lesson4

Contents

LESSON 4 1

THE PROBLEM WE WANT TO SOLVE 1

PUT THE SYSTEM OF ODE IN A FUNCTION GALLILEO 2

SOLVE THE SYSTEM USING ODE45 4

ADDITIONAL REMARKS 6

End lesson 4 6

LESSON 4

Related Assignments:
Preread + participation activities:
After class challenge activities:
Also: hw4c

```
% reduce needless whitespace
format compact
% reduce irritations
more off
% start a diary
diary lectureN.txt
```

THE PROBLEM WE WANT TO SOLVE

We want to solve Galileo's problem of dropping iron spheres from a tall building and seeing how fast they fall.
Without air resistance, all spheres would reach the ground in the same time. This time follows from the relation

```
 _s_ = 0.5 _g_ _t_^2
```

The distance traveled s is the height of the tower of Pisa, 60 m. The acceleration of gravity g is about 9.81 m/sˆ2. Putting in these numbers gives the time for the sphere to reach the ground as 3.5 s.
With air resistance included, the above simple formula is no longer correct. Now we must solve Newton's second law to find the velocity v and from that the distance s traveled. The two equations to solve are

```
d_s_/d_t_ = v
d_v_/d_t_ = (FGravity - FAir)/_m_
```

where the second equation is Newton's second law for the sphere after dividing by the mass. The above two equations are "a system of first order differential equations" (ODE). "Differential equations" because there are detivatives in it. "First order" because there are no second or higher order derivatives in them. "System" because there is more than one equation.
Note that if you replace v by ds/dt you get

```
d^2_s_/d_t_^2 = (FGravity - FAir)/_m_
```

which is a single second order differential equation. But most numerical software, including the relevant Matlab function ode45, solves only first order equations, not second order ones. So the system it is.
To solve the two equations, we must first note that the mass of a iron sphere of radius r is

```
m = (4/3) pi _r_^3 rhoIron
```

while its "frontal area" is

```
A = pi _r_^2
```

where rhoIron is the density of iron, 7,820 kg/mˆ3. Also, the force of gravity is given by

```
FGravity = _m__g_
```

while the drag force exerted by the air is

```
FAir = Cd A 0.5 rhoAir _v_^2
```

Here rhoAir is the density of air, 1.225 kg/mˆ2 under standard sea-level conditions. Also, Cd is the so-called drag coefficient, which can be assumed to be 0.5 as long as the diameter of the sphere is not much more than 10 cm.
To solve a system of first order ODE, we must put the unknowns, here s and v, in a vector. Calling the vector 'unknowns', its first component unknowns(1)=s and its second component unknowns(2)=v.

PUT THE SYSTEM OF ODE IN A FUNCTION GALLILEO

```
function unknownsDot = Galileo(t,unknowns,r)

% Function that describes the ordinary differential
% equations governing Gallileo's falling iron spheres.
%
% Input: t: the time since the start of the fall.
%        unknowns: vector with two components:
%            unknowns(1): the distance 's' that the
%                         sphere has traveled down.
%            unknowns(2): the downward velocity 'v' of
%                         the sphere.
%        r: radius of the iron sphere.
%
% Output: unknownsDot: the time derivatives of the
%         unknowns, to be used by function ode45:
%            unknownsDot(1) = ds/dt = v
%            unknownsDot(2) = dv/dt = (FGravity - FAir)/m
%         where FGravity is the force of gravity, FAir
%         the force of air resistance, and m the mass of
%         the iron sphere.

% take s and v out of unknowns for readability
s=unknowns(1);
v=unknowns(2);

% acceleration of gravity
g=9.81;

% density of air at sea level
rhoAir=1.225;

% density of iron
rhoIron=7272;

% mass of the iron sphere
m=(4/3)*pi*r^3*rhoIron;

% frontal area of the iron sphere
A=pi*r^2;

% approximate drag coefficient of a normal size sphere
Cd=0.5;

% force of gravity
```

```
FGravity=m*g;

% force of air resistance
FAir=Cd*A*0.5*rhoAir*v^2;

% derivative ds/dt
dsdt=v;

% derivative dvdt
dvdt=(FGravity-FAir)/m;

% return them as a *column* vector
unknownsDot=[dsdt dvdt]';

end
```

SOLVE THE SYSTEM USING ODE45

First we must select a starting vector y0 at time 0. Here that is easy, as both the initial distance traveled s and the initial velocity v are zero.
Next we call ode45 and tell it to find the solution for times greater than zero up to 3.5 seconds.

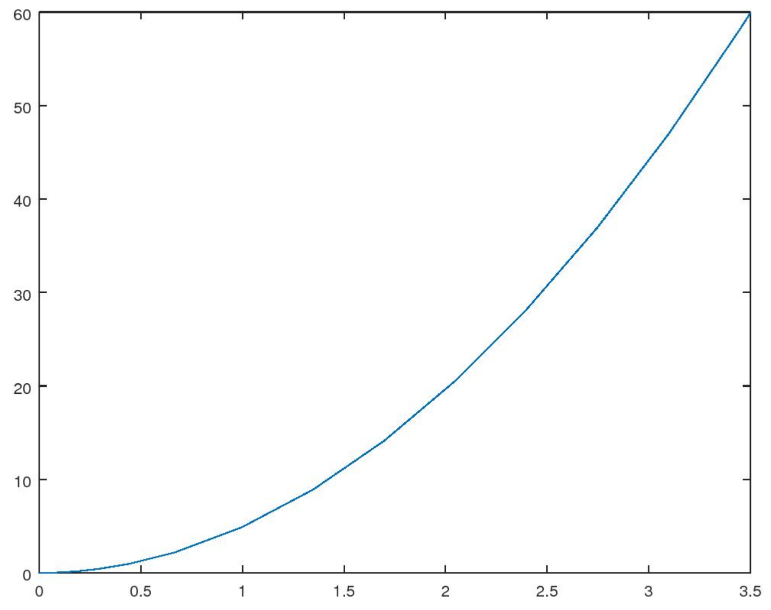
```
% the distance in 3.5 seconds without air resistance
sNoAir=0.5*9.81*3.5^2

% set the initial condition
unknowns0=[0 0]';

% try a 20 cm radius
r=0.2
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],unknowns0);
plot(tValues,unknownsValues(:,1))
hold on
% print out the final value
unknownsValues(end,1)

% try a 10 cm radius
r=0.1
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
plot(tValues,unknownsValues(:,1))
hold on
```

```
% print out the final value
unknownsValues(end,1)

% try a 5 cm radius
r=0.05
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
plot(tValues,unknownsValues(:,1))
% print out the final value
unknownsValues(end,1)

% try a 1 cm radius
r=0.01
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
plot(tValues,unknownsValues(:,1))
% print out the final value
unknownsValues(end,1)
```

```
error: 'y0' undefined near line 2 column 62
  in:
```

```
% the distance in 3.5 seconds without air resistance
sNoAir=0.5*9.81*3.5^2

% set the initial condition
unknowns0=[0 0]';

% try a 20 cm radius
r=0.2
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],unknowns0);
plot(tValues,unknownsValues(:,1))
hold on
% print out the final value
unknownsValues(end,1)

% try a 10 cm radius
r=0.1
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
```

```
plot(tValues,unknownsValues(:,1))
hold on
% print out the final value
unknownsValues(end,1)

% try a 5 cm radius
r=0.05
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
plot(tValues,unknownsValues(:,1))
% print out the final value
unknownsValues(end,1)

% try a 1 cm radius
r=0.01
% call ode45 to find the solution to t=3.5
[tValues, unknownsValues] = ode45(@(t,y) Galileo(t,y,r)
    ,[0 3.5],y0);
plot(tValues,unknownsValues(:,1))
% print out the final value
unknownsValues(end,1)
```



ADDITIONAL REMARKS

If the system of first order differential equations describes, say, a set of chemical reactions, there may be a problem with using ode45. Typically, some reactions proceed very quickly and others much more slowly. The slow reactions imply that you have to solve the evolution for a relatively long time. But ode45 must compute accurately over the shortest time scales in order not to get the fast reactions all wrong. Having to compute accurately over very many short time intervals is a problem for ode; the computation may take excessive computational time.
Such a problem, and any other problem where there is a very large spread in typical time scales, is called "stiff". For stiff problems you want to use a solver dedicated to such problems. One basic one provided by Matlab is ode15s.

End lesson 4