GIAN Workshop on Theory and Computation of SPDEs, December 2017

Lab 1: Finite difference methods on uniform meshes

Niall Madden (Niall.Madden@NUIGalway.ie)

1 Boundary value problems

The general form a second-order, two point, linear BVP is

$$-u''(x) + a(x)u'(x) + b(x)u(x) = f(x) \quad 0 < x < 1$$

$$u(0) = \alpha, \quad u(1) = \beta.$$

There are built-in functions for solving these, but we'll look at how to solve them using our own *finite difference method*.

1.1 The algorithm

- Choose N, the number of *mesh intervals*
- Set up a set of $N+1$ equally spaced points:

  $$0 = x_0 < x_1 < x_2 < x_3 \cdots < x_{n-1} < x_n = 1.$$
- Construct A, a $(N+1)\times(N+1)$ matrix of zeros, except for
  - $A_{1,1} = 1$
  - For $i = 2, 3, \ldots, N$

    $$A_{i,j} = \begin{cases} -1/h^2 & j = i-1 \\ 2/h^2 + b_k & j = i \\ -1/h^2 & j = i+1 \\ 0 & \text{otherwise.} \end{cases}$$
  - $A_{N,N+1} = 1$

  In MATLAB this could be implemented as

```
A(1,1) = 1;
for i=2:N
    A(i,i-1) = -1/h^2;
    A(i,i)   = 2/h^2 + r(x(i));
    A(i,i+1) = -1/h^2;
end
A(N+1,N+1)=1;
```

  (We would **not** do this in practice: it is very slow).
- Solve the linear system: `u = A \ B` where `B(i)=f(x(i))`.

Download the script FiniteDifference.m from `http://www.maths.nuigalway.ie/~niall/TCSPDEs2017/` and try it out.

Consider the problem:

$$-u''(x) + u(x) = 1 + x \text{ on } (0,1), \tag{1a}$$

$$u(0) = u(1) = 0. \tag{1b}$$

The solution to this is

$$u(x) = 1 + x - \big(e^{-x}(e^2 - 2e) + e^x(2e-1)\big)/(e^2-1).$$

Use this to test the code. In particular, does the error tend to zero as $N \to \infty$? If so, how rapidly? (These two questions can also be rephrased as "Does the method converge? If so, how quickly?)

1.2 The Profiler

This is not a good way to construct a linear system. Whenever you write a MATLAB program, particularly for solving differential equations, you should use the **profiler** to find any bottle-necks in the code.

If most of the time is **not** spent solving the linear system, then there is a problem.

Another simple method for code-timing are the `tic` and `toc` functions.

1.3 Some Optimisations

To improve, and speed up this code, initialise the matrix `A` and vector `b`:

`A = zeros(N+1, N+1);    b = zeros(N+1,1)`

However, the real improvement is to avoid using loops to initialise matrices or vectors.

For vectors, this is easy:

`b = [alpha; r(x(2:N)); beta];`

For Matrices, we need *sparse matrices.* To initialise:

`A = sparse(N+1, N+1);`

However, the best way to use it is as:

`S = sparse(i,j,s)`

which sets `S(i(k),j(k)) = s(k)`. This can be used as follows:

```
A = sparse(2:N, 1:N-1, -1/h^2) + ...
     sparse(2:N, 2:N, 2/h^2+r(x(2:N))) + ...
      sparse(2:N, 3:N+1, -1/h^2);
```

2 Exercises

1. Change the equation in (1a) to include a non-zero convective term (if you like, remove the reaction term entirely, i.e., set $b = 0$).
2. Produce a program like that above that solves this method using standard central differences. Verify that the solution is oscillatory.
3. Now ally upwinding. Verify the order of convergence.