

## Outline

### SIAM student workshop on Matlab and differential equations

Mike Sussman

January 31, 2009

#### 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](mailto: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 Equations (PDEs) using Matlab ODE solvers.
  - ▶ Finite-difference discretizations
  - ▶ One and two space dimension, one time dimension



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

- ▶ Log in
- ▶ Start up Matlab



Outline

Initial Value Problem (IVP)

Introduction

Ordinary Differential Equations (ODEs)  
Options for controlling ode solvers

Partial Differential Equations (PDEs)  
Heat equation  
Burgers' equation

$$\begin{aligned} \dot{u} &= f(t, u) \\ u(t_0) &= u_0. \end{aligned}$$

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



- ▶ 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 focusses on the error.

$$\begin{aligned} \dot{u} &= f(t, u) \\ u(t_0) &= u_0. \end{aligned}$$

1. Write a Matlab m-file to define the function  $f$ .
2. Choose a Matlab ODE solver



Matlab ODE solvers

Matlab ODE solvers and support	
ode23	non-stiff, low order
ode113	non-stiff, variable order
ode15s	stiff, variable order, includes DAE
ode23s	stiff, low order
ode23t	trapezoid rule
ode23tb	stiff, low order
ode45	non-stiff, medium order (Runge-Kutta)
odeset	sets options for all ODE solvers
odeget	gets current options

A first example

$$\begin{aligned} \frac{du}{dt} &= \sin t - u \\ u(0) &= 1 \end{aligned}$$

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

```
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]=ode45(@ex1_ode,[0,15],1);
```

You can then plot it using the normal plot commands

```
plot(t,u)
```

or compare it with other solutions

```
plot(t,u,t,sin(t))
```



## A stiff example

- ▶ Widely-different time scales
- ▶ Modify example 1 to be  $\dot{u} = 1000 * (\sin(t) - u)$
- ▶ 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

```
tic;[t45,u45]=ode45(@ex2_ode,[0,15],1);toc
[t15s,u15s]=ode15s(@ex2_ode,[0,15],1);
tic;[t15s,u15s]=ode15s(@ex2_ode,[0,15],1);toc
```

Never time the first use of a function!

- ▶ Solution is same.

```
plot(t45,u45,t15s,u15s)
```

- ▶ Time difference comes from number of steps

```
length(u45)
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

$$\begin{aligned}
 v_1 &= u \\
 v_2 &= du/dt \\
 v_3 &= d^2u/dt^2 \\
 v_4 &= d^3u/dt^3
 \end{aligned}$$

and write

$$\begin{aligned}
 \dot{v}_1 &= v_2 \\
 \dot{v}_2 &= v_3 \\
 \dot{v}_3 &= v_4 \\
 \dot{v}_4 &= (\sin x - v_1 - 2v_2 - 3v_3 - 4v_4)/5
 \end{aligned}$$



# 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)
      -A*(u(1)^2-1)*u(2)-u(1)];
```

- ▶ There are other ways to make column vectors.
  - ▶ Not stiff. Use ode45
  - ▶ First component of solution is  $u$ , second is  $\dot{u}$
- ```
plot(t,u(:,1))
```



# Options: odeset

| Some options for odeset |                           |         |
|-------------------------|---------------------------|---------|
| Option name             | value                     | default |
| AbsTol                  | positive scalar or vector | 1e-6    |
| RelTol                  | positive scalar           | 1e-3    |
| OutputFcn               | function_handle           |         |
| OutputSel               | vector of integers        |         |
| Stats                   | on   off                  | off     |
| InitialStep             | positive scalar           |         |
| MaxStep                 | positive scalar           |         |
| MaxOrder                | 1   2   3   4   5         | 5       |
| Jacobian                | matrix   function_handle  |         |
| JPattern                | sparse matrix             |         |
| Vectorized              | on   off                  | off     |
| Mass                    | matrix   function_handle  |         |
| MvPattern               | sparse matrix             |         |
| MassSingular            | yes   no   maybe          | maybe   |
| InitialSlope            | vector                    |         |
| Events                  | function_handle           |         |



# Using options

## ▶ Confusing!

```
ode45(@ex3_ode,[0,15],[5;0]);
```

## ▶ Not

```
opt=odeset('OutputSel',1);
ode45(@ex3_ode,[0,15],[5;0],opt);
```



## Extra parameters

► [ex4\\_ode.m](#)

```
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 ...

```
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!

```
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](#)

```
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

```
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:

```
[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





ex6\_ode.m

**Note:** Now there is an  $x_N$  because of the new boundary condition.

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

With  $u_{\text{left}} = 1$  and  $\frac{\partial u}{\partial x}|_{\text{right}} = 0$  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  $\frac{\partial u}{\partial x}|_{\text{right}} = 0$  approximated by  $u_{\text{right}+} = u_N$ .

```

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
end

```

Navigation icons: back, forward, search, etc.

Navigation icons: back, forward, search, etc.

## Running ex6

- ▶  $\nu = .001$  solution is relatively smooth.

```

N=500;
init=(1-linspace(0,1,N)).^3;
nu=.001;
[t,u]=ode45(@ex6_ode,0:.01:1.5,init,[],nu);

```

- ▶ See the wave steepen as it moves

```

for k=1:length(u(:,1));plot(u(k,:));
axis([0,N,0,2]);pause(.1);end

```

- ▶  $\nu = .0001$  causes numerics to break down

```

nu=.0001;
[t,u]=ode45(@ex6_ode,0:.01:1.5,init,[],nu);

```

- ▶ See the oscillations grow

```

for k=1:length(u(:,1));plot(u(k,:));
axis([0,N,0,2]);pause(.1);end

```

Navigation icons: back, forward, search, etc.