Particle Filter Implementation Report

The report outlines the implementation of a particle tracking algorithm based on observations from a doppler radar at discrete time intervals.

Parts 1 and 2 are just initialization phases. Most of my report and the main algorithm is outlined in Part 3 along with certain conclusions.The code for this report can be found on my Github.

Part 1 (Initialization)

Helper function ‘funbk’ does the job of computing the angular position of the particle and the radial velocity of the particle at time t.

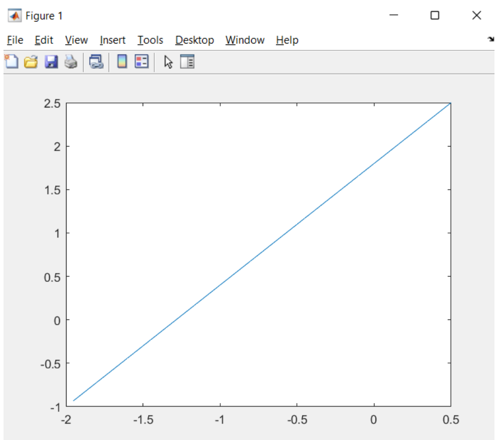
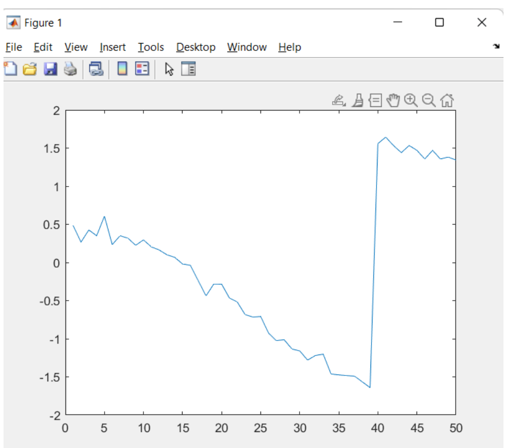
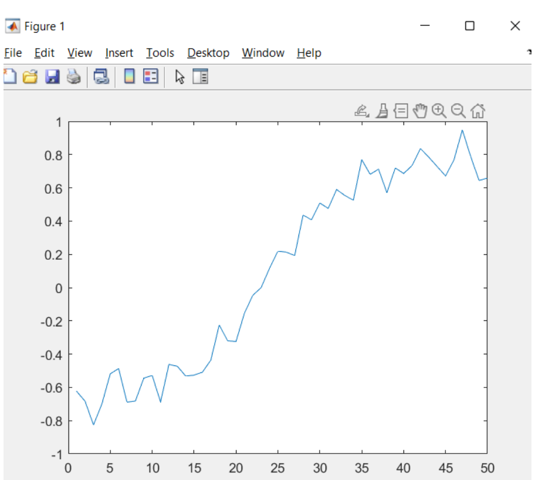
x_star is the actual position of the particle at time t while v_star represents the velocity of the particle at time t. These factors determine the vector b which represents the angular position and radial velocity of the particle.

Graphs on next page.

Code

```
%Part 1
%Initial vector and constant veclocity
x_0 = [-2;-1];
v_star = [0.5;0.7];
%Time step and sigma value
dt = 0.1;
sig = 0.07;
%Observations b_k
b = zeros(2,50);
%True positions
x_star = zeros(2,50);

%Calculate b_k and x_star at different time points
for k = 1:50
    t_k = dt*k;
    x_star(:,k) = x_0 + v_star*t_k;
    b(:,k) = funbk(x_star(:,k),v_star) + sig*randn(2,1);
end
```

```
%Function that calculates the observation vectors.
function b_k = funbk(x,v)
    theta_t = atan(x(2)./x(1));
    r_p = (v'*x)./norm(x,2);
    b_k = [theta_t;r_p];
end
```

Graph	Explanation
	**Plot of x_star(position of the particle at time t) -** Graph appears linear as should be expected since we are starting from an initial position and moving ahead with a constant velocity. X-axis is the x component of x-star Y-axis is the y component of x-star
	**Plot of theta_k(angular position at time t).** This is computes using arctan(y/x) where (x,y) is the position of the particle at time t
	**Plot of r'(t)/velocity of the particle at time t .**This is the radial velocity of the particle at any point of time t

Part 2 (Initialization)

Initialize the positions, velocities, angular positions and weights of the particles that will determine the position of our target .The v_0 and x_0 vectors will be used for setting off the forward euler approximations initially. These also determine the initial guesses for the position of the particle. V0bar is the initial guess for velocity and r_0 and theta_0 together pretty much guesses the location of the particle. These values can be changed. I use a bad guess on purpose to show the effectiveness of the algorithm.

```
%Part 2
%no of particles
n = 500;
%Initialize the observations
b_0 = funbk(x_0,v_star) + sig*randn(2,1);
%Calculate the angular positions
theta_0 = b_0(1) + sig*randn(n,1);
r_0 = 1.5 + .05*randn(n,1);
%Calculate initial positions
x01 = (r_0.*cos(theta_0))';
x02 = (r_0.*sin(theta_0))';
x_0 = [x01; x02];
%Initial Velocity guess
v0bar = [2;5];
gamma = 0.3;
v_0 = zeros(2,n);
for i = 1:n
    v_0(:,i) = v0bar + gamma*randn(2,1);
end
%Initialize the weights
w_0 = (1/n).*ones(n,1);
```

Part 3 (Main Algorithm)

The entire algorithm is divided into 5 parts. Explanations and intuitions have been explained for each part followed by graphs showing the efficacy of the algorithm.

Initialize the vectors. Note here ni refers to the number of iterations. After using the algorithm for 50 iterations, I later use the algorithm for 500-1500 iterations with bad guesses to show how the distance of our estimated location from the original particle decreases drastically even after a bad initial guess. n refers to the number of particles, I specifically used 500 particles.

```
xhat = zeros(2,n,ni);
vhat = zeros(2,n,ni);
eta = zeros(1,n,ni);
x = zeros(2,n,ni);
v = zeros(2,n,ni);
w = zeros(1,n,ni);
```

Part I

Move forward using a forward euler approximation. Use the initial guess for the first iteration and after that point move forward using the information gathered from the previous observation and previous particles. x and v will be updated using the information gathered about the target particle by the doppler radar in Part IV of the algorithm.

```
for j = 1:50
    %Part a
    if j==1
        xhat(:,:,1) = x_0 + dt.*v_0;
        vhat(:,:,1) = v_0;
    else
        xhat(:,:,j) = x(:,:,j-1) + dt.*v(:,:,j-1);
        vhat(:,:,j) = v(:,:,j-1);
    end
```

Part II

Calculate the fitness weights and normalize them. Notice that points closer to the observations will have higher fitness weights. This is because we initially compute the distance of the particle from the observed location (temp2) and then the steps after that ensure that there is an inverse relationship between the distance and the weights ensuring that particles closer to the observation are given higher fitness weights.

```
%Part b
%Compute fitness weights
for k = 1:5000
 temp1 = funbk(xhat(:,k,j),vhat(:,k,j));
 temp2 = norm(b(:,j)-temp1,2).^2;
 eta(1,k,j)= ((1./sig.^2).*temp2);
end
%Normalize fitness weights
min_eta = min(eta(1,:,j));
eta(1,:,j) = exp(-eta(1,:,j))./exp(-min_eta);
eta(1,:,j) = eta(1,:,j)./norm(eta(1,:,j),1);
```

Part III

Draw I_k; Use an inbuilt function to draw indices from a discrete distribution; Discrete distribution is the distribution generated from eta in the previous bit and I use a helper function to generate points from the distribution. The idea is to penalize the points that are closer to the observations since the distribution will be centered towards those points. This will become more evident in Part IV where points with these indices will be given higher weightage.

```
%Part c
idx = [];
for i = 1:n
    xk = discrete([1:n],eta(1,:,j));
    idx = [idx;xk];
end
```

Part IV

Calculate x and v from xhat and vhat; Use the indices drawn from the discrete distribution to draw points from xhat and vhat to generate vectors x and v. In a sense we trust particles that are closer to the observations and start ignoring the ones away from the observation. x and v are then used in later iterations for moving the particle in bit a.

```
    %Part d
    alpha = 0.01;
    for k = 1:5000
        x(:,k,j) = xhat(:,idx(k),j) + alpha.*randn(2,1);
        v(:,k,j) = vhat(:,idx(k),j) + alpha.*randn(2,1);
    end

```

Part V

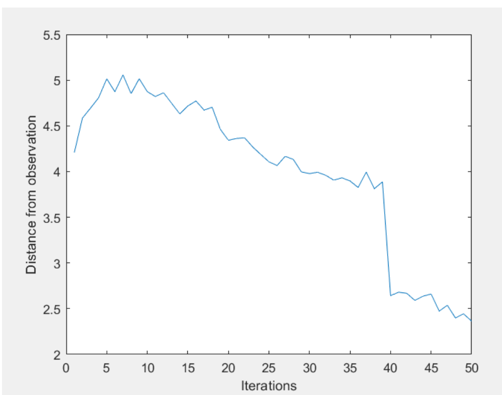
Calculate the weights of the particles. In the end I use them for computing the angular positions and radial velocities. I take two approaches - (a) compute the weighted mean of these components using the weights computed in this part; (b) trust the particle with the highest weight. Results from both these observations are graphed later with certain conclusions.

```
  %Part e
  for k = 1:5000
      temp3 = funbk(x(:,k,j),v(:,k,j));
      temp4 = norm(b(:,j)-temp3,2).^2;
      temp5 = funbk(xhat(:,idx(k),j),vhat(:,idx(k),j));
      temp6 = norm(b(:,j)-temp5,2).^2;
      w(1,k,j) = (1./sig.^2).*(temp4-temp6);
  end

   min_w = min(w(1,:,j));
   w(1,:,j) = exp(-w(1,:,j)+min_w);
   w(1,:,j) = w(1,:,j)./norm(w(1,:,j),1);
end
```

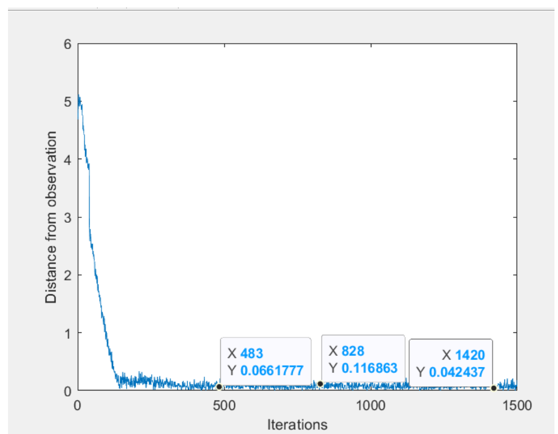
Code for computing distance using weighted mean using normalized weights:

```
idx = zeros(ni,1);
mvals = zeros(ni,1);
dists = zeros(ni,1);
for i = 1:ni
    [m ind] = max(w(1,:,i));
    mvals(i) = m;
    idx(i) =ind;
    mean_x1 = sum(x(1,:,i).*w(1,:,i));
    mean_x2 = sum(x(2,:,i).*w(1,:,i));
    mean_v1 = sum(v(1,:,i).*w(1,:,i));
    mean_v2 = sum(v(2,:,i).*w(1,:,i));
    dists(i) = norm(funbk([mean_x1;mean_x2],[mean_v1;mean_v2])-b(:,i),2);

end
```

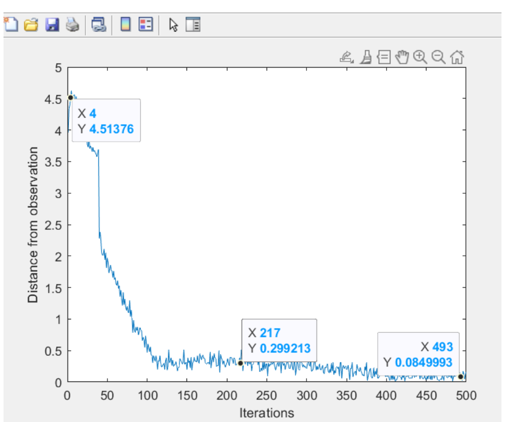
<table>
<tr><th>50 iterations; 500 particles</th><th>1500 iterations; 500 particles</th></tr>
<tr><td>



</td><td>



</td></tr>
<tr><th>500 iterations; 500 particles</th><th>Observations</th></tr>
<tr><td>



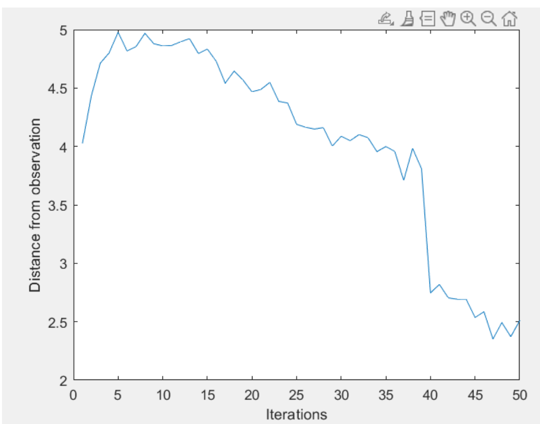
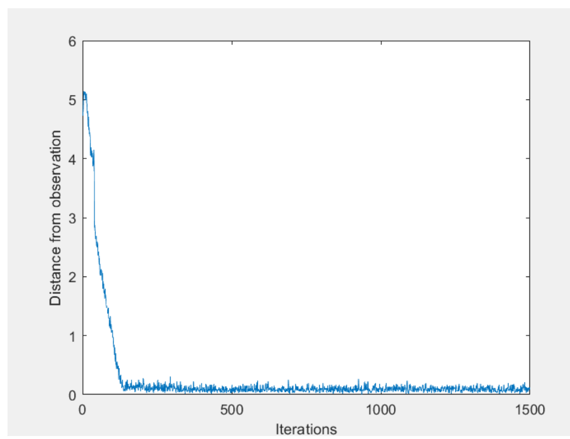
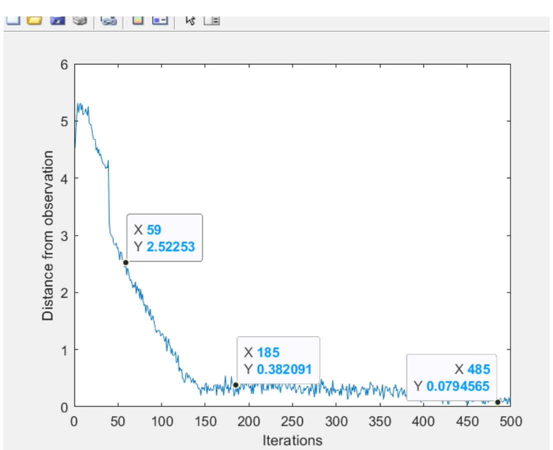
</td><td>

The algorithm is able to successfully track down the particle with more no of iterations as shown by the graph. The distance between our target particle and tracker particle reduces to almost 0 at ~500 iterations.

</td></tr>
</table>

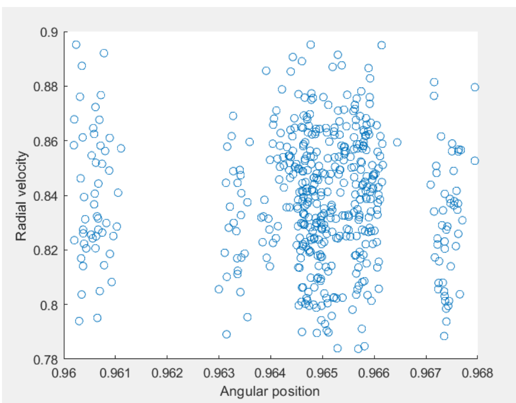
Code for computing the distance using the particle with the highest weight:

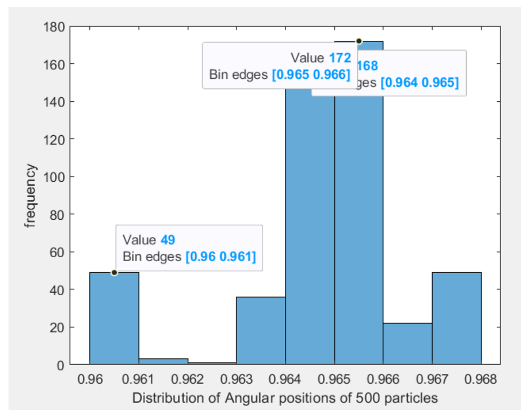
```
idx = zeros(ni,1);
mvals = zeros(ni,1);
dists = zeros(ni,1);
for i = 1:ni
    [m ind] = max(w(1,:,i));
    mvals(i) = m;
    idx(i) =ind;
    dists(i) = norm(funbk(x(:,idx(i),i),v(:,idx(i),i))-b(:,i),2);

end
```

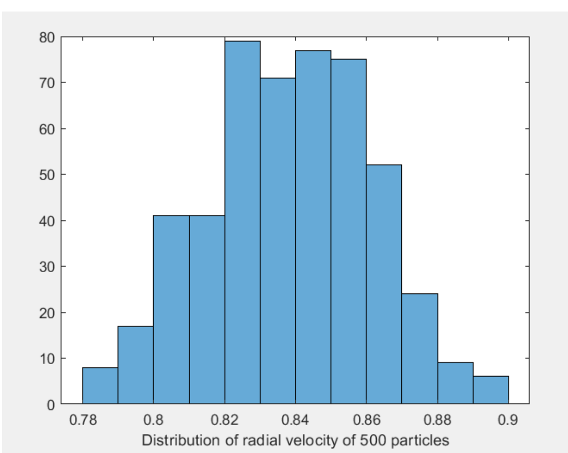
<table>
<tr><th>50 iterations; 500 particles</th><th>1500 iterations; 500 particles</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><th>500 iterations; 500 particles</th><th>Observations</th></tr>
<tr><td>
</td><td>The algorithm is able to successfully track down the particle with more no of iterations as shown by the graph. The distance between our target particle and tracker particle reduces to almost 0 at ~500 iterations.</td></tr>
</table>

These graphs first illustrate that the algorithm does a really good job following the subject particles as the distance from the observed particle becomes negligible with time even with a really bad guess.

Secondly, notice how the graphs follow a similar pattern for both the weighted mean of 500 particles and the tracker which just uses the particle with the highest weight. This shows that the particles by the end of max iterations become certain about the location of the subject particle converging to the same point. This is further illustrated by the histograms and scatter plots below.

<table>
<tr><td>Scatterplot of the 500 particles at the end of 1500 iterations</td><td>Histogram of the angular positions at the end of 1500 iterations</td></tr>
<tr><td>


</td><td>


</td></tr>
<tr><td>Histogram of the radial velocity at the end of 1500 iterations</td><td>Explanations</td></tr>
<tr><td>


</td><td>Notice that the radial velocities of the 500 particles are in the range of 0.78-0.9 with most of the particles centered around 0.85 ensuring that particles become more and more certain about the radial velocity. This is close to the actual radial velocity in our case.

Notice that the angular position of the 500 particles are in the range of 0.96-0.97 with most of the particles centered around 0.965 ensuring that particles become more and more certain about the angular position. This is close to the actual angular position in our case.</td></tr>
</table>