Lecture 22: Beginning of the End, the End of the Beginning...starting to put it all together #1

Outline

1) Motivation: mathematical models (ODE's and PDE's)
2) An Example: 1-D thermal evolution with radiation and heating
   A) The Basics: steady state 2-pt BVP’s
   B) Themes and variations
      Boundary conditions, non-linearity
      spatially variable coefficients
      time-dependence, PDE's
3) Solution of 2-point Boundary value problems
   A) Shooting Methods
   B) Finite-Difference methods
   C) Spectral - Collocation methods
   D) Galerkin Finite Elements

Basic Numerical Methods: building blocks for mathematical modeling and scientific computation

Points:

1) One of the principal goals of this course is to give you enough understanding of the fundamental components of computational math to begin putting them together to solve problems arising in an array of application areas (e.g. physics, engineering, Earth Sciences etc. etc)

2) Many (but not all) physical models can be described as ODE's and PDE's

3) Many of the basic tools in this course (interpolation, quadrature, Linear algebra, root-finding...) are the building blocks for numerical techniques for PDE's

4) The last two weeks we'll start putting them together to give you understand and a taste of things to come.
An Example problem: 1-D thermal evolution of a rod with heating and radiation.

\[
\frac{\partial u}{\partial t} = f(x) + \frac{\alpha}{2} \frac{\partial^2 u}{\partial x^2} - \beta
\]

\[
\frac{\partial^2 u}{\partial x^2} = -u(x,t)
\]

An Example problem: Simplest Steady-state 2-point boundary value problem

\[
0 = u_{xx} + f(x) - \xi \\
u(0) = \alpha \\
u(L) = \beta
\]

\[
u(x) = \frac{x^2}{2} + A
\]

\[
u(x) = x^3 + x^2 + B
\]

\[
u(x) = e^{x} - 1 + 1 = 0 \\
u(0) = A e^{0} + B e^{0}
\]

\[
u(x) = x + A + B
\]

\[
u(x) = \beta = A e^{x} + B e^{-x}
\]
Themes and variations: (adding model complexity)

1) Changing Boundary conditions: (e.g. insulating)

\[
\begin{align*}
\text{Dirichlet} & \quad u(0) = \alpha, \quad u(L) = \beta \\
\text{Neumann} & \quad J = -k \frac{\partial u}{\partial x}
\end{align*}
\]

2) Non-linear radiation term (black-body)

\[
-k \frac{\delta^2 u}{\delta x^2} + u = f(x) \quad p = 4
\]

3) Non-constant thermal diffusivity

\[
\frac{\partial}{\partial x} k(x) \frac{\partial u}{\partial x} + u = f(x)
\]

4) Time dependence

\[
\frac{\partial u}{\partial t} = \frac{\delta^2 u}{\delta x^2} + f(x) - u
\]

5) Higher spatial dimensions

\[
\frac{\partial^2 u}{\partial x^2} = \nabla \cdot (\kappa \nabla u) + f(x) - u
\]

6) Da woiks!
Solving 2-pt ODE Boundary Value Problems (BVPs)

Our basic example:

\[
\begin{align*}
-\frac{d^2u}{dx^2} + u &= f(x) \\
\text{subject to } u(a) &= 0, \quad u(b) = \beta
\end{align*}
\]

\[f(x) = Ae^{-\frac{(x-b)^2}{\sigma^2}}\]

1) Shooting method
2) Finite Difference
3) Higher order EN $\rightarrow$ Spectral or Collocation methods
4) FEM, Galerkin Finite Element methods

Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #1: Shooting Schemes

Basic idea: reuse adaptive ODE IVP schemes (but in $x...and with a catch)
Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #1: Shooting Schemes

Matlab: Need 3 functions

\[
\text{function} \quad \text{dudx} = \text{shootRHS}(x,u,\text{func}); \quad \% \text{integrand for ODE integrator}
\]

\[
\text{function} \quad \text{residual} = \text{targetShoot}(uprime,\text{func},xRange,\alpha,\beta,\text{options})
\]

\[
[t,y]=\text{ode45}(@(t,y) \text{shootRHS}(t,y,\text{func}),xRange, [\alpha \text{ uprime}],\text{options});
\]

\[
\text{residual} = y(\text{end},1) - \beta;
\]

\[
\text{function} \quad [u,x,uprime] = \text{solveBVPshoot}(xRange,\text{func},\alpha,\beta,uprime0)
\]

\[
\text{uprime} = \text{fzero}(@(y) \text{targetShoot}(y,\text{func},xRange,\alpha,\beta,\text{odeOptions}),\text{uprime0},\text{fzeroOptions})
\]

\[
[x,y]=\text{ode45}(@(t,y) \text{shootRHS}(t,y,\text{func}),xRange, [\alpha \text{ uprime}],\text{odeOptions});
\]

\[
u = y(:,1); \quad \% \text{return } u \text{ as first variable in IVP}
\]

\[
\text{uprime} = y(1,2); \quad \% \text{return value of } u'(0)
\]

Method #2: Finite Difference Schemes

Basic idea: Discretize onto mesh and approximate Derivatives using numerical differentiation
Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #2: Finite Difference Schemes

Finite Differences and Linear algebra: Boundary conditions

Theme and variations #1: Changing boundary conditions
Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #2: Finite Difference Schemes

Matlab implementation:

```matlab
function [ D1, D2, x ] = centerDifference(N,xBounds)
% CenterDifference - generates 2nd order sparse Difference matrices and coordinates for
% N evenly spaced points in the domain xBounds = [xMin xMax]
%
% [ D1, D2, x ] = centerDifference(N,xBounds)
%
% D1: first derivative Matrix 1/2h [-1 0 1];
% D2: second derivative matrix 1/h^2 [ 1 -2 1 ]
% x: uniform grid
%
% N: number of grid points
% xBounds: array holding coordinates of domain [xMin xMax]
%
% generate uniform grid
x = linspace(xBounds(1),xBounds(2),N);
x=x(:);
% make column vector
h = x(2)-x(1);
% grid spacing
% generate sparse Difference matrices
e=ones(N,1);
D1=spdiags([-e e], [-1 1],N,N)/2/h;
% First derivative matrix
D2=spdiags([ e -2*e e], -1:1,N,N)/h^2;
% second derivative matrix
% fix boundary points for consistent 1-sided derivatives
D1(1,1:3) = [-3 4 -1]/2/h;
D1(end,end-2:end) = [1 -4 3]/2/h;
D2(1,1:3) = [1 -2 1]/h/h;
D2(end,end-2:end) = [1 -2 1]/h/h;
```

Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #2: Finite Difference Schemes

Matlab implementation:

```matlab
function [A,x] = finiteDifferenceOperator(N,xBounds,bcType)
% FINITEDIFFERENCEOPERATOR - Calculates the sparse difference matrix
% for -u''+u with N points and variable Boundary conditions on the
% domain x in [xMin xMax]
%
% A = finiteDifferenceOperator(N,xBounds,bcType)
%
% N: number of grid points in domain
% xBounds: array with [xMin xMax] defining x domain
% bcType: array with flags for boundary condition types for left and right
% edges
% A: sparse matrix form of I-d_xx (uses SPDIAGS)
% x: vector of evenly spaced grid points
%
% get grid and centered difference matrices
[D1,D2,x] = centerDifference(N,xBounds);
%
% clear first and last rows of D2 for dirichlet BC's
D2([1 end],:) = zeros(2,N);
%
% set sparse finite Difference operator for -u''+u = -D2+I
A = speye(N) - D2;
%
% correct A for Neumann BC's
if (bcType(1) == 1)
A(1,:) = D1(1,:);
end
if (bcType(2) == 1)
A(end,:) = D1(end,:);
end
```
Solving 2-pt ODE Boundary Value Problems (BVPs)

Method #2: Finite Difference Schemes

Matlab implementation:

```matlab
function  [u,x] = solveBVPfd(N,xBounds,func,alpha,beta,bcType)
% get the differential operator matrix and grid
[A,x] = finiteDifferenceOperator(N,xBounds,bcType);

% evaluate the rhs function
f = func(x);
f = f(:);  % convert to column vector

% set boundary conditions
f(1)=alpha;
f(end)=beta;

% solve the linear system
u=A;
```

Finite Differences: pros and cons

Pros:
- Easy to implement (and debug)...good for quick an dirty solutions
- Easy to modify
- Relatively easy to add non-linearities
- Fast

Cons:
- low order FD requires significant number of points for accuracy
- 2nd order accuracy only guaranteed for uniform mesh
Solving 2-pt ODE Boundary Value Problems (BVPs)

Theme and variation #2: adding non-linearity

Let's change the radiation model to a generalized black-body radiation:

Method #1: Shooting --easy

Solving 2-pt ODE Boundary Value Problems (BVPs)

Theme and variation #2: adding non-linearity

Method #2: Non-linear finite difference: still replace differential operators with their discrete approximations, however the resulting equations are no longer linear