notes.tex

Henrik Linder

September 22, 2022

Unless you got some more help that I couldn't see, this task is poorly formulated and very hard for a course like this.

Not sure what kind of background you guys have in differential equations, but you got this one

$$a_z = \frac{B}{m} - g - \left(\frac{D}{m}\right) |v_z| v_z, \tag{1}$$

which seems fairly appropriate for the problem at hand.

This equation can be rewritten as

$$a_z = \alpha - \beta |v_z| v_z, \tag{2}$$

if we define $\alpha = \frac{B}{m} - g$ and $\beta = \left(\frac{D}{m}\right)$.

Now just move everything to the Left Hand Side (LHS) and rename $v = y$, $a = y'$ and we have the differential equation on standard form:

$$y' - \alpha + \beta y^2 = 0. \tag{3}$$

That was the easy part! The issue is that this is a non-homogeneous, nonlinear differential equation. Those are in general *very* hard to solve analytically. You can see the analytical solution for this particular one here, but trust me it's a headache.

You can however, fairly easily, solve it numerically by taking a small step forward in time, checking what the acceleration was at the last time step, and increasing your velocity by that much. Now you have your new velocity, and you can use that to find your new acceleration, which is used to find the velocity in the next time step, etc...

For example, if our time step is 0.1 s, then at the start, the algorithm would be

- **t=0** $v_0 = 0$
- **t=0** $a_0 = \frac{B}{m} - g$
- **t=0.1** $v_{0.1} = v_0 + a_0 \cdot dt = 0 + (\frac{B}{m} - g) \cdot 0.1$
- **t=0.1** $a_{0.1} = \frac{B}{m} - g - \left(\frac{D}{m}\right) v_{0.1}^2$

- **t=t**$_n$ $\quad v_n = t_{n-1} + a_{n-1} \cdot dt$
- **t=t**$_n$ $\quad a_n = \frac{B}{m} - g - \left(\frac{D}{m}\right) v_n^2$

What I described there is roughly the Forward Euler method. If you're interested in the fascinating world of numerical integration, id recommend reading about Runge-Kutta methods. If you're interested in music, I'd recommend watching Yunge Kutta.

I also did a quick and dirty implementation of that algorithm in a matlab script, also pasted below.

```
%I usually close and clear at start of main script,
%It can save a lot of headaches
close all
clear

m = 1200e-3;      %g ??   SI units? I wrote in kg now
D = 2.23;         %kg/m   ? assumption?
g = 9.82;         %m/s^2
d = 1.11;         %m      ? assumption?
B = 71.63;        %N      ? assumption?
rho = 1.293;      %kg/m^3 ? just says k/m^3

% Anonymous function for acceleration as a function of velocity.
% Just to make it easier to read in the loop
acceleration_function = @(v) B/m - g - (D/m) *abs(v)*v;

% Time step, good enough for this simulation
dt = .01

%initialize variables
t = linspace(0,5,1/dt);
t_length = length(t);
v = zeros(t_length,1);
a = zeros(t_length,1);
z = zeros(t_length,1);

%acceleration is not zero at t=0,
%so we define an initial acceleration for v=0
a(1) = acceleration_function(v(1));

%loop where we update the current velocity
%based on previous velocity and acceleration
for i = 2:t_length
  v(i) = v(i-1) + a(i-1)*dt;
  z(i) = z(i-1) + v(i-1)*dt;
  a(i) = acceleration_function(v(i));
end

%plot the results
plot(t,v)
xlabel("Time [s]")
ylabel("Velocity [m/s]")
grid on

figure
plot(t,z)
```

```
xlabel("Time [s]")
ylabel("Height [m]")
grid on
%text(2.5,4,"Note terminal velocity up here")
```