. In this regard, the finite-difference method can have a speed advantage over the finite-element one, since the latter ends up performing an *exact* solution of an approximate problem at each iterative step (with this *exact* solution at each step being more accurate than necessary), whereas the former is performing an approximate solution all the way along (finally converging to nearly an exact solution). One further distinction that hasn't been mentioned thus far has to do with scaling of the grids, which is straightforward for finite-difference computations. One can start with a rather coarse grid, then after some given convergence criterion has been reached the size of the grid (in all dimensions) can easily be doubled, the solution then further iterated to some new convergence criterion, etc. For the case of finite-element computations, with their complicated (non-rectangular) grids, it is not at all straightforward to double a grid size, and hence that sort of procedure is generally not a standard one in the finite-element packages.

References:

[1] J. D. Jackson, *Classical Electrodynamics*, 3rd ed. (Wiley, New York, 1999).
[2] http://en.wikipedia.org/wiki/Newton's_method
[3] R. M. Feenstra, *Electrostatic Potential for a Hyperbolic Probe Tip near a Semiconductor*, J. Vac. Sci. Technol. B **21**, 2080 (2003); http://www.andrew.cmu.edu/user/feenstra/semitip_v6/