Optimal Estimation Methods

(Lecture 18 – Particle Filtering: Part I)

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

Particle Filtering
(Sequential Monte Carlo Estimation)

“I think that a particle must have a separate reality independent of the measurements. That is an electron has spin, location and so forth even when it is not being measured. I like to think that the moon is there even if I am not looking at it.”
Albert Einstein

Einstein may be right in the physics world, but for our particles the measurements are extremely useful (used to compute weights)

Particle Filtering – Does distribution estimation rather than state estimation

- PFs perform sequential Monte Carlo (SMC) estimation
  - Dates back to 1950's!

  Hammersley, J.M., and Morton, K.W., "Poor Man's Monte Carlo," *Journal of the Royal Statistical Society*, Vol. 16, 1954, pp. 23-38.

  - Most applications used plain sequential importance sampling, which degenerates over time
  - We now have the tools and computer power for PFs
- Posterior distributions can't be analytical obtained
  - PFs approximate the continuous posterior distribution using a set of **weighted** particles
  - Each particle corresponds to a possible value of the state
  - Particles constitute random support of the approximating discrete distribution
  - PFs do not provide measures of uncertainty, such as mean and covariance ← May have non-gaussian distribution, so could be useless

Monte Carlo Integration (i)

- Simple example: consider $\tilde{\mathbf{y}} = \mathbf{h}(\mathbf{x}) + \mathbf{v}$, and let's say we are given the prior density $p(\mathbf{x})$ and $\mathbf{v} \sim N(\mathbf{0}, R)$
  - $p(\tilde{\mathbf{y}})$ is not Gaussian but $p(\tilde{\mathbf{y}}|\mathbf{x}) \sim N(\mathbf{h}(\mathbf{x}),\, R)$
  - We wish to determine the integral for various reasons

$$G = \int \mathbf{f}(\mathbf{x})\, p(\mathbf{x}|\tilde{\mathbf{y}})\; d\mathbf{x}$$

  - For example, choosing $\mathbf{f}(\mathbf{x}) = \mathbf{x}$ gives the estimate of $\mathbf{x}$
- Perfect Monte Carlo integration
  - Draw $N \gg 1$ samples $\{\mathbf{x}^{(i)};\, i = 1, \ldots, N\}$ from $p(\mathbf{x}|\tilde{\mathbf{y}})$
  - Estimate of G is given by

$$\hat{G} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{f}(\mathbf{x}^{(i)})$$

  - Note: This is not a function of $p(\mathbf{x}|\tilde{\mathbf{y}})$, but we must be able to draw from $p(\mathbf{x}|\tilde{\mathbf{y}})$

Monte Carlo Integration (ii)

- If Central Limit Theorem holds (key to choose N!)
  - Error of estimate on the order of $N^{-1/2}$
    - Rate of estimate convergence independent of the dimension of the integrand (samples come from regions important for the integration result)
    - Usual “numerical integration” has rate of convergence decreasing as the dimension increases
  - Bayesian estimation, $p(\mathbf{x}|\tilde{\mathbf{y}})$ is the posterior density
    - Can’t draw from this in practice since it’s too complicated!!!
- Generate samples from $q(\mathbf{x})$, *Importance Density*
  - Importance density is similar to $p(\mathbf{x}|\tilde{\mathbf{y}})$
  - If $p(\mathbf{x}|\tilde{\mathbf{y}})/q(\mathbf{x})$ is upper bounded, then

$$G = \int \mathbf{f}(\mathbf{x}) p(\mathbf{x}|\tilde{\mathbf{y}})\, d\mathbf{x} = \int \mathbf{f}(\mathbf{x}) \frac{p(\mathbf{x}|\tilde{\mathbf{y}})}{q(\mathbf{x})} q(\mathbf{x})\, d\mathbf{x}$$

  - Draw $N \gg 1$ samples $\{\mathbf{x}^{(i)};\, i = 1, \ldots, N\}$ from $q(\mathbf{x})$

Importance Sampling (i)

- Bayes' rule

known!

$$p(\mathbf{x}|\tilde{\mathbf{y}}) = \frac{p(\tilde{\mathbf{y}}|\mathbf{x})\,p(\mathbf{x})}{p(\tilde{\mathbf{y}})} = \frac{p(\tilde{\mathbf{y}}|\mathbf{x})\,p(\mathbf{x})}{\int p(\tilde{\mathbf{y}}|\mathbf{x})p(\mathbf{x})\,d\mathbf{x}} \equiv \frac{p(\tilde{\mathbf{y}}|\mathbf{x})\,p(\mathbf{x})}{\text{normalizing constant}}$$

  - So, we can *evaluate* $p(\mathbf{x}|\tilde{\mathbf{y}})$ to within a constant!
  - Substituting into G gives

$$G = \int \mathbf{f}(\mathbf{x})p(\mathbf{x}|\tilde{\mathbf{y}})\ d\mathbf{x} = \frac{\displaystyle\int \mathbf{f}(\mathbf{x})\frac{p(\tilde{\mathbf{y}}|\mathbf{x})p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\ d\mathbf{x}}{\displaystyle\int \frac{p(\tilde{\mathbf{y}}|\mathbf{x})p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\ d\mathbf{x}}$$

- Importance sampling uses weighted samples
  - All quantities in the integral can now be calculated
  - Weight the importance of the particles
  - We'll use the same approximation approach as was done for the perfect Monte Carlo integration approach to find an estimate of G

Importance Sampling (ii)

- Estimate given by

$$\hat{G} = \frac{N^{-1}\sum_{i=1}^{N} \mathbf{f}(\mathbf{x}^{(i)})\tilde{\varpi}^{(i)}}{N^{-1}\sum_{i=1}^{N} \tilde{\varpi}^{(i)}}, \qquad \tilde{\varpi}^{(i)} \equiv \frac{p(\tilde{\mathbf{y}}|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$$

or

$$\boxed{\hat{G} = \sum_{i=1}^{N} \mathbf{f}(\mathbf{x}^{(i)})\varpi^{(i)}, \qquad \varpi^{(i)} = \frac{\tilde{\varpi}^{(i)}}{\sum_{j=1}^{N} \tilde{\varpi}^{(i)}}}$$

  - Note: $\sum_{i=1}^{N} \varpi^{(i)} = 1$ (probablites must equal 1)
  - Also

$$p(\tilde{\mathbf{y}}) = \int p(\tilde{\mathbf{y}}|\mathbf{x})\, p(\mathbf{x})\; d\mathbf{x} = \int \frac{p(\tilde{\mathbf{y}}|\mathbf{x})\, p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x})\; d\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N} \tilde{\varpi}^{(i)}$$

- Choice of $q(\mathbf{x})$
  - Easiest choice: $q(\mathbf{x}) = p(\mathbf{x})$, which gives (more later on this)

$$\boxed{\tilde{\varpi}^{(i)} = p(\tilde{\mathbf{y}}|\mathbf{x}^{(i)})}$$

Simple “Sequential” Example

- Consider $\tilde{y} = e^{-x\,t} + v$
  - High noise case $\Rightarrow \sigma = 0.01$
  - Low noise case $\Rightarrow \sigma = 0.001$
  - Generate 101 measurements with $\Delta t = 0.1$ seconds
  - Choose 2,000 particles $\Rightarrow N = 2000$
  - True value given by $x = 2$
  - Assume that $q(x) = p(x)$ is a uniform distribution from 0 to 3
    - Draw 2,000 samples from this distribution
  - Weight updated “sequentially” with each new measurement
    - Start with $\varpi_0^{(i)} = 1/N = 0.0005$

Gaussian

Particle estimator

$$\varpi_{k+1}^{(i)} = \varpi_k^{(i)} \exp\left[-\frac{(\tilde{y}_k - e^{-x^{(i)} t_k})^2}{2\sigma^2}\right]$$

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

$$\boxed{\hat{x}_k = \sum_{i=1}^{N} x^{(i)} \varpi_k^{(i)}}$$

Code (i)

```
clear
% Truth and Measurements
% High Noise
r=0.01^2;ylimit_movie=[0 0.01];
% Low Noise
%r=0.001^2;ylimit_movie=[0 0.1];
t=[0:0.1:10]';m=length(t);
x_true=2;
y=exp(-x_true*t);
ym=y+sqrt(r)*randn(m,1);

% Particles and Weights
m_part=2000;x_est=zeros(m,1);
x_particle=3*rand(m_part,1);[x_particle_sort,ix]=sort(x_particle);
w=ones(m_part,1)/m_part;
```

Code (ii)

```
% Settings for Movie
clf
clear m_get
% Low Noise
% set(gca,'xlim',[0 3],'ylim',[0 0.3],'NextPlot','replace','Visible','on')
% set(gca,'ytick',[0 0.05 0.1 0.15 0.2 0.25 0.3]);
% High Noise
set(gca,'xlim',[0 3],'ylim',[0 0.04],'NextPlot','replace','Visible','on')
set(gca,'nextplot','replacechildren');
```

Code (iii)

```
% Update Weights and Get Estimate
for i=1:m,
 w_nonnorm=w.*exp(-(ym(i)-exp(-x_particle*t(i))).^2/(2*r));
 w=w_nonnorm/sum(w_nonnorm);
 x_est(i)=sum(x_particle.*w);
 h=stem(x_particle_sort,w(ix));
 set(gcf,'color',[1 1 1]);
 set(gca,'fontsize',16);
 m_get(:,i)=getframe(gcf);
end
movie2gif(m_get,'out.gif','DelayTime',0.1)

% Show Estimate at Final Time
x_xest_final_time=x_est(m)

% Plot Results
plot(t,x_est,'*')
set(gca,'Fontsize',16);
ylabel('Estimate')
xlabel('Time (Sec)')
```

Code (iv)

```
function movie2gif(mov, gifFile, varargin)
% Movie2Gif ver. 1.0
% ===================
% Matlab movie to GIF Converter.
%
% Syntax: movie2gif(mov, gifFile, prop, value, ...)
% ====================================================
% The list of properties is the same like for the command 'imwrite' for the
% file format gif:
%
% 'BackgroundColor' - A scalar integer. This value specifies which index in
%                     the colormap should be treated as the transparent
%                     color for the image and is used for certain disposal
%                     methods in animated GIFs. If X is uint8 or logical,
%                     then indexing starts at 0. If X is double, then
%                     indexing starts at 1.
%
% 'Comment' - A string or cell array of strings containing a comment to be
%             added to the image. For a cell array of strings, a carriage
%             return is added after each row.
```

Code (v)

```
% 'DelayTime' - A scalar value between 0 and 655 inclusive, that specifies
%              the delay in seconds before displaying the next image.
%
% 'DisposalMethod' - One of the following strings, which sets the disposal
%                    method of an animated GIF: 'leaveInPlace',
%                    'restoreBG', 'restorePrevious', or 'doNotSpecify'.
%
% 'LoopCount' - A finite integer between 0 and 65535 or the value Inf (the
%              default) which specifies the number of times to repeat the
%              animation. By default, the animation loops continuously.
%              For a value of 0, the animation will be played once. For a
%              value of 1, the animation will be played twice, etc.
%
% 'TransparentColor' - A scalar integer. This value specifies which index
%                      in the colormap should be treated as the transparent
%                      color for the image. If X is uint8 or logical, then
%                      indexing starts at 0. If X is double, then indexing
%                      starts at 1
% *************************************************************
% Copyright 2007 Nicolae CINDEA
```

Code (vi)

```
if (nargin < 2)
  error('Too few input arguments');
end
if (nargin == 2)
    h = waitbar(0, 'Generate GIF file...');
    frameNb = size(mov, 2);
    isFirst = true;
    for i = 1:frameNb
        waitbar((i-1)/frameNb, h);
        [RGB, colMap] = frame2im(mov(i));
        [IND, map] = aRGB2IND(RGB);
        if isFirst
            imwrite(IND, map, gifFile, 'gif');
            isFirst=false;
        else
            imwrite(IND, map, gifFile, 'gif', 'WriteMode', 'append');
        end
    end
    close(h);
end
```

Code (vii)

```
if (nargin > 2)
    h = waitbar(0, 'Generate GIF file...');
    frameNb = size(mov, 2);
    isFirst = true;
    for i = 1:frameNb
        waitbar((i-1)/frameNb, h);
        [RGB, colMap] = frame2im(mov(i));
        [IND, map] = aRGB2IND(RGB);
        if isFirst
            imwrite(IND, map, gifFile, 'gif', varargin{:});
            isFirst=false;
        else
            imwrite(IND, map, gifFile, 'gif', 'WriteMode', 'append', ...
                varargin{:});
        end
    end
    close(h);
end
```

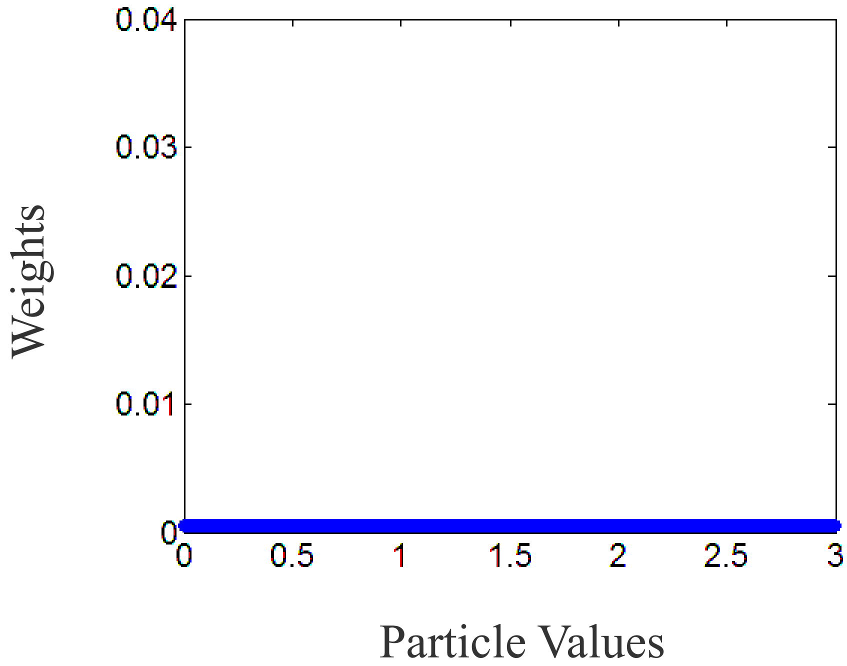
Code (viii)

```
function [X, map] = aRGB2IND(RGB)
% written by Nicolae CINDEA
m = size(RGB, 1);n = size(RGB, 2);X = zeros(m, n);
map(1,:) = RGB(1, 1, :)./255;
for i = 1:m
    for j = 1:n
        RGBij = double(reshape(RGB(i,j,:), 1, 3)./255);
        isNotFound = true;
        k = 0;
        while isNotFound && k < size(map, 1)
            k = k + 1;
            if map(k,:) ==  RGBij
                isNotFound = false;
            end
        end
        if isNotFound, map = [map; RGBij];end
        X(i,j) = double(k);
    end
end
```

Simple Example (i)

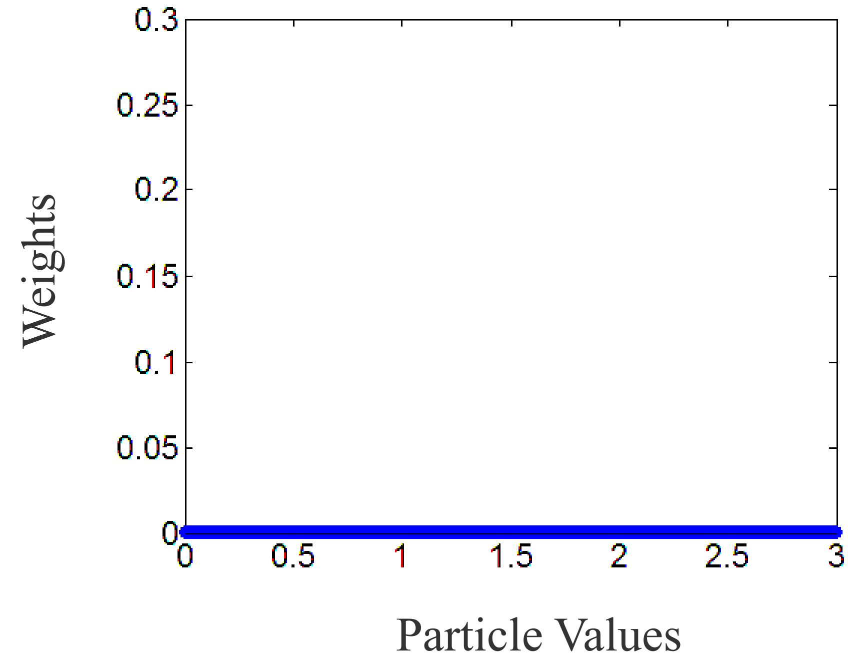
True Value is 2

$p(x^{(i)}|\tilde{y})$ with $\sigma = 0.01$



x_est(m) = 2.0095

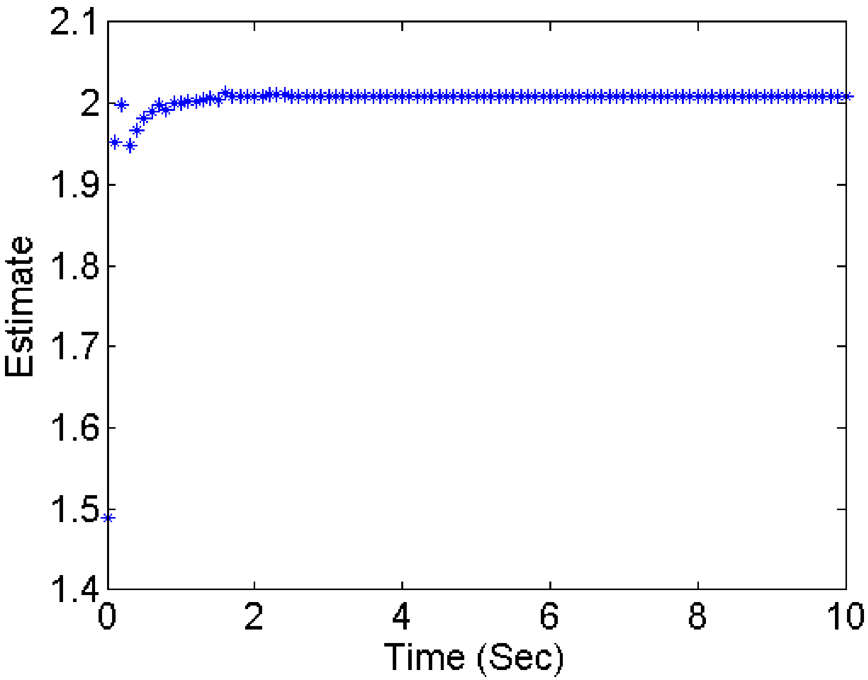
$p(x^{(i)}|\tilde{y})$ with $\sigma = 0.001$



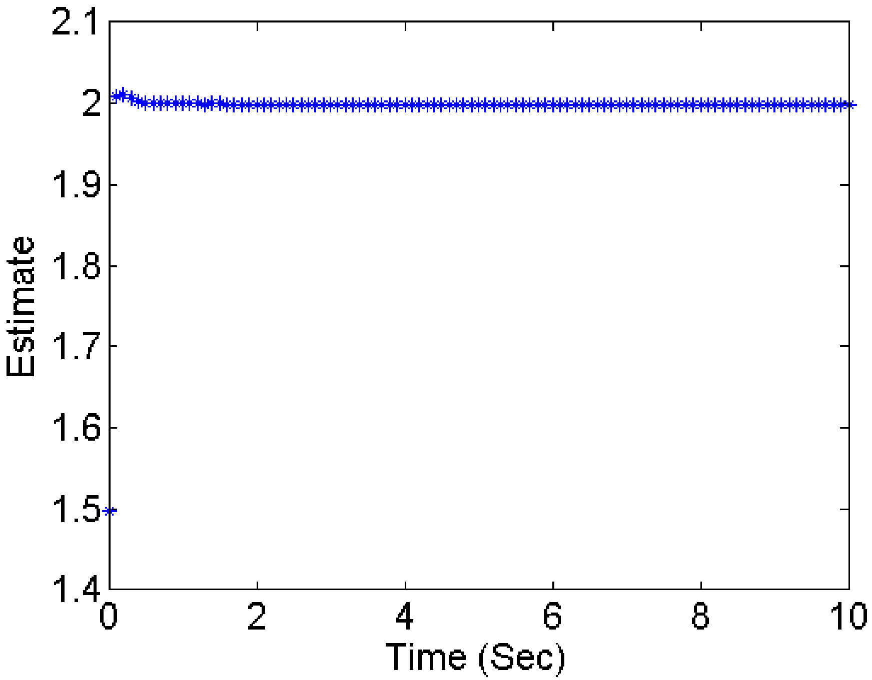
x_est(m) = 1.9987

Simple Example (ii)

$\sigma = 0.01$



$\sigma = 0.001$



Multivariable Example

- Consider $\tilde{y} = e^{-x_1 t} \sin(x_2 t) + v$
  - Measurement noise $\Rightarrow \sigma = 0.01$
  - Generate 101 measurements with $\Delta t = 0.1$ seconds
  - Choose 4,000 × 2 particles $\Rightarrow N = 4000$
  - True value given by $\mathbf{x} = [1 \quad 1.5]^T$
  - Assume that $q(\mathbf{x}) = p(\mathbf{x})$ is a uniform distribution from 0 to 3
    - Draw 4,000 × 2 samples from this distribution
  - Weight updated "sequentially" with each new measurement
    - Start with $\varpi_0^{(i)} = 1/N = 0.00025$

$e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

Gaussian

$$\varpi_{k+1}^{(i)} = \varpi_k^{(i)} \exp\left[-\frac{\left(\tilde{y}_k - \overbrace{e^{-x_1^{(i)} t_k} \sin(x_2^{(i)} t_k)}^{\text{Mean}}\right)^2}{2\sigma^2}\right]$$

$$\boxed{\hat{\mathbf{x}}_k = \sum_{i=1}^{N} \mathbf{x}^{(i)} \varpi_k^{(i)}}$$

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

Code (i)

```
clear
% Truth and Measurements
r=0.01^2;ylimit_movie=[0 0.1];
t=[0:0.1:10]';m=length(t);
x_true=[1;1.5];
y=exp(-x_true(1)*t).*sin(x_true(2)*t);
ym=y+sqrt(r)*randn(m,1);

% Particles and Weights
m_part=4000;x_est=zeros(m,2);
x_particle=3*rand(m_part,2);
[x_particle_sort1,ix1]=sort(x_particle(:,1));
[x_particle_sort2,ix2]=sort(x_particle(:,2));
w=ones(m_part,1)/m_part;

% Settings for Movie
clf
clear m_get
set(gca,'xlim',[0 3],'ylim',[0 0.8],'NextPlot','replace','Visible','on')
set(gca,'nextplot','replacechildren');
```

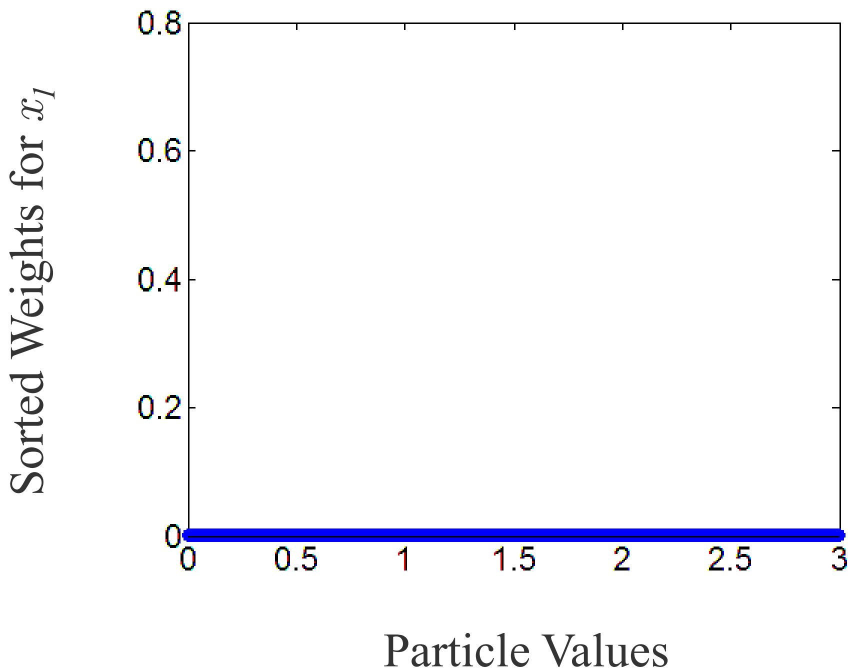
Code (ii)

```
% Update Weights and Get Estimates
for i=1:m,
 w_nonnorm=w.*exp(-(ym(i)-exp(-x_particle(:,1)*t(i)).*sin(x_particle(:,2)*t(i))).^2/(2*r));
 w=w_nonnorm/sum(w_nonnorm);
 x_est(i,:)=sum(x_particle.*[w w]);
 h=stem(x_particle_sort1,w(ix1));
 %h=stem(x_particle_sort2,w(ix2));
 set(gcf,'color',[1 1 1])
 set(gca,'fontsize',16);
 m_get(:,i)=getframe(gcf);
end
movie2gif(m_get,'out.gif','DelayTime',0.1)

% Show Estimate at Final Time
x_xest_final_time=x_est(m,:)
```

Multivariable Example (i)

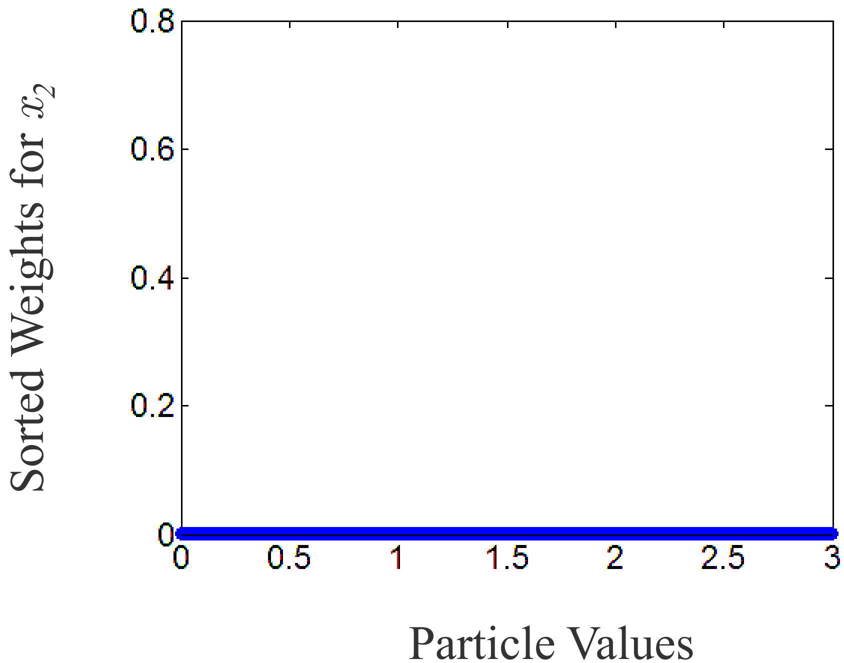
True Values are [1 1.5]

$p(x_1^{(i)}|\tilde{y})$ with $\sigma = 0.01$



x_est(m,1) = 0.9823

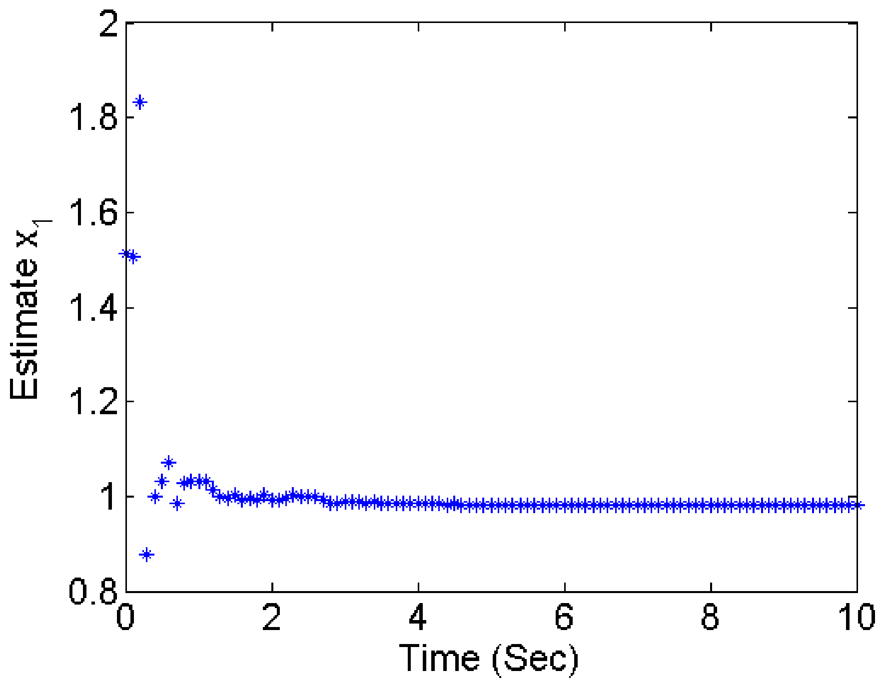
$p(x_2^{(i)}|\tilde{y})$ with $\sigma = 0.01$



x_est(m,1) = 1.5414

Multivariable Example (ii)

Estimates



Covariance Estimate

- A Particle filter does not compute covariance
  - Computed as the sample covariance

$$\boxed{P_k = \sum_{i=1}^{N} \varpi_k^{(i)} \left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right)^T}$$

- Last example

$$P^{\text{mle}} = \begin{bmatrix} 6.2962 \times 10^{-4} & 2.2154 \times 10^{-5} \\ 2.2154 \times 10^{-5} & 1.2705 \times 10^{-3} \end{bmatrix}$$

$$P_{11} = \begin{bmatrix} 5.4594 \times 10^{-4} & 1.9367 \times 10^{-4} \\ 1.9367 \times 10^{-4} & 1.8413 \times 10^{-3} \end{bmatrix}$$

  - What happened? Need more particles $\Rightarrow N = 500{,}000$

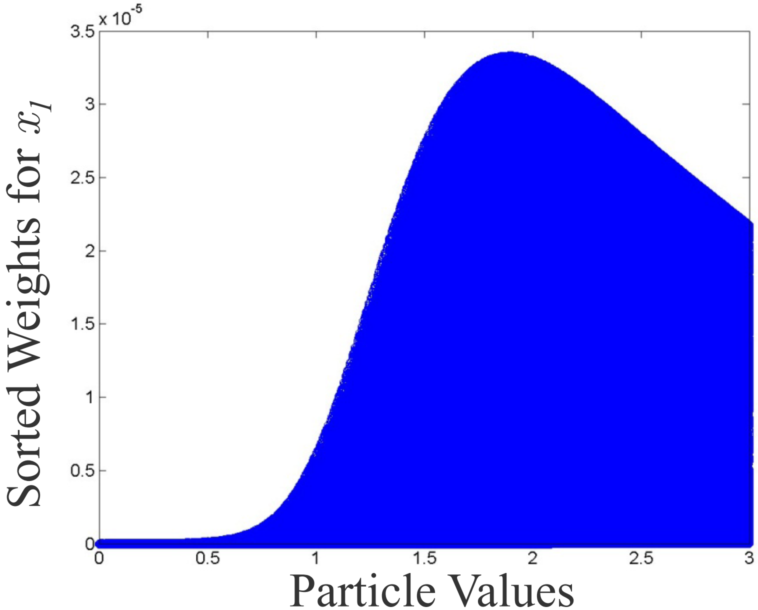
$$P_{11} = \begin{bmatrix} 6.2445 \times 10^{-4} & 3.7379 \times 10^{-5} \\ 3.7379 \times 10^{-5} & 1.2333 \times 10^{-3} \end{bmatrix}$$

  - Actually needed *a lot* more particles
    - Signs of the famous “Curse”

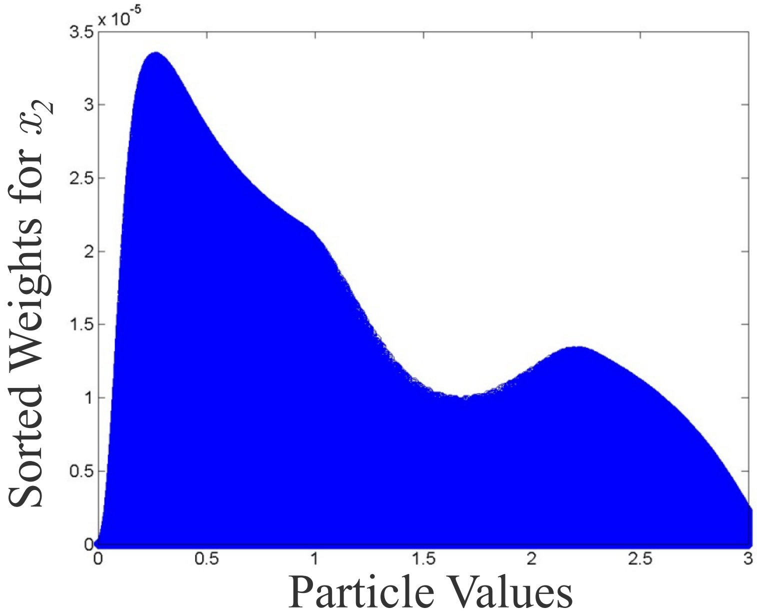
Multivariable Example

- Do again with true values $[1 \quad 0.1]$ and $N = 500{,}000$

$p(x_1^{(i)}|\tilde{y})$ with $\sigma = 0.01$



$p(x_2^{(i)}|\tilde{y})$ with $\sigma = 0.01$



- Highly non-Gaussian $\Rightarrow$ computed mean $\neq$ MLE in this case

$$\hat{\mathbf{x}} = [2.4512 \;\; 1.0094]^T, \quad \hat{\mathbf{x}}^{\text{mle}} = [1.8934 \;\; 0.2649]$$

- Requires more data points to find estimate because we have a lower frequency sinusoid than before $\Rightarrow m = 1001$ works better

Simple Non-Gaussian Example

- Consider $\tilde{y} = e^{-x\,t} + v$
  - Noise v is uniform $[-0.01 \;\; 0.01] \Rightarrow v \sim U[-0.01, 0.01]$
  - Generate 11 measurements with $\Delta t = 1$ second; truth is $x = 2$
  - Choose 2,000 particles $\Rightarrow N = 2000$
  - Assume that $q(x) = p(x)$ is a uniform distribution from 0 to 3
  - Weight updated "sequentially" with each new measurement
    - Start with $\varpi_0^{(i)} = 1/N = 0.0005$
  - By the definition of uniform distribution the probability that a point is outside this bound is zero
    - Set weights to zero that have the property

$$|\tilde{y}_k - \underbrace{e^{-x^{(i)} t_k}}_{\text{mean}}| > 0.01 \qquad w^{(i)} = 0$$

    - The points inside this bound cannot be discriminated; they are equally good
    - Compared with the Gaussian distribution, the uniform distribution only has two grades of data: good or bad $\Rightarrow$ pass or fail

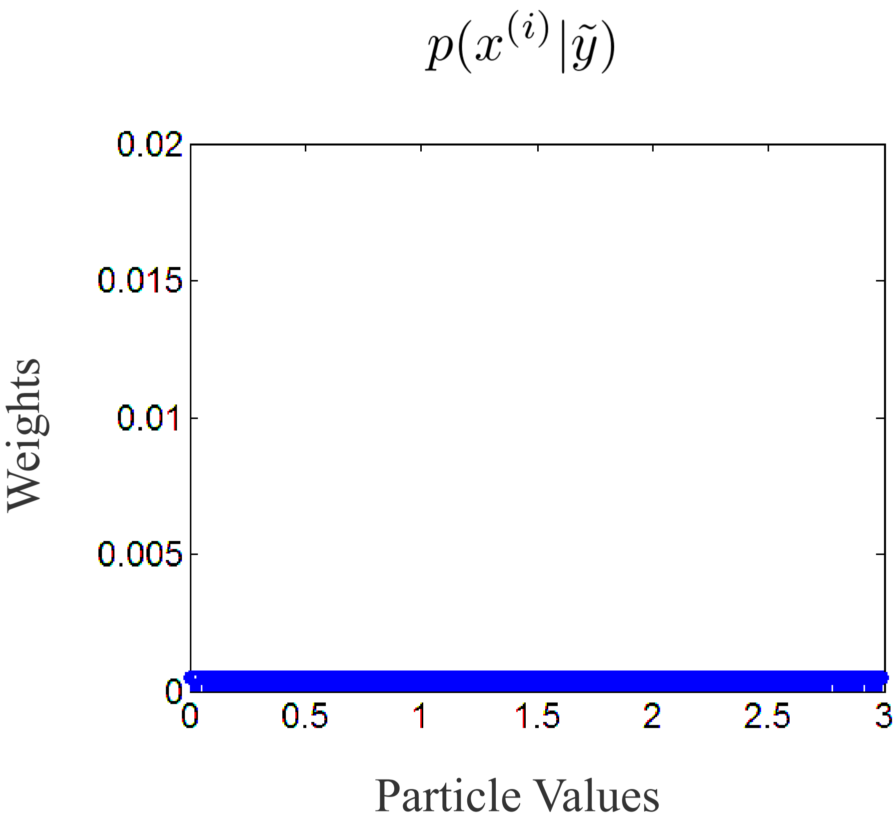
Code (i)

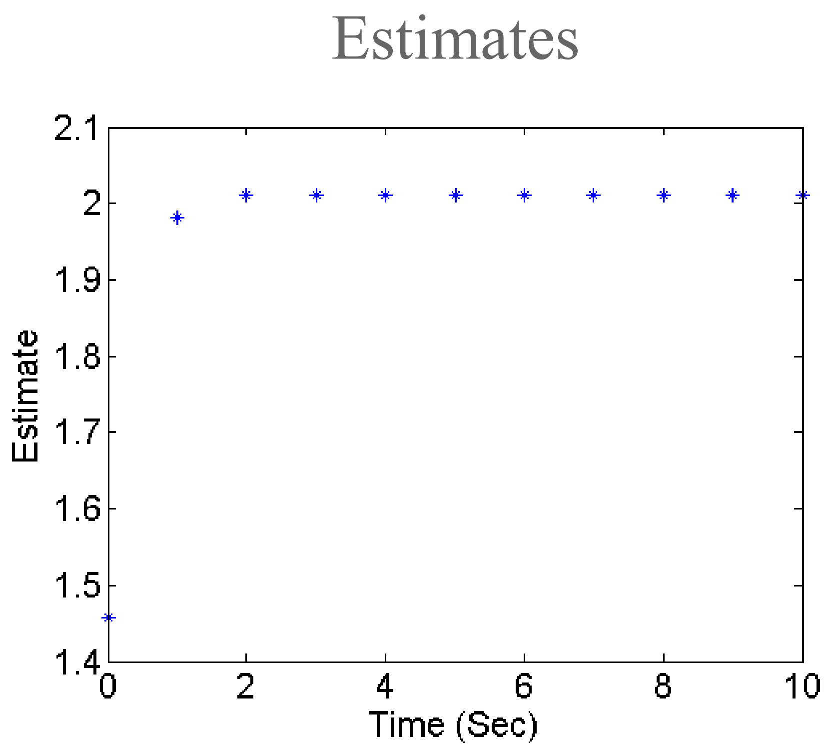
```
clear
% Truth and Measurements
a=0.01;x_true=2;
t=[0:1:10]';m=length(t);
y=exp(-x_true*t);
ym=y+a*sign(randn(m,1)).*rand(m,1);

% Particles and Weights
m_part=2000;x_est=zeros(m,1);ylimit_movie=[0 0.1];
x_particle=3*rand(m_part,1);
w=ones(m_part,1)/m_part;

% Settings for Movie
clf
clear m_get
set(gca,'xlim',[0 3],'ylim',[0 0.02],'NextPlot','replace','Visible','on')
set(gca,'nextplot','replacechildren');
```

Code (ii)

```
% Update Weights and Get Estimate
for i=1:m,
 w_nonnorm = w;
 res = ym(i)-exp(-x_particle*t(i));
 j = find(abs(res)>a);
 w_nonnorm(j) = 0;
 w = w_nonnorm/sum(w_nonnorm);
 x_est(i) = sum(x_particle.*w);
 h = stem(x_particle,w);
 set(gcf,'color',[1 1 1])
 set(gca,'fontsize',16);
 m_get(:,i)=getframe(gcf);
end
movie2gif(m_get,'out.gif','DelayTime',1)

% Show Estimate at Final Time
x_est(m)
```

Non-Gaussian Example (i)





x_est(m) = 2.0110

Non-Gaussian Example (ii)

- Some notes
  - When a is too small and m is not too large, it is possible that all the initial points are outside the interval with nonzero likelihood
    - Then all the weights vanish after a single update
  - Consider the simpler equation $\tilde{y} = e^{-x} + v$ with $x = 2$
    - If the magnitude of v is smaller than 1×10^{-4}, then the particles with distance from x larger than 1×10^{-3} (approximately) would be assigned zero weight
    - Because the prior distribution of x is in $[0, 3]$, $2 \times 10^{-3} / 3$ is the probability that a particle drawn from the uniform distribution falls in the small neighborhood of the true value
    - For 2,000 particles, only about 2 particles can have nonzero weights
    - As a keeps decreasing, the problem will become much more severe
  - The Particle filter is finite and discrete
    - It does not seem to be as good at very small or very large numbers since it is based on Monte Carlo simulation

Optimal Filtering

- Truth model

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$$
$$\tilde{\mathbf{y}}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k)$$

$$\mathbf{x}_0 \sim p(\mathbf{x}_0), \quad \mathbf{w}_k \sim p(\mathbf{w}_k)$$
$$\mathbf{v}_k \sim p(\mathbf{v}_k)$$

- Objective of optimal filtering
  - To construct posterior probability distribution $p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k)$

    all measurements at time k

    where $\tilde{\mathbf{Y}}_k = \{\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, \ldots, \tilde{\mathbf{y}}_k\}$
- Recursion of optimal filtering

$$\left.\begin{array}{c} \boxed{p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k)} \\ p(\mathbf{x}_{k+1}|\mathbf{x}_k) \\ p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}) \end{array}\right\} \Rightarrow \boxed{p(\mathbf{x}_{k+1}|\tilde{\mathbf{Y}}_{k+1})}\ ?$$

Approximate Filtering (i)

- Let $\mathbf{X}_\mathrm{k} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$; joint posterior approximation is

$$p(\mathbf{X}_k|\tilde{\mathbf{Y}}_k) \approx \sum_{i=1}^{N} \varpi_k^{(i)}\, \delta(\mathbf{X}_k - \mathbf{X}_k^{(i)}), \quad \sum_{i=1}^{N} \varpi_k^{(i)} = 1$$

  - Say $\mathbf{X}_k^{(i)}$ are drawn from an importance density $q(\mathbf{X}_k|\tilde{\mathbf{Y}}_k)$, then

$$\varpi_k^{(i)} \propto \frac{p(\mathbf{X}_k^{(i)}|\tilde{\mathbf{Y}}_k)}{q(\mathbf{X}_k^{(i)}|\tilde{\mathbf{Y}}_k)} \tag{1}$$

  - We now choose the importance density at time t_{k+1} to be factorized by

$$\overbrace{q(\mathbf{X}_{k+1}|\tilde{\mathbf{Y}}_{k+1})}^{\text{current time}} = q(\mathbf{x}_{k+1}|\mathbf{X}_k, \tilde{\mathbf{Y}}_{k+1})\, q(\mathbf{X}_k|\tilde{\mathbf{Y}}_k) \tag{2}$$

  - Now we can obtain samples $\mathbf{X}_{k+1}^{(i)} \sim q(\mathbf{X}_{k+1}|\tilde{\mathbf{Y}}_{k+1})$ by augmenting each of the existing samples $\mathbf{X}_k^{(i)} \sim q(\mathbf{X}_k|\tilde{\mathbf{Y}}_k)$ with the new state $\mathbf{x}_{k+1}^{(i)} \sim q(\mathbf{x}_{k+1}|\mathbf{X}_k, \tilde{\mathbf{Y}}_{k+1})$
- Use Bayes' rule to derive weights as before

Approximate Filtering (ii)

- Formal solution (Bayes' formula)

$$
\begin{aligned}
p(\mathbf{X}_{k+1}|\tilde{\mathbf{Y}}_{k+1}) &= \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{X}_{k+1},\tilde{\mathbf{Y}}_k)\, p(\mathbf{X}_{k+1}|\tilde{\mathbf{Y}}_k)}{p(\tilde{\mathbf{y}}_{k+1}|\tilde{\mathbf{Y}}_k)} \\
&= \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{X}_{k+1},\tilde{\mathbf{Y}}_k)\, p(\mathbf{x}_{k+1}|\mathbf{X}_k,\tilde{\mathbf{Y}}_k)\, p(\mathbf{X}_k|\tilde{\mathbf{Y}}_k)}{p(\tilde{\mathbf{y}}_{k+1}|\tilde{\mathbf{Y}}_k)} \\
&= \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1})\, p(\mathbf{x}_{k+1}|\mathbf{x}_k)}{p(\tilde{\mathbf{y}}_{k+1}|\tilde{\mathbf{Y}}_k)} p(\mathbf{X}_k|\tilde{\mathbf{Y}}_k) \\
&\propto p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1})\, p(\mathbf{x}_{k+1}|\mathbf{x}_k)\, p(\mathbf{X}_k|\tilde{\mathbf{Y}}_k) \qquad (3)
\end{aligned}
$$

- Substituting Eqs. (2) and (3) into Eq. (1) at time t_{k+1} gives

$$
\begin{aligned}
\varpi_{k+1}^{(i)} &\propto \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i)})\, p(\mathbf{x}_{k+1}^{(i)}|\mathbf{x}_k^{(i)})\, p(\mathbf{X}_k^{(i)}|\tilde{\mathbf{Y}}_k^{(i)})}{q(\mathbf{x}_{k+1}^{(i)}|\mathbf{X}_k^{(i)},\tilde{\mathbf{Y}}_{k+1})\, q(\mathbf{X}_k^{(i)}|\tilde{\mathbf{Y}}_k)} \\
&= \varpi_k^{(i)} \frac{p(\tilde{\mathbf{y}}_{k+1}|\mathbf{x}_{k+1}^{(i)})\, p(\mathbf{x}_{k+1}^{(i)}|\mathbf{x}_k^{(i)})}{q(\mathbf{x}_{k+1}^{(i)}|\mathbf{X}_k^{(i)},\tilde{\mathbf{Y}}_{k+1})}
\end{aligned}
$$

Sequential Importance Sampling

- Assume that $q(\mathbf{x}_{k+1}|\mathbf{X}_k, \tilde{\mathbf{Y}}_{k+1}) = q(\mathbf{x}_{k+1}|\mathbf{x}_k, \tilde{\mathbf{y}}_{k+1})$
  - Useful since posterior $p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k)$ is required for filtering
    - Only $\mathbf{x}_k^{(i)}$ need to be stored
  - Weight is given by

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

  - Posterior density is given by

$$p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k) \approx \sum_{i=1}^{N} \varpi_k^{(i)}\, \delta(\mathbf{x}_k - \mathbf{x}_k^{(i)})$$

  - As $N \rightarrow \infty$ this approximation becomes exact!
- Approach is known as *Sequential Importance Sampling* (SIS)
  - Basis for all Particle filtering approaches

SIS Pseudocode

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

- FOR $i = 1 : N$
  - Draw $\mathbf{x}_{k+1}^{(i)} \sim q(\mathbf{x}_{k+1} | \mathbf{x}_{k}^{(i)}, \tilde{\mathbf{y}}_{k+1})$
  - Evaluate non-normalized weights

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

- 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

- State Estimate and Covariance
  - State Estimate

$$\hat{\mathbf{x}}_k = \sum_{i=1}^{N} \varpi_k^{(i)} \mathbf{x}_k^{(i)}$$

1. EKF
2. UKF
3. Particle filter

  - Covariance (not needed for PF algorithm)

$$P_k = \sum_{i=1}^{N} \varpi_k^{(i)} \left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right) \left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right)^T$$

Degeneracy Problem

- The degeneracy phenomenon
  - It can be shown that the variance of the importance weights can only increase over time
  - After a certain number of time steps, all but one particle will have negligible weight
  - Unfortunately impossible to avoid!
- Measure of degeneracy
  - Effective sample size

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (\varpi_k^{(i)})^2}$$

  - We can easily show that $1 \leq N_{\text{eff}} \leq N$
    - $N_{\text{eff}} = N \Rightarrow$ uniform weights (all are equal to $1/N$)
    - $N_{\text{eff}} = 1 \ \Rightarrow$ one weight is equal to 1 (others are zero)
  - Choose a threshold $N_{\text{eff}} < \varepsilon$ to indicate action is needed
    - Usual action is *resampling*

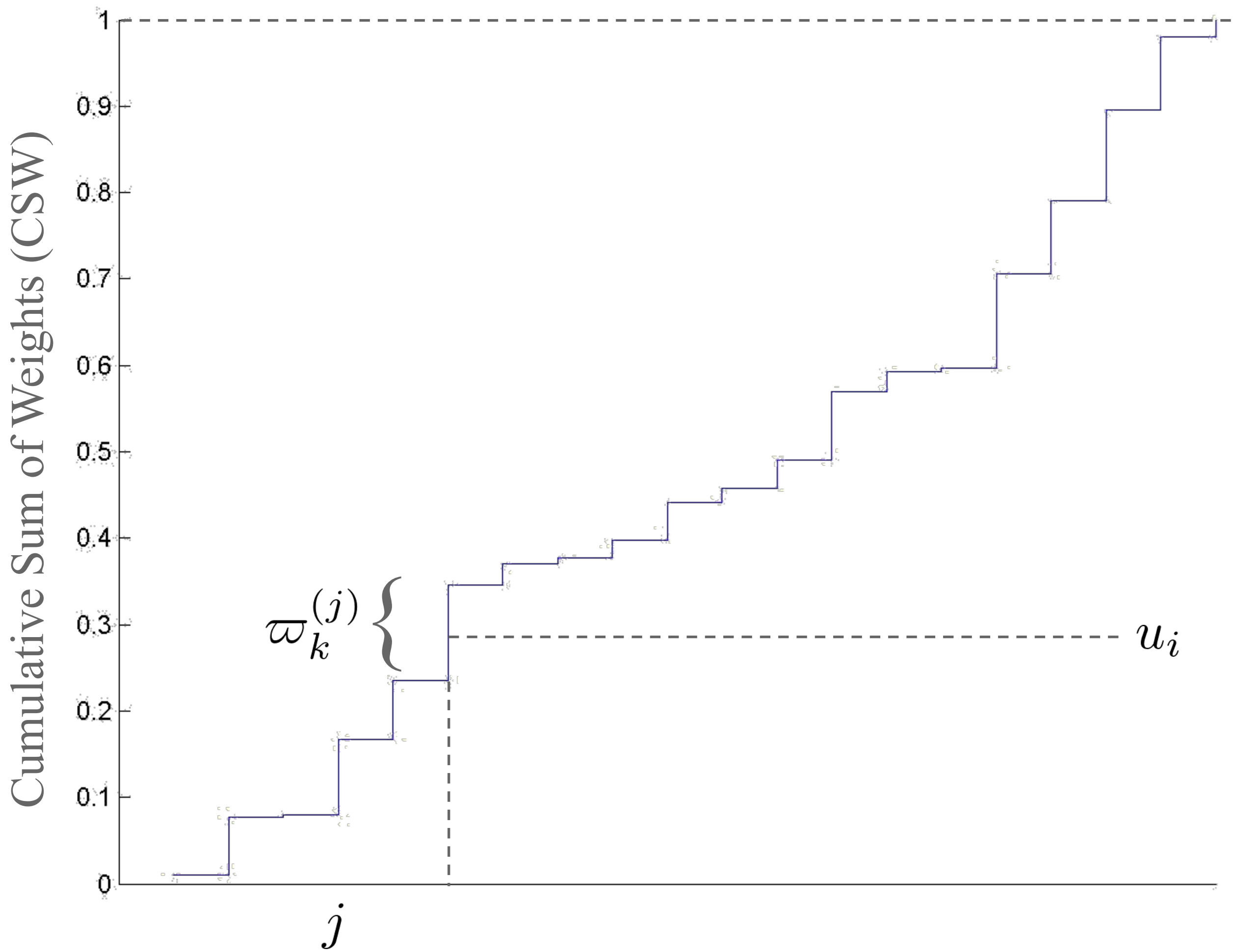
Resampling (i)

- Resampling algorithms
  - Basic idea: to discard particles with small weights and multiply particles with large weights
  - Maps random measure $\{\mathbf{x}_k^{(i)}, \varpi_k^{(i)}\} \Rightarrow$ random measure $\{\mathbf{x}_k^{(i)*}, 1/N\}$
  - New set formed by resampling (with replacement) N times from an approximate discrete representation $p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k)$

$$p(\mathbf{x}_k|\tilde{\mathbf{Y}}_k) \approx \sum_{i=1}^{N} \delta(\mathbf{x}_k - \mathbf{x}_k^{(i)})$$

    - So $P\{\mathbf{x}_k^{(i)*} = \mathbf{x}_k^{(i)}\} = \varpi_k^{(i)}$
  - Gives an independent and identically distributed (i.i.d.) sample from this pdf
    - Therefore, the weights must be equal
  - Selection is based on the cumulative sum of weights (CSW)

Resampling (ii)



$u_i \sim U[0,\,1]$ maps into index j ; since $\varpi_k^{(j)}$ has a high value, its corresponding particle has a good chance of being selected and multiplied

Resampling Schemes

- Direct implementation
  - Generate N i.i.d. variables from a uniform distribution
  - Sort these in ascending order
  - Compare them with the cumulative sum of normalized weights
  - Complexity is $\mathcal{O}(N \ln(N))$
- Many efficient approaches exist though $\approx \mathcal{O}(N)$
  - Stratified Sampling
  - Residual Sampling
  - Systematic Sampling (shown here)
    - Simple to implement
    - Minimizes the Monte Carlo variation

Cappé, O., Douc, R., and Moulines, E., “Comparison of Resampling Schemes for Particle Filtering,” *Fourth International Symposium on Image and Signal Processing and Analysis (ISPA)*, Zagreb, Croatia, Sept. 2005.

Resampling Pseudocode

$$[\{\mathbf{x}_k^{(j)*},\ \varpi_k^{(j)},\ i^{(j)}\}_{i=1}^N] = \mathrm{RESAMPLE}[\{\mathbf{x}_k^{(i)},\ \varpi_k^{(i)}\}_{i=1}^N]$$

- Initialize the CSW: $c_1 = \varpi_k^{(1)}$
- FOR $i = 2 : N$
  - Construct CSW: $c_i = c_{i-1} + \varpi_i^{(i)}$
- END FOR
- Start at $i = 1$ and draw $u_1 \sim U[0,\ N^{-1}]$
- FOR $j = 1 : N$
  - Move along the CSW: $u_j = u_1 + N^{-1}(j-1)$
  - WHILE $u_j > c_i$, $i = i + 1$, END WHILE
  - New Sample and Weight: $\mathbf{x}_k^{(j)*} = \mathbf{x}_k^{(i)}$, $\varpi_k^{(j)} = N^{-1}$; Parent: $i^{(j)} = i$
- END FOR

MATLAB Code (i)

```
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);
```

MATLAB Code (ii)

```
% 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);
```

Resampling

- Note: calculate *estimates* before resampling!
- Resampling reduces degeneracy… BUT
  - Limits opportunity to parallelize implementation
    - All particle must be combined
  - Particles with high weights are statistically selected many times
    - Leads to loss of diversity since resultant sample will contain many repeated points $\Rightarrow$ known as *Sample Impoverishment*
    - Severe in the case of small process noise
  - Other problems too
- Many methods to overcome sample impoverishment
  - Markov Chain Monte Carlo (MCMC)
    - Sound theoretical foundations
  - Regularized PF
    - Found to improve performance despite a less rigorous derivation

Regularized PF (i)

- Resamples from a continuous approximation of $p(\mathbf{x}_k | \tilde{\mathbf{Y}}_k)$
  - Effectively "jitters" the resampled values
  - Samples are drawn from

$$p(\mathbf{x}_k | \tilde{\mathbf{Y}}_k) \approx \sum_{i=1}^{N} \varpi_k^{(i)} \frac{1}{h^n} K\left(\frac{\mathbf{x}_k - \mathbf{x}_k^{(i)}}{h}\right)$$

  - $K(\cdot)$ is rescaled kernel density and h is the bandwidth
  - Kernel density is a symmetric PDF

$$\int \mathbf{x}\, K(\mathbf{x})\; d\mathbf{x} = \mathbf{0}, \quad \int ||\mathbf{x}||^2 K(\mathbf{x})\; d\mathbf{x} < \infty$$

  - Optimal bandwidth is chosen to minimize the mean-integrated square-error between true posterior and the approximated one shown above
  - Note: kernel approximation becomes increasingly less appropriate as the dimension of the state increases

Regularized PF (ii)

- For equally spaced weights, optimal choice is given by Epanechnikov kernel (not shown here)
  - Can determine the optimal h when the underlying density is Gaussian with unit covariance
    - Can still be used in general case $\Rightarrow$ suboptimal filter
- Reduce computational load by using a Gaussian kernel
  - Optimal bandwidth is given by

$$h_{\text{opt}} = \left(\frac{4}{n+2}\right)^{\frac{1}{n+4}} N^{-\frac{1}{n+4}}$$

  - After resampling $\boxed{\mathbf{x}_k^{(j)*} \leftarrow \mathbf{x}_k^{(j)*} + h_{\text{opt}} D_k\, \mathbf{g}, \quad \mathbf{g} \sim N(\mathbf{0}, I_{n\times n})}$

$$D_k\, D_k^T = P_k = \sum_{i=1}^{N} \varpi_k^{(i)} \left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k^{(i)} - \hat{\mathbf{x}}_k\right)^T$$

    - Can use Cholesky decomposition to determine D_k
    - Note: the empirical covariance is computed before resampling