**Particle Filter**

P ~ Type I Extreme Value(mean=2000, cov=0.2)

E ~ Lognormal(mean=30000, cov=0.1)

I ~ Normal(mean=10, cov=0.05)

Estimate ; = C.x1/(x2.x3) + using Particle Filter

The actual system response is



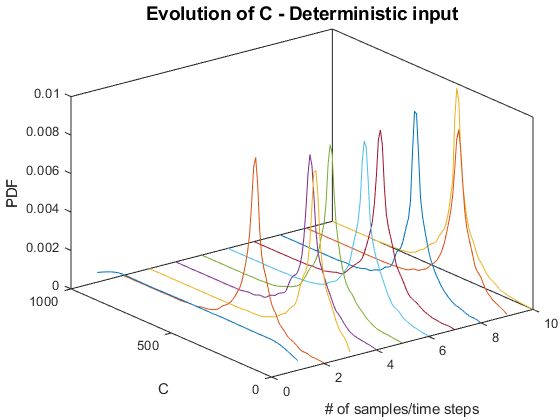
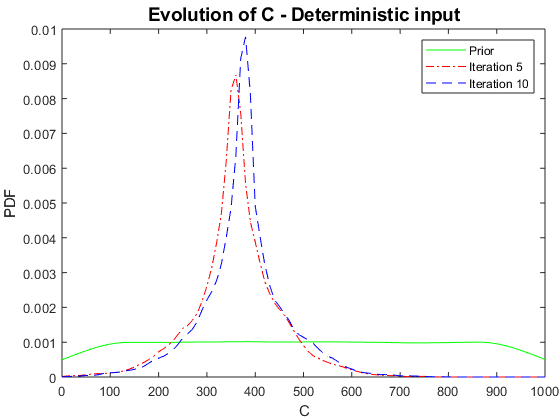
The errors are assumed to be normally distributed with zero mean and standard deviation **.**

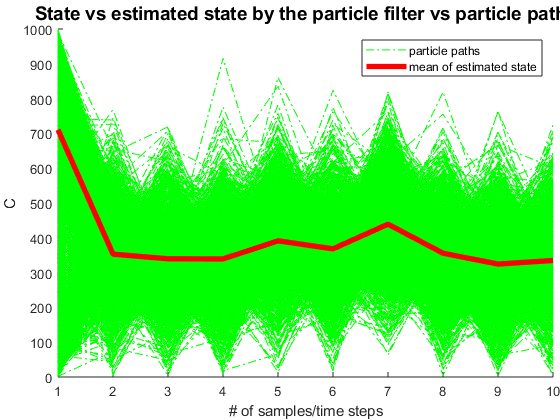


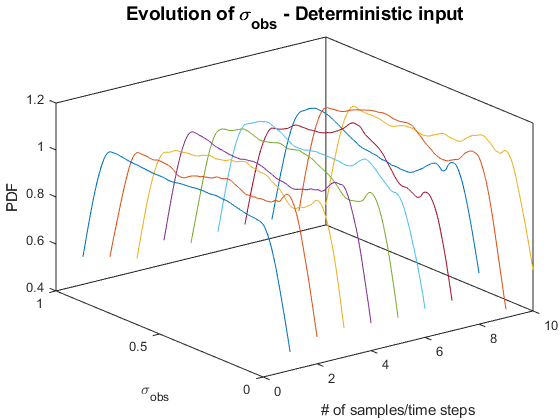
**Case I**

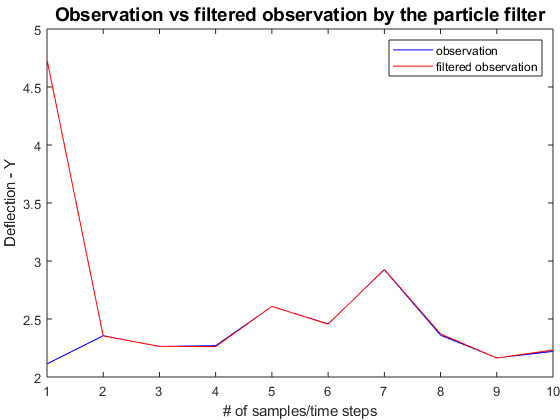
* **,**

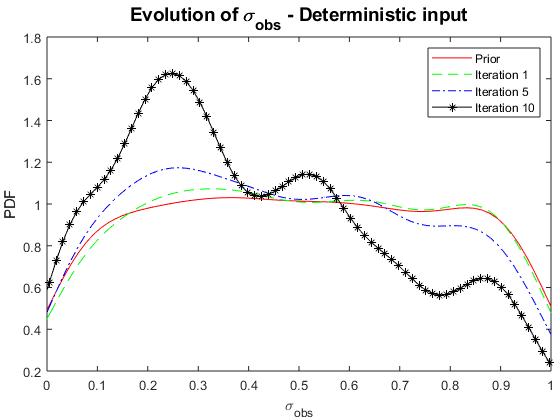
1. Generate initial particles
2. Evaluate the new collection of state particles – weights
3. Resample the current sample according to the sizes of the weights (Importance Sampling)
4. Do the same loop for each observation (we have 10 observations).

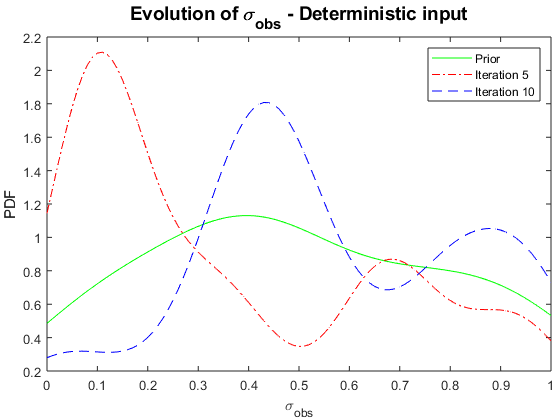












Results for from two different codes are shown above.

%% clear memory, screen, and close all figures

clear, clc, close all;

%% Inputs

% P~Type I

mu\_x1 = 2000;

% E~Lognormal

mu\_x2 = 30000;

% I~Normal

mu\_x3 = 10;

%% Inputs for the Observation/Measurement/Process Model

L = 10; % length of the beam

% mean and std of beam deflection Euler-Bernoulli

mu\_y = mu\_x1\*L^3/(3\*mu\_x2\*mu\_x3); sigma\_y = 0.3;

%% Number of time steps - Generate 10 samples of Y~N(mu\_y,sigma\_y)

T = 10;

%% Process equation x[k] = sys(k, x[k-1], u[k]); -> Pseudo-time dependent

nx = 2; % number of states

% sys = @(k, xkm1, uk) normrnd(mu\_y,sigma\_y,1,1) + uk;

cmin = 0; cmax = L^3; % define a range for starting values

smin = 0; smax = 1; % define a range for starting values

sys = @(k, xkm1, uk) [unifrnd(cmin, cmax); unifrnd(smin, smax)];

% sys = @(k, xkm1, uk) xkm1/2 + 25\*xkm1/(1+xkm1^2) + 8\*cos(1.2\*k) + uk; % (returns column vector)

%% Observation equation y[k] = obs(k, x[k], v[k]);

ny = 1; % number of observations

obs = @(k, xk, vk) xk(1,:)\*mu\_x1/(mu\_x2\*mu\_x3) + vk; % (returns column vector)

%% PDF of process noise and noise generator function

nu = 1; % size of the vector of process noise

sigma\_u = sqrt(0);

p\_sys\_noise = @(u) normpdf(u, 0, sigma\_u);

gen\_sys\_noise = @(u) normrnd(0, sigma\_u); % sample from p\_sys\_noise (returns column vector)

%% PDF of observation noise and noise generator function

nv = 1; % size of the vector of observation noise

sigma\_v = sqrt(0);

p\_obs\_noise = @(v,xk) normpdf(v, 0, xk(2,:));

gen\_obs\_noise = @(xk) normrnd(0, xk(2,:)); % sample from p\_obs\_noise (returns column vector)

%% Initial PDF

% p\_x0 = @(x) normpdf(x, 0,sqrt(10)); % initial pdf

% gen\_x0 = @(x) normrnd(0, sqrt(10)); % sample from p\_x0 (returns column vector)

gen\_x0 = @(x) [unifrnd(cmin, cmax); unifrnd(smin, smax)];

%% Transition prior PDF p(x[k] | x[k-1])

% (under the suposition of additive process noise)

% p\_xk\_given\_xkm1 = @(k, xk, xkm1) p\_sys\_noise(xk - sys(k, xkm1, 0));

%% Observation likelihood PDF p(y[k] | x[k])

% (under the suposition of additive process noise)

p\_yk\_given\_xk = @(k, yk, xk) p\_obs\_noise(yk - obs(k, xk, 0), xk);

%% Separate memory space

x = zeros(nx,T); y = zeros(ny,T);

u = zeros(nu,T); v = zeros(nv,T);

%% Simulate system

% xh0 = 0; % initial state

xh0 = sys(1, x(:,1), 0);

% u(:,1) = 0; % initial process noise

% v(:,1) = gen\_obs\_noise(xh0); % initial observation noise

x(:,1) = xh0;

% y(:,1) = obs(1, xh0, v(:,1));

y(:,1) = normrnd(mu\_y,sigma\_y,1,1);

for k = 2:T

% here we are basically sampling from p\_xk\_given\_xkm1 and from p\_yk\_given\_xk

% u(:,k) = gen\_sys\_noise(); % simulate process noise

% v(:,k) = gen\_obs\_noise(x(:,k-1)); % simulate observation noise

x(:,k) = sys(k, x(:,k-1), 0); % simulate state

% y(:,k) = obs(k, x(:,k), v(:,k)); % simulate observation

y(:,k) = normrnd(mu\_y,sigma\_y,1,1);

end

%% Separate memory

xh = zeros(nx, T); xh(:,1) = xh0;

yh = zeros(ny, T); yh(:,1) = obs(1, xh0, 0);

pf.k = 1; % initial iteration number

pf.Ns = 1000; % number of particles

pf.w = zeros(pf.Ns, T); % weights

pf.particles = zeros(nx, pf.Ns, T); % particles

pf.gen\_x0 = gen\_x0; % function for sampling from initial pdf p\_x0

pf.p\_yk\_given\_xk = p\_yk\_given\_xk; % function of the observation likelihood PDF p(y[k] | x[k])

pf.gen\_sys\_noise = gen\_sys\_noise; % function for generating system noise

%pf.p\_x0 = p\_x0; % initial prior PDF p(x[0])

%pf.p\_xk\_given\_ xkm1 = p\_xk\_given\_xkm1; % transition prior PDF p(x[k] | x[k-1])

%% Estimate state

for k = 2:T

fprintf('Iteration = %d/%d\n',k,T);

% state estimation

pf.k = k;

%[xh(:,k), pf] = particle\_filter(sys, y(:,k), pf, 'multinomial\_resampling');

[xh(:,k), pf] = particle\_filter(sys, y(:,k), pf, 'systematic\_resampling');

% filtered observation

yh(:,k) = obs(k, xh(:,k), 0);

end

%% Make plots of the evolution of the density

figure

hold on;

xi = 1:T;

yi = 0:10:1000;

[xx,yy] = meshgrid(xi,yi);

den = zeros(size(xx));

xhmode = zeros(size(xh));

for i = xi

% for each time step perform a kernel density estimation

den(:,i) = ksdensity(pf.particles(1,:,i), yi,'kernel','epanechnikov');

[~, idx] = max(den(:,i));

% estimate the mode of the density

xhmode(i) = yi(idx);

plot3(repmat(xi(i),length(yi),1), yi', den(:,i));

end

view(3);

box on;

title('Evolution of the state density','FontSize',14)

title('Evolution of C - Deterministic input','FontSize',14)

xlabel('# of samples/time steps');

ylabel('C'); zlabel('PDF');

figure

% for each time step perform a kernel density estimation

i=1;

den(:,i) = ksdensity(pf.particles(1,:,i), yi,'kernel','epanechnikov');

[~, idx] = max(den(:,1));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', den(:,i),'-g');

hold on;

i=5;

den(:,i) = ksdensity(pf.particles(1,:,i), yi,'kernel','epanechnikov');

[~, idx] = max(den(:,i));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', den(:,i),'-.r');

hold on;

i=10;

den(:,i) = ksdensity(pf.particles(1,:,i), yi,'kernel','epanechnikov');

[~, idx] = max(den(:,i));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', den(:,i),'--b');

title('Evolution of C - Deterministic input','FontSize',14)

xlabel('C');

ylabel('PDF'); zlabel('PDF');

legend('Prior','Iteration 5','Iteration 10')

% figure

% mesh(xx,yy,den);

% title('Evolution of the state density','FontSize',14)

%% plot of the state vs estimated state by the particle filter vs particle paths

figure

hold on;

h1 = plot(1:T,squeeze(pf.particles(1,:,:)),'-.g');

% h2 = plot(1:T,x,'b','LineWidth',4);

h3 = plot(1:T,xh(1,:),'r','LineWidth',4);

% h4 = plot(1:T,xhmode(1,:),'g','LineWidth',4);

% legend([h1(pf.Ns) h2 h3 h4],'particle paths','state','mean of estimated state','mode of estimated state');

legend([h1(pf.Ns) h3],'particle paths','mean of estimated state');

title('State vs estimated state by the particle filter vs particle paths','FontSize',14);

xlabel('# of samples/time steps');

ylabel('C');

%% plot of the observation vs filtered observation by the particle filter

figure

plot(1:T,y,'b', 1:T,yh,'r');

legend('observation','filtered observation');

title('Observation vs filtered observation by the particle filter','FontSize',14);

xlabel('# of samples/time steps');

ylabel('Deflection - Y');

%% Make plots of the evolution of the density - Sigma

figure

hold on;

xi = 1:T;

yi = 0:0.01:1;

[xx,yy] = meshgrid(xi,yi);

den = zeros(size(xx));

xhmode = zeros(size(xh));

for i = xi

% for each time step perform a kernel density estimation

den(:,i) = ksdensity(pf.particles(2,:,i), yi,'kernel','epanechnikov');

[~, idx] = max(den(:,i));

% estimate the mode of the density

xhmode(i) = yi(idx);

plot3(repmat(xi(i),length(yi),1), yi', den(:,i));

end

view(3);

box on;

title('Evolution of the state density','FontSize',14)

title('Evolution of \sigma\_{obs} - Deterministic input','FontSize',14)

xlabel('# of samples/time steps');

ylabel('\sigma\_{obs}'); zlabel('PDF');

%% 2D

figure

% for each time step perform a kernel density estimation

i=1;

dens(:,i) = ksdensity(pf.particles(2,:,i), yi);

[~, idx] = max(dens(:,i));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', dens(:,i),'-g');

hold on;

i=5;

dens(:,i) = ksdensity(pf.particles(2,:,i), yi);

[~, idx] = max(dens(:,i));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', dens(:,i),'-.r');

hold on;

i=10;

dens(:,i) = ksdensity(pf.particles(2,:,i), yi);

[~, idx] = max(dens(:,i));

% estimate the mode of the density

xhmode(1,i) = yi(idx);

plot(yi', dens(:,i),'--b');

title('Evolution of \sigma\_{obs} - Deterministic input','FontSize',14)

xlabel('\sigma\_{obs}');

ylabel('PDF');

legend('Prior','Iteration 5','Iteration 10')

% %% plot of the state vs estimated state by the particle filter vs particle paths

% figure

% hold on;

% h1 = plot(1:T,squeeze(pf.particles(2,:,:)),'-.g');

% h3 = plot(1:T,xh(2,:),'r','LineWidth',4);

% legend([h1(pf.Ns) h3],'particle paths','mean of estimated state');

% title('State vs estimated state by the PF vs particle paths','FontSize',14);

% xlabel('# of samples/time steps');

% ylabel('\sigma\_{obs}');

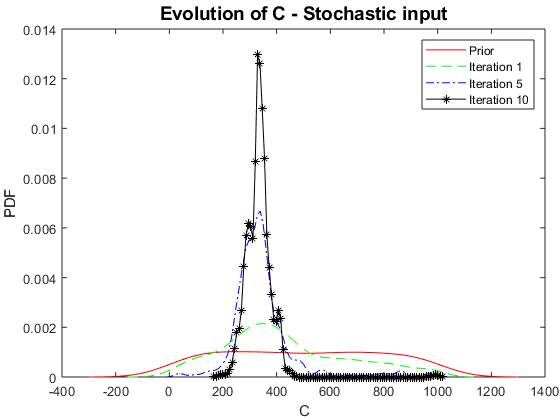
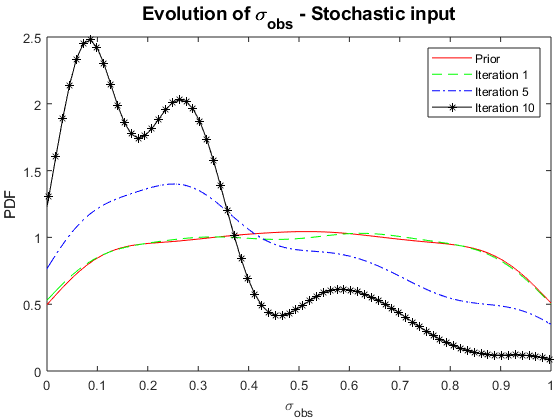
return

**Case II**

* **,**

1. Generate initial particles
2. Do Monte Carlo Sampling for each stochastic input
3. Use ksdensity to find the pdf of the observed data
4. Evaluate the new collection of state particles – weights for the MC samples
5. Resample the current sample according to the sizes of the weights (Importance Sampling)
6. Do the same loop for each observation (we have 10 observations).

Note: Latin hypercube sampling is used for and also for instead of uniformly distributed random samples, because it gives a better estimates, especially for .



The results are in agreement with the measured results. Because of the uncertainty in observed values the results are showing small amounts of variability.

%% Inputs

% P~Type I

mu\_x1 = 2000;

% E~Lognormal

mu\_x2 = 30000;

% I~Normal

mu\_x3 = 10;

%% Inputs for the Observation/Measurement/Process Model

L = 10; % length of the beam

% mean and std of beam deflection Euler-Bernoulli

mu\_y = mu\_x1\*L^3/(3\*mu\_x2\*mu\_x3); sigma\_y = 0.3;

% Y\_obs = Y\_m - eps\_obs

y\_obs = normrnd(mu\_y,sigma\_y,1,10);

% # of obs.

obs = length(y\_obs);

% # of particles

N = 1000;

% Uniform numbers

u = rand(N,2);

% Initial, prior particles

initParticles = zeros(N,2);

initParticles(:,1) = 1000 \* u(:,1);

initParticles(:,2) = u(:,2);

% Initial weight for each particle

w = 1/N \* ones(N,1);

Particles = zeros(N,2,obs+1);

Particles(:,:,1) = initParticles;

for i=1:obs

tic

CurrState = Particles(:,:,i);

w = myWeight(y\_obs(i), CurrState(:,1), CurrState(:,2));

Particles(:,:,i+1) = Resampling(CurrState, w);

toc

end

%% Figures

figure

[p,f] = ksdensity(Particles(:,1,1));

[p1,f1] = ksdensity(Particles(:,1,2));

[p5,f5] = ksdensity(Particles(:,1,5));

[p10,f10] = ksdensity(Particles(:,1,11));

plot(f,p,'-r',f1,p1,'--g',f5,p5,'-.b',f10,p10,'-\*k');

title('Evolution of C - Stochastic input','FontSize',14)

xlabel('C');

ylabel('PDF');

legend('Prior','Iteration 1','Iteration 5','Iteration 10')

%% Figures

figure

[p,f] = ksdensity(Particles(:,2,1));

[p1,f1] = ksdensity(Particles(:,2,2));

[p5,f5] = ksdensity(Particles(:,2,5));

[p10,f10] = ksdensity(Particles(:,2,11));

plot(f,p,'-r',f1,p1,'--g',f5,p5,'-.b',f10,p10,'-\*k');

title('Evolution of \sigma\_{obs} - Stochastic input','FontSize',14)

xlabel('\sigma\_{obs}');

ylabel('PDF');

legend('Prior','Iteration 1','Iteration 5','Iteration 10')

function wk = myWeight(y\_obs, C, sigma\_obs)

Nn = length(C);

w = zeros(Nn,1);

for i=1:Nn

C\_i = C(i,1);

sigma\_obs\_i = sigma\_obs(i,1);

%% # of MCS for X

N = 1000;

% Uniform numbers (LHS)

r = [lhsdesign(N,4)];

u = 0.5772;

sigmaBar = sqrt(6)\*(2000\*0.2)/pi;

muBar=2000-u\*sigmaBar;

muN\_x2 = log(30000 / sqrt(1 + 3000^2/30000^2));

sigmaN\_x2 = sqrt(log(1 + 3000^2/30000^2));

%% MCS

%--------------------------------------------------------------------------

x1 = icdf('GeneralizedExtreme Value', r(:,1), 0, sigmaBar, muBar);

x2 = icdf('Lognormal', r(:,2), muN\_x2, sigmaN\_x2);

x3 = icdf('Normal', r(:,3), 10, 0.5);

% Y\_obs = Y\_m - eps\_obs

eps\_obs = icdf('Normal', r(:,4), 0, abs( sigma\_obs\_i ));

Y\_m = C\_i\*x1./(x2.\*x3);

Y\_obs = Y\_m - eps\_obs;

% find pdf values of corresponding y\_obs

[fobs, yobs] = ksdensity(Y\_obs);

[~,ind] = min(abs(yobs-y\_obs));

w(i,1) = fobs(ind);

end

wk = w/sum(w);

end

function NextParticles = Resampling(CurrState, w)

% n = length(CurrState);

% edges = min([0 cumsum(w)'],1); % protect against accumulated round-off

% edges(end) = 1; % get the upper edge exact

% u1 = rand/n;

%

% % this works like the inverse of the empirical distribution and returns

% % the interval where the sample is to be found

% [~, idx] = histc(u1:1/n:1, edges);

% Particles = CurrState(idx,:); % extract new particles

NextParticles = zeros(size(CurrState));

len = length(w);

u = rand(len,1);

myCumsum = cumsum(w);

myCumsum(end) = 1;

for i=1:len

[~,ind] = min(abs(myCumsum - u(i)));

NextParticles(i,:) = CurrState(ind,:);

end

end