EGR 280

Mechanics Laboratory Exercise

Numerical Integration of Acceleration

Introduction

We have seen that the position, velocity and acceleration of a particle are related through the definitions:

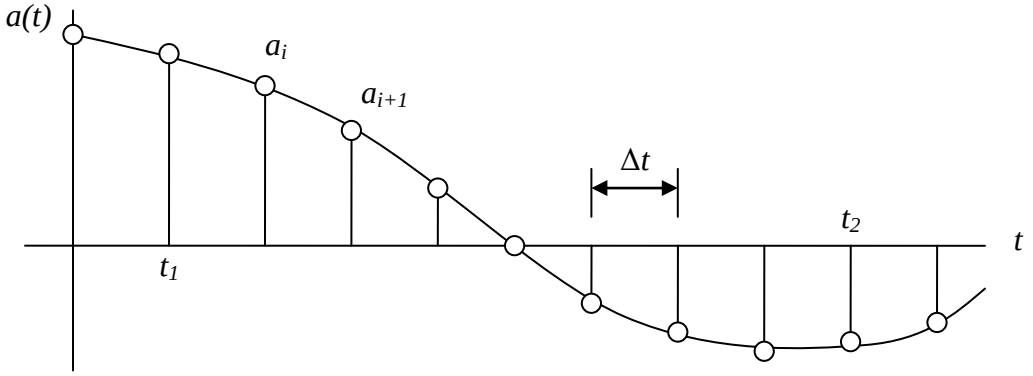
$$v = dx/dt \quad a = dv/dt$$

where *x(t)* is the position coordinate of the particle, *v(t)* is the velocity of the particle and *a(t)* is the acceleration of the particle. Depending on what we know about the acceleration in a particular problem, we can integrate the equations above to find the velocity and position as functions of time.

Sometimes we know the acceleration not as an analytical function, but only as discrete numerical values, for example when we read a sensor such as an accelerometer. In these cases, we can still integrate to find the velocity and position of the particle. This type of integration is call *numerical integration* and is a particular application of the field of *numerical methods*.

Numerical Integration

All of the techniques of numerical integration (there are several types, all with their own advantages and disadvantages) rely on the concept that the integral of a function is the "area under the curve." Consider the function below, which we know only by the discrete values



A simple way to approximate the area under the curve between any two points is as a trapezoid, with area $(1/2)(a_i + a_{i+1})\Delta t$. Adding these areas from some time t_1 to another time t_2 gives

$$v(t_2) - v(t_1) = \int_{t_1}^{t_2} a(t)dt \approx \sum_{t_1}^{t_2} \frac{1}{2}(a_i + a_{i+1})\Delta t$$

Once we obtain the velocity, also as discrete values in time, we can apply the same procedure to find the position of the particle.

This numerical integration technique is called the *Trapezoidal Rule*. The local error (the error in each trapezoid calculation is of the order of $(\Delta t)^2$, and the global error (the error that accumulates over the entire calculation) is of the order of the time step Δt.

There are many other types of numerical integration schemes that are much more involved and with the potential for better accuracy. The Trapezoidal Rule is the simplest technique of an entire class of numerical integration schemes known as the *Newton-Cotes Formulas*.

Exercise 1 – Integrating a Sine Function

To begin, we will numerically integrate to find a known function, $s(t) = A \sin(ft)$, where A is the magnitude and f is the frequency. We can easily find that

$$s(t) = A \sin(ft)$$
$$v(t) = ds/dt = Af \cos(ft)$$
$$a(t) = dv/dt = -Af^2 \sin(ft)$$

MATLAB Listing 1

```
%  Numerical integration using the Trapezoidal Rule - Example 1
%  Target function: s = A sin (ft)
%  We know:
%       velocity; v = Af*cos(ft)
%       acceleration: a = -Af^2*sin(ft)
%  Plot s,v,a as functions of time
%
clear;
dt=0.05;
A=1;
f=0.5;
%  generate acceleration data
for i=1:20/dt
    t(i)=i*dt;
    a(i)=-A*f^2*sin(f*t(i));
end
%  initial conditions
v(1)=A*f*cos(f*dt);
s(1)=A*sin(f*dt);
for i=2:length(a)
%  numerically integrate acceleration to find velocity
    v(i)=v(i-1)+0.5*(a(i-1)+a(i))*dt;
%  numerically integrate velocity to find position
    s(i)=s(i-1)+0.5*(v(i-1)+v(i))*dt;
end
plot(t,s,t,v,t,a);
xlabel('time (sec)');
legend('position','velocity','acceleration')
grid on;
```

Referring to Listing 1, define the time step, magnitude and frequency, then generate 20 seconds of data (the vectors *t* and *a* automatically re-dimension to hold the data, the semicolons are there so that the calculations are not echoed to the command window). The initial conditions are

defined, then the acceleration is integrated to find velocity, and velocity is integrated to find position all within the same for-loop. Note that we are using the length of the vector *a* as the termination condition on the loop.

Once the data is generated and the functions are integrated, plot the position, velocity and acceleration as functions of time, all on the same set of axes.

Things to explore:

1. How do you know that the two integrations are accurate? Are there characteristics of derivatives and integrals that you can use to make sure that the results make sense?
2. Change the value of the time step, looking at the results when it is both larger and smaller. Do you see any effect on the quality of the results?
3. Change the values of the magnitude and/or the frequency and adjust the time step, if necessary, to achieve a good output. Is there any relationship between the magnitude and the time step? Is there a relationship between the frequency and the time step?

Exercise 2 – Dealing with Noisy Signals

The code in Listing 2 superimposes random noise on top of the acceleration. Use this code to generate the acceleration data and explore the effects of different levels of noise. What happens to the noise as the signal is integrated? Change the value of the time step. Is there any effect on the quality of the integration?

MATLAB Listing 2

```
nl=0.10; % magnitude of the noise level
%  generate acceleration data
for i=1:20/dt
    t(i)=i*dt;
    a(i)=-A*f^2*sin(f*t(i));
%  The following lines superimpose 10% random noise onto the acceleration
    if(i>1)
        a(i)=a(i) + A*f^2*nl*(rand-0.5)/0.5;
    end
end
```

Exercise 3 – Two-Dimensional Acceleration Data

Since acceleration is a vector, we can integrate the components of acceleration independently of each other, then combine the results to obtain multi-dimensional velocities and/or positions. The MATLAB code in Listing 3 generates acceleration data for a two-dimensional rectilinear motion of a particle, with initial velocity of 90 m/s at an angle of 50 degree to the horizontal, and initial position at the origin. In this case, we will plot s_y versus s_x to observe the trajectory of the particle.

MATLAB Listing 3

```
%  Numerical integration using the Trapezoidal Rule - Example 3
%  Two-dimensional integration, particle trajectory
%  Acceleration: a = (0i -9.81j)m/s
%  Initial conditions:
%      initial velocity: 90 m/s at 50 degrees to horizontal
%      inital position: the origin
%  Plot x,y position
%
clear;
dt=0.05;
v0=90;
ang=50;
%  generate acceleration data
for i=1:20/dt
    t(i)=i*dt;
    ax(i)=0;
    ay(i)=-9.81;
end
%  initial conditions
vx(1)=v0*cos(pi*ang/180);
vy(1)=v0*sin(pi*ang/180);
sx(1)=0;
sy(1)=0;
%  integrate numerically
for i=2:length(t)
    vx(i)=vx(i-1)+0.5*(ax(i-1)+ax(i))*dt;
    vy(i)=vy(i-1)+0.5*(ay(i-1)+ay(i))*dt;
    sx(i)=sx(i-1)+0.5*(vx(i-1)+vx(i))*dt;
    sy(i)=sy(i-1)+0.5*(vy(i-1)+vy(i))*dt;
end
plot(sx,sy);
xlabel('x position (m)');
ylabel('y position (m)');
grid on;
```

Again, change the value of the time step to see its effect on the final trajectory.

Exercise 4 – Integration of Arbitrary Acceleration Data

For this last exercise, as practice in reading and using real acceleration data, you will integrate two-dimensional data that is stored as ASCII values in text files “axtestdata.txt” and “aytestdata.txt”. Read these data files with the following MATLAB commands:

MATLAB Listing 4

```
ax=load('axtestdata.txt','-ascii');
ay=load('aytestdata.txt','-ascii');
```

Using a time step dt=0.01 and initial conditions v(1) = (-475**i** – 75**j**), s(1) = 0**i** +0**j**, integrate these acceleration data, then generate the following plots one at time:

1. x-acceleration vs time
2. y-acceleration vs time
3. x-velocity vs time
4. y-velocity vs time
5. x-position vs time
6. y-position vs time
7. y-position vs x-position

Print out a copy of your MATLAB code for Exercise 4 with your name as a comment at the beginning, and print each of the 7 plots listed above. Have your lab instructor sign your printout and hand in this listing.