

**Your Homework Assignment**  
**Finite Differences and Finite Elements:**  
**Getting to Know You**  
**50 points. Due Feb 22**  
Dianne P. O'Leary

Finite differences and finite elements are two common approaches to solving differential equations. In this homework, we explore the nuts-and-bolts of the two methods for a simple problem:

$$-(a(x)u'(x))' + c(x)u(x) = f(x) \text{ for } x \in (0, 1)$$

with the functions  $a$ ,  $c$ , and  $f$  given and  $u(0) = u(1) = 0$ .

We will assume that  $a(x) \geq a_0 > 0$  and  $c(x) \geq 0$  for  $x \in [0, 1]$ .

Read through the entire assignment before beginning. In particular, notice the "Tools" section at the end.

### The Finite Difference Method

We rewrite our equation as

$$-a(x)u''(x) - a'(x)u'(x) + c(x)u(x) = f(x)$$

and approximate each derivative of  $u$  by a finite difference:

$$\begin{aligned} u'(x) &= \frac{u(x) - u(x-h)}{h} + O(h), \\ u''(x) &= \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} + O(h^2). \end{aligned}$$

(We'll compute  $a'(x)$  analytically, so we won't need an approximation to it.)

The finite difference approach is to choose *mesh points*  $x_j = jh$ , where  $h = 1/(M-1)$  for some large integer  $M$ , and solve for  $u_j \approx u(x_j)$  for  $j = 1, \dots, M-2$ . We write one equation for each unknown, by substituting our finite difference approximations for  $u''$  and  $u'$ , and then evaluating the equation at  $x = x_j$ .

**Problem 1.** Let  $M = 6$ , and write the 4 finite difference equations for  $u$  at  $x = .2, .4, .6$ , and  $.8$ .

Notice that the matrix you constructed in Problem 1 has nonzeros on only three bands around the main diagonal; all other elements are zero. The full matrix requires  $(M-2)^2$  storage locations, but, if we are careful, we can instead store all of the data in  $O(M)$  locations by agreeing to store only the nonzero elements, along with their row and column indices. This is a standard technique for storing *sparse matrices*, those whose elements are mostly zero.

Let's see how this finite difference method is implemented.

**Problem 2.** The Matlab function `finitediff1.m`, found on the website, implements the finite difference method for our equation. The inputs are the parameter  $M$  and the functions  $\mathbf{a}$ ,  $\mathbf{c}$ , and  $\mathbf{f}$  that define the equation. Each of these functions takes a vector of points as input and returns a vector of function values. (The function  $\mathbf{a}$  also returns a second vector of values of  $a'$ .) The outputs of `finitediff1.m` are a vector `ucomp` of computed estimates of  $u$  at the mesh points `xmesh`, along with the matrix  $\mathbf{A}$  and the right-hand side  $\mathbf{g}$  from which `ucomp` was computed, so that  $\mathbf{A} \mathbf{ucomp} = \mathbf{g}$ . Add documentation to the function `finitediff1.m` so that a user could easily use it, understand the method, and modify the function if necessary.

There is a mismatch in `finitediff1.m` between our approximation to  $u''$ , which is second order in  $h$ , and our approximation to  $u'$ , which is only first order. We can compute a better solution, for the same cost, by using a second order (central difference) approximation to  $u'$ , so next we will make this change to our function.

**Problem 3.** Define a central difference approximation to the first derivative by

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + O(h^2),$$

Modify the function of Problem 2 to produce a function `finitediff2.m` that uses this approximation in place of the first order approximation.

### The Finite Element Method

We'll use a *Galerkin* approach to solving our problem with finite elements. In particular, we notice that

$$-(a(x)u'(x))' + c(x)u(x) = f(x) \text{ for } x \in (0, 1)$$

implies that

$$\int_0^1 (-(a(x)u'(x))' + c(x)u(x))\phi(x)dx = \int_0^1 f(x)\phi(x)dx$$

for all functions  $\phi$ . Now use integration by parts on the first term, recalling that our boundary values are zero:

$$\int_0^1 (a(x)u'(x))\phi'(x) + c(x)u(x)\phi(x)dx = \int_0^1 f(x)\phi(x)dx.$$

If  $a$ ,  $c$ , and  $f$  are smooth functions (i.e., their first few derivatives exist), then the solution to our differential equation satisfies the boundary conditions and has a first derivative, with the integral of  $(u'(x))^2$  on  $[0, 1]$  finite. We call the space of all such functions  $H_0^1$ , and that is also the space we draw  $\phi$  from.

How does this help us solve the differential equation? We will choose a subspace  $S_h$  of  $H_0^1$  that contains functions that are good approximations to every function in  $H_0^1$ , and we will look for a function  $u_h \in S_h$  so that

$$\int_0^1 (a(x)u_h'(x))\phi_h'(x) + c(x)u_h(x)\phi_h(x)dx = \int_0^1 f(x)\phi_h(x)dx.$$

for all functions  $\phi_h \in S_h$ . This will give us an approximate solution to our problem.

A common choice for  $S_h$  is the set of functions that are continuous and linear on each interval  $[jh, (j+1)h]$ ,  $j = 0, \dots, M-2$  (piecewise linear elements), where  $h = 1/(M-1)$ . We can construct our solution using *any* basis for  $S_h$ , but one basis is particularly convenient: the set of *hat functions*  $\phi_j$ ,  $j = 1, \dots, M-2$ , where

$$\phi_j(x) = \begin{cases} \frac{x-x_{j-1}}{x_j-x_{j-1}} & x \in [x_{j-1}, x_j] \\ \frac{x-x_{j+1}}{x_j-x_{j+1}} & x \in [x_j, x_{j+1}] \\ 0 & \text{otherwise} \end{cases}$$

These are designed to satisfy  $\phi_j(x_j) = 1$  and  $\phi_j(x_k) = 0$  if  $j \neq k$ .

Then we can express our solution  $u_h$  as

$$u_h(x) = \sum_{j=1}^{M-2} u_j \phi_j(x)$$

where  $u_j$  is our approximation to  $u(x_j)$ .

Because the  $\phi_j$  form a basis, our new problem becomes: Find  $u_h$  satisfying

$$a(u_h, \phi_j) = (f, \phi_j)$$

for  $j = 1, \dots, M-2$ , where

$$\begin{aligned} a(u, v) &= \int_0^1 (a(x)u'(x)v'(x) + c(x)u(x)v(x))dx, \\ (f, v) &= \int_0^1 f(x)v(x)dx. \end{aligned}$$

Putting the unknowns  $u_j$  in a vector  $\mathbf{u}$  we can write the resulting system of equations as  $\mathbf{A}\mathbf{u} = \mathbf{g}$  where the  $(j, k)$  entry in  $\mathbf{A}$  is  $a(\phi_j, \phi_k)$  and the  $j$ th entry in  $\mathbf{g}$  is  $(f, \phi_j)$ .

**Problem 4.** Write a function `fe_linear.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise linear elements. Remember to store  $\mathbf{A}$  as a sparse matrix.

It can be shown that the computed solution is within  $O(h^2)$  of the exact solution, if the data is smooth enough. Better accuracy can be achieved if we use *higher order elements*; for example, piecewise quadratic elements would produce a result within  $O(h^3)$  for smooth data. A convenient basis for this set of elements is the piecewise linear basis plus  $M-1$  quadratic functions  $\psi_j$  that are zero outside  $[x_{j-1}, x_j]$  and satisfy

$$\begin{aligned} \psi_j(x_j) &= 0 \\ \psi_j(x_{j-1}) &= 0 \\ \psi_j(x_{j-1} + h/2) &= 1 \end{aligned}$$

for  $j = 1, \dots, M-1$ .

**Problem 5.** Write a function `fe_quadratic.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise quadratic elements. In order to keep the number of unknowns comparable to the number in the previous functions, let the number of intervals be  $m = \lfloor M/2 \rfloor$ . If you order the basis elements as  $\psi_1, \phi_1, \dots, \psi_{m-1}, \phi_{m-1}, \psi_m$  then the matrix  $\mathbf{A}$  will have 5 nonzero bands around the main diagonal. Compute one additional output `uval` which is the finite element approximation to the solution at the  $m - 1$  interior mesh points and the  $m$  midpoints of each interval, where the  $2m - 1$  equally spaced points are ordered smallest to largest. (In our previous methods, this was equal to `ucomp`, but now the values at the midpoints of the intervals are a linear combination of the linear and quadratic elements.)

Now we have four solution algorithms, so we define a set of functions for experimentation:

$$\begin{aligned}
 u_1(x) &= x(1-x)e^x, \\
 u_2(x) &= \begin{cases} u_1(x) & \text{if } x \leq 2/3 \\ x(1-x)e^{2/3} & \text{if } x > 2/3 \end{cases} \\
 u_3(x) &= \begin{cases} u_1(x) & \text{if } x \leq 2/3 \\ x(1-x) & \text{if } x > 2/3 \end{cases} \\
 a_1(x) &= 1, \\
 a_2(x) &= 1 + x^2, \\
 a_3(x) &= \begin{cases} a_2(x) & \text{if } x \leq 1/3 \\ (x - 1/3) + 10/9 & \text{if } x > 1/3 \end{cases} \\
 a_4(x) &= \begin{cases} a_2(x) & \text{if } x \leq 1/3 \\ 1 + 2x^2 & \text{if } x > 1/3 \end{cases} \\
 c_1(x) &= 0, \\
 c_2(x) &= 2, \\
 c_3(x) &= 2x.
 \end{aligned}$$

**Problem 6.** Use your four algorithms to solve 7 problems:

- $a_1$  with  $c_j$  ( $j = 1, 2, 3$ ) and true solution  $u_1$ .
- $a_j$  ( $j = 2, 3$ ) with  $c_1$  and true solution  $u_1$ .
- $a_1$  and  $c_1$  with true solution  $u_j$  ( $j = 2, 3$ ).

Compute three approximations for each algorithm and each problem, with the number of unknowns in the problem chosen to be 9, 99, and 999. For each approximation, print  $\|\mathbf{u}_{computed} - \mathbf{u}_{true}\|_\infty$  where  $\mathbf{u}_{true}$  is the vector of true values at the  $M - 1$  mesh points.

Discuss the results:

- How easy is it to program each of the four methods? Estimate how much work Matlab does to form and solve the linear systems. (The work to solve the tridiagonal systems should be about  $5M$  multiplications, and the work to solve the 5-diagonal systems should be about  $11M$  multiplications, so you just need to estimate the work in forming each system.)
- For each problem, note the observed convergence rate  $r$ : if the error drops by a factor of  $10^r$  when  $M$  is increased by a factor of 10, then the observed convergence rate is  $r$ .
- Explain any deviations from the theoretical convergence rate:  $r = 1$  and  $r = 2$  for the two finite difference implementations, and  $r = 2$  and  $r = 3$  for the finite element implementations.

In doing this work, we begin to understand the complexities of implementation of finite difference and finite element methods. We have left out many features that a practical implementation should contain. In particular, the algorithm should be adaptive, estimating the error on each mesh interval and subdividing the intervals (or raising the order of polynomials) where the error is too high. And we need to handle partial differential equations, too. Luckily, there are good implementations of these methods for 2- and 3-dimensional domains, so we don't need to write our own.

**How to submit your homework:** Please submit your work on paper. (Email submission is acceptable only in an emergency.) Put your work in this order:

- A listing of the output produced by your experiments.
- A discussion for Problem 6.
- A listing of each of your Matlab functions. (Remember to document them.)

Grading:

- Problem 1: 0 points (just an exercise worth doing).
- Clear output: 5 points.
- Discussion in Problem 6: 15 points.
- Design and correctness of Matlab programs: 20 points.
- Documentation: 10 points.

**Tools:**

For Problem 1, refer to the “Guidelines for Writing Scientific Computing Software” on the website.

In doing Problems 2 and 3, it is helpful to experiment with a test problem in order to see what the functions are doing. Look ahead to Problem 6 for sample problems.

Problem 2 uses the Matlab function `spdiags` to construct a sparse matrix. If you have never used sparse matrices in Matlab, print the matrix `A` to see that the data structure for it contains the row index, column index, and value for each nonzero element. If you have never used `spdiags`, type `help spdiags` to see the documentation.

Use Matlab’s `quad` to compute the integrals for the entries in the matrix and right-hand side for the finite element formulations.

Before tackling the programming for Problems 4 and 5, take some time to understand exactly where the nonzeros are in the matrix, and exactly what intervals of integration should be used. The program is short, but it is easy to make mistakes if you don’t understand what it is supposed to compute. In Problem 6, we measure *work* by counting the number of multiplications. One alternative is to count the number of floating point computations, but this usually gives a count of about twice the number of multiplications, since typically multiplications and additions are paired in computations. Computing time is another possible measure of work, but it can be contaminated by the effects of other users or other processes on the computer. In determining and understanding the convergence rate in Problem 6, plotting the solutions or the error norms might be helpful.

If you have questions about the homework, please ask me, by email or during office hours.

Watch the FAQ page for corrections and clarifications of the homework.

A good introduction to the theory of finite difference and finite element methods is given by Stig Larsson and Vidar Thomée<sup>1</sup>.

1. Stig Larsson and Vidar Thomée, *Partial Differential Equations with Numerical Methods*, Springer, 2003, Chapter 2, Sections 4.1 and 5.1.