CMPS 251 - Assignment 10

Kinan Dak Al Bab

November 23, 2014

This assignment was solved using Octave. Matlab is unavailable in the labs at the moment.

1 Stability

1.1 Question

Consider the problem

$$y' = -10y, y(0) = 1; \text{for } 0 \leq t \leq 10$$

- Integrate the ODE with the explicit forward Euler method using h = 0.25 and h = 0.1. Comment and explain the difference in behavior.
- Integrate the ODE with the implicit backward Euler method using the same values of h above. Compare with the solutions above and comment.
- Comment, in a sentence, on the relative merits of explicit vs. implicit methods for integrating ODEs.

1.2 Code

```
%% FORWARD EULER
y1 = [1];
y2 = [1];

t1 = 0:0.25:10; % h1 = 0.25
t2 = 0:0.1:10; % h2 = 0.1

l1 = length(t1) - 1;
l2 = length(t2) - 1;
for i = 1:l2
    if i ≤ l1
        y1 = [y1 y1(i) + 0.25 * (-10 * y1(i))];
    end

    y2 = [y2 y2(i) + 0.1 * (-10 * y2(i))];
end

y1(l1+1)
y2(l2+1)
```

```
%% BACKWARDS EULER
y1 = [1];
y2 = [1];

t1 = 0:0.25:10; % h1 = 0.25
t2 = 0:0.1:10; % h2 = 0.1

l1 = length(t1) - 1;
l2 = length(t2) - 1;
for i = 1:l2
    if i ≤ l1
        y1 = [y1 y1(i) / (1 - 0.25 * - 10)];
    end

    y2 = [y2 y2(i) / (1 -  0.1 * -10)];
end

y1(l1+1)
y2(l2+1)
```

1.3 Output

```
ans =  1.1057e+07 % forward for h = 0.25, 0.1
ans = 0

ans =  1.7269e-22 % backwards for h = 0.25, 0.1
ans =  7.8886e-31
```

1.4 Comment

The actual solution should be close to zero (from wolframalpha solver).

- In the first Part, we are using the Forward Euler. As seen in class, we have a condition on h to make this method stable for this form of ODEs:

$$y_{i+1} = (1 + \lambda h)y_i$$
$$|1 + \lambda h| \leq 1$$
$$h \leq \frac{2}{|\lambda|} \implies h \leq 0.2$$

  For $h = 0.25$ that condition doesn't hold and therefore the result we get has a huge magnification in error. due to instability of forward euler.

  For $h = 0.1$ we notice that we get the solution to be zero, which is due to the fact that h satisfies the stability condition and therefore the errors are no longer magnified.

- In the second part, we are using the backwards Euler, which again as seen in class gives us unconditional stability for $\lambda \leq 0$ (-10 in our case). which is why we get close results to zero, the accuracy improves as we make h smaller (truncation error decrease).

Forward Euler is faster and less "Complicated" but is only stable for certain h depending on the problem (sometimes that h can be unpracticale), Backwards Euler is more expensive computationally and more complicated (involves solving a non linear equation, in our case it was linear due to luck) but it gives us better stability.

2 Fall with parachute

2.1 Question

A paratrooper weighing m = 80 kg is dropped from an airplane at a height of 600 m. After 5s the parachute opens. The paratrooper's height as a function of time, y(t) is the solution of the IVP:

$$my'' = -mg + \alpha(t), y(0) = 600m, y'(0) = 0$$

Where g is the acceleration due to gravity and the air resistance α is proportional to the square of the velocity, with different proportionality constants before and after the chute opens:

$$\alpha(t) = \begin{cases} K_1`y'^2 & \text{if } t < 5s \\ K_2`y'^2 & \text{if } x \geq 5s \end{cases}$$

- Obtain the analytical solution for free-fall (i.e., no air resistance, $K_1 = K_2 = 0$) and plot the height vs time. Find:
  - the height at which the parachute opens
  - how long it takes to reach the ground
  - the impact velocity
- Consider the case K1 = 1/15 and K2 = 4/15. Obtain a numerical solution using an appropriate integrator from the matlab ODE suite. Plot the height vs time and compare it with the free-fall solution. At what height does the chute open? How do the time-to-ground and impact velocity change? Is this expected?

2.2 Analytical Solution

So we know that the solution of this ODE is:

$$y(t) = c_1 + c_2 t - \frac{g}{2}t^2$$

We can determine c_1 and c_2, by replacing t with values 0, 1. we get that c_1 is the original height and c_2 is the original velocity (equals to zero). Also $g \neq 9.8$

$$y(t) = -\frac{9.8}{2}t^2 + 600$$

By Differentiating we also get the equation for velocity

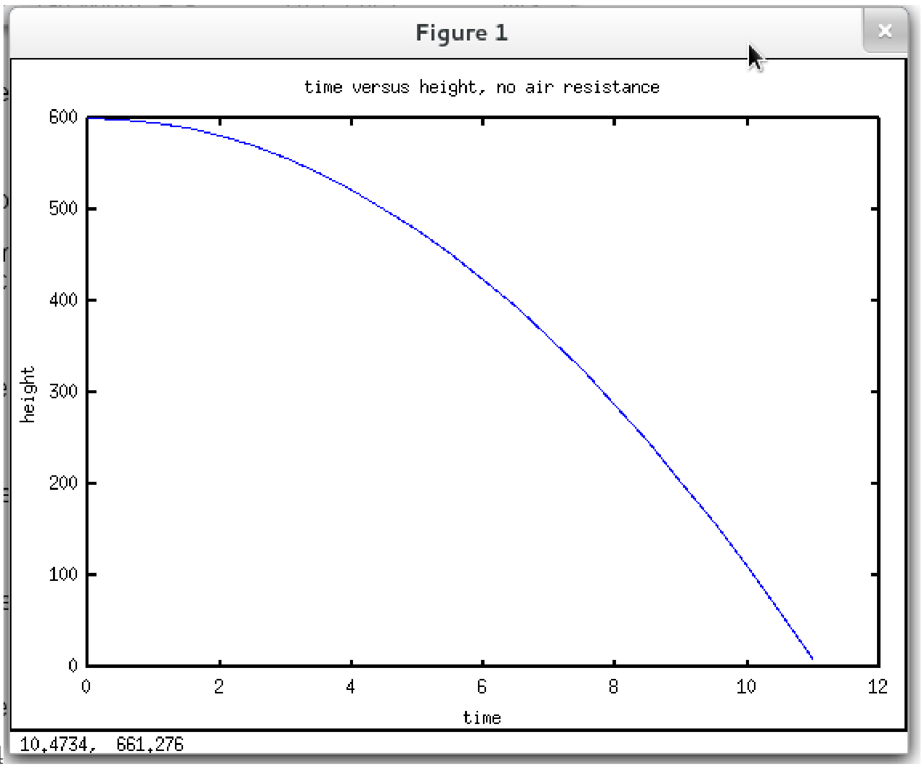
$$v(t) = v_0 - gt = -9.8t$$

2.3 Code

```
y = @(t)(- (9.8 / 2) * t^2 +  600);

para_h = feval(y, 5) % Height at which parachute opens.
t_impact = fzero(y, 5) % Time until impact
v_impact = - 9.8 * t_impact % Velocity at time of impact

ys = [];
for t = 0:0.5:11
    ys = [ys feval(y, t)];
end

plot(0:0.5:11, ys)
title('time versus height, no air resistance');
xlabel('time');
ylabel('height');
```

2.4 Output

```
para_h =  477.50
t_impact = -11.066
v_impact =  108.44
```

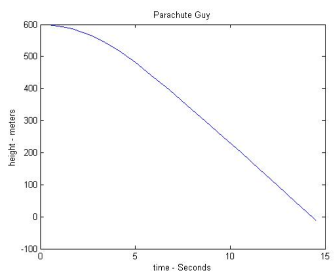


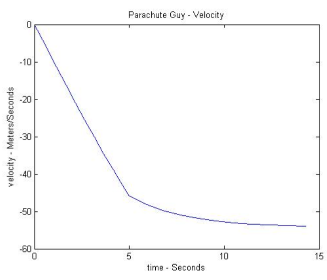
/4/4

2.5 Code - part 2

```
f0 = @(t,y)[y(2); -9.8  + (1/15) * y(2)^2 / 80];
f1 = @(t,y)[y(2); -9.8  + (4/15) * y(2)^2 / 80];

t0 = [0 5];
t1 = [5 14.3];

[T1 Y1] = ode23(f0, t0, [600; 0]);
[T2 Y2] = ode23(f1, t1, Y1(length(T1), 1:2)' );

T = [T1; T2];
Y = [Y1; Y2];

plot(T, Y(1:length(T), 1));
title('Parachute Guy');
xlabel('time - Seconds');
ylabel('height - meters');

plot(T, Y(1:length(T), 2));
title('Parachute Guy - Velocity');
xlabel('time - Seconds');
ylabel('velocity - Meters/Seconds');

Y1(length(T1), 1) % height in which parachute opens
Y2(length(T2), 2) % velocity near the ground
```

2.6 Output and plot





```
ans = 481.4550 % height in which parachute opens
ans = -53.9127 % velocity near the ground
```

2.7 Comment

Notice How the velocity slows down over all, the velocity is a bit slower before opening the parachute because of the extra 1/15 constant for air resistance, and after the parachute the increase of velocity slows down even more eventually reaching a stable level. This is due to the decrease in acceleration due to the extra force against gravity (air resistance).

3 Stiff ODEs

3.1 Question

If you light a match, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface. Flame propagation may be modeled by the following ODE

$$y' = y^2 - y^3, y(0) = \gamma, 0 \leq t \leq \frac{2}{\gamma}.$$

y(t) represents the radius of the ball. The terms y^2 and y^3 come from the surface area and the volume. The critical parameter is the initial radius,γ which is “small”. We seek the solution over a length of time that is inversely proportional to γ.
This problem becomes stiff as the solution approaches steady-state. When the problem is stiff, explicit methods will need to take very tiny time steps while implicit methods can use much larger ones to achieve the same accuracy. The goal of this problem is to observe the behavior of an explicit and an implicit method on this nonlinear problem.

- With $\gamma = 10^{-4}$, use $ode45$ to solve the problem to a relative error of 10^{-4}, Use $odeset(RelTol, 1.e-4)$ and pass the returned value to $ode45$. Plot the solution and zoom in on it near where it first approaches steady state. Comment.
- Repeat the solution with the implicit solver $ode23s$. Compare the number of time steps needed to obtain the same accuracy. Comment.

3.2 Code

```
f = @(t,y)[y^2 - y^3];
t = 2/(10^-4); % 2 / gamma

[T1 Y1] = ode45(f, [0 t], [10^-4], odeset('RelTol',1e-4));
[T2 Y2] = ode23s(f, [0 t], [10^-4], odeset('RelTol',1e-4));


length(T1)
length(T2)

plot(T1, Y1, T2, Y2);

title('Stiff Stuff');
xlabel('time');

xlim ([9750 10050]) % To zoom in on the thing
```

3.3 Output

```
ans = 12161 % steps by ode45
ans = 100 % steps by ode23s
```

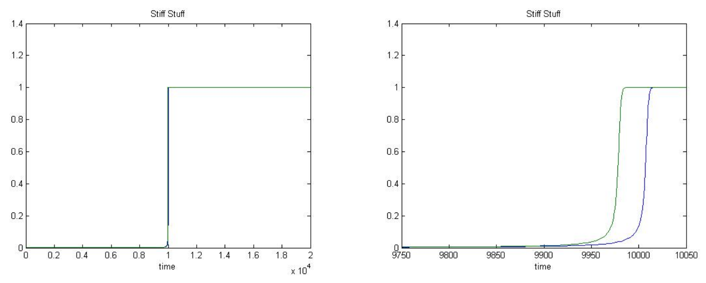


Figure 1: Blue : ode45, Green ode23s

3.4 Comment

To reach the same accuracy ode23s used barely 100 steps, however ode45 used more than 10000, this is because stiff solvers are implicit and have more stability, while explicit solvers need to go for smaller steps to gain stability.

4 Least Squares

4.1 Question

Consider the following set of 51 points:

$$t = 0 : 0.01 : 1;$$

$$y = cos(pi * t) + 0.8 * sin(2 * pi * t)$$

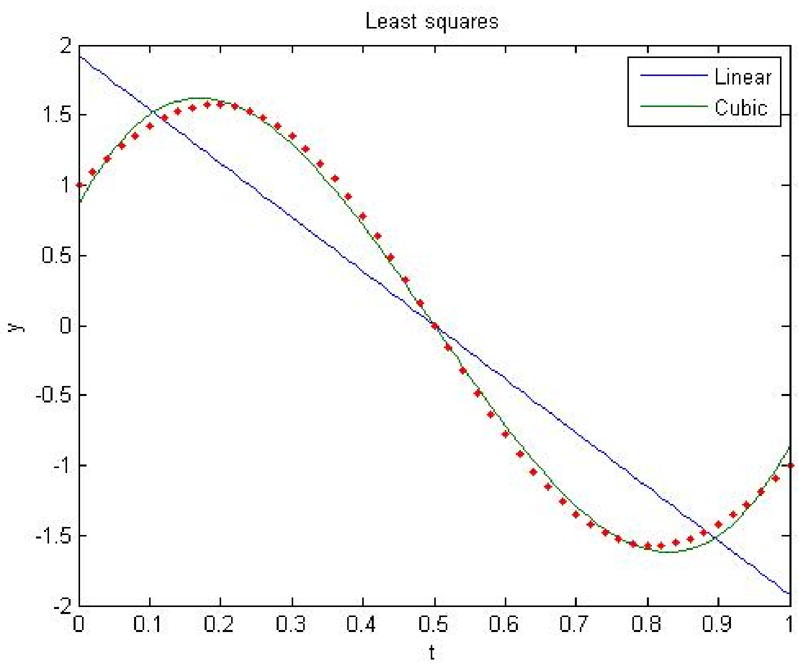
- Generate a least squares straight line fit to the data. Plot the points and the fit
- Generate a least squares cubic polynomial that best fits these data. Plot the points and the cubic fit.

4.2 Code

```
t = 0:0.02:1;
y = cos(pi * t) + 0.8 * sin(2*pi*t);
y = y';

o = ones(length(t), 1);
t = t';
A = [o t];

c1 = A \ y; % line
A = [A t.^2 t.^3];

c2 = A \ y; % cubic

x = 0:0.01:1;
y1 = c1(1) + x * c1(2);
y2 = c2(1) + x * c2(2) + x.^2 * c2(3) + x.^3 * c2(4);

```

```
plot(x, y1, x, y2, t, y, '.');
title('Least squares');
xlabel('t');
ylabel('y');
legend('Linear', 'Cubic');
```

4.3 Output



4.4 Comment

All I can say is, Thank the mighty god for cubic functions!!