Solving PDE’s with FEniCS

Finite difference methods

L. Ridgway Scott
The Institute for Biophysical Dynamics, The Computation Institute, and the Departments of Computer Science and Mathematics, The University of Chicago
Consider the two-point boundary-value problem

\[-\frac{d^2 u}{dx^2} = f \text{ in } (0, 1)\]

\[u(0) = g_0, \quad u'(1) = g_1.\]  \hspace{1cm} (1)

FDM: differential operator $\longrightarrow$ difference operator.

In this way we get the approximation for (1)

\[-u(x - h) + 2u(x) - u(x + h) \approx h^2 f(x)\]  \hspace{1cm} (2)

where $h > 0$ is the mesh size to be used.
Choosing $x = x_n := nh$ for $n = 0, 1, \ldots, N = 1/h$, we get a system of linear equations

$$-u_{n-1} + 2u_n - u_{n+1} = h^2 f(x_n)$$

where $u_n \approx u(x_n)$.

Same as piecewise linear finite element discretization.

Boundary condition at $x = 0$ translates into $u_0 = g_0$.

Thus equation (3) for $n = 1$ becomes

$$2u_1 - u_2 = h^2 f(x_1) + g_0.$$  

But derivative boundary conditions more complex.
Derivative boundary conditions

Derivative boundary condition at $x = 1$ must be approximated by a difference equation.

A natural one to use is

$$u_{N+1} - u_{N-1} = 2hg_1 \quad (5)$$

using a difference over an interval of length $2h$ centered at $x_N$.

Using (5), equation (3) for $n = N$ becomes

$$-2u_{N-1} + 2u_N = h^2 f(x_N) + 2hg_1. \quad (6)$$

Now we summarize these equations as a matrix equation.
Algebraically, we can express the finite difference method as

\[ AU = F \]  

(7)

where

- \( U \) is the vector with entries \( u_n \) and
- \( F \) is the vector with entries \( h^2 f(x_n) \)
- appropriately modified at \( n = 1 \) and \( n = N \) using boundary data.

Finite difference matrix \( A \) has very regular pattern.
- appropriately modified at \( n = 1 \) and \( n = N \)

as we now describe.
Finite difference matrix $A$ diagonal entries equal to 2.

First sub- and super-diagonal entries equal to $-1$

\[
A = \begin{pmatrix}
  2 & -1 & 0 & 0 & 0 & \cdots \\
  -1 & 2 & -1 & 0 & 0 & \cdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  \cdots & -1 & 2 & -1 & 0 & \cdots \\
  \cdots & 0 & -1 & 2 & -1 & \cdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  0 & 0 & \cdots & 0 & -2 & 2 \\
\end{pmatrix}
\]

except the last sub-diagonal entry, which is $-2$.

Matrix $A$ created in \texttt{octave} by various techniques.
For simplicity, we consider the case where the differential equation to be solved is

\[-u''(x) = \sin(x) \quad \text{for } x \in [0, \pi]\]

\[ u(0) = u(\pi) = 0. \quad \text{(8)} \]

Then \( u(x) = \sin(x) \) for \( x \in [0, \pi] \).

To create difference operator \( A \) in a sparse format, one must specify only the non-zero entries of the matrix, that is, you give a list of triples: \((i, j, A_{ij})\).

Operation \texttt{sparse} amalgamates these triples into a sparse matrix.
octave code for solving (8)

dx=pi/(N+1);
i(1:N)=1:N;
j(1:N)=1:N;
v(1:N)= 2/(dx*dx);
i(N+(1:(N-1)))=(1:(N-1));
j(N+(1:(N-1)))=1+(1:(N-1));
v(N+(1:(N-1)))= -1/(dx*dx);
i((2*N-1)+(1:(N-1)))=1+(1:(N-1));
j((2*N-1)+(1:(N-1)))=(1:(N-1));
v((2*N-1)+(1:(N-1)))= -/(dx*dx);
A=sparse(i,j,v);
F(1:N)=sin(dx*(1:N));
In Octave, the solution to (7) can be achieved simply by writing

\[ U = A \backslash F \]

Critical to use vector constructs in Octave to insure optimal performance.

Code executes much more rapidly, but code is not shorter, more readable, or less prone to error.
For the approximation \((u_n)\) (3) of the solution \(u\) of equation (3) it can be shown [1] that

\[
\max_{1 \leq n \leq N} |u(x_n) - u_n| \leq C_f h^2
\]

\[
\left( h \sum_n (u(x_n) - u_n)^2 \right)^{1/2} \leq C_f h^2
\]

where \(C_f\) is a constant depending only on \(f\).

Experiment with a known \(u\) can determine if relationship (9) holds as \(h\) is decreased.

If \(\log e_h\) is plotted as a function of \(\log h\), then the resulting plot should be linear, with a slope of two.
Consider the boundary value problem (8), that is, 
\[-u'' = f \text{ on } [0, \pi] \text{ with } f(x) = \sin x \text{ and } u(0) = u(\pi) = 0.\]

Mean-squared error plotted in Figure 1.

The line \( e_h = 0.1h^2 \) has been added for clarity.

Thus we see for \( h \geq 10^{-4} \), the error diminishes quadratically.

However, when mesh size is much less than \( 10^{-4} \), round-off error causes accuracy to diminish, and error even increases as mesh size is further decreased.
Figure 1: Error in $L^2([0, \pi])$ for the finite difference approximation of the boundary value problem for the differential equation $-u'' = \sin(x)$ on the interval $[0, \pi]$, with boundary conditions $u(0) = u(\pi) = 0$, as a function of the mesh size $h$. The solid line has a slope of 2 as a reference.
All discretization methods can suffer from the effects of finite precision arithmetic.

Typical cause is increasing condition number of linear system $A$ as $h$ tends to zero.

Condition number grows proportional to $N^2 \approx h^{-2}$.

<table>
<thead>
<tr>
<th># of grid points</th>
<th>mesh size</th>
<th>condition #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>2.9e-01</td>
<td>3.1e-02</td>
</tr>
<tr>
<td></td>
<td>4.8e+01</td>
<td>4.1e+03</td>
</tr>
</tbody>
</table>

Table 1: Condition number as a function of $N$ (mesh size $h = \pi/(N + 1)$).
Pitfall: Low Accuracy

Error decreases quadratically to a point, then hits wall.

Accuracy then behaves randomly and even decreases as the mesh size is further decreased.

Greatest accuracy achieved is

\[ \text{condition number multiplied by machine } \epsilon = 2.22 \times 10^{-16}. \]

Thus with \( N = 10,000 \), we cannot expect accuracy better than about \( 10^{-9} \).

The simple way to avoid this difficulty is to avoid very small \( h \) values.
The equivalent accuracy can be achieved by using a larger $h$ and a more accurate discretization method.

Indeed, “spectral” methods rely entirely on increasing the degree of approximation on a fixed mesh.

<table>
<thead>
<tr>
<th>degree</th>
<th>number of grid intervals</th>
<th>error</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td>1.20e-08</td>
<td>0.137</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>1.14e-09</td>
<td>0.083</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>9.57e-11</td>
<td>0.079</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>4.08e-10</td>
<td>0.079</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1.02e-09</td>
<td>0.076</td>
</tr>
</tbody>
</table>

Table 2: Error in $L^2$ norm for the problem $-u'' = \pi^2 \sin(\pi x)$ on $[0, 1]$ with boundary conditions $u(0) = u(1) = 0$ as a function of degree, number of grid points. Time is in seconds.
Two-dimensional problems

Same as seen in two-dimensional problem.

<table>
<thead>
<tr>
<th>degree</th>
<th>mesh number</th>
<th>$L^2$ error</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1024</td>
<td>2.07e-06</td>
<td>22.5</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>2.11e-09</td>
<td>18.4</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>4.95e-12</td>
<td>3.0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>3.98e-12</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Table 3: Computational experiments with solving two-dimensional problem. Degree refers to the polynomial degree, mesh number indicates the number of edges along each boundary side, $L^2$ error is the error measured in the $L^2([0, 1]^2)$ norm.
Higher-order methods do not solve the problem of round-off error.

Maximum accuracy about same independent of degree of approximation.

Condition number is linked to the accuracy of the approximation.

The condition number of the linear system depends on the largest eigenvalue of the PDE that is resolved.

The better the resolution, the larger the condition number.
This is a feature of elliptic systems, like Poisson’s equation, whereas in time-stepping methods increasing the accuracy always reduces the effects of round-off errors.

On the other hand, higher-order methods do reduce the run time required to achieve maximal accuracy, as indicated in Table 3.

For this reason, higher-order methods would allow the use of more accurate precision arithmetic at a manageable cost.
Simplest time discretization method for heat equation uses forward (or explicit) Euler difference method:

\[ u^{n+1}(x) = u^n(x) + \Delta t \frac{\partial^2 u^n}{\partial x^2}(x, t) \quad \forall x \in [0, 1], \]
\[ u^0(x) = u_0(x) \quad \forall x \in [0, 1] \]
\[ u^n(0) = g_0(t) \quad \text{and} \quad u^n(1) = g_1(t) \quad \forall n > 0 \]

(10)

where \( u^n(x) \) denotes an approximation to \( u(x, n\Delta t) \).

Applying the finite difference or finite element approximation (3) to (10) yields a simple algorithm.

Difficulty: \textit{unstable} unless \( \Delta t \) is sufficiently small.
Using the approximations (10) and (2), we get the following finite difference method for the heat equation:

\[
  u_{j}^{n+1} = u_{j}^{n} + \frac{\Delta t}{\Delta x^2} (u_{j-1}^{n} - 2u_{j}^{n} + u_{j+1}^{n}) \\
  = ru_{j-1}^{n} + (1 - 2r)u_{j}^{n} + ru_{j+1}^{n},
\]

where \( r = \Delta t / \Delta x^2 \).

Here \( u(j \Delta x, n \Delta t) \approx u_{j}^{n} \).

Very appealing since there are no equations to solve.

However, there is a strict stability limitation.
Stability limitation

Start with the initial condition \( u_j^0 = (-1)^j \).

Although this is a rough function, we know that the heat equation should smooth it out.

For the moment, let us ignore boundary conditions and assume that (11) holds for all \( j \) and \( n \).

This is equivalent to assuming that \( \Omega = \mathbb{R} \).

Then we will prove by induction that

\[
    u_j^n = (1 - 4r)^n (-1)^j.
\]  \( \quad \text{(12)} \)

First, by assumption this holds for \( n = 0 \).
Using (11) and the induction hypothesis, we have

\[
u_{j}^{n+1} = r u_{j-1}^{n} + (1 - 2r) u_{j}^{n} + r u_{j+1}^{n} = r(1 - 4r)^{n}(-1)^{j-1} + (1 - 2r)(1 - 4r)^{n}(-1)^{j} + r(1 - 4r)^{n}(-1)^{j+1} = (1 - 4r)^{n}(-1)^{j}(-r + (1 - 2r) - r) = (1 - 4r)^{n+1}(-1)^{j},\]

completing the induction step. So (12) holds \(\forall n, j\).

\[r = \frac{\Delta t}{\Delta x^2} > 0 \text{ so } |1 - 4r| \leq 1 \text{ only for } r \leq 1/2.\]

For \(r < 1/2\), then \(u_{j}^{n}\) decreases rapidly to zero.
Approximation blow up

If \( r > 1/2 \), then \( |1 - 4r| > 1 \), and (12) blows up exponentially. Most insideously, we have

\[
 u(x, t) = u(j \Delta x, n \Delta t) \approx u^n_j = (1 - 4r)^{t/\Delta t}(-1)^j,
\]

so the blow up accelerates as \( \Delta t \to 0 \).

Thus the explicit method (11) for the heat equation is restricted by the stringent requirement

\[
 \Delta t \leq \frac{\Delta x^2}{2}.
\]

Therefore the implicit methods are more efficient in many cases, allowing larger time steps.
The solution of the Jeffrey-Hamel problem can be effected using a difference method for the differential operator as before:

\[-u_{n-1} + 2u_n - u_{n+1} + 4h^2 u_n + 6h^2 u_n^2 = h^2 C\]  \hspace{1cm} (14)

where \( u_n \approx u(x_n) \).

Since this system of equations is nonlinear, we cannot solve it directly.

A standard algorithm to use is Newton’s method, which can be written as follows.
Newton’s method

First, we write the system of equations as $F((u_i)) = 0$ where

$$f_n := -u_{n-1} + 2u_n - u_{n+1} + 4h^2 u_n + 6h^2 u_n^2 - h^2 C \quad (15)$$

Newton’s iteration takes the form

$$u \leftarrow u - J_f(u)^{-1} f(u) \quad (16)$$

where $J_f(u)$ denotes the Jacobian of the mapping $f$ evaluated at $u$.

This can be written in octave as follows.
Suppose that $A$ is defined as before.

Then $f$ can be written

$$
\delta = \frac{(n+1)}{\alpha_f} \cdot \frac{(n+1)}{\alpha_f};
$$

$$
f = \delta \cdot A \cdot u_{h_f} - 4 \cdot u_{h_f} - 6 \cdot u_{h_j} \cdot u_{h_j} + c_{vec};
$$

where

$$
c_{vec} = C \cdot \text{ones}(n,1);
$$

The Jacobian $J$ of $f$ is

$$
J = \delta \cdot A - 4 \cdot \text{eye}(n) - 12 \cdot \text{diag}(u_{h_j},0);
$$
Newton’s method takes the following form in `octave`.

```octave
JA = delta*A - 4*eye(n);
ujh = -JA\cvec;
enorm = 1;
while (enorm >> .00000000000001)
f = JA*ujh - 6*ujh.*ujh + cvec;
J = JA - 12*diag(ujh,0);
x = ujh - J\f;
enorm = norm(ujh-x)/(norm(ujh)+norm(x));
ujh=x;
end
```
Solution of the Jeffrey-Hamel equation with $C = 0.1$. 
Solution of the Jeffrey-Hamel equation with $C = 0.01$. 