Homework 7 involved estimating a financial time series using a particle filter and an EKF. For this task, a second particle filter with roughening added was also considered. The financial time series was a scalar problem with dynamics and measurement equations as follows:

$$x_k = \frac{1}{2}x_{k-1} + \frac{25x_{k-1}}{1+x_{k-1}^2} + 8\cos[1.2(k-1)] + w_k$$

$$y_k = \frac{1}{20}x_k^2 + v_k$$

The system is very nonlinear as can be seen from the model equations and the measurement equation is also nonlinear. Therefore, it is predicted that normal EKF will not perform very well in this problem.

5 different figures were created after running the simulation for all three filters. The first figure shows the plots of the estimation with 95% and 5% confidence bounds for the particle filter, EKF and particle filter with roughening respectively. The next figure shows the estimation error for all three filters. The third figure shows the covariance values of each filter and the fourth and last figures show the evolution of the pdf's of the particle filters.

Figure 1 (below) shows the plots of the actual state and the filter estimations with upper and lower confidence bounds. The top plot is of the standard particle filter, the middle is the EKF and the bottom plot is the particle filter with roughening. It is easy to see that the estimation of both particle filters is far better than the EKF for most of the state trajectory (generally an order of magnitude better). It is somewhat difficult to draw conclusions from the two particle filter plots due to a difference in scaling. From this figure, it appears that there is only slight differences between the standard particle filter and the particle filter with roughening. It is possible that roughening might only have a small effect given this specific problem and the number of sample points (500). The estimation error plots in figure 2 will reveal more information about the filter accuracy.

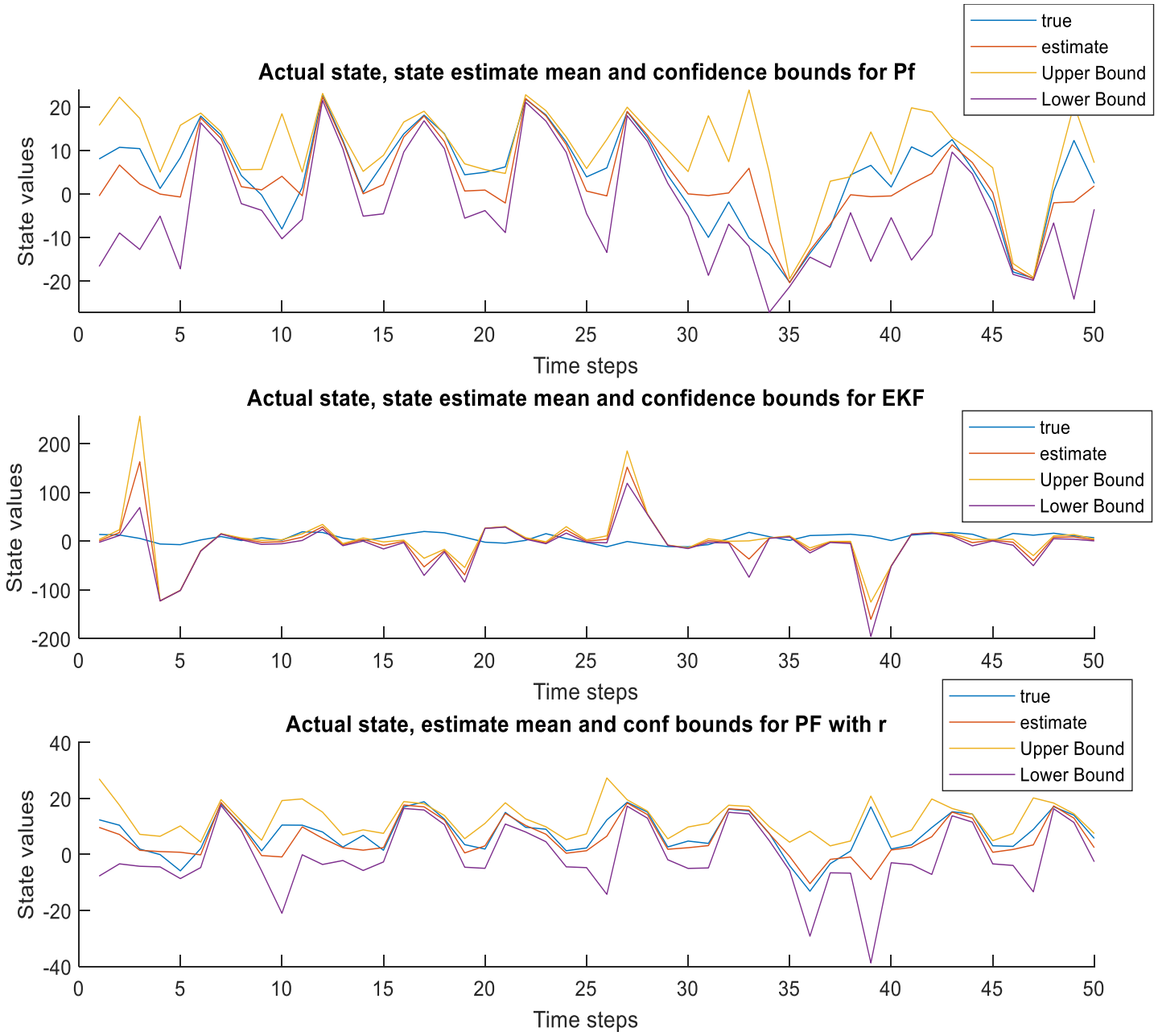
Actual state, state estimate mean and confidence bounds for Pf
true
estimate
Upper Bound
Lower Bound
State values
Time steps
Actual state, state estimate mean and confidence bounds for EKF
true
estimate
Upper Bound
Lower Bound
State values
Time steps
Actual state, estimate mean and conf bounds for PF with r
true
estimate
Upper Bound
Lower Bound
State values
Time steps

Fig. 1

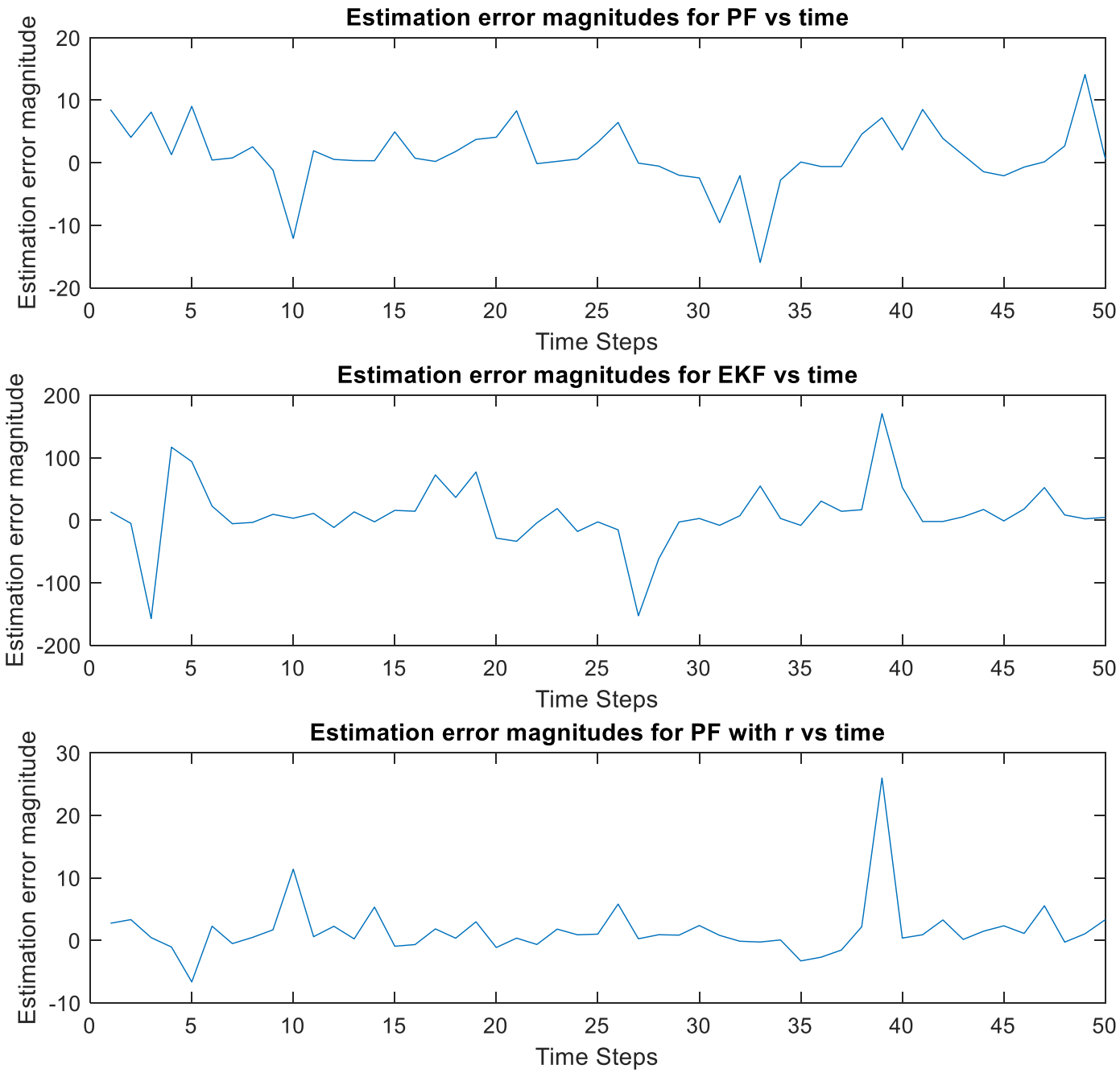

Fig. 2

Figure 2, above, shows the estimation errors of each filter over time. Again, it is not difficult to see the advantage that the particle filters have over the EKF, the estimation error being about an order of magnitude better for most of the simulation time. While the particle filter plots in this figure suffer some scaling issues as well, a clearer comparison of each filter's performance can be readily seen. The

particle filter with roughening estimation errors are noticeably smaller in this plot (except for one spot near the 40 second mark) than the regular particle filter. I think this is because of the improved distributions of the particles inherent in the roughening technique. However, the large increase near the 39 second mark is of some concern. I'm not entirely sure what caused this spike in error. A similar spike can be seen in the EKF so perhaps it is a symptom of the system dynamics.

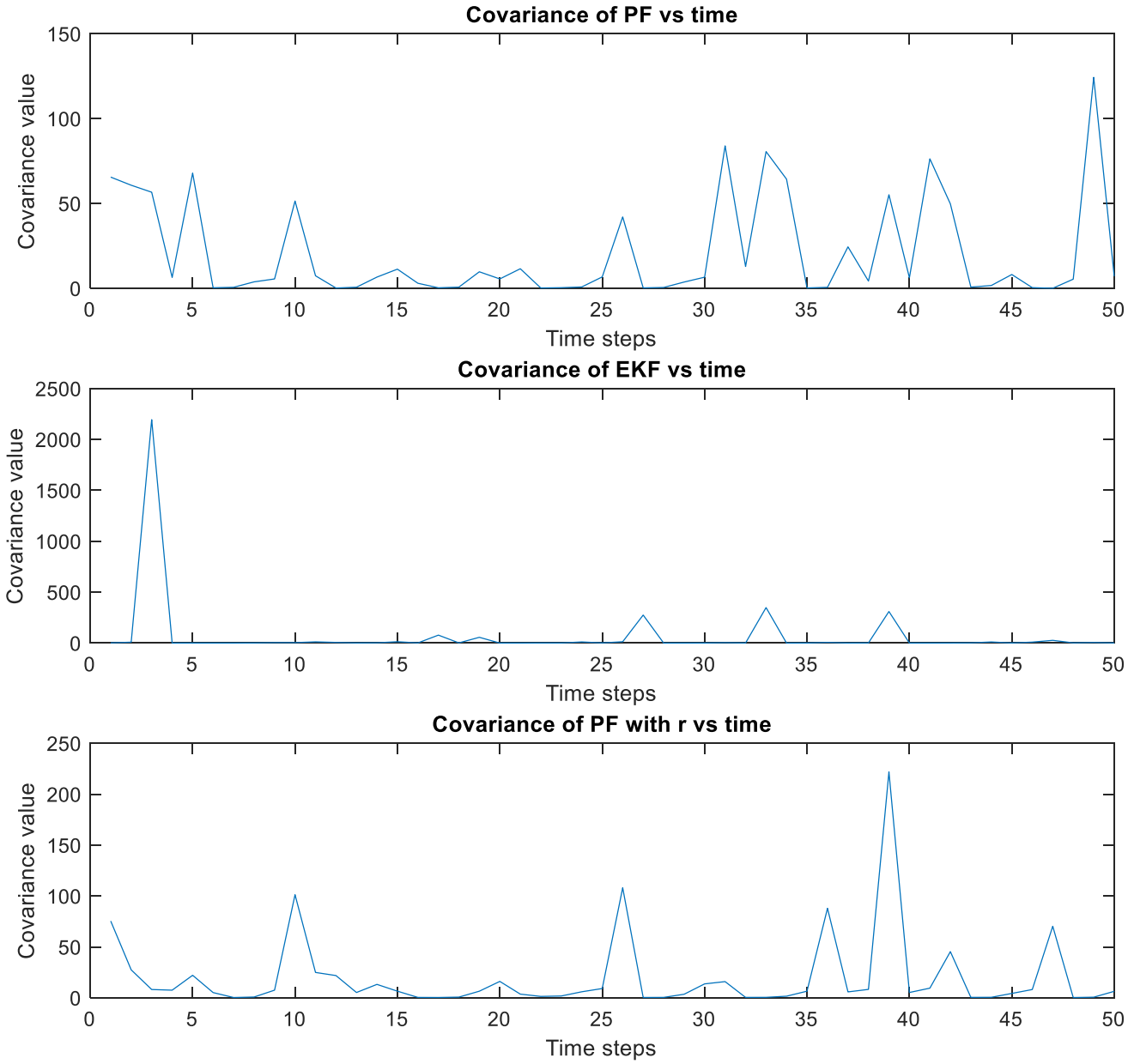

Fig. 3

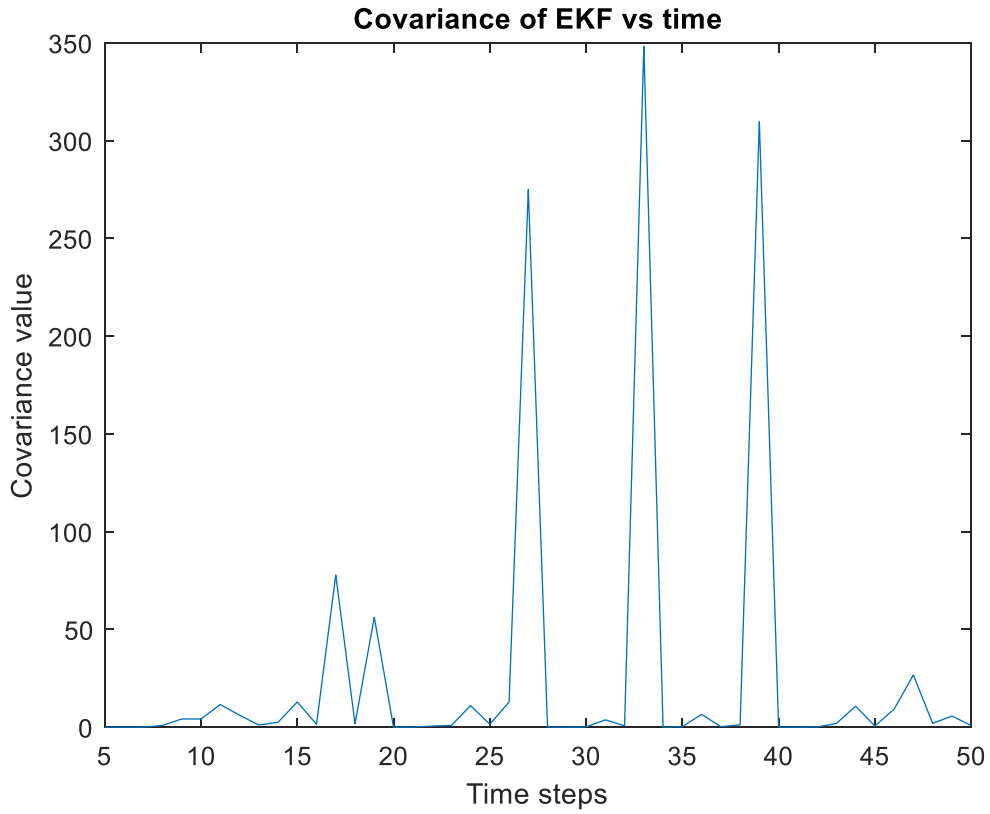

Fig. 3 Auxiliary Plot

Figure 3, above, shows the covariance for each filter. Because of the scaling issue in the EKF plot, an auxiliary plot was made showing the values from 5 to 50 seconds. It is interesting to note that all three plots show a spike in uncertainty around the 39 second mark, which is where the largest error for the particle filter with roughening was observed. Otherwise, the covariance plots for the three filters look fairly similar. Not surprisingly, EKF shows the highest uncertainty of the three filters. What is of more concern is that the EKF frequently shows areas of low uncertainty that do not match up with the estimation plots. This means that the filter thinks its estimation is very good even when it is not. The particle filter with roughening appears to be the most confident except for a few specific points. This makes sense as it had the smallest average estimation error of the three filters.

The last two figures, figures 4 and 5 shown below, depict the progression of the particle distribution over time for both the standard particle filter and the particle filter with roughening. It is interesting to observe the difference in PDF's for the two filters at equal time steps. I think adding roughening to the filter generally causes the PDF to take a more gaussian look at most time steps. I also noticed that the PDF's from the roughened particle filter generally have a larger spread of values. I believe this is proof of the concept of roughening working to prevent the samples from converging to one point. This is most apparent when comparing PDF's at k=35. The spread for the standard particle filter is only 3 while the particle filter with roughening has a spread of 15.

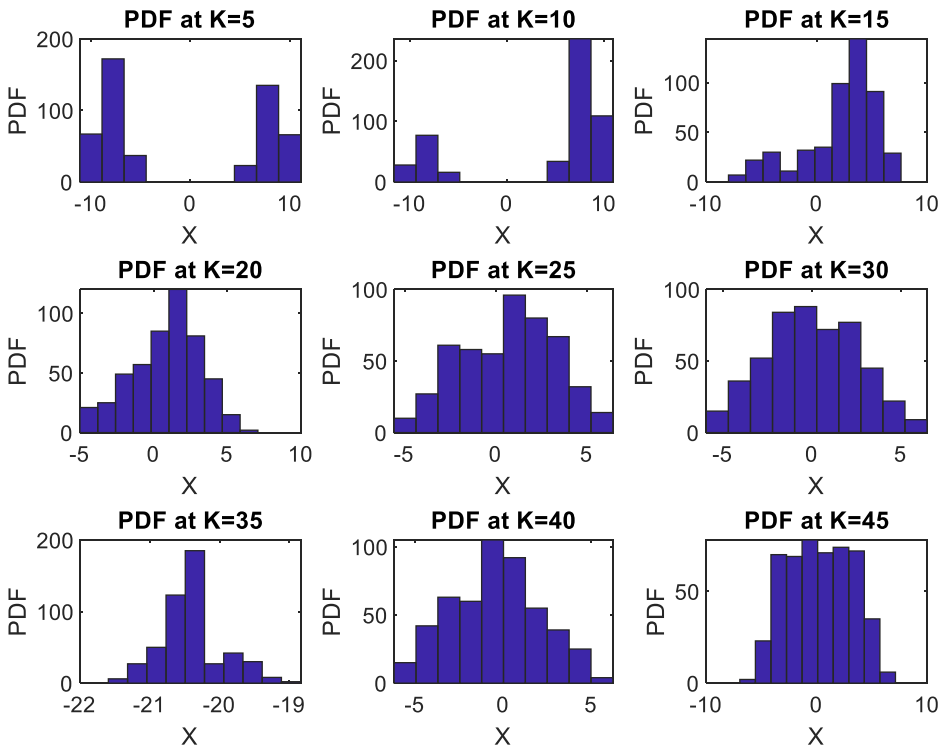

Fig. 4: PDF Progression of Standard Particle Filter

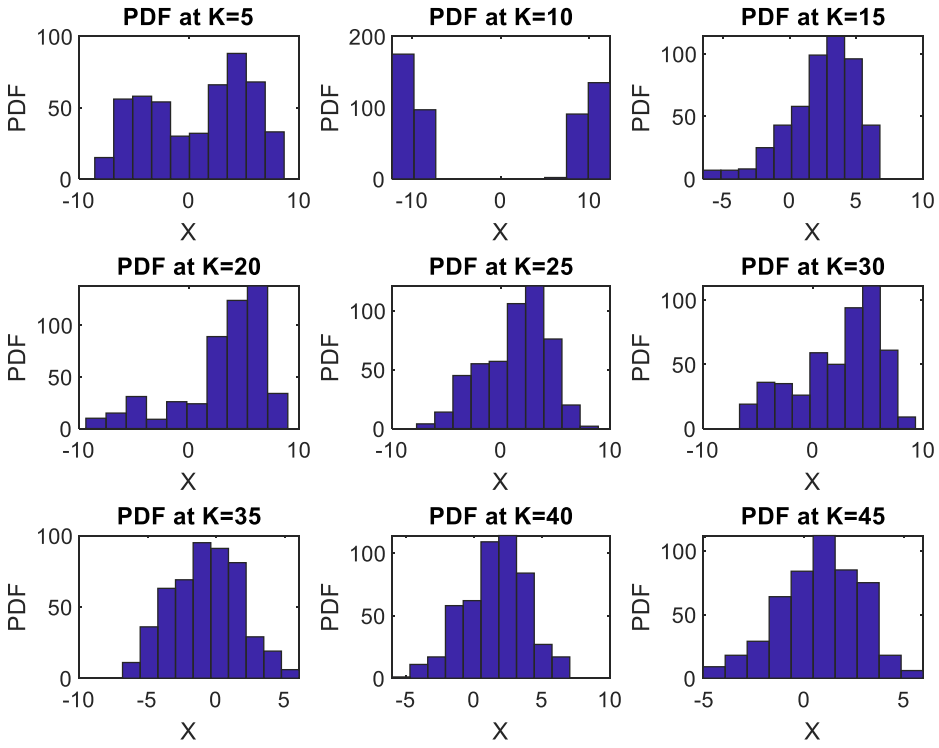

Fig. 5: PDF Progression of Particle Filter with Roughening

```
>> type disc_EKF.m

function [xact,xhatp,Pp,T] = disc_EKF(x0,xhat0,P0,w,v,Q,R,r,q,t)
%This function simulates a system and implements a discrete extended kalman
%filter
%This function will need to be manually updated on a problem by problem
%basis
%x0 is the actual system initial conditions, nx1
%x_hat0 is the initial estimate of the system, nx1
%P0 is the initial estimate of the covariance matrix, nxn
%w is the ceoff matrix for the system noise, nx1
%v is the ceoff matrix for the measurement noise, m x 1
%q is the variance for the system noise
%r is the variance for the measurement noise
%t is the amount of time (in seconds to simulate the system)

%define constants used in problem
T=0.1; %seconds
num_steps=1:T:t;
N=[20 0];
E=[0 20];

%storage for important variables
xact=zeros(length(x0),length(num_steps));
xhatm=zeros(length(xhat0),length(num_steps));
xhatp=zeros(length(xhat0),length(num_steps));
[rp,cp]=size(P0);
Pm=zeros(rp,cp,length(num_steps));
Pp=zeros(rp,cp,length(num_steps));

%discrete EKF algorithm
for i=1:length(num_steps)
    %simulate actual system
    if i==1
        %initialization
        xact(:,i)=x0;
    else
        %noise
        W=normrnd(0,q)*w;
        V=normrnd(0,r)*v;
        %system matrices
        f=[1 0 T 0;0 1 0 T;0 0 1 0;0 0 0 1];
        h=[sqrt(((xact(1,i-1)-N(1))^2)+((xact(2,i-1)-E(1))^2));...
            sqrt(((xact(1,i-1)-N(2))^2)+((xact(2,i-1)-E(2))^2))];
        xact(:,i)=f*xact(:,i-1)+W;
        y=h+V;
    end
    if i==1
        %initialization
        xhatp(:,i)=xhat0;
        Pp(:,:,i)=P0;
```

```
    else
        %partial differential matrices
        F=f;
        L=eye(4,4);
        %covariance minus and xhat minus
        Pm(:,:,i)=F*Pp(:,:,i-1)*F'+L*Q*L';
        xhatm(:,i)=f*xhatp(:,i-1);
        h_xhatm=[sqrt(((xhatm(1,i)-N(1))^2)+((xhatm(2,i)-E(1))^2));...
            sqrt(((xhatm(1,i)-N(2))^2)+((xhatm(2,i)-E(2))^2))];
        h11=(xhatm(1,i)-N(1))/h_xhatm(1);
        h12=(xhatm(2,i)-E(1))/h_xhatm(1);
        h21=(xhatm(1,i)-N(2))/h_xhatm(2);
        h22=(xhatm(2,i)-E(2))/h_xhatm(2);
        H=[h11, h12 0 0;h21 h22 0 0];
        M=eye(2,2);
        %gain xhatp and Pp
        K=Pm(:,:,i)*H'/(H*Pm(:,:,i)*H'+M*R*M');
        xhatp(:,i)=xhatm(:,i)+K*(y-h_xhatm);
        Pp(:,:,i)=(eye(4,4)-K*H)*Pm(:,:,i);
    end
end

>>
```

```
>> type particle_filter.m

function [x,u_xhatp,xhatp,P] = particle_filter(x0,t)
%This function implements a particle filter given a set of initial
%conditions. Due to the complexity of most problems, this algorithm will
%need to be updated on a per-problem basis
%note this function is currently setup for a scalar problem

%calculate number of time steps
num_steps=1:t;

%choose particles
%set number of particles, choose 500 for this problem
N=500;

%create storage
%note that the storage was setup for a scalar problem and that it will need
%to be modified for a vector problem by implementing 3D matrices
x=zeros(length(x0),length(num_steps));
y=zeros(1,length(num_steps));
xhatm=zeros(N,length(num_steps));
xhatp=zeros(N,length(num_steps));
q=zeros(N,length(num_steps));
qs=zeros(N,length(num_steps));
u_xhatp=zeros(1,length(num_steps));
P=zeros(1,length(num_steps));

%useful variables
%m=number of measurement equation dimensions
m=1;
R=1;
Q=10;

%Initial pdf of X
u_xhatp0=0;
var_xhatp0=2;

%generate initial particles
xhatp0=normrnd(u_xhatp0,sqrt(var_xhatp0),N,1);

%particle filter algorithm
for k=1:t
    %random variables
    W=normrnd(0,sqrt(Q),1,1);
    V=normrnd(0,sqrt(R),1,1);
    %actual system simulation
    if k==1
        %initialize system
        uk=8*cos(1.2*(num_steps(k)-1));
        x(:,1)=(0.5*x0+(25*x0)/(1+x0^2)+uk+W);
    else
```

```
    uk=8*cos(1.2*(num_steps(k)-1));
    x(:,k)=(0.5*x(:,k-1)+(25*x(:,k-1))/(1+x(:,k-1)^2)+uk+W);
end
%measurement
y(:,k)=((x(:,k)^2)/20)+V;
%time propogation
if k==1
    for i=1:N
        %generate noise for every particle
        W=normrnd(0,sqrt(Q),1,1);
        xhatm(i,k)=(0.5*xhatp0(i)+(25*xhatp0(i))/(1+xhatp0(i)^2)+uk+W);
    end
else
    for i=1:N
        %generate noise for every particle
        W=normrnd(0,sqrt(Q),1,1);
        xhatm(i,k)=(0.5*xhatp(i,k-1)+...
            (25*xhatp(i,k-1))/(1+xhatp(i,k-1)^2)+uk+W);
    end
end
%compute relative likelihood qi of each particle
for i=1:N
    h=((xhatm(i,k)^2)/20);
    q1=((2*pi)^(m/2)*abs(R)^(1/2))^-1;
    q2=exp(0.5*(-(y(:,k)-h)'*(R^-1)*(y(:,k)-h)));
    q(i,k)=q1*q2;
end
%scale relatively likelihoods
qsum=0;
for i=1:N
    qsum=qsum+q(i,k);
end
qs(:,k)=q(:,k)/qsum;
%generate a posteriori particles on the basis of the relative
%likelihoods (resampling step)
for i=1:N
    qsum2=0;
    r=rand; %uniform random number on interval (0,1)
    for j=1:N
        qsum2=qsum2+qs(j,k);
        if qsum2<abs(r)
            continue
        else
            break
        end
    end
    xhatp(i,k)=xhatm(j,k);
end
%compute desired statistical measurements
%compute mean of particles
u_xhatp(k)=mean(xhatp(:,k));
```

```
    %calculate sample covariance
    cov_sum=0;
    for i=1:N
        cov=(xhatp(i,k)-u_xhatp(k))*(xhatp(i,k)-u_xhatp(k))';
        cov_sum=cov_sum+cov;
    end
    P(:,k)=cov_sum/N;
end

>>
```

```
>> type pf_wr.m

function [x,u_xhatp,xhatp,P] = pf_wr(x0,t)
%This function implements a particle filter with roughening given a set
%of initialconditions. Due to the complexity of most problems,
%this algorithm will need to be updated on a per-problem basis
%note this function is currently setup for a scalar problem

%calculate number of time steps
num_steps=1:t;

%choose particles
%set number of particles, choose 500 for this problem
N=500;

%create storage
%note that the storage was setup for a scalar problem and that it will need
%to be modified for a vector problem by implementing 3D matrices
x=zeros(length(x0),length(num_steps));
y=zeros(1,length(num_steps));
xhatm=zeros(N,length(num_steps));
xhatp=zeros(N,length(num_steps));
q=zeros(N,length(num_steps));
qs=zeros(N,length(num_steps));
u_xhatp=zeros(1,length(num_steps));
P=zeros(1,length(num_steps));

%useful variables
%m=number of measurement equation dimensions
m=1;
R=1;
Q=10;

%Initial pdf of X
u_xhatp0=0;
var_xhatp0=2;

%generate initial particles
xhatp0=normrnd(u_xhatp0,sqrt(var_xhatp0),N,1);

%particle filter algorithm
for k=1:t
    %random variables
    W=normrnd(0,sqrt(Q),1,1);
    V=normrnd(0,sqrt(R),1,1);
    %actual system simulation
    if k==1
        %initialize system
        uk=8*cos(1.2*(num_steps(k)-1));
        x(:,1)=(0.5*x0+(25*x0)/(1+x0^2)+uk+W);
    else
```

```
    uk=8*cos(1.2*(num_steps(k)-1));
    x(:,k)=(0.5*x(:,k-1)+(25*x(:,k-1))/(1+x(:,k-1)^2)+uk+W);
end
%measurement
y(:,k)=((x(:,k)^2)/20)+V;
%time propogation
if k==1
    for i=1:N
        %generate noise for every particle
        W=normrnd(0,sqrt(Q),1,1);
        xhatm(i,k)=(0.5*xhatp0(i)+(25*xhatp0(i))/(1+xhatp0(i)^2)+uk+W);
    end
else
    for i=1:N
        %generate noise for every particle
        W=normrnd(0,sqrt(Q),1,1);
        xhatm(i,k)=(0.5*xhatp(i,k-1)+...
            (25*xhatp(i,k-1))/(1+xhatp(i,k-1)^2)+uk+W);
    end
end
%compute relative likelihood qi of each particle
for i=1:N
    h=((xhatm(i,k)^2)/20);
    q1=((2*pi)^(m/2)*abs(R)^(1/2))^-1;
    q2=exp(0.5*(-(y(:,k)-h)'*(R^-1)*(y(:,k)-h)));
    q(i,k)=q1*q2;
end
%scale relatively likelihoods
qsum=0;
for i=1:N
    qsum=qsum+q(i,k);
end
qs(:,k)=q(:,k)/qsum;
%generate a posteriori particles on the basis of the relative
%likelihoods (resampling step)
for i=1:N
    qsum2=0;
    r=rand; %uniform random number on interval (0,1)
    for j=1:N
        qsum2=qsum2+qs(j,k);
        if qsum2<abs(r)
            continue
        else
            break
        end
    end
    xhatp(i,k)=xhatm(j,k);
end
%Roughening step
%designed to prevent sample impoverishment
%sample impoverishment occurs when not enough a priori particles are
```

```
    %selected to become a posteriori particles due to differences in the
    %prior and posterior pdf
    %the idea is to add artifical noise to every a posteriori particle
    %such that the posterior pdf is broadened. This will in turn broaden
    %the prior pdf in the next time step.
    K=0.2;      %a user defined hyperparameter that determines how much
    %'jitter' is given to each particle
    n=1; %dimensionality of state space (this problem is scalar)
    for i=1:N    %loop through every particle
        for m=1:n    %loop through every dimension
            M=max(xhatp(:,k))-min(xhatp(:,k));  %only one dimension
            deltx=normrnd(0,K*M*N^(-1/n));
            xhatp(i,k)=xhatp(i,k)+deltx;
        end
    end
    %compute desired statistical measurements
    %compute mean of particles
    u_xhatp(k)=mean(xhatp(:,k));
    %calculate sample covariance
    cov_sum=0;
    for i=1:N
        cov=(xhatp(i,k)-u_xhatp(k))*(xhatp(i,k)-u_xhatp(k))';
        cov_sum=cov_sum+cov;
    end
    P(:,k)=cov_sum/N;
end
>>
```