12.010 Computational Methods of Scientific Programming

Lecturers
Thomas A Herring
Chris Hill

Overview Today

- Solution of ordinary differential equations with Mathematica and Matlab.
- Examine formulations:
  - Mathematica 2-nd order (and higher order) ODE can be directly solved with NDsolve
  - Matab solved first order differential equations and so second order equations need to be reduced to pairs of first order equations.
- Both program allow specific results to be found such as a zero crossing.

Codes used in today's lecture

- Mathematic notebook
  http://geoweb.mit.edu/~tah/12.010/Lec19_NDsolve.nb
- Matlab solutions are
  http://geoweb.mit.edu/~tah/12.010/Lec19_ODE.m
  http://geoweb.mit.edu/~tah/12.010/Lec19_animate.m
  http://geoweb.mit.edu/~tah/12.010/Lec19_hit.m
  http://geoweb.mit.edu/~tah/12.010/Lec19_bacc.m

Equations to be solved

- Problem: High flying ballistic missile where changes in gravity with height and drag are important. The boundary conditions are initial velocity and launch angle which need to be set to hit a target at a specified distance.
- Two parts to this problem:
  - Solving a pair of second order differential equations and determining a precise end to the trajectory.
  - Once this is solved, an iteration can be set up to work out what initial velocity and launch angle is needed to reach target distance.

Differential equations

- Basic equation to solve is

$$\ddot{x} - k(\dot{x})^2 - hx - g = 0$$

where *x* is position vector, superscript double is second time derivative, superscript dot is first time derivative, *k* is drag force coefficient, *h* is coefficient which shows an acceleration dependent on position and *g* is constant acceleration.

- If k and h are zero, then this equation can be solve analytically.

More basic equations

- We solve the problem in 2-D so that vector x has components x and z.
- Equation for gravity

$$F = \frac{GMm}{r^2} = \frac{GMm}{(R+h)^2} \approx \frac{GMm}{R^2}(1-2h/R) = (9.806 - 3.0784 \times 10^{-6} h)m$$

- Drag effects $\mathbf{F}_d = -\frac{1}{2} C_d \rho \mathbf{V}^2 A \hat{\mathbf{V}}$

where V is velocity, ρ is density of air (1.29 kg/m^3 with an exponential decay height of 7.5 km)

Mathematica Setup

- To solve this problem in Mathematic use NDSolve.
- solution = NDSolve[
  {z''[t] == grav[z[t]] + dragz[cd, x'[t], z'[t], z[t]],
  x''[t] == dragx[cd, x'[t], z'[t], z[t]],
  x[0] == 0, z[0] == 0,
  x'[0] == vx, z'[0] == vz},
  {x, z}, {t, 0, 1000}];
  hz[t_] := Evaluate[z[t] /. solution];
  hx[t_] := Evaluate[x[t] /. solution];
- First two equations are 2nd order differential equations to solve (z'' and x''); grav and dragz/x are functions); the terms below this are boundary conditions. Last trwo entries are one way to access the values of the solution (i.e., hz[100] will return value of z at time 100 seconds.

Mathematic setup

- The functions needed here are:
  Gravity here depends on height (x - coordinate) *)
  grav[z_] := -9.806 + 3.0786 10^(-6) z;
  (* Drag also depends on height because the air density decreases with height *)
  dragz[cd_, xd_, zd_,
    z_] := -(1.29*Exp[-z/(7500.)]*Sqrt[xd^2 + zd^2]*zd*cd*
      xarea)/(2 mass);
  dragx[cd_, xd_, zd_,
    z_] := -(1.29*Exp[-z/(7500.)]*Sqrt[xd^2 + zd^2]*xd*cd*
      xarea)/(2 mass);
- The cd_ (drag coefficient is used in the functions so that Manipulate[] can be used to generate dynamic plots.
- The Lec19_NDsolve.nb note book implements this solution

Additional Mathematica feature

- In this problem we want to hit a specific target distance and to do that we use FIndRoot
- The solution on the earlier slides provides values of x and z as a function of time; we now want to find the time when z is zero again (any height could be chosen) and then we change the initial velocity so that x is a specific value at this value of z.
- This is done with:
  zerot = t /. FindRoot[hz[t] == 0 , {t, 20, 500}];
  derr = targdist - First[hx[zerot]];
  (First[] is needed here because hx[t] returns a list (which in this case contains just 1 item).

Matlab setup

- The Matlab solution to this problem is a little different because Matlab solvers only solve first-order differential equations, so we need to repose a 2nd order equation as two first order one.

$$x''(t) = f(x'(t), x(t), C) \text{ becomes 2 equations with}$$
$$x''(t) = y(t)$$
$$y''(t) = f(y(t), x(t), C)$$

With Matlab we solve for y(t) and x(t). In our case these are vector quantities.

Matlab setup

- The ODE solvers in Matlab take a vector containing the variables to be solved, the initial values of the vector contain the initial values (boundary conditions at time zero). We call this vector y.
- For the ballistic problem with 2 2nd order equations and hence 4 1st order equations, y is 4 elements long.
- An m-file function is supplied that given the the vector y, returns dy/dt. Other parameters are often needed for this calculation such gravity, drag coefficients, area etc and these can be passed into to m-file or by declared global (easiest approach in general).
- In our case the vector y is: y(1) - x-position; y(2) - z-position, y(3) - x velocity and y(4) - z velocity;
  dy/dy called dy is dy(1) - x velocity (y(3)); dy(2) - z velocity (y(4)); dy(3) - x acceleration (drag) and dy(4) - z acceleration (gravity and drag).

ODE solvers

- In addition to the acceleration m-file, and m-file can also be specified that allows events to be detected.
- In our case here that event is the missile hitting the ground again (ie., the height becoming zero).

```
function
  [value,isterminal,direction]=Lec19_hit(t,y)
% Locate the time when height passes through zero
% in a decreasing direction
% and stop integration.
value = y(2);      % detect height = 0
isterminal = 1;    % stop the integration
direction = -1;    % negative direction
```

ODE Solver

- The name of the event function and other characteristics of the ODE solution are set with the odeset command
- These options allow the tolerances on the solution accuracy to be set. These can be set as relative or absolute accuracy.
- There are a number of ODE solvers that use different order of integration and some are posed to solve stiff problems (i.e., problems where solution vary slowly but can have nearby solutions that vary rapidly. These problems need to careful with the size of step they take to avoid unstable results and to run rapidly.

ODE Solvers

- Use of ODE Solvers in Matlab (demonstrated in class)
- Vector y is 2-d position and velocity (1:4).

```
y0 = [0.0; 0.0; vx; vz];
[t,y,te,ye,ie] = ode23(@Lec19_bacc,[0:1:tmax],y0,options);
```

  - The Lec19_bacc routine computes accelerations. dy/dt is returned so that dy[1]=d(pos)/dt=y[3]; dy[2]=y[4]; and dy[3] and dy[4] are new accelerations

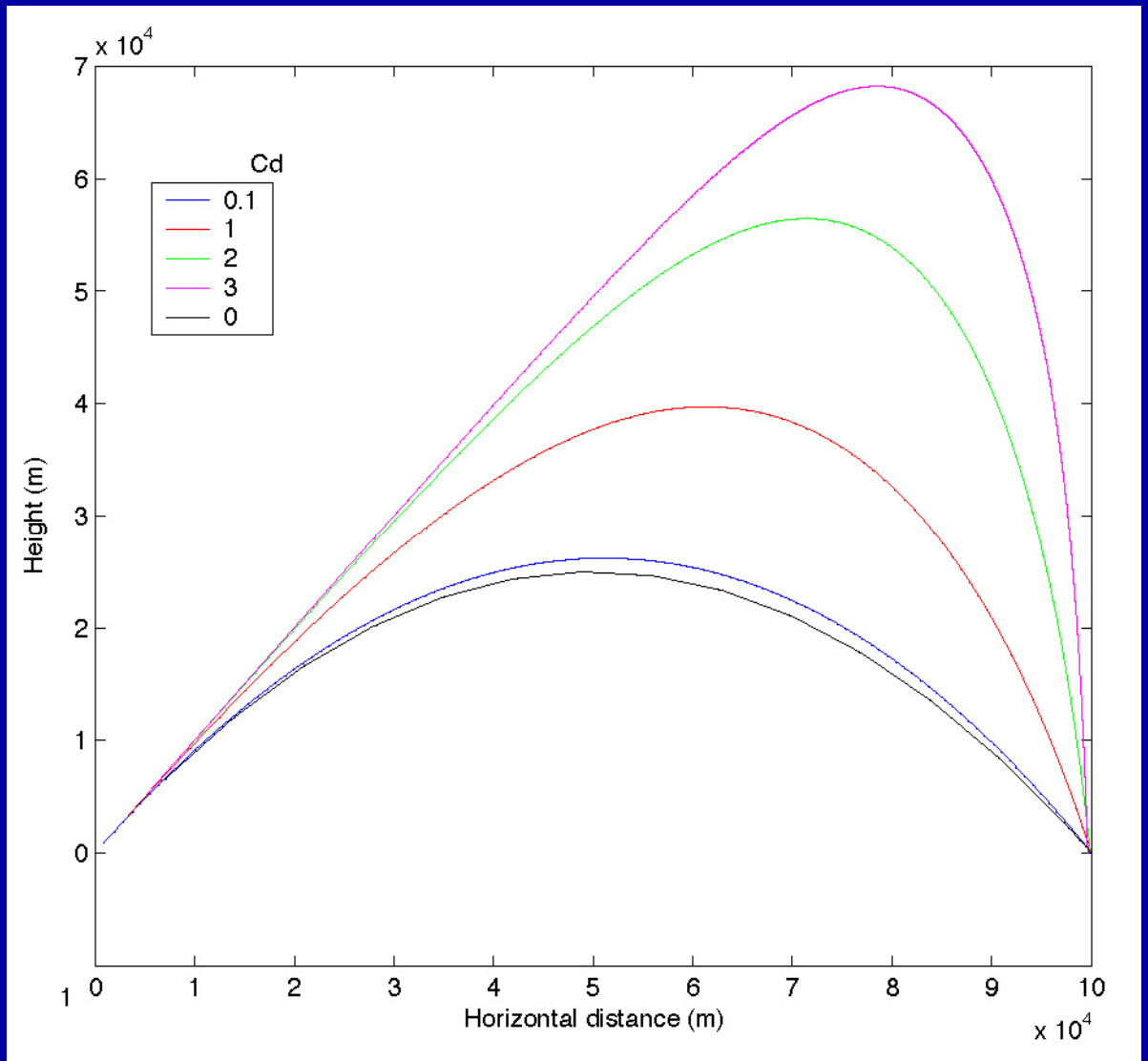
```
function dy = Lec19_bacc(t, y)
% Lec19_bacc: Computes ballistic accelerations
```

  - Options sets ability to detect event such as hitting ground

```
options = odeset('AbsTol',[terr 1 1 1],'Events',' Lec19_hit');
function [value,isterminal,direction] = Lec19_hit(t,y)
Value returns the height.
```

  - Look through Matlab help and use demo program
  - Solutions in Lec19_ODE.m, Lec19_bacc.m, Lec19_hit.m and Lec19_animate.m

Example solutions



Here Cd is changed and effects on trajectory can be seen

Summary

- Examined solution of differential equations using NDsolve and Findroots in Mathematica
- Using ODExx in Matlab and the options that allow events to be dected.
- Example of multiple events is given with ballode command in Matlab

12.010 Computational Methods of Scientific Programming
Fall 2011