Simulating swarm/flock behaviour

Supervisor: Graham Kells

The aim of this project is to write matlab code to simulate the swarm (or flock) like behaviour observed in birds/bees/fish etc. The flocking behaviour is not something we program directly but is what is called *emergent* behaviour. This is when simple rules are followed to generate complicated and often beautiful phenomena.

We want to examine the behaviour in 3-dimensions so we first set the initial position (x), velocity (v) and acceleration (a) components of our N elements. We randomly generate initial position and velocity components of our N elements through the commands

\[
\begin{align*}
x &= X \cdot \text{randn}(3,N); \\
v &= V \cdot \text{randn}(3,N);
\end{align*}
\]

These commands generate a Gaussian spread of particles about the (0,0,0) with a Gaussian spread of velocities. The size of the spread can be controlled by the the values we give X and V. Each of these matrices are $3 \times N$ in size. For example we address the y velocity of the 10th particle as \(v(2,10)\).

The main engine of this simulation will be our outer (primary) for loop. This will generate the dynamics through the Euler-Cromer update rule. That is we generate and plot new position and velocity matrices through the relation

\[
\begin{align*}
v_{\text{new}} &= v + a \cdot \text{delta}; \\
x_{\text{new}} &= x + v_{\text{new}} \cdot \text{delta}; \\
v &= v_{\text{new}}; \\
x &= x_{\text{new}};
\end{align*}
\]

\[
\text{plot3}(x(1,:),x(2,:),x(3,:),'k+','Markersize',.5); \\
\text{drawnow};
\]

where \text{delta} is our time step. Of course it is how we calculate \(a\) that gives the system of N elements its character and this will be the main challenge of this project.

The first basic assumption we make is that each element (bird/bee/fish) of our swarm is in complete control over its x, y and z acceleration components. Each element then decides, based on the positions of (not necessarily all) other elements, where is wants to accelerate to. The most widely known algorithm for flocking is that of *boids* by Craig Reynolds [1]. In this model the value chosen for acceleration of each element depends on 3 main criteria

1. *Alignment:* Elements move towards the average heading of local flock-mates.
2. *Cohesion:* Elements move towards the average position of local flock-mates
3. *Separation:* Elements of the swarm don’t collide with each other. (Unless they’re stupid — this is easier to program than you might think).
Figure 1: A swarm for N=400, c1=0.3, c2=0.3, c3=5, c4=5; A=10, V=10 and only cares about objects within a radius of 10 to the front and side.

The most basic way that I can think of to implement these rules is create secondary and tertiary nested for loops that each run from 1 to N. That is

```matlab
a1=zeros(3,N)
a2=zeros(3,N)
a3=zeros(3,N)

for n=1:N
    for m=1:N
        if m ~= n
            r = x(:,m)-x(:,n);
            vr = v(:,m)-v(:,n);
            rmag=sqrt(r(1,1)^2+r(2,1)^2+r(3,1)^2);
            a1(:,n) = a1(:,n) - c1*r/rmag^2
            a2(:,n) = a2(:,n) + c2*r ;
            a3(:,n) = a3(:,n) + c3*vr;
        end
    end
a4(:,n) = c4*randn(3,1);
```

The inside for loop sums up the contributions of all the other elements (except itself) on the \(n\)th element.

1. The matrix \(a1\) tends to steer elements towards the headings (relative velocities) of others.
2. The matrix \(a2\) tends to steer elements towards the relative positions of others.
3. The matrix \(a3\) is a separating acceleration that only has significant values if elements get too close.
4. The matrix \(a4\) introduces a little randomness into the motion.
The total acceleration is the sum of these four accelerations.

(a) Write the program and examine its behaviour for different values of \( N, \) \( \delta, c_1, c_2, c_3 \), \( X \) and \( V \). Try to find values that give reasonably realistic behaviours.

(b) One glaring inaccuracy in this program is that elements pay attention to all other elements. In reality they should only care about other elements immediately in front and to the sides of them. If we make the reasonable assumption that elements face the direction they are travelling in it is a simple task to implement this. **Hint:** Use an `if` statement inside the tertiary `for` loop, the normalised velocity vectors, the normalised relative position vectors and the dot product.

(c) When one is running the simulation for large values of \( N \) it can be difficult to see any patterns emerging because it is slow. One way to get around this is to use the matlab `moviein` function to make a film. Before the the outer for loop write \( M = \text{moviein}(Z) \); \( Z \) is the number of frames you want in the film.

Inside the outer for loop and after the plot commands type \( M(:,k) = \text{getframe} \) where \( k \) is an integer that increases by one each time we iterate the loop. To play the movie simply write \( \text{movie}(M) \) in the command window.

(d) Write the report. Include all relevant matlab codes and figures. Keep them tidy and make sure they do what you say they do.

**References**