Tutorial 5: A 1d PDE example. The Fisher Equation

COMP5930M Scientific Computation

Today

The Fisher Equation

Find $u(x, t)$ satisfying

$$\frac{\partial u}{\partial t} = u\frac{\partial^2 u}{\partial x^2} + u\,(1 - u)$$

on $x \in [X_1, X_2]$ and for $t > 0$, with suitable initial and boundary conditions.

- A nonlinear reaction-diffusion equation
- Note that $u(x, t) \equiv 0$ and $u(x, t) \equiv 1$ are solutions (fixed points)

Physical meaning

- One of many ecological population models
- $u(x,t)$ represents a population density ($0 \leq u \leq 1$)
- Physical processes captured are redistribution/birth/death within a homogeneous population
- We require $0 \leq u \leq 1$ for all x and t

Method-of-Lines approach

- Approximate in space
  - Semi-discrete system
- Approximate in time
  - Fully discrete system
- Discrete solution algorithm
  - Numerical solution

Spatial approximation

Define a uniformly spaced grid

$$x_i = X_1 + (i-1)h, \qquad i = 1, 2, ..., n$$

where $h = \frac{X_2 - X_1}{n-1}$

Approximate $\frac{\partial^2 u}{\partial x^2}$ with the FD stencil

$$\boxed{\frac{\partial^2 u}{\partial x^2}(x_i) \approx \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2}}$$

Approximation in time

Implicit Euler approximation for $\frac{d\mathbf{u}}{dt} = f(t, \mathbf{u})$

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^{k}}{\Delta t} = f(\mathbf{u}^{k+1})$$

The discrete model

On a grid of m nodes with grid size h and time step Δt the nonlinear equation at node i, $F_i = 0$, can be written as

$$\begin{aligned} F_i &= \frac{u_i^{k+1} - u_i^k}{\Delta t} - u_i^{k+1}\frac{u_{i+1}^{k+1} - 2u_i^{k+1} + u_{i-1}^{k+1}}{h^2} - u_i^{k+1}\left(1 - u_i^{k+1}\right) \\ &= 0 \end{aligned}$$

3

where $u_i^{k+1} \equiv u(x_i, t^{k+1})$

Our $n = m - 2$ unknowns $U = \{u_i^{k+1}, i = 2, 3, ..., m - 1\}$

Discrete system structure

Analytical Jacobian

4

We can calculate analytically all the elements of row i:

$$
\begin{aligned}
\frac{\partial F_i}{\partial U_i - 1} &= -\frac{U_i}{h^2} \\
\frac{\partial F_i}{\partial U_i} &= \frac{1}{\Delta t} - \frac{U_{i+1} - 4U_i + U_{i-1}}{h^2} - 1 + 2U_i \\
\frac{\partial F_i}{\partial U_{i+1}} &= -\frac{U_i}{h^2} \\
\frac{\partial F_i}{\partial U_j} &= 0, \text{ for all } |j - i| > 1
\end{aligned}
$$

Numerical solution algorithm

Time-stepping

- The overall algorithm evolves the solution in time
- At each time-step a nonlinear system is solved

```
function [ u,t ] = fisherModel( dtIn,nTimeSteps, mIn )

global uOld dt n
dt = dtIn; m = mIn; n = m-2;

[t,u,x] = initialData( m );

for k = 1:nTimeSteps
      t = t + dt;
      uOld = u;
      [u,f] = newtonAlgorithm( @fisherFDM, uOld, 1e-10, ...
               @fdJacobian, @linearSolve, maxIts );
      fprintf(' Completed step %d',k);

end
```

- Initial guess for $\mathbf{U}^{k+1}$ at each step is $\mathbf{U}^{k}$

Components of the nonlinear algorithm

- The solution algorithm: `newtonAlgorithm.m`
- Define the nonlinear system: `fisherFDM.m`
- Build the Jacobian: `fdJacobian.m`
- Solve the linear system: `linearSolve.m`

5

Using the solution algorithm:

```
[u,f] = newtonAlgorithm( @fisherFDM, u0, tol,...
                         @fdJacobian, @linearSolve, maxits );
```

- The component functions are passed as arguments

Implementation of the FDM

- The nonlinear system is a set of $n = m - 2$ equations
- Straightforward Matlab implementation

```
function f = fisherFDM( u )
global uOld dt n h uL uR
f = zeros(n,1);

f(1) = ( u(1) - uOld(1) )/dt ...
      - u(1)*( u(2)-2*u(1) + uL )/h^2 ...
      - u(1)*( 1 - u(1) );

for i = 2:n-1
      f(i) = ( u(i) - uOld(i) )/dt ...
          - u(i)*( u(i+1)-2*u(i) + u(i-1) )/h^2 ...
          - u(i)*( 1 - u(i) );
end

f(n) = ( u(n) - uOld(n) )/dt ...
      - u(n)*( uR-2*u(n) + u(n-1) )/h^2 ...
      - u(n)*( 1 - u(n) );

end
```

Switching components of the numerical solver

Provided we use the same input and output function arguments we can implement new algorithms for each component to replace the standard ones used here

For example

- Tridiagonal numerical Jacobian fdTridiagJacobian.m
- A Thomas algorithm Thomas.m for tridiagonal systems

Using the solution algorithm:

```
[u,f] = newtonAlgorithm( @fisherFDM, u0, tol, ...
                         @fdTridiagJacobian, @Thomas, maxIts );
```

Next time...

Self-study

- Implement the Thomas algorithm for tridiagonal systems

Lecture

- Improved time-stepping methods