SIAM student workshop on Matlab and differential equations

Mike Sussman

January 31, 2009
Outline

Introduction

Ordinary Differential Equations (ODEs)
- Options for controlling ode solvers

Partial Differential Equations (PDEs)
- Heat equation
- Burgers’ equation
Who am I?

- Mike Sussman
- **email:** sussmanm@math.pitt.edu
  There is an “m” at the end of “sussman”.
- **Web page:** http://www.math.pitt.edu/~sussmanm
- Retired from Bettis Laboratory in West Mifflin.
- Part-time instructor at Pitt: 2070, 2071, 3040
- Interests: numerical partial differential equations, particularly the Navier-Stokes equations and applications
Objectives

- Matlab Ordinary Differential Equation (ODE) solvers and application
  - Solving ODEs with default options
  - Writing m-files to define the system
  - Advanced options
- Solving time-dependent Partial Differential Equationss (PDEs) using Matlab ODE solvers.
  - Finite-difference discretizations
  - One and two space dimension, one time dimension
Non-objective

- Will not discuss the Matlab PDE toolbox
- GUI for creating complicated mesh
- Limited set of differential equations, not including Navier-Stokes.
Start up Matlab

- Log in
- Start up Matlab
Introduction

Ordinary Differential Equations (ODEs)
Options for controlling ode solvers

Partial Differential Equations (PDEs)
Heat equation
Burgers’ equation
Initial Value Problem (IVP)

\[ \dot{u} = f(t, u) \]
\[ u(t_0) = u_0. \]

- \( u \in \mathbb{R}^n \)
- \( \dot{u} \) is shorthand for the derivative \( du/dt \)
- \textit{Explicit} because \( \dot{u} \) can be written explicitly as a function of \( t \) and \( u \)
- \textit{First-order} because the highest derivative that appears is the first derivative \( \dot{u} \)
- Higher-order equations can be written as first-order systems
- \textit{IVP} because \( u_0 \) is given and solution is \( u(t) \) for \( t > t_0 \)
Solutions

- An *analytic solution* is a formula $u(t) \in C^p$ for some $p$
- A *numerical solution* of an ODE is a table of times and approximate values $(t_k, u_k)$, possibly with an interpolation rule
- In general, a numerical solution is *always wrong*, and numerical analysis focuses on the error.
Steps for basic solution

\[ \dot{u} = f(t, u) \]
\[ u(t_0) = u_0. \]

1. Write a Matlab m-file to define the function \( f \).
2. Choose a Matlab ODE solver
Matlab ODE solvers

<table>
<thead>
<tr>
<th>Matlab ODE solvers and support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode23</td>
</tr>
<tr>
<td>ode113</td>
</tr>
<tr>
<td><strong>ode15s</strong></td>
</tr>
<tr>
<td>ode23s</td>
</tr>
<tr>
<td>ode23t</td>
</tr>
<tr>
<td>ode23tb</td>
</tr>
<tr>
<td><strong>ode45</strong></td>
</tr>
<tr>
<td>odeset</td>
</tr>
<tr>
<td>odeget</td>
</tr>
</tbody>
</table>
A first example

\[
\frac{du}{dt} = \sin t - u
\]

\[u(0) = 1\]

The Matlab m-file is `ex1_ode.m`.

```matlab
function udot=ex1_ode(t,u)
% udot=ex1_ode(t,u)
% computes the right side of the ODE du/dt=sin(t)−u
% t,u are scalars
% udot is value of du/dt

udot=sin(t)-u;

Use this command line:

ode45(@ex1_ode,[0,15],1)
```
More first example

If you want to get access to the solution values, use the following command line

\[ [t,u]=\text{ode45(@ex1_ode, [0,15], 1);} \]

You can then plot it using the normal plot commands

\[ \text{plot}(t,u) \]

or compare it with other solutions

\[ \text{plot}(t,u,t,\sin(t)) \]
Widely-different time scales
Modify example 1 to be $\dot{u} = 1000 \times (\sin(t) - u)$
Changing name of function requires changing name of file!

```
ex2_ode.m
```

Looks like $\sin t$.
```
[t,u]=ode45(@ex2_ode,[0,15],1);
plot(t,u,t,sin(t))
```

But different for first 100 time steps!
```
plot(t(1:100),u(1:100),t(1:100),sin(t(1:100)))
```
How to tell stiffness

- Easiest way is to time a non-stiff solver and a stiff solver
  \[
  \text{tic;} \quad [t45,u45]=\text{ode45}(\text{@ex2_ode},[0,15],1); \text{toc}
  \]
  \[
  [t15s,u15s]=\text{ode15s}(\text{@ex2_ode},[0,15],1);
  \]
  \[
  \text{tic;} \quad [t15s,u15s]=\text{ode15s}(\text{@ex2_ode},[0,15],1); \text{toc}
  \]

  Never time the first use of a function!

- Solution is same.
  \[
  \text{plot(t45,u45,t15s,u15s)}
  \]

- Time difference comes from number of steps
  \[
  \text{length(u45)}
  \]
  \[
  \text{length(u15s)}
  \]
High order ODEs as systems

Consider a fourth-order differential equation

\[ 5 \frac{d^4 u}{dt^4} + 4 \frac{d^3 u}{dt^3} + 3 \frac{d^2 u}{dt^2} + 2 \frac{du}{dt} + u = \sin(t) \]

The first step is to define new variables

\[ \begin{align*}
    v_1 &= u \\
    v_2 &= \frac{du}{dt} \\
    v_3 &= \frac{d^2 u}{dt^2} \\
    v_4 &= \frac{d^3 u}{dt^3}
\end{align*} \]

and write

\[ \begin{align*}
    \dot{v}_1 &= v_2 \\
    \dot{v}_2 &= v_3 \\
    \dot{v}_3 &= v_4 \\
    \dot{v}_4 &= (\sin x - v_1 - 2v_2 - 3v_3 - 4v_4)/5
\end{align*} \]
van der Pol’s equation

\[ \ddot{u} + a(u^2 - 1)\dot{u} + u = 0 \]

- Assume \( a > 0 \). We will use \( a = 1 \).
- \( u < 1 \), system behaves as negatively-damped oscillator
- \( u > 1 \), system behaves as damped oscillator
- tunnel diodes, beating heart
van der Pol solution

- **ex3_ode.m**

  function udot=ex3_ode(t,u)
  \%
  \% udot=ex3_ode(t,u)
  \%
  \% van der Pol ode with \( A=1 \)

  \[
  A=1;
  udot=[ u(2)
  \quad -A*(u(1)^2-1)*u(2)-u(1)]
  \]

- There are other ways to make column vectors.
- Not stiff. Use ode45

  \[
  [t \ u]=ode45(@ex3_ode,[0,75],[5;0])
  \]

- First component of solution is \( u \), second is \( \dot{u} \)

  plot(t,u(:,1))
Some options for odeset

<table>
<thead>
<tr>
<th>Option name</th>
<th>value</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTol</td>
<td>positive scalar or vector</td>
<td>1e-6</td>
</tr>
<tr>
<td>RelTol</td>
<td>positive scalar</td>
<td>1e-3</td>
</tr>
<tr>
<td>OutputFcn</td>
<td>function_handle</td>
<td></td>
</tr>
<tr>
<td>OutputSel</td>
<td>vector of integers</td>
<td></td>
</tr>
<tr>
<td>Stats</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>InitialStep</td>
<td>positive scalar</td>
<td></td>
</tr>
<tr>
<td>MaxStep</td>
<td>positive scalar</td>
<td></td>
</tr>
<tr>
<td>MaxOrder</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Jacobian</td>
<td>matrix</td>
<td>function_handle</td>
</tr>
<tr>
<td>JPattern</td>
<td>sparse matrix</td>
<td></td>
</tr>
<tr>
<td>Vectorized</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>Mass</td>
<td>matrix</td>
<td>function_handle</td>
</tr>
<tr>
<td>MvPattern</td>
<td>sparse matrix</td>
<td></td>
</tr>
<tr>
<td>MassSingular</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>InitialSlope</td>
<td>vector</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>function_handle</td>
<td></td>
</tr>
</tbody>
</table>
Using options

- Confusing!
  \[
  \text{ode45 (@ex3_ode, [0, 15], [5; 0])};
  \]

- Not
  \[
  \text{opt = odeset (‘OutputSel’, 1);} \\
  \text{ode45 (@ex3_ode, [0, 15], [5; 0], opt);} \\
  \]
Extra parameters

▶ **ex4_ode.m**

```matlab
function udot=ex4_ode(t,u,a)
% udot=ex4_ode(t,u,a)
% van der Pol ode with parameter = a
% default value of a is 1

if nargin < 3
    a=1;
end

udot=[   u(2)
         -a*(u(1)^2-1)*u(2)-u(1)];
```

▶ **Timing test ...**

```matlab
tic;[t,u]=ode45 (@ex4_ode,[0,750],[5;0],[],1);toc
tic;[t,u]=ode15s(@ex4_ode,[0,750],[5;0],[],1);toc
tic;[t,u]=ode45 (@ex4_ode,[0,750],[5;0],[],25);toc
tic;[t,u]=ode15s(@ex4_ode,[0,750],[5;0],[],25);toc
```
Accuracy

**Solutions must be critically evaluated!**
In this case, default options are too coarse to pick up the nonzero initial value!

```matlab
opt=odeset('AbsTol',1.e-14,'RelTol',1.e-10);
[td,ud]=ode15s(@ex4_ode,[0,200],[1.e-5;0],[] ,100);
[to,uo]=ode15s(@ex4_ode,[0,200],[1.e-5;0],opt,100);
plot(td,ud(:,1),to,uo(:,1))
```
Critical events

- **ex4_event.m**
  ```matlab
  function [value, isterminal, direction] = ex4_event(t, u, dummy)
  % [value, isterminal, direction] = ex4_event(t, u, dummy)

  isterminal = 1;
  direction = -1; % peak: derivative decreases to zero
  value = u(2); % Event is that derivative is zero
  ```

- Use any of the integrators with it
  ```matlab
  opt = odeset('OutputSel', 1, 'Events', @ex4_event);
  ode45(@ex4_ode, [0, 25], [2.5; 0], opt);
  ```

- To pick up and continue for one more cycle:
  ```matlab
  [t0, u0] = ode45(@ex4_ode, [0, 25], [2.5; 0], opt);
  [t1, u1] = ode45(@ex4_ode, [t0(end), 25], u0(end,:), opt);
  plot(t0, u0(:,1), 'b', t1, u1(:,1), 'r');
  ```
Outline

Introduction

Ordinary Differential Equations (ODEs)
  Options for controlling ode solvers

Partial Differential Equations (PDEs)
  Heat equation
  Burgers’ equation
Heat equation: continuous

- Imagine a rod of some sort of metal,
  - Part of it might be heated in some manner
  - Its ends are kept at a constant temperature
  - It starts out with some distribution of temperature

- Temperature is $u(x, t)$, $x_{\text{left}} \leq x \leq x_{\text{right}}$, and $t \geq t_{\text{initial}}$.

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left( k(u, x, t) \frac{\partial u}{\partial x} \right) + f(x, t)$$

- Boundary conditions $u(x_{\text{left}}, t) = u_{\text{left}}(t)$, $u(x_{\text{right}}, t) = u_{\text{right}}(t)$.
- Initial condition $u(x, t_{\text{initial}}) = u_{\text{initial}}(x)$. 
Spatial discretization

- $x_{\text{left}} = 0$, $x_{\text{right}} = 1$, $t_{\text{initial}} = 0$, $k = 2x + t + 1$
- $u_{\text{left}} = u_{\text{right}} = 0$
- Choose $N$, and set $\Delta x = 1/N$
- $x_n = n\Delta x$ for $n = 0, 1, \ldots, N$
- $u_n(t) \approx u(x_n, t)$.
- $\dot{u}_n = \left[ k(x_{n+1/2}, t) \left( \frac{u_{n+1} - u_n}{\Delta x} \right) - k(x_{n-1/2}, t) \left( \frac{u_n - u_{n-1}}{\Delta x} \right) \right] / \Delta x$
Matlab spatial discretization

\[ \dot{u}_n = \left[ k(x_{n+1/2}, t) \left( \frac{u_{n+1} - u_n}{\Delta x} \right) - k(x_{n-1/2}, t) \left( \frac{u_n - u_{n-1}}{\Delta x} \right) \right] / \Delta x \]

\[ dx = 1/N; \]
for \( n=1:N-1 \)
  kright = (2*(x(n)+dx/2)+t+1);
  kleft = (2*(x(n)-dx/2)+t+1);
  if \( n==1 \) \% left
    udot(n,1) = (kright*(u(n+1)-u(n))-kleft*(u(n)-uleft))/(dx^2);
  elseif \( n<N-1 \) \% interior
    udot(n,1) = (kright*(u(n+1)-u(n))-kleft*(u(n)-u(n-1)))/(dx^2);
  else \% right
    udot(n,1) = (kright*(uright-u(n))-kleft*(u(n)-u(n-1)))/(dx^2);
  end
end

Full code is in ex5_ode.m
Heat equation results

$$[t,u]=\text{ode15s(@ex5_ode, [0, .1, .2, .3, .4, .5], 100*\text{ones}(99,1))}$$
figure(1)
for k=1:6
    plot(u(k,:))
    hold on
end
hold off
title('Temperature distribution at several times')
figure(2)
plot(u(:,50))
xlabel('time')
ylabel('temperature')
title('Temperature vs. time in the middle')
Burgers’ equation

\[ \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \]

With \( u_{\text{left}} = 1 \) and \( \frac{\partial u}{\partial x} \bigg|_{\text{right}} = 0 \) and \( u \in [0, 1] \).

\[ \frac{du_n}{dt} = \nu \frac{u_{n+1} - 2u_n + u_{n-1}}{\delta x^2} - u_n \frac{u_{n+1} - u_{n-1}}{2\Delta x} \]

With \( u_{\text{left}} = 1 \) and \( \frac{\partial u}{\partial x} \bigg|_{\text{right}} = 0 \) approximated by \( u_{\text{right+}} = u_N \).
Note: Now there is an $x_N$ because of the new boundary condition.

dx=1/N;
x=(1:N)*dx;
for n=1:N
    if n==1 %left
        udot(n,1)=nu*(u(n+1)-2*u(n)+uleft )/dx^2- ...
        u(n)*(u(n+1)-uleft )/(2*dx);
    elseif n<N-1 %interior
        udot(n,1)=nu*(u(n+1)-2*u(n)+u(n-1))/dx^2- ...
        u(n)*(u(n+1)-u(n-1))/(2*dx);
    else %right
        % approximate Neumann b.c.
        uright=u(n);
        udot(n,1)=nu*(uright-2*u(n)+u(n-1))/dx^2- ...
        u(n)*(uright-u(n-1))/(2*dx);
    end
end
Running ex6

- $\nu = .001$ solution is relatively smooth.
  
  \[
  \begin{align*}
  N &= 500; \\
  \text{init} &= (1 - \text{linspace}(0, 1, N))^3; \\
  \nu &= .001; \\
  [t, u] &= \text{ode45}(@\text{ex6}_\text{ode}, 0:.01:1.5, \text{init}, [], \nu);
  \end{align*}
  \]

- See the wave steepen as it moves
  
  \[
  \text{for } k = 1: \text{length}(u(:, 1)); \text{plot}(u(k, :)); \\
  \text{axis}([0, N, 0, 2]); \text{pause}(0.1); \text{end}
  \]

- $\nu = .0001$ causes numerics to break down
  
  \[
  \begin{align*}
  \nu &= .0001; \\
  [t, u] &= \text{ode45}(@\text{ex6}_\text{ode}, 0:.01:1.5, \text{init}, [], \nu);
  \end{align*}
  \]

- See the oscillations grow
  
  \[
  \text{for } k = 1: \text{length}(u(:, 1)); \text{plot}(u(k, :)); \\
  \text{axis}([0, N, 0, 2]); \text{pause}(0.1); \text{end}
  \]