Lab 6 - Particle Filter

Submitted By:
Shashi Shivaraju
C88650674

Clemson University
November 20, 2018

Abstract

This report explains the process involved in designing an Particle Filter to mitigate tractable non-Gaussian noise in sensor measurements to estimate the parameters of a tractable model used to track the object.

1 Introduction

This report considers the problem of designing a Particle Filter.A filter is a mathematical tool that uses an expected dynamic model to help mitigate noise in sensor data.We know that the Kalman Filter is only intended for linear systems and the Extended Kalman Filter works on nonlinear systems.However,both these filters assume that the state distribution, dynamic noise and observation noise are all Gaussian in nature.There are many problems for which the distribution is not Gaussian and thus the Kalman and Extended Kalman Filters are not suitable. Whereas Particle Filter can be used to mitigate the tractable non-Gaussian noises from the sensor measurements to track an object which is represented by a tractable model.

The Particle Filter utilizes concepts such as recursive Bayesian estimation,Monte Carlo approximation, and sequential importance sampling for its implementation.According to recursive Bayesian estimation, current observation of a system only depends upon the current state.Also next state depends only upon the current state, and not upon all the previous history of state.Particle filter approximates using Monte Carlo approximation in which a set of samples is utilized to approximate a distribution. Each sample is given a state and a weight.

This report describes designing a Particle Filter for given data set "magnets-data.txt" by considering a tractable model,which describes a system following a motion pattern,where the position zig-zags back and forth on a line.

2 Methods

Similar to Kalman filtering,Particle filtering is also a continuous cycle of predict-update. It involves four main steps:prediction,measurement,updation and particle resampling. Before we apply the Particle filtering to the data,one has to determine the a mathematical model which describes the behavior of the object being tracked. This follows the below steps:

1. Determine the state variables,which describes the property of the object being tracked.
2. Determine the state transition equations,which provides description of nominal expected behavior of the state variables.
3. Define the dynamic noises,which determine the possible deviations during a state transition.
4. Determine the observation variables,which are sensor measurement used to track the object.
5. Define the observation equations,which relate the sensor readings to the state variables.
6. Define the measurement noises,which describes the possible corruptions during a sensor reading.

Using these mathematical models,the Particle Filter is implemented. The design and implementation of Particle Filter for the given data-set is described in below section.

2.1 Particle Filter design

The dataset ”magnets-data.txt” describes the position of an object moving in between two attractors, such as magnets. The system follows a motion pattern where the object position zig-zags back and forth on a line.Figure 1 represents the system. The sensor on the system detects a field strength that is the sum of the distances from two fixed-position magnets.

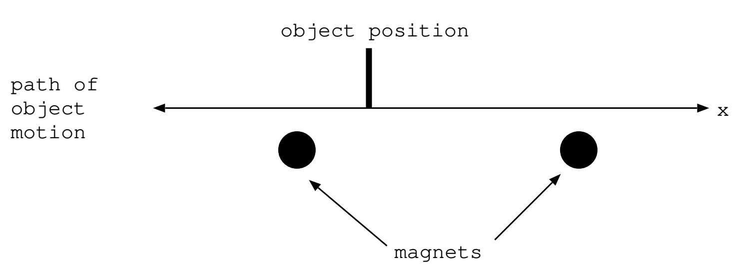


Figure 1: System in consideration.

In particle filter,the samples are called particles. They are denoted as:

$$\chi = \{x^{(m)}, w^{(m)}\}_{m=1}^{M} \tag{1}$$

where $x^{(m)}$ represents the state of particle m, $w^{(m)}$ represents the weight of particle m and M represents the total number of particles.For this model,M was selected as 1000 and each particle's weight was initialised with $1/M$.

Let the state transition equations for this model be:

$$f(x_t, a_t) = \begin{bmatrix} x_{t+1} = x_t + \dot{x}_t T \\ \dot{x}_{t+1} = \begin{cases} 2 & \text{if } x_t < -20 \\ \dot{x}_t + |a_t| & \text{if } -20 \leq x_t < 0 \\ \dot{x}_t - |a_t| & \text{if } 0 \leq x_t \leq 20 \\ -2 & \text{if } x_t > 20 \end{cases} \end{bmatrix} \tag{2}$$

where x_t represents position,$\dot{x}_t$ represents velocity and T represents time period. The velocity equation is a piecewise function that adds or subtracts a random amount a_t to the current velocity,depending on the current position. The dynamic noise a_t is drawn from a zero-mean Gaussian distribution $N(0, \sigma_a^2)$,where the value of $\sigma_a = 2^{-4} = 0.0625$. The goal of the state transition equation is to keep the position oscillating about zero but between -20 and 20.

Let the the state X_t of the system be defined as:

$$X_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} \tag{3}$$

Since the given data is the sensor readings that measures the total magnetic strength,let us denote it as y_t:

$$Y_t = \begin{bmatrix} y_t \end{bmatrix} \tag{4}$$

Two magnets are placed at $x_{m1} = -10$ and $x_{m2} = 10$.The observation equations for this model is:

$$g(x_t, n_t) = \begin{bmatrix} y_t = \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t - x_{m1})^2}{2\sigma_m^2}) + \frac{1}{\sqrt{2\pi}\sigma_m} \exp(\frac{-(x_t - x_{m2})^2}{2\sigma_m^2}) + n_t \end{bmatrix} \tag{5}$$

where n_t is a random sample drawn from $N(0, \sigma_n^2)$ representing measurement noise.The value of $\sigma_n = 2^{-8} = 0.003906$.The value $\sigma_m = 4.0$.

The above equations formulate the model for the filtering problem of this system. The equations provide a theoretical model of the system being tracked and describes its expected behavior.

The implementation of extended Particle Filter for this model is explained below.

2.2 Implementation of Particle Filter

The Particle Filter is a continuous cycle of predict-update. The following equations form the main loop of the Particle Filter applied to the model described above:

1. Each particle m is propagated through the state transition equation:

$$\{x_t^{(m)} = f(x_{t-1}^{(m)}, a_t^{(m)})\}_{m=1}^{M} \tag{6}$$

   The value $a_t^{(m)}$ represents the dynamic noise from $t-1$ to t, and is randomly and independently calculated for each particle m. It may be envisioned as each particle taking a different "guess" at the dynamic noise undertaken for the current iteration.

2. Using the new measurement vector y_t, the weight for each particle is updated:

$$\tilde{w}_t^{(m)} = w_{t-1}^{(m)} \cdot p(y_t|x_t^{(m)}) \tag{7}$$

   The value $p(y_t|x_t^{(m)})$ is determined by the measurement noise. It is calculated by taking the ideal measurement of the particle, and comparing it against the actual measurement, in the model of the measurement noise. The ideal measurement of the particle is calculated as follows:

$$g(x_t^{(m)}, 0) = \left[y_t^{(m)} = \tfrac{1}{\sqrt{2\pi}\sigma_m} \exp(\tfrac{-(x_t^{(m)} - x_{m1})^2}{2\sigma_m^2}) + \tfrac{1}{\sqrt{2\pi}\sigma_m} \exp(\tfrac{-(x_t^{(m)} - x_{m2})^2}{2\sigma_m^2}) \right] \tag{8}$$

   The ideal measurement is then compared against the actual measurement in the model of the measurement noise as follows:

$$p(y_t|x_t^{(m)}) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp(\frac{-(y_t^{(m)} - y_t)^2}{2\sigma_n^2}) \tag{9}$$

3. Normalize the updated weights, so they sum to 1:

$$w_t^{(m)} = \frac{\tilde{w}_t^{(m)}}{\sum_{m=1}^{M} \tilde{w}_t^{(m)}} \tag{10}$$

4. Compute the desired output:

$$E[x_t] \approx \sum_{m=1}^{M} x_t^{(m)} \cdot w_t^{(m)} \tag{11}$$

5. Check if sampling is necessary, and if so, resample.

   In order to determine if resampling is needed, the coefficient of variation statistic can be calculated as:

$$\mathrm{CV} = \frac{\mathrm{VAR}(w^{(m)})}{E^2[w^{(m)}]} = \frac{\dfrac{1}{M} \displaystyle\sum_{m=1}^{M} \left(w^{(m)} - \frac{1}{M} \sum_{m=1}^{M} w^{(m)} \right)^2}{\left(\dfrac{1}{M} \displaystyle\sum_{m=1}^{M} w^{(m)} \right)^2} = \frac{1}{M} \sum_{m=1}^{M} (M \cdot w^{(m)} - 1)^2 \tag{12}$$

The effective sample size can then be calculated as:

$$\text{ESS} = \frac{M}{1 + \text{CV}} \tag{13}$$

The effective sample size represents the number of particles which have an appreciable weight.In order to check if resampling is necessary, the effective sample size is tested against the number of particles:

```
if (ESS < 0.5 M)
  resample
```

For this model, the threshold is set to 50% of the particles. Please refer section 2.3 for implementation of resampling.

6. Loop : now t becomes t + 1

Please refer the Appendix for the Matlab implementation of Particle Filter for model described above.

2.3 Resampling

Resampling is accomplished by a method called select with replacement. The logic is to kill off particles with negligible weights, and replace them with copies of particles that have large weights. Below is the pseudo code for resampling:

```
Assume particle states in P[1...M], weights in W[1...M].

Q=cumsum(W);           calculate the running totals
t=rand(M+1);           t is an array of M+1 uniform random numbers 0 to 1
T=sort(t);             sort them smallest to largest
T[M+1]=1.0;            boundary condition for cumulative hist
i=j=1;                 arrays start at 1
while (i<=M)
  if (T[i] < Q[j])
    Index[i]=j;
    i=i+1;
  else
    j=j+1;
  end if
end while

loop (i=1; i<=M; i=i+1)
```

```
    NewP[i]=P[index[i]];
    NewW[i]=1/M;
  end loop
```

This algorithm computes a list of indices of particles.In creating the list, particles with lower weights are less likely to get copied, and particles with higher weights are more likely to get multiple copies.After computing the list, it creates a new list of particles of equal weights.

3 Results

The Particle Filter was implemented for the given model to track the position of the object with M = 1000 and resampling threshold set to 50% of the total particles.The resultant plots of inphase object position tracking,out-of-phase object position tracking and the distribution of particle weights during filter iterations is shown in figures 2,3 and 4 respectively:

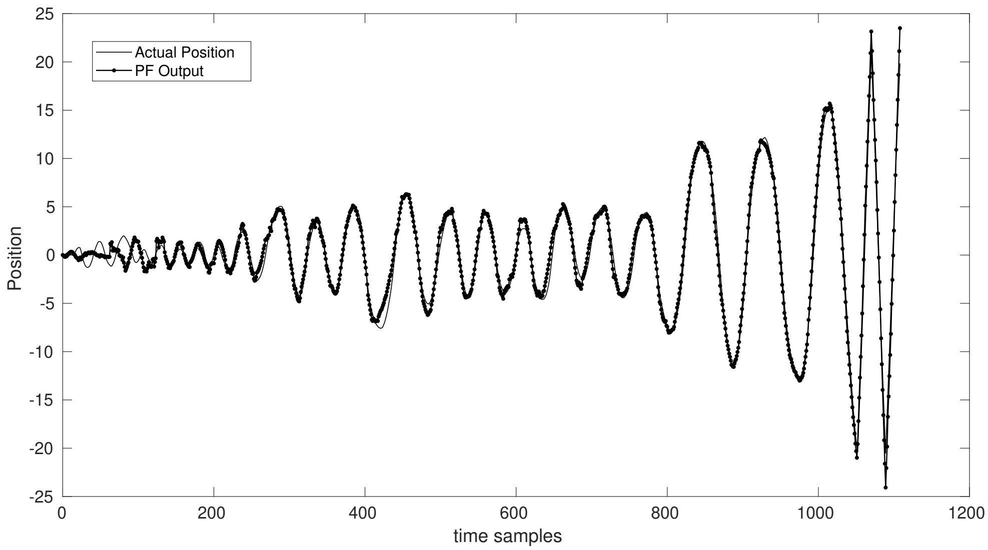


Figure 2: Inphase Tracking of Position

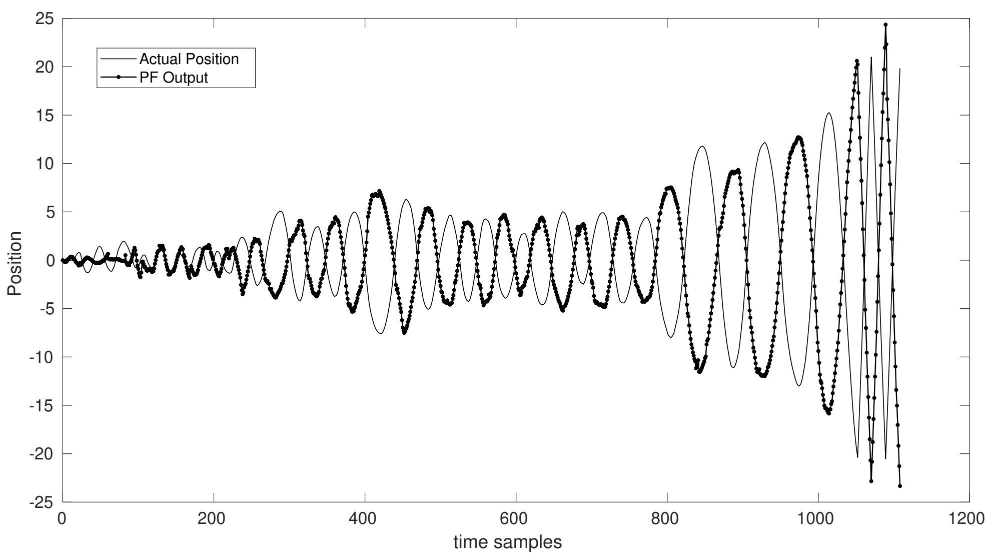
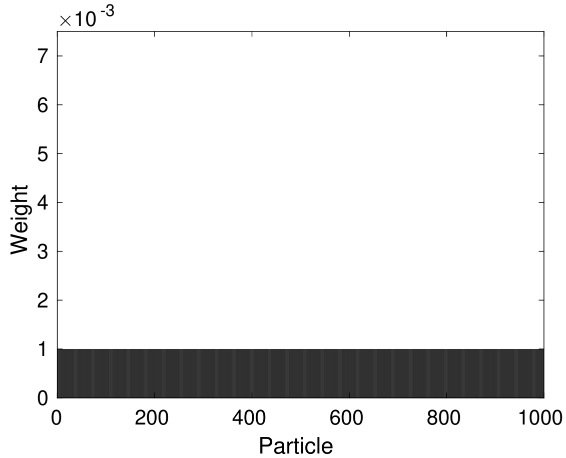
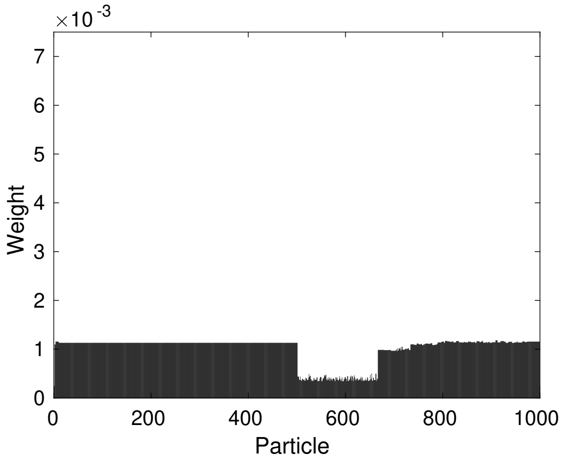


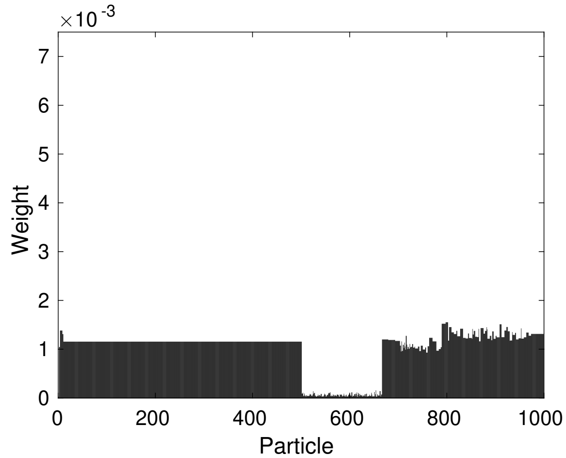
Figure 3: Out-of-Phase Tracking of Position



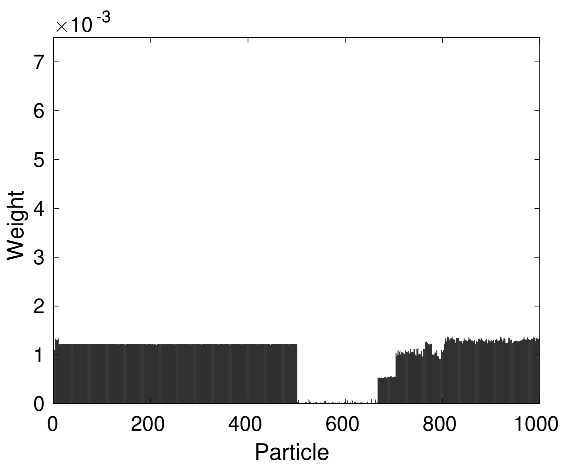
Resampled at iteration =189 & ESS =305.5329



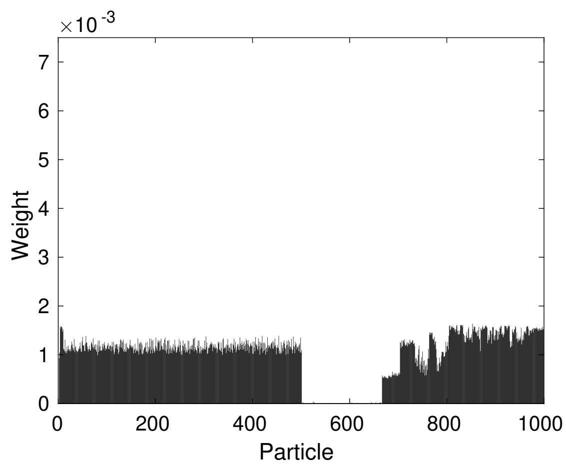
Iteration =190 & ESS =929.2959



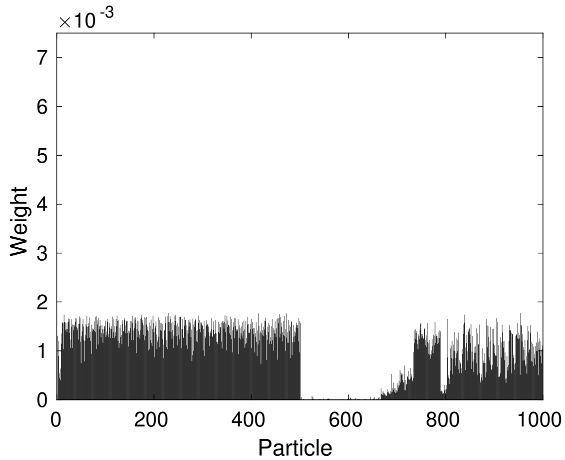
Iteration =191 & ESS =843.4607



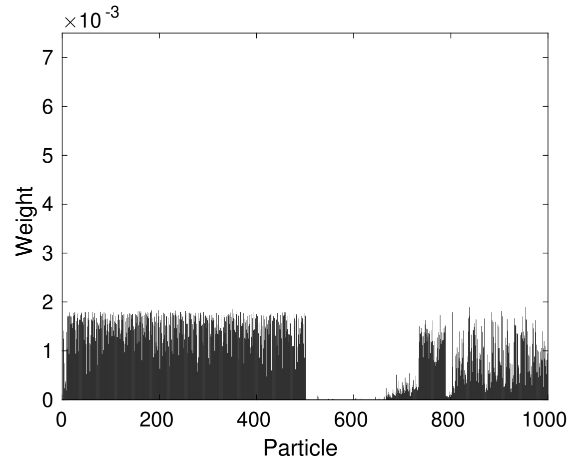
Iteration =192 & ESS =824.4521



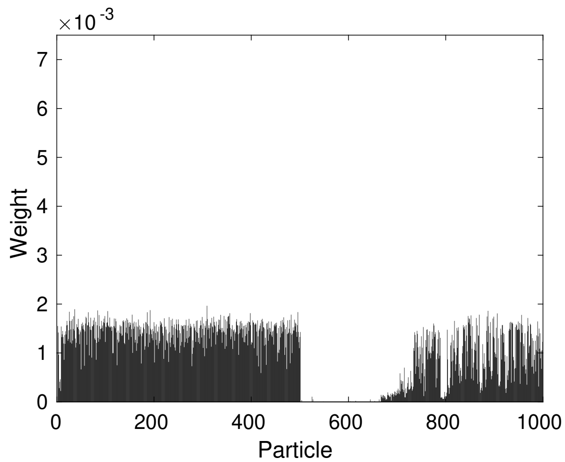
Iteration =193 & ESS =804.4512



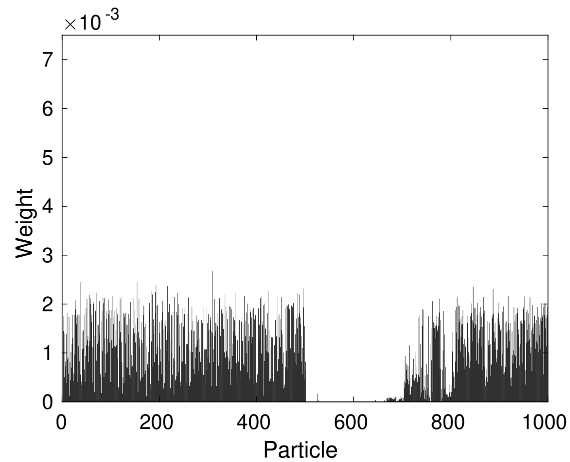
Iteration =194 & ESS =741.2224



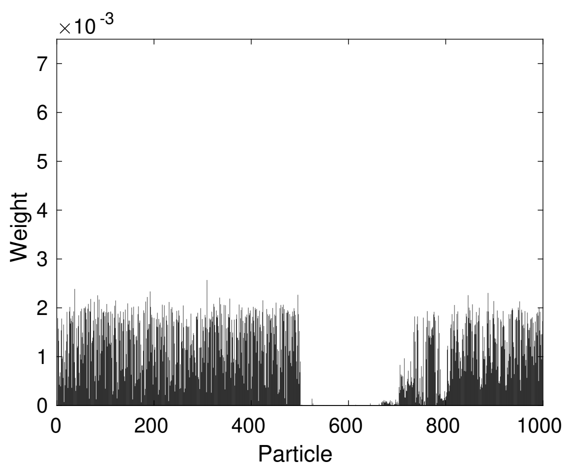
Iteration =195 & ESS =699.3194



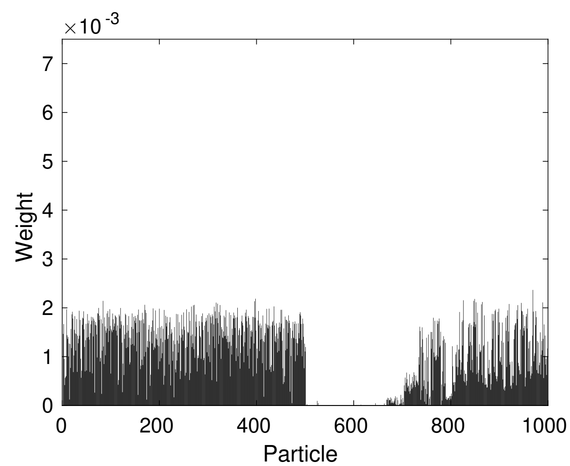
Iteration =196 & ESS =716.5442



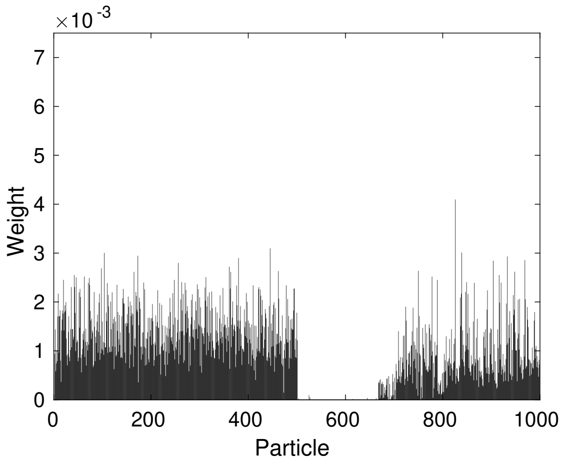
Iteration =197 & ESS =654.6847



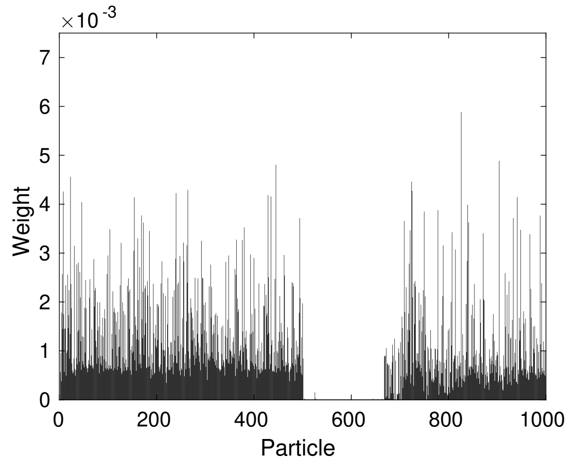
Iteration =198 & ESS =650.5192



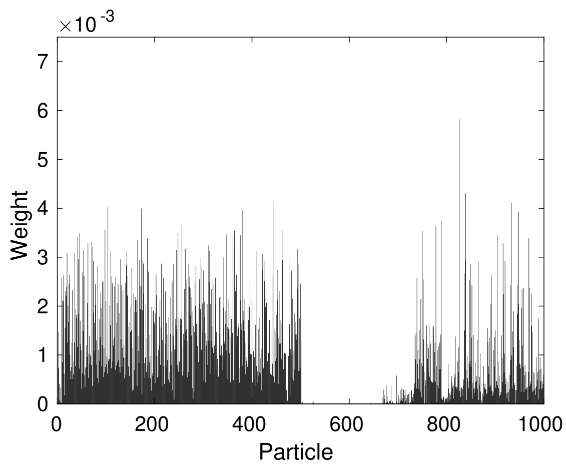
Iteration =199 & ESS =677.5877



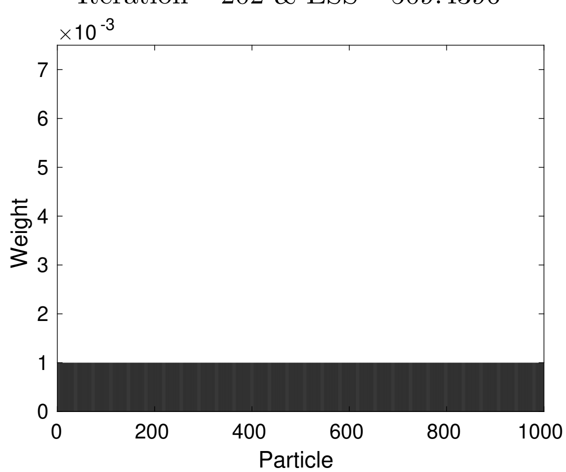
Iteration =200 & ESS =653.8899



Iteration =201 & ESS =540.2736



Iteration =202 & ESS =509.4396



Resampled at iteration =203 & ESS =395.6746

Figure 4: Distribution of particle weights inbetween resampling events

4 Conclusion

The report describes in detail the analytical process involved in designing and implementing a Particle Filter to predict the given model with tractable non-Gaussian measurement noise.It was often observed that the output of the Particle Filter varied with the actual ground truth by being out-of-phase due to the symmetrical positions of the two magnets with respect to the object being tracked.

Appendix

Matlab Code for Particle Filter

```
% FILE NAME      : Particle_Filter.m
%
% DESCRIPTION    : Code to implement Particle filter.
%
% PLATFORM              : Matlab
%
% DATE                       NAME
% 19th-Nov-2018      Shashi Shivaraju

clc; %clear all the varaibles
close all;%close all windows
clear; %clear the screeen

%Read data from file
Data = dlmread("magnets-data.txt");
ActualPosition = Data(:,1);%Actual position of data for reference
ActualVelocity = Data(:,2);%Actual velocity of data for reference
SensorMeasurement = Data(:,3);%Sensor measurement data

%Positions of Magnets
xm1 = -10;
xm2 = 10;

%Number of Particles in Filter
M = 1000;

%Standard Deviations
SigmaA = 0.0625; %Dynamic noise
SigmaN = 0.003906; %Measurement noise
SigmaM = 4;

%State of each particle
XPos = zeros(1,M);
XVel = zeros(1,M);
XPrevPos = zeros(1,M);
XPrevVel = zeros(1,M);

%Weight of each particle
weight = ones(1,M) * 1/M;
```

```
weightPrev = ones(1,M) * 1/M;
weightUpdated = ones(1,M) * 1/M;

%Ideal measurement for each particle
Sensor_Ideal = zeros(1,M);

%Particle Filter Output
PF_Output = zeros(1,length(SensorMeasurement));
%Index of Weights
Weight_Index= 1 : M;

%Resampling
Q = zeros(1,M); %calculate the running totals
T = zeros(1,M+1);%Array of M+1 uniform random numbers 0 to 1
RSCount = 0;% Total number of resampling done during filtering
PlotWeights = 0;%Flag for plotting the weights between one resample cycle
Select_ResampleCount = 27; %Select the resample count after which weights have to be plotted

%loop through the each sensor reading
for t = 1:length(SensorMeasurement)

    %loop through each particle in filter
    for i = 1:M

            % Update each particle as per state transition equation
            XPos(i) = XPrevPos(i) + XPrevVel(i) ;
            if( XPrevPos(i) < -20)
                XVel(i) = 2;
            elseif (XPrevPos(i) > 20)
                XVel(i) = -2;
            elseif (XPrevPos(i) >= 0 && XPrevPos(i) <= 20 )
                XVel(i) = XPrevVel(i) - abs(randn * SigmaA);
            elseif (XPrevPos(i) >= -20 && XPrevPos(i) < 0)
                XVel(i) = XPrevVel(i) + abs(randn * SigmaA);
            end

            %Ideal measurement of the particle
            Sensor_Ideal(i) = (1 / (sqrt(2*pi) * SigmaM))  * exp( -((XPrevPos(i) - xm1  )^2) / (2 * (SigmaM^2) )) + (1 / (sqrt(2*pi) * SigmaM))  * exp( -((XPrevPos(i) - xm2  )^2) / (2 * (SigmaM^2) ));
```

```
        %Calculate Probability by comparing the ideal measurement against the actual measurement
        Prob_Yt_Xtm = ((1 / (sqrt(2*pi) * SigmaN)) * exp ( -((Sensor_Ideal(i) - SensorMeasurement(t) )^2) / (2 * (SigmaN^2) )));
        %Update weight of the particle
        weightUpdated(i) = weightPrev(i) * Prob_Yt_Xtm;

        %Store the previous values
        XPrevPos(i) = XPos(i);
        XPrevVel(i) = XVel(i);
        weightPrev(i) = weightUpdated(i);
    end

    %Initialization
    Index = zeros(1,M);

    %Find the cumulative weight
    weightSum = 0;
    for k = 1:M
        weightSum = weightSum + weightUpdated(k) ;
    end

    %Normalize the weights so they add up to 1
    for k = 1:M
        weight(k) = weightUpdated(k) / weightSum ;
    end

    %Calculate the Particle Filter Output and Coefficient of Variation
    Filter_Output = 0;
    CV = 0;
    for k = 1:M
        %Expected Filter Output
        Filter_Output = Filter_Output + (weight(k) * XPos(k));
        %Coefficient of Variation
        CV = CV + (((M * weight(k)) - 1) ^ 2);
    end

    %Coefficient of Variation
    CV = 1/M * CV;
    %Particle Filter Output
    PF_Output(t) = Filter_Output;
    %Effective Sampling Size
    ESS = M / (1 + CV);
```

```
    %Plot weights if flag is on
    if(PlotWeights)
        figure(t)
        bar(Weight_Index, weight,'k')
        axis([0 M 0 0.0075])
        xlabel("Particle");
        ylabel("Weight");
        set(gca,'FontSize',14)
        disp(strcat('iteration = ',num2str(t),' ESS = ',num2str(ESS)));
    end

     %Resample the weights
     if( ESS < 0.5 * M )

        %Calculate the running totals
        Q(1) = weight(1);
        for k = 2:M
            Q(k) = Q(k-1) + weight(k);
        end
        %T is an array of M+1 uniform random numbers 0 to 1
        T = rand(1,M);
        T(k+1) = 1; %Boundary condition for cumulative hist
        %Sort them smallest to largest
        T = sort(T);
        %Arrays start at 1
        i = 1;
        j = 1;
        while( i <= M )
            if( T(i) < Q(j) )
                Index(i) = j;
                i = i+1;
            else
                j = j+1;
            end
        end

        %Update the states and weights of the particles
        for i = 1:M
            XPos(i) = XPos(Index(i));
            XPrevPos(i) = XPrevPos(Index(i));
            XVel(i) = XVel(Index(i));
            XPrevVel(i) = XPrevVel(Index(i)) ;
            weight(i) = 1/M;
```

```
            weightPrev(i) = 1/M;
            weightUpdated(i) = 1/M;
        end

        %Increase the resample count
        RSCount = RSCount + 1;

        %Update the flag to plot the weights during resampling
        if(RSCount == Select_ResampleCount)
            PlotWeights = 1;
        end

        %Plot resampled weights
        if(PlotWeights)
            figure(t)
            bar(Weight_Index, weight,'k')
            axis([0 M 0 0.0075])
            xlabel("Particle");
            ylabel("Weight");
            set(gca,'FontSize',14)
        end

        %Reset the flag to plot weights
        if(RSCount == Select_ResampleCount + 1)
           PlotWeights = 0;
        end

    end
end

%Plot the actual position and position obtained from Particle filter
PF_Index = 0 : length(ActualPosition)-1;   %Index of Particle Filter output
figure(1)
plot(PF_Index,ActualPosition,'k','LineWidth', 0.7);
hold on
plot(PF_Index,PF_Output,'k','LineWidth', 1,'Marker','.','MarkerSize',10);
hold off
xlabel("time samples");
ylabel("Position");
set (gca,"FontSize",14);
legend("Actual Position", "PF Output");
```

References

1.Lecture notes of Dr.Adam Hoover

http://cecas.clemson.edu/ ahoover/ece854/lecture-notes/lecture-pf.pdf

2.TeX Live - TeX Users Group

https://www.tug.org/texlive/