Optimal Estimation Methods

(Lecture 19 – Particle Filtering: Part II)

Dr. John L. Crassidis

University at Buffalo – State University of New York
Department of Mechanical & Aerospace Engineering
Amherst, NY 14260-4400
johnc@buffalo.edu
http://www.buffalo.edu/~johnc

University at Buffalo The State University of New York

Importance Density

- One that minimizes variance of importance weights

$$q(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)},\tilde{\mathbf{y}}_{k+1})_{\mathrm{opt}} = p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)},\tilde{\mathbf{y}}_{k+1})_{\mathrm{opt}}$$
$$= \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1},\mathbf{x}_k^{(i)})\,p(\mathbf{x}_{k+1},\mathbf{x}_k^{(i)})}{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_k^{(i)})}$$

  - Leads to

$$\boxed{\varpi_{k+1}^{(i)} \propto \varpi_k^{(i)} p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_k^{(i)})}$$

weights can be computed before particles are propagated

  - Problems
    - Must be able to sample from $p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)},\tilde{\mathbf{y}}_{k+1})$
    - To within a normalizing constant we must evaluate

$$p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_k^{(i)}) = \int p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1})\,p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)})\;d\mathbf{x}_k$$

    - Both are not easy to do!
  - Special case: additive Gaussian process noise and linear measurement equation $\Rightarrow$ can use optimal importance density

Popular Choice

- Most widely used choice given by transitional prior

$$q(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)}, \tilde{\mathbf{y}}_{k+1}) = p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)})$$

  - Suboptimal, but it's easy to compute if process noise is additive

$$p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)}) = N(\mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k), \Upsilon_k Q_k \Upsilon_k^T)$$

  - Weights are given by

$$\varpi_{k+1}^{(i)} \propto \varpi_k^{(i)} p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i)})$$

weights must be computed after particles are propagated

  - Known as "Bootstrap" or "Sampling Importance Resampling" (SIR) filter
- Main problem
  - If $p(\mathbf{x}_{k+1}|\mathbf{x}_k)$ is a much broader distribution than the likelihood, $p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1})$, then only a few particles will be assigned high weight
    - Particles will degenerate very quickly
    - Auxiliary Particle Filter (discussed later) may help in this case

SIR Filter Pseudocode

$$[\{\mathbf{x}_{k+1}^{(i)}, \varpi_{k+1}^{(i)}\}_{i=1}^{N}] = \mathrm{SIR}[\{\mathbf{x}_{k}^{(i)}, \varpi_{k}^{(i)}\}_{i=1}^{N}, \tilde{\mathbf{y}}_{k+1}]$$

- FOR $i = 1 : N$
  - Draw $\mathbf{x}_{k+1}^{(i)} \sim p(\mathbf{x}_{k+1}|\mathbf{x}_{k}^{(i)})$
  - Evaluate non-normalized weights $\tilde{\varpi}_{k+1}^{(i)} = \varpi_{k}^{(i)} p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i)})$
- END FOR
- Calculate total weight $\varpi_{\text{tot}} = \sum_{i=1}^{N} \tilde{\varpi}_{k+1}^{(i)}$
- FOR $i = 1 : N$
  - Normalize: $\varpi_{k+1}^{(i)} = \tilde{\varpi}_{k+1}^{(i)} / \varpi_{\text{tot}}$
- END FOR
- Resample
- Regularize if Desired

A Simple Linear Example

- Truth model

$$x_{k+1} = x_k + w_k, \quad w_k \sim N(0, 10)$$
$$\tilde{y}_k = x_k + v_k, \quad v_k \sim N(0, 1)$$

- Bootstrap filter with $x_0^{(i)} \sim N(0, 5), \quad i = 1,\, 2,\, \ldots,\, 500$
  - Prediction

$$x_{k+1}^{(i)} = x_k^{(i)} + w_k^{(i)}, \quad w_k^{(i)} \sim N(0, 10)$$

  - Update

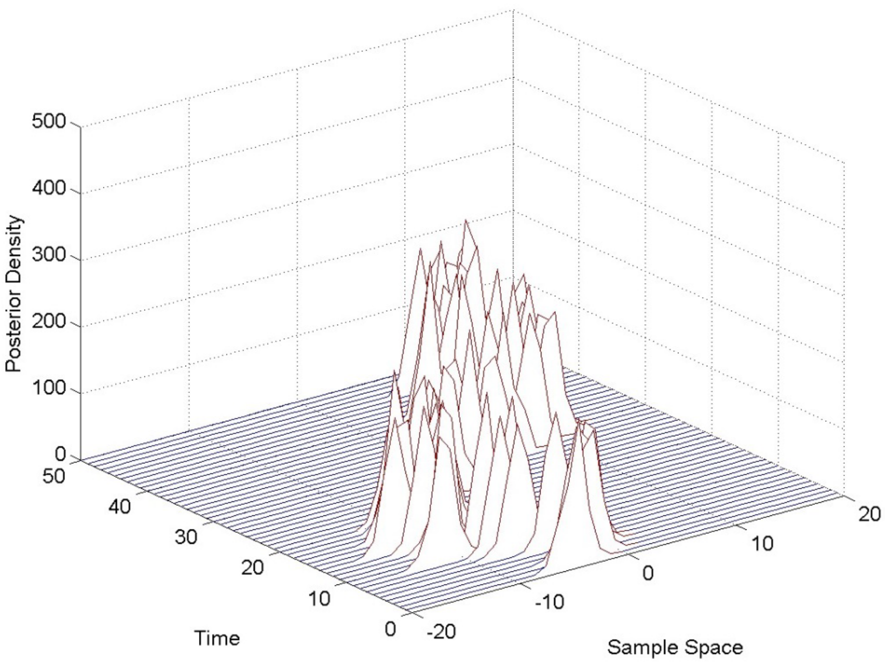
Gaussian

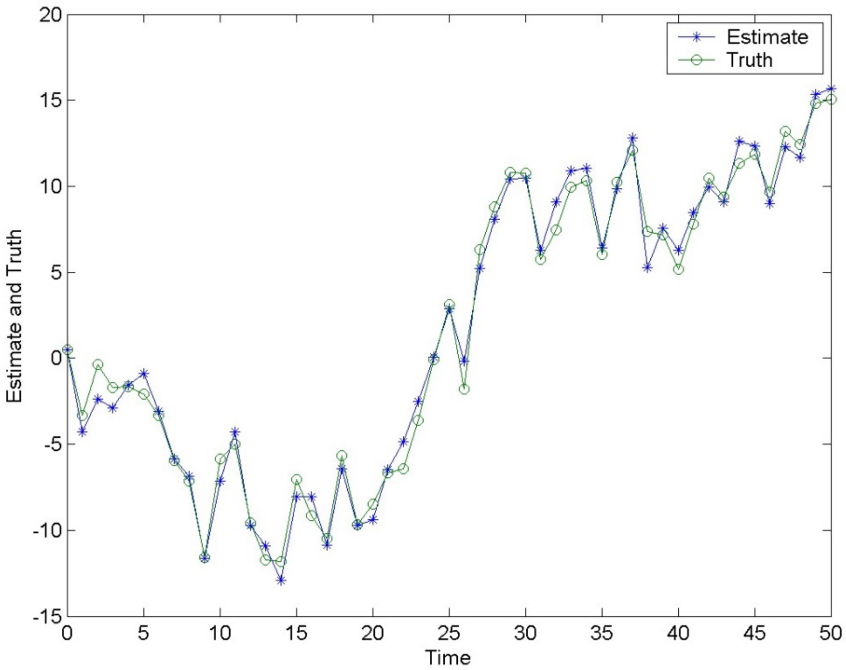
$$\varpi_{k+1}^{(i)} = \varpi_k^{(i)} \exp\left[-\frac{(\tilde{y}_{k+1} - x_{k+1}^{(i)})^2}{2 \times 1}\right]$$

Measurement Variance

$$\varpi_{k+1}^{(i)} = \varpi_{k+1}^{(i)} / \textstyle\sum_{i=1}^{N} \varpi_{k+1}^{(i)}$$

Linear Example Results





The posterior distribution looks Gaussian

Code (i)

```
% Time
m=51;t=[0:1:50]';

% Standard Devations
sig_x=sqrt(5);sig_y=1;sig_w=sqrt(10);

% States and Measurements
x=zeros(m,1);x(1)=sig_x*randn(1);x_est=zeros(m,1);
ym=zeros(m,1);ym(1)=x(1)+sig_y*randn(1);

% Particles
n=500;x_particle=sig_x*randn(n,1);
x_est(1)=mean(x_particle);
w_particle=inv(n)*ones(n,1);

clf;hold on
```

Code (ii)

```
% Main Loop
for i = 1:m-1
  x(i+1) = x(i) + sig_w*randn(1);ym(i+1) = x(i+1) + sig_y*randn(1);
  x_particle = x_particle + sig_w*randn(n,1);
  w_particle = w_particle.*exp(-(ym(i+1)-x_particle).^2/(2*sig_y^2));
  w_particle = w_particle/sum(w_particle);
   x_est(i+1)=sum(x_particle.*w_particle);

  [x_particle,w_particle]=resample_pf(x_particle,w_particle);

  caxis([0 1]); [range,domain]=hist(x_particle,[-20:1:20]);
  set(gca,'fontsize',12);waterfall((domain),i,range)
end

hold off

axis([-20 20 0 50 0 500]);
ylabel('Time','fontsize',12);
xlabel('Sample Space','fontsize',12);
zlabel('Posterior Density','fontsize',12)
```

Code (iii)

```
function [x_resamp,w_resamp,index]=resample_pf(x_particle,w_particle);

% Get Length of Particles
n=length(x_particle);

% Cumulative Sum of Particles
w_particle=w_particle(:);
c=cumsum(w_particle);

% Compute u Vector
u=zeros(n,1);
u(1)=inv(n)*rand(1);
u(2:n)=u(1)+inv(n)*(1:n-1)';

% Pre-allocate Index
index=zeros(n,1);
```

Code (iv)

```
% Compute Index for Resampling
i=1;
for j=1:n
    while u(j)>c(i)
        i=i+1;
    end
    index(j)=i;
end

% Resampled Data
x_resamp=x_particle(index,:);
w_resamp=inv(n)*ones(n,1);
```

A Simple Nonlinear Example

- Truth model (Nando de Freitas)

$$x_{k+1} = \frac{x_k}{2} + \frac{25x_k}{1+x_k^2} + 8\cos(1.2k) + w_k, \quad w_k \sim N(0,10)$$

$$\tilde{y}_k = \frac{x_k^2}{20} + v_k, \quad v_k \sim N(0,1)$$

- Bootstrap filter with $x_0^{(i)} \sim N(0,5), \quad i = 1,\, 2,\, \ldots,\, 500$
  - Prediction

$$x_{k+1}^{(i)} = \frac{x_k^{(i)}}{2} + \frac{25x_k^{(i)}}{1+\left(x_k^{(i)}\right)^2} + 8\cos(1.2k) + w_k^{(i)}, \quad w_k^{(i)} \sim N(0,10)$$

  - Update

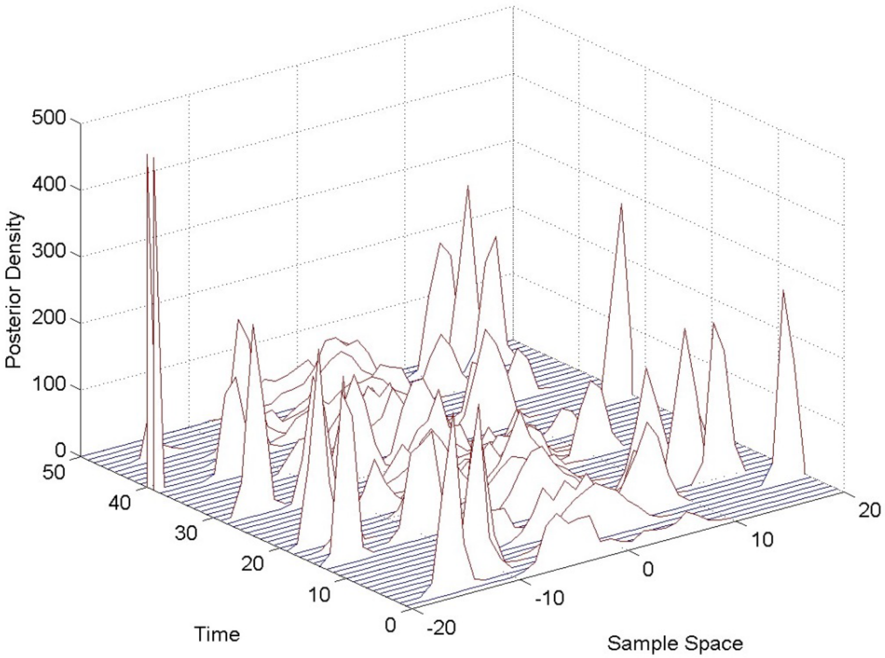
Gaussian

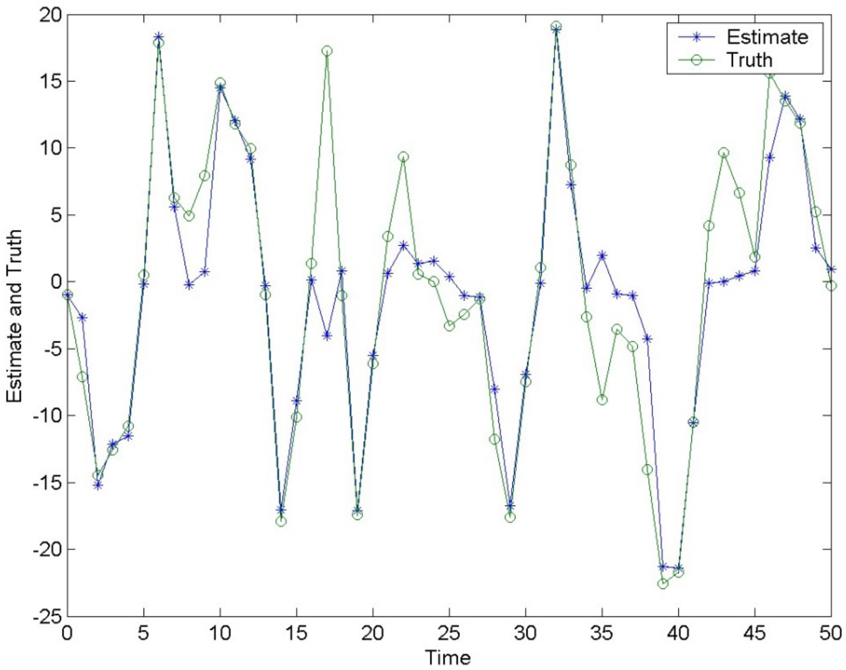
$$\varpi_{k+1}^{(i)} = \varpi_k^{(i)} \exp\left[-\frac{\left(\tilde{y}_{k+1} - \left(x_{k+1}^{(i)}\right)^2/20\right)^2}{2\times 1}\right]$$

Measurement Variance

$$\varpi_{k+1}^{(i)} = \varpi_{k+1}^{(i)} / \sum_{i=1}^{N} \varpi_{k+1}^{(i)}$$

Nonlinear Example Results





The posterior distribution is strongly non-Gaussian (multi-peaked) though all the noise is Gaussian

Code (i)

```
% Time
m=51;t=[0:1:50]';

% Standard Devations
sig_x=sqrt(5);sig_y=1;sig_w=sqrt(10);

% States and Measurements
x=zeros(m,1);x(1)=sig_x*randn(1);x_est=zeros(m,1);
ym=zeros(m,1);ym(1)=x(1)^2/20+sig_y*randn(1);

% Particles
n=500;x_particle=sig_x*randn(n,1);
x_est(1)=mean(x_particle);
w_particle=inv(n)*ones(n,1);

clf;hold on
```

Code (ii)

```
% Main Loop
for i = 1:m-1
  x(i+1) = x(i)/2 + 25*x(i)/(1+x(i)^2) + 8*cos(1.2*i) + sig_w*randn(1);
  ym(i+1)= x(i+1)^2/20+ sig_y*randn(1);
 x_particle = x_particle/2 +25*x_particle./(1+x_particle.^2) + ...
                   8*cos(1.2*i) + sig_w*randn(n,1);
  w_particle = w_particle.*exp(-(ym(i+1)-x_particle.^2/20).^2/(2*sig_y^2));
  w_particle = w_particle/sum(w_particle);
   x_est(i+1)=sum(x_particle.*w_particle);

  [x_particle,w_particle]=resample_pf(x_particle,w_particle);

  caxis([0 1]); [range,domain]=hist(x_particle,[-20:1:20]);
  set(gca,'fontsize',12);waterfall((domain),i,range)
end

hold off; axis([-20 20 0 50 0 500]);
ylabel('Time','fontsize',12)
xlabel('Sample Space','fontsize',12)
zlabel('Posterior Density','fontsize',12)
```

Additive Gaussian & Linear

- Truth model

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) + \overbrace{\Upsilon_k \mathbf{w}_k}^{\text{additive Gaussian}}, \quad \mathbf{w}_k \sim N(\mathbf{0}, Q_k)$$

$$\tilde{\mathbf{y}}_k = H_k \mathbf{x}_k + \underbrace{\mathbf{v}_k}_{\text{linear}}, \quad \mathbf{v}_k \sim N(\mathbf{0}, R_k)$$

- We can show that

$$\boxed{\begin{aligned} p(\mathbf{x}_{k+1} | \mathbf{x}_k, \tilde{\mathbf{y}}_{k+1}) &= N(\mathbf{a}_{k+1}, \Sigma_{k+1}) \\ p(\tilde{\mathbf{y}}_{k+1} | \mathbf{x}_k) &= N(\mathbf{b}_{k+1}, S_{k+1}) \end{aligned}}$$

where

$$\mathbf{a}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) + \Sigma_{k+1} H_{k+1}^T R_{k+1}^{-1} (\tilde{\mathbf{y}}_{k+1} - \mathbf{b}_{k+1})$$

$$\Sigma_{k+1} = \Upsilon_k (Q_k - Q_k \Upsilon_k^T H_{k+1}^T S_{k+1}^{-1} H_{k+1} \Upsilon_k Q_k) \Upsilon_k^T$$

$$S_{k+1} = H_{k+1} \Upsilon_k Q_k \Upsilon_k^T H_{k+1}^T + R_{k+1}$$

$$\mathbf{b}_{k+1} = H_{k+1} \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k)$$

Example (i)

- The system is described by

$$x_{k+1} = \frac{x_k^2}{1 + x_k^3} + w_k, \quad w_k \sim N(0, q)$$

$$\tilde{y}_k = x_k + v_k, \quad w_k \sim N(0, r)$$

  - The truth is generated using an initial condition sampled from $p(x_0) \sim N(0,10)$ and $q = 0.1$
  - A set of 51 time observations is obtained, and synthetic measurements are obtained using $r = 1$
  - The number of particles for the PF is 500, which are sampled from $p(x_0) \sim N(0,10)$, same as the generated initial condition
  - Since $H_k = 1$ and $\Upsilon_k = 1$ for this system, then

$$\Sigma = q(1 - q/s)$$

$$S_k \equiv s = q + r$$

Example (ii)

- Also

$$\mathbf{b}_{k+1}^{(j)} = f_k(x_k^{(j)}) = \frac{(x_k^{(j)})^2}{1+(x_k^{(j)})^2}$$

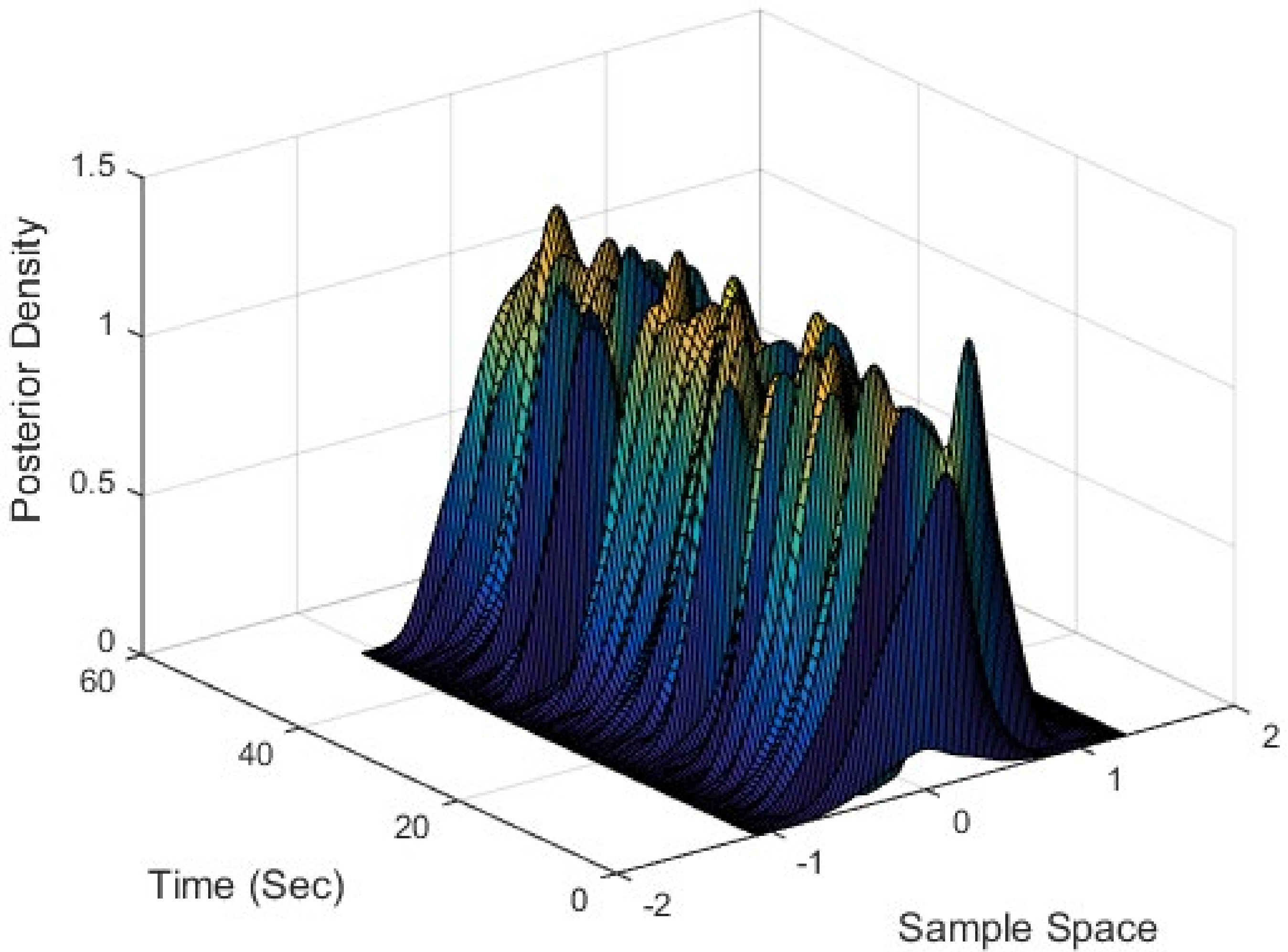
$$\mathbf{a}_{k+1}^{(j)} = f_k(x_k^{(j)}) + \frac{\Sigma}{r}(\tilde{y}_{k+1} - f_k(x_k^{(j)}))$$

- Then $p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_k^{(j)})$ reduces down to

$$p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_k^{(j)}) = \frac{1}{\sqrt{2\pi\, s}} \exp\left(-\frac{(\tilde{y}_{k+1} - f_k(x_k^{(j)}))^2}{2s}\right)$$

- Note that the term $\sqrt{2\pi\, s}$ is not required because it is constant and cancels out when the weights are normalized, but is shown here for completeness
- We plot the posterior pdfs as they evolve over time
  - Shows that the posterior pdf is well approximated by a Gaussian function, even though the state function is highly nonlinear

Example (iii)



Posterior Density
1.5
1
0.5
0
60
40
20
0
Time (Sec)
-2
-1
0
1
2
Sample Space

Code (i)

```
% Time
t=[0:1:50]';m=length(t);

% Standard Deviations
sig_x=sqrt(10);sig_y=1;sig_w=sqrt(0.1);

% States and Measurements
x=zeros(m,1);x(1)=sig_x*randn(1);
ym=zeros(m,1);ym(1)=x(1)+sig_y*randn(1);

% Generate Truth
for i = 1:m-1
  x(i+1) = x(i)^2/(1+x(i)^3) + sig_w*randn(1);
  ym(i+1)= x(i+1)+ sig_y*randn(1);
end

% Particles
n=500;x_particle_opt=sig_x*randn(n,1);x_particle_bf=x_particle_opt;
```

Code (ii)

```
% Optimal Filter
x_est_opt=zeros(m,1);
x_est_opt(1)=mean(x_particle_opt);
w_particle_opt=inv(n)*ones(n,1);

% Sigma and S Matrices are Constant
s=sig_w^2+sig_y^2;
sigma=sig_w^2-sig_w^4*inv(s);

% Density
fff=zeros(m,100);
xi=zeros(m,100);
[ff,xii]=ksdensity(x_particle_opt);
fff(1,:)=ff;xi(1,:)=xii';
```

Code (iii)

```
% Main Loop
for i = 1:m-1
  f=x_particle_opt.^2./(1+x_particle_opt.^3);
  a=f+sigma*1/sig_y^2*(ym(i+1)-f);
  x_particle_opt=a+sqrt(sigma)*randn(n,1);
  w_particle_opt=w_particle_opt.*exp(-(ym(i+1)-f).^2/(2*s));
  w_particle_opt=w_particle_opt/sum(w_particle_opt);
  x_est_opt(i+1)=sum(x_particle_opt.*w_particle_opt);

% Density
 [ff,xii]=ksdensity(x_particle_opt);
 fff(i+1,:)=ff;xi(i+1,:)=xii';

end
```

Code (iv)

```
% Plot Results
surf(xii,t,fff)
ylabel('Time (Sec)','fontsize',12)
xlabel('Sample Space','fontsize',12)
zlabel('Posterior Density','fontsize',12)

pause
```

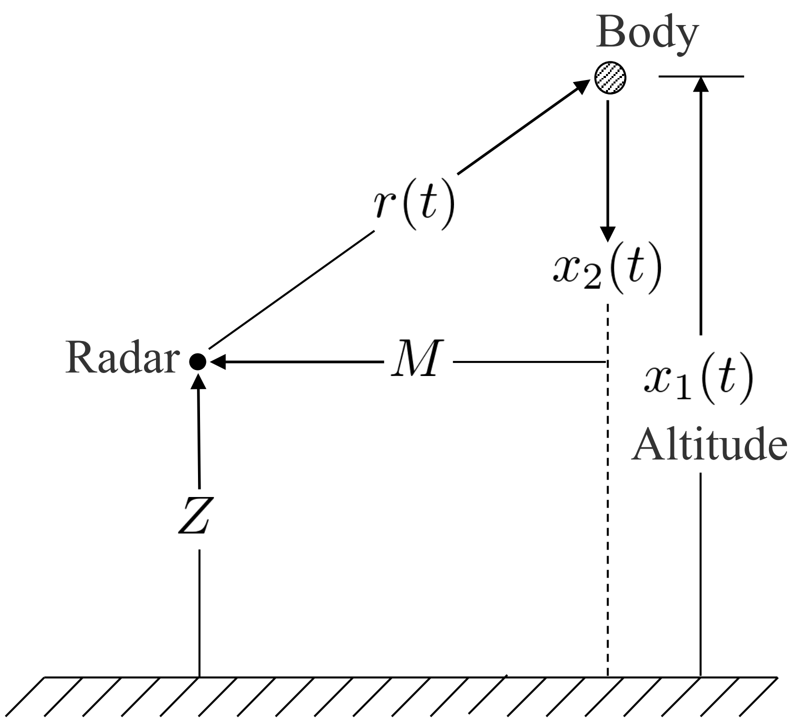
Code (v)

```
% Bootstrap Filter (to compare to the optimal one)
x_est_bf=zeros(m,1);
x_est_bf(1)=mean(x_particle_bf);
w_particle_bf=inv(n)*ones(n,1);

% Density
fff=zeros(m,100);
xi=zeros(m,100);
[ff,xii]=ksdensity(x_particle_bf);
fff(1,:)=ff;xi(1,:)=xii';
```

Code (vi)

```
% Main Loop
for i = 1:m-1
  x_particle_bf=x_particle_bf.^2./(1+x_particle_bf.^3)+sig_w*randn(n,1);
  w_particle_bf=w_particle_bf.*exp(-(ym(i+1)-x_particle_bf).^2/(2*sig_y^2));
  w_particle_bf=w_particle_bf/sum(w_particle_bf);
  x_est_bf(i+1)=sum(x_particle_bf.*w_particle_bf);

 % Resample
 [x_particle_bf,w_particle_bf]=resample_pf(x_particle_bf,w_particle_bf);

 [ff,xii]=ksdensity(x_particle_bf);
 fff(i+1,:)=ff;xi(i+1,:)=xii';

end

% Plot Results
surf(xii,t,fff)
ylabel('Time (Sec)','fontsize',12)
xlabel('Sample Space','fontsize',12)
zlabel('Posterior Density','fontsize',12)
```

Free Falling Body Example



$$\dot{x}_1(t) = -x_2(t)$$

$$\dot{x}_2(t) = -e^{-\alpha\, x_1(t)} x_2^2(t)\, x_3(t)$$

$$\dot{x}_3(t) = 0$$

$$\tilde{y}_k = \sqrt{M^2 + (x_{1_k} - Z)^2} + v_k$$

- Estimate ballistic coefficient, x_3, of a vertical falling body
- Measure range by a radar
- M and Z are constants
- α relates air density with altitude (constant)
- Process noise added here
  - PF has issues without it
- UF and PF comparisons
  - Same initial covariances
  - Monte Carlo simulation with 100 runs

Truth (i)

```
% Parameters and Time
gam=5e-5;dt=1;t=[0:dt:60]';m=length(t);x=zeros(m,3);x(1,:)=[3e5 2e4 1e-3];

% Covariances
r=1e4;q2=1e-12;q1=0.0001;
q=[q1*dt^3/3 q1*dt^2/2 0;q1*dt^2/2 q1*dt 0;0 0 q2*dt];
[v_q,e_q]=eig(q);

% True State
for i=1:m-1
 w_uncorr=kron(diag(e_q)'.^(0.5),ones(1,1)).*randn(1,3);noise=(v_q*w_uncorr');
 f1=dt*athansfun(x(i,:)',gam,noise);
 f2=dt*athansfun(x(i,:)'+0.5*f1,gam,noise);
 f3=dt*athansfun(x(i,:)'+0.5*f2,gam,noise);
 f4=dt*athansfun(x(i,:)'+f3,gam,noise);
 x(i+1,:)=x(i,:)+1/6*(f1'+2*f2'+2*f3'+f4');
end

% Measurements
y=(1e5^2+(x(:,1)-1e5).^2).^(0.5);ym=y+sqrt(r)*randn(m,1);
```

Truth (ii)

```
function f=athansfun(x,gam,noise)

% Function Routine for Athans Problem
[m,n]=size(x);
f=zeros(m,n);
f(1,:)=-x(2,:)+noise(1,:);
f(2,:)=-exp(-gam*x(1,:)).*(x(2,:).^2).*x(3,:)+noise(2,:);
f(3,:)=zeros(1,n)+noise(3,:);
```

Note: Process noise is added to the first state, which is a kinematic state. This addition is due to the conversion of the continuous-time process noise covariance to discrete-time (this can be neglected if the sampling interval is “small”)

Free Falling Example UF (i)

```
% Initial Conditions
x_est_uf=zeros(m,3);
x_est_uf(1,:)=[3e5 2e4 3e-5];
pcov=diag([1e6 4e6 1e-4]);p_uf=zeros(m,3);p(1,:)=diag(pcov)';

% Unscented Filter Parameters
alp=1;beta=2;kap=0;bigl=6;
lam=alp^2*(bigl+kap)-bigl;
w0m=lam/(bigl+lam);
w0c=lam/(bigl+lam)+(1-alp^2+beta);
wim=1/(2*(bigl+lam));
yez=zeros(1,6);

for i=1:m-1
% Covariance Decomposition
 psquare=chol([pcov zeros(3);zeros(3) q])';
 sigv=real([sqrt(bigl+lam)*psquare -sqrt(bigl+lam)*psquare]);
 xx0=x_est_uf(i,:)';
 xx=sigv(1:3,:)+kron(x_est_uf(i,:)',ones(1,2*bigl));
 noise_w=[zeros(3,1) sigv(4:6,:)];
```

Free Falling Example UF (ii)

```
% Calculate Mean Through Propagation
 f1=dt*athansfun([xx0 xx],gam,noise_w);
 f2=dt*athansfun([xx0 xx]+0.5*f1,gam,noise_w);
 f3=dt*athansfun([xx0 xx]+0.5*f2,gam,noise_w);
 f4=dt*athansfun([xx0 xx]+f3,gam,noise_w);
 xx0=xx0+1/6*(f1(:,1)+2*f2(:,1)+2*f3(:,1)+f4(:,1));
 xx=xx+1/6*(f1(:,2:2*bigl+1)+2*f2(:,2:2*bigl+1)+2*f3(:,2:2*bigl+1)+f4(:,2:2*bigl+1));

 x_est_uf(i+1,:)=w0m*xx0'+wim*sum(xx,2)';

% Covariance
 pp0=w0c*(xx0-x_est_uf(i+1,:)')*(xx0-x_est_uf(i+1,:)')';
 pmat=xx-kron(x_est_uf(i+1,:)',ones(1,2*bigl));
 pcov=pp0+wim*pmat*pmat';
```

Free Falling Example UF (iii)

```
% Output
 for j = 1:2*bigl
  yez(j)=sqrt(1e5^2+(xx(1,j)-1e5)^2);
 end
  ye0=sqrt(1e5^2+(xx0(1)-1e5)^2);

 ye=w0m*ye0+wim*sum(yez,2);

% Calculate pyy
 pyy0=w0c*(ye0-ye)*(ye0-ye)';
 pyymat=yez-ye0;
 pyy=pyy0+wim*pyymat*pyymat';

% Calculate pxy
 pxy0=w0c*(xx0-x_est_uf(i+1,:)')*(ye0-ye);
 pxy=pxy0+wim*pmat*pyymat';

% Innovations Covarinace
 pvv=pyy+r;
```

Free Falling Example UF (iv)

```
% Gain and Update
 gain=real(pxy*inv(pvv));
 pcov=pcov-gain*pvv*gain';
 pcov=0.5*(pcov+pcov');
 p_uf(i+1,:)=diag(pcov)';
 x_est_uf(i+1,:)=x_est_uf(i+1,:)+(gain*(ym(i+1)-ye))';

end
```

Note: The covariance sometimes loses symmetry. To overcome this problem, the line pcov=0.5*(pcov+pcov') was added (simple but effective).

Free Falling Example PF (i)

```
% Initial Condition
x_est_pf=zeros(m,3);
x_est_pf(1,:)=[3e5 2e4 3e-5];
p_pf=zeros(m,3);

% Particles
n=500;nx=3;
h_opt=(4/(nx+2))^(1/(nx+4))*n^(-1/(nx+4));
x_particle=[1000*randn(n,1)+x_est_pf(1,1)
2000*randn(n,1)+x_est_pf(1,2) 2e-3*rand(n,1)];
w_particle=inv(n)*ones(n,1);

% Initial Covariance
x_diff=x_particle-kron(ones(n,1),x_est_pf(1,:));
p_part=x_diff'*(x_diff.*kron(w_particle,[1 1 1]));
p_pf(1,:)=diag(p_part)';

% Threshold for Resampling
epsilon=250;
```

Free Falling Example PF (ii)

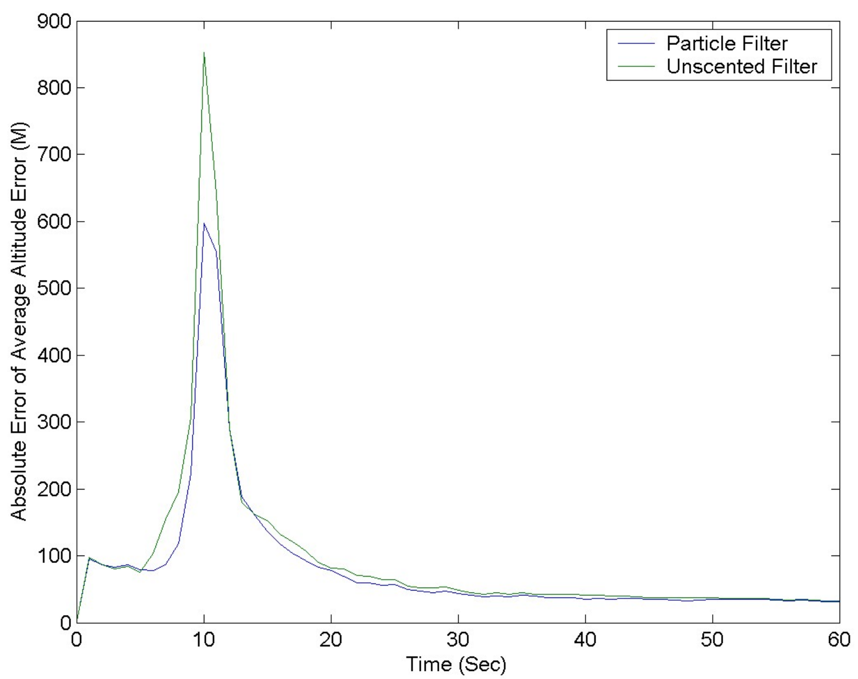
```
% Particle Filtering
for i=1:m-1

% Get Correlated Noise and Integrate Particles
 w_uncorr=kron(diag(e_q)'.^(0.5),ones(n,1)).*randn(n,3);
 noise=(v_q*w_uncorr');
 f1=dt*athansfun(x_particle',gam,noise);
 f2=dt*athansfun(x_particle'+0.5*f1,gam,noise);
 f3=dt*athansfun(x_particle'+0.5*f2,gam,noise);
 f4=dt*athansfun(x_particle'+f3,gam,noise);
 x_particle=x_particle+1/6*(f1'+2*f2'+2*f3'+f4');

% Output and Weights
 ye=(1e5^2+(x_particle(:,1)-1e5).^2).^(0.5);
 w_particle = w_particle.*exp(-(ym(i+1)-ye).^2/(2*r));
 w_particle = w_particle/sum(w_particle);
 x_est_pf(i+1,:)=sum(x_particle.*kron(w_particle,[1 1 1]));
```

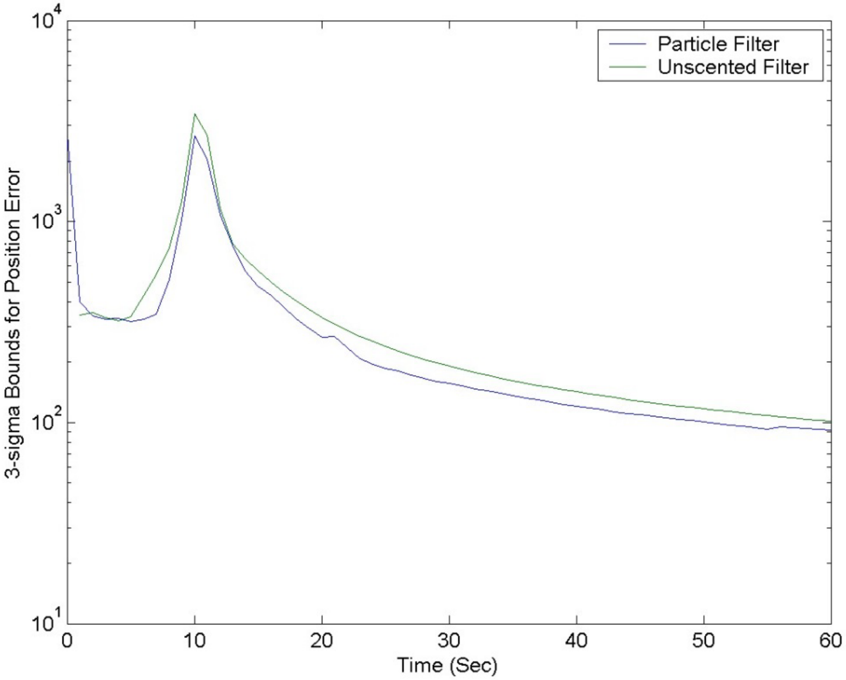
Free Falling Example PF (iii)

```
% Covariance
x_diff=x_particle-kron(ones(n,1),x_est_pf(i+1,:));
p_part=x_diff'*(x_diff.*kron(w_particle,[1 1 1]));
p_pf(i+1,:)=diag(p_part)';

% Resampling and Regularization
 n_eff=1/sum(w_particle.^2);
 if n_eff < epsilon
  [x_particle,w_particle]=resample_pf(x_particle,w_particle);
  x_particle=x_particle+h_opt*(chol(p_part)'*randn(3,n))';
 end

end
```

Monte Carlo Results (i)

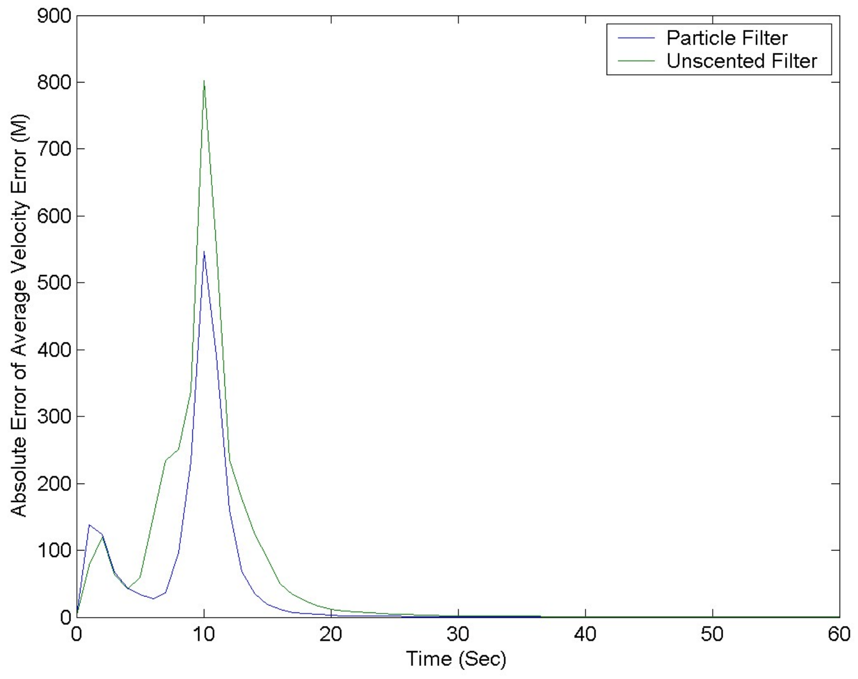
Average Error Results Over 100 Monte Carlo Runs

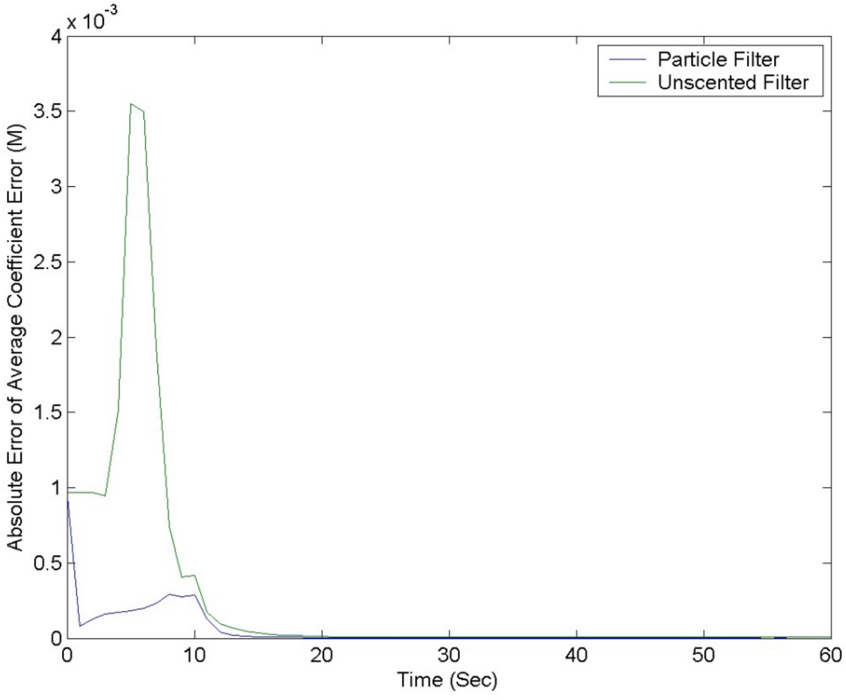




Monte Carlo Results (ii)

Average Error Results Over 100 Monte Carlo Runs





Auxiliary SIR

- Auxiliary SIR filter
  - Importance density $q(\mathbf{x}_{k+1}, i|\tilde{\mathbf{Y}}_{k+1}) \propto p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)})\, p(\tilde{\mathbf{y}}_{k+1}|\boldsymbol{\mu}_{k+1}^{(i)})$
    - i is an introduced auxiliary index
    - $\boldsymbol{\mu}_{k+1}^{(i)}$ is considered a representative point at t_{k+1} evolved from $\mathbf{x}_k^{(i)}$
  - Then, using $i^{(j)}$ draw $\mathbf{x}_{k+1}^{(i^{(j)})}$ from $p(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i^{(j)})})$
  - Corresponding weights

$$\varpi_{k+1}^{i^{(j)}} = \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i^{(j)})})}{p(\tilde{\mathbf{y}}_{k+1}|\boldsymbol{\mu}_{k+1}^{(i^{(j)})})}$$

- Good news and bad news
  - For small process noise $p(\mathbf{x}_{k+1}|\mathbf{x}_k)$ is well characterized by $\boldsymbol{\mu}_{k+1}^{(i)}$
    - Filter is less sensitive to outliers than SIR filter and weights are more even
  - For large process noise $\mathbf{x}_k^{(i)}$ will be mapped through the dynamic model to a rather large region, which we have no reason to believe can be effectively represented by a single point

Local Linearized PF

- Approximates optimal importance density by incorporating current measurement with an EKF or an UF
  - Use a separate EKF or UF to generate and propagate a Gaussian importance distribution

$$q(\mathbf{x}_{k+1}|\mathbf{x}_k^{(i)}, \tilde{\mathbf{y}}_{k+1}) = N(\hat{\mathbf{x}}_{k+1}^{(i)}, \hat{P}_{k+1}^{(i)})$$

  - Gives a Local Linearized Particle Filter (LLPF)
  - UF has been shown to provide better results than an EKF in many cases

LLPF Pseudocode

$[\{\mathbf{x}_{k+1}^{(j)},\, P_{k+1}^{(j)}\}_{j=1}^{N}] = \text{LLPF}[\{\mathbf{x}_k^{(i)},\, P_k^{(i)}\}_{i=1}^{N},\, \tilde{\mathbf{y}}_{k+1}]$

- FOR $i = 1 : N$
  - Run EKF or UF: $[\hat{\mathbf{x}}_{k+1}^{(i)}, \hat{P}_{k+1}^{(i)}] = \text{EKF/UF}[\mathbf{x}_k^{(i)}, P_k^{(i)}, \tilde{\mathbf{y}}_{k+1}]$
  - Draw $\mathbf{x}_{k+1}^{(i)} \sim N(\hat{\mathbf{x}}_{k+1}^{(i)}, \hat{P}_{k+1}^{(i)})$
  - Evaluate $\tilde{\varpi}_{k+1}^{(i)} = \dfrac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i)})\, p(\mathbf{x}_{k+1}^{(i)}|\mathbf{x}_k^{(i)})}{N(\hat{\mathbf{x}}_{k+1}^{(i)}, \hat{P}_{k+1}^{(i)})}$

Multiplying by $\varpi_k^{(i)}$ not necessary since we resample each time

- END FOR
- Compute normalized weights $\varpi_{k+1}^{(i)}$
- Resample $[\{\mathbf{x}_{k+1}^{(j)},\, -,\, i^{(j)}\}_{i=1}^{N}] = \text{RESAMPLE}[\{\mathbf{x}_{k+1}^{(i)},\, \varpi_{k+1}^{(i)}\}_{i=1}^{N}]$
- FOR $j = 1 : N$
  - Assign covariance $P_{k+1}^{(j)} = \hat{P}_{k+1}^{(i^{(j)})}$
- END FOR

Curse of Dimensionality (i)

- The famous “Curse of Dimensionality”
  - First coined by Richard Bellman for nonlinear filtering applications
    - Describes the exponential growth of computational complexity as a function of the dimension of the state vector
    - Kalman filter computational complexity grows as the cube of the dimension
    - General nonlinear filters grow exponentially
  - It has been stated that PFs do not suffer from this curse
    - Daum and Huang argue that this is not true though

Daum, F., and Haung, J., “Curse of Dimensionality and Particle Filters,” *Proceedings of 2003 IEEE Aerospace Conference*, Big Sky, MT, Vol. 4, March, 2003, pp. 1979-1993.

- Variance of error estimating mean (y) of x using N statistically independent samples of x is given by

$$\mathrm{var}(y) = \mathrm{var}(x)/N$$

Curse of Dimensionality (ii)

- Suppose conditional density of $\mathbf{x}$ is vaguely Gaussian
  - Not necessarily Gaussian, but close to it
  - Bulk of the volume of the state space with significant probability is bounded by elliptical curves
    - Can convert this to spherical contours using principal axes through eigenvectors of covariance (same idea as inertia matrix)
    - For an n-dimensional sphere we assume that the significant volume is bound by $k\sigma$ contours
    - Popular choice is $k = 6$, however it can be shown that k is a weakly increasing function of n
  - Daum and Huang have shown that for large n

$$\boxed{N \approx c\,n/\mathrm{var}(y)k^2}$$

where c is a constant independent of n
    - Linear function of n
    - Therefore, the number of particles is important even for quasi-Gaussian problems

Rao-Blackwellization

- For poorly chosen densities
  - Variance of error is exponential in n, which leads to the “Curse of Dimensionality”
- Rao-Blackwellization
  - Can reduce the number of particles for some problems
  - Partition state vector so a part of it is conditionally linear and Gaussian
    - Use the standard Kalman filter for the linear-Gaussian part
    - Use the Particle filter for the other part

      Gyros Accels bias scale factors are linear
  - Nice application for inertial navigation (27 states in this paper)

Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., and Nordlund, P.-J., “Particle Filters for Positioning, Navigation, and Tracking,” *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, Feb. 2002, pp. 425-437.

Daum's Comparison

Item	EKF	UKF	PF
Statistics propagated by algorithm	Mean vector and covariance matrix	Mean vector and covariance matrix	Complete pdf conditioned on the measurements
Prediction of statistics from one measurement time to the next	Linear approximation of dynamics	Approximation of the multidimensional integrals using the "unscented transformation"	Monte Carlo integration using importance sampling
Correction of statistics at measurement time	Linear approximation of the measurement equations	Approximation of the multidimensional integrals using the "unscented transformation"	Monte Carlo sampling of the conditional density using both importance sampling and resampling
Accuracy of state vector estimate	Often ~~Sometimes~~ good but often poor compared with theoretically optimal accuracy	Often provides significant improvement relative to the EKF, but sometimes it does not	Optimal performance for low dimensional problems, but can be highly suboptimal for higher dimensions
Computational complexity of real-time algorithm	On the order of n^3 for estimating state vectors of dimension n	Roughly the same as the EKF	Beats the curse of dimensionality for "nice" problems with a carefully designed PF, but not otherwise

Daum, F., "Nonlinear Filters: Beyond the Kalman Filter," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 20, No. 8, Part 2, Aug. 2005, pp. 57-69.

Some References

Ristic, B., Sanjeev Arulampalam, S., and Gordon, N., *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House, Boston, MA, 2004.

Doucet, A., de Freitas, J.F.G., and Gordon, N.J., Editors, *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, New York, NY, 2001.

Arulampalam, M.S., Maskell, S., Gordon, N., and Clapp, T., "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, Feb. 2002, pp. 174-188.

Liu, J.S., *Monte Carlo Strategies in Scientific Computing*, Springer-Verlag, Berlin, 2001.

Doucet, A., Godsill, S.J., and Andrieu, C., "On Sequential Monte Carlo Sampling Methods for Bayesian Filtering", *Statistics in Computing*, Vol. 10, No. 3, 2000, pp. 197-208.

http://en.wikipedia.org/wiki/Particle_filter

http://www-sigproc.eng.cam.ac.uk/smc/

http://www.cs.ubc.ca/~nando/software.html

Ripley, B., *Stochastic Simulation*, Wiley, New York, NY, 1987.