a) Cointegration

```
clear ; close all; clc
load Data_Canada
Y = Data(:,3:end);

figure
plot(dates,Y,'LineWidth',2)
xlabel('Year')
ylabel('Percent')
names = series(3:end);
legend(names,'location','NW')
title('{\bf Canadian Interest Rates, 1954-1994}')
axis tight
grid on

y1 = Y(:,1); % Short-term rate

% Levels data:
fprintf('=== Test y1 for a unit root ===\n\n')
[h1,pVal1] = adftest(y1,'model','ARD') % Left-tail probability

fprintf('\n=== Test y1 for stationarity ===\n\n')
[h0,pVal0] = kpsstest(y1,'trend',false) % Right-tail probability

% Differenced data:
fprintf('\n=== Test (1-L)y1 for a unit root ===\n\n')
[h1D,pVal1D] = adftest(diff(y1),'model','ARD') % Left-tail
probability

fprintf('\n=== Test (1-L)y1 for stationarity ===\n\n')
[h0D,pVal0D] = kpsstest(diff(y1),'trend',false) % Right-tail
probability

figure
plot(dates(2:end),diff(Y),'LineWidth',2)
names = series(3:end);
legend(names,'location','NW')
title('{\bf Differenced Data}')
axis tight
grid on

% Run the test with both "tau" (t1) and "z" (t2) statistics:
```

```
fprintf('\n=== Engle-Granger tests for cointegration ===\n\n')
[hEG,pValEG] = egcitest(Y,'test',{'t1','t2'})

% Return the results of the cointegrating regression:
[~,~,~,~,reg] = egcitest(Y,'test','t2');

c0 = reg.coeff(1);
b = reg.coeff(2:3);
figure
C = get(gca,'ColorOrder');
set(gca,'NextPlot','ReplaceChildren','ColorOrder',circshift(C,3));
plot(dates,Y*[1;-b]-c0,'LineWidth',2);
title('{\bf Cointegrating Relation}');
axis tight
grid on

% Permutations of the data variables:
P0 = perms([1 2 3]);
[~,idx] = unique(P0(:,1)); % Rows of P0 with unique regressand y1
P = P0(idx,:); % Unique regressions
numPerms = size(P,1);

% Preallocate:
T0 = size(Y,1);
HEG = zeros(1,numPerms);
PValEG = zeros(1,numPerms);
CIR = zeros(T0,numPerms);

% Run all tests:
for i = 1:numPerms

    YPerm = Y(:,P(i,:));
    [h,pVal,~,~,reg] = egcitest(YPerm,'test','t2');
    HEG(i) = h;
    PValEG(i) = pVal;
    c0i = reg.coeff(1);
    bi = reg.coeff(2:3);
    CIR(:,i) = YPerm*[1;-bi]-c0i;

end

fprintf('\n=== Different Engle-Granger tests, same data ===\n\n')
HEG,PValEG
```

```
% Plot the cointegrating relations:
figure
C = get(gca,'ColorOrder');
set(gca,'NextPlot','ReplaceChildren','ColorOrder',circshift(C,3))
plot(dates,CIR,'LineWidth',2)
title('{\bf Multiple Cointegrating Relations}')
legend(strcat({'Cointegrating relation  '}, ...
     num2str((1:numPerms)')),'location','NW');
axis tight
grid on

fprintf('\n=== Johansen tests for cointegration ===\n')
[hJ,pValJ] = jcitest(Y,'model','H1','lags',1:2);

[~,~,~,~,mles] = jcitest(Y,'model','H1','lags',2,'display','params');

B = mles.r2.paramVals.B % Cointegrating relations with rank = 2
restriction

fprintf('\n=== Test y1, y2,y3 for stationarity ===\n\n')
[h0J,pVal0J] = jcontest(Y,1,'BVec',{[1 0 0]',[0 1 0]',[0 0 1]'})



clear;
echo off;

N = 500 ;                    % Number of particles.
T = 100;                     % Number of time steps.

n_x = 4;                    % Continuous state dimension.
n_z = 3;                    % Number of discrete states.
n_y = 4;                    % Dimension of observations.

parameter.A = zeros(n_x,n_x,n_z);
parameter.B = zeros(n_x,n_x,n_z);
parameter.C = zeros(n_y,n_x,n_z);
parameter.D = zeros(n_y,n_y,n_z);
parameter.E = zeros(n_x,n_x,n_z);
parameter.F = zeros(n_x,1,n_z);
parameter.G = zeros(n_y,1,n_z);
ST = 1;                % sampling period
u = ones(1,T);                           % input signal(Deterministic)
```

b)particle filter, rao-blackwellized particle filter, extended kalman filter

```
%Defining parameters
for i=1:n_z
  parameter.A(:,:,i) = [1 ST 0 0; 0 1 0 0; 0 0 1 ST; 0 0 0 1];
  parameter.C(:,:,i) = eye(n_x);
  parameter.B(:,:,i) = 0.2*eye(n_x,n_x);
  parameter.D(:,:,i) = sqrt(3)*diag([20,1,20,1]);
  parameter.G(:,:,i) = zeros(n_y,1);

% input and control vectors as given in paper
parameter.F(:,1,1)  = [0 0 0 0]';
parameter.F(:,1,2) = [-1.225, -0.35, 1.225, 0.35]';
parameter.F(:,1,3) = [1.225, 0.35, -1.225, -0.35]';

% Transition matrix
parameter.T = [0.9 0.05 0.05;
               0.05 0.9 0.05;
               0.05 0.05 0.9];

parameter.pz0 = [0 1 0];                          % Initial r
markov chain
parameter.mu0 = zeros(n_x,1);                  % Initial Gaussian mean
parameter.S0  = 1*eye(n_x,n_x);                % Initial Gaussian
covariance

% Generating ground truth
x = zeros(n_x,T);
z = zeros(1,T);
y = zeros(n_y,T);

x(:,1) = parameter.mu0 + sqrtm(parameter.S0)*randn(n_x,1);
z(1) = length(find(cumsum(parameter.pz0')<rand))+1;

for t=2:T
  z(t) = length(find(cumsum(parameter.T(z(t-1),:)')<rand))+1;
  x(:,t) = parameter.A(:,:,z(t))*x(:,t-1) +
parameter.B(:,:,z(t))*randn(n_x,1) + parameter.F(:,:,z(t))*u(:,t);
  y(:,t) = parameter.C(:,:,z(t))*x(:,t) +
parameter.D(:,:,z(t))*randn(n_y,1) + parameter.G(:,:,z(t))*u(:,t);
end
```

```
%%%%Plotting ground truth and noisy observations
figure; hold on
plot(x(1,:), x(3,:), '*');
plot(y(1,:), y(3,:), '.');
drawnow

[styles, colors, symbols, str] =  plotColors;

%%%% Plotting change in the state of target as it maneuvers
figure; hold on
for k=1:3
  ndx=find(z==k);
  plot(x(1,ndx), x(3,ndx), sprintf('%s%s', colors(k), symbols(k)));
  %plot(x(1,ndx), x(3,ndx));
end
title('data')
drawnow

%%%% particle filter jmls
[zest_pf, xest_pf, zamples_pf, xsamples_pf] = pfSlds(N, par, y, u);

%%%% Rao-Balckwellized particle filter
[zest_rbpf, xest_rbpf, zsamples_rbpf] = rbpfSlds(N, par, y, u);

%%%% Extended Kalman filter
[zest_ekf, xest_ekf, zsamples_ekf] = ekf(N, par, y, u);

%%%%Calculating Mean square error
mse_ekf = mean((xest_ekf(1,:)-x(1,:)).^2) +
mean((xest_ekf(3,:)-x(3,:)).^2);
mse_pf = mean((xest_pf(1,:)-x(1,:)).^2) +
mean((xest_pf(3,:)-x(3,:)).^2);
mse_rbpf = mean((xest_rbpf(1,:)-x(1,:)).^2) +
mean((xest_rbpf(3,:)-x(3,:)).^2);

[junk,z_max_pf] = max(zest_pf,[],1);

%%%%Plotting extended Kalman filter estimate
[junk,z_max_ekf] = max(zest_ekf,[],1);
figure; hold on
for k=1:n_z
  ndx=find(z_max_ekf==k);
```

```
  plot(xest_ekf(1,ndx), xest_ekf(3,ndx));
end
title(sprintf('ekf, mse %5.3f', mse_ekf));

%%%%Plotting Particle filter estimate
figure; hold on
for k=1:n_z
  ndx=find(z_max_pf==k);
  plot(xest_pf(1,ndx), xest_pf(3,ndx));
end
title(sprintf('pf, mse %5.3f', mse_pf));


%%%%Plotting rbpf estimate
[junk,z_max_rbpf] = max(zest_rbpf,[],1);
figure; hold on
for k=1:n_z
  ndx=find(z_max_rbpf==k);
  plot(xest_rbpf(1,ndx), xest_rbpf(3,ndx));
end
title(sprintf('rbpf, mse %5.3f', mse_rbpf));

{
figure;
plot(1:T,z,'k',1:T,z,'ko',1:T,z_max_rbpf,'r+',1:T,z_max_pf,'bv','line
width',1);
legend('','True state','RBPF MAP estimate','PF MAP estimate');
% axis([0 T+1 0.5 n_z+0.5])
}

% error rate Z
detect_error_pf   = sum(z~=z_max_pf)/T;
detect_error_rbpf = sum(z~=z_max_rbpf)/T;
z_ind = dummyEncoding(z(:), n_z);
figure;
subplot(1,3,1); imagesc(z_ind);
title('truth');
subplot(1,3,2); imagesc(zest_pf');
title(sprintf('pf, error rate %5.3f', detect_error_pf));
subplot(1,3,3); imagesc(zest_rbpf');
title(sprintf('rbpf, error rate %5.3f', detect_error_rbpf));
subplot(1,4,4); imagesc(zest_ekf');
title(sprintf('ekf, error rate %5.3f', detect_error_ekf));
```

```
fprintf('PF: misclassification rate %5.3f, mse %5.3f, time %5.3f\n',
...
  detect_error_pf, mse_pf, time_pf);
fprintf('RBPF: misclassification rate %5.3f, mse %5.3f, time
%5.3f\n', ...
  detect_error_rbpf, mse_rbpf, time_rbpf);

fprintf('EKF: misclassification rate %5.3f, mse %5.3f, time %5.3f\n', ...
  detect_error_ekf, mse_ekf, time_ekf);

function [zest, xest, zsamples, w] = rbpfSlds(N, par, y, u,
resamplingScheme)
if nargin < 5, resamplingScheme = 2; end

 [nz,T] = size(y);
 [n_y, n_x, n_z] = size(parameter.C);
 z_rbpf = ones(1,T,N);
z_rbpf_pred = ones(1,T,N);     % One-step-ahead predicted values of
z.
mu = 0.01*randn(n_x,T,N);      % Kalman mean of x.
mu_pred = 0.01*randn(n_x,N);
Sigma = zeros(n_x,n_x,N);      % Kalman covariance of x.
Sigma_pred = zeros(n_x,n_x,N);
S = zeros(n_y,n_y,N);          % Kalman predictive covariance.
y_pred = 0.01*randn(n_y,T,N);
w = ones(T,N);                 % Importance weights.
initz = 1/n_z*ones(1,n_z);
xest = zeros(n_x,T); % KPM
zest = zeros(n_z,T); % KPM
for i=1:N
  Sigma(:,:,i) = 1*eye(n_x,n_x);
  Sigma_pred(:,:,i) = Sigma(:,:,i);
  z_rbpf(:,1,i) = length(find(cumsum(initz')<rand))+1;
  S(:,:,i) =
parameter.C(:,:,z_rbpf(1,1,i))*Sigma_pred(:,:,i)*parameter.C(:,:,z_rb
pf(1,1,i))' + ...

parameter.D(:,:,z_rbpf(1,1,i))*parameter.D(:,:,z_rbpf(1,1,i))';
end
```

```
for t=2:T
  %fprintf('RBPF :  t = %i / %i  \r',t,T); fprintf('\n');

  % Sequential Importance Sampling Step:
  for i=1:N
    % sample z(t)~p(z(t)|z(t-1))
    z_rbpf_pred(1,t,i) =
length(find(cumsum(parameter.T(z_rbpf(1,t-1,i),:)')<rand))+1;

    % Kalman prediction:
    mu_pred(:,i) = parameter.A(:,:,z_rbpf_pred(1,t,i))*mu(:,t-1,i) +
...
                   parameter.F(:,:,z_rbpf_pred(1,t,i))*u(:,t);

Sigma_pred(:,:,i)=parameter.A(:,:,z_rbpf_pred(1,t,i))*Sigma(:,:,i)*pa
rameter.A(:,:,z_rbpf_pred(1,t,i))'...
                       +
parameter.B(:,:,z_rbpf_pred(1,t,i))*parameter.B(:,:,z_rbpf_pred(1,t,i
))';
    S(:,:,i)=
parameter.C(:,:,z_rbpf_pred(1,t,i))*Sigma_pred(:,:,i)*parameter.C(:,:
,z_rbpf_pred(1,t,i))' + ...

parameter.D(:,:,z_rbpf_pred(1,t,i))*parameter.D(:,:,z_rbpf_pred(1,t,i
))';
    y_pred(:,t,i) = parameter.C(:,:,z_rbpf_pred(1,t,i))*mu_pred(:,i)
+ ...
                      parameter.G(:,:,z_rbpf_pred(1,t,i))*u(:,t);
  end
  % Evaluate importance weights.
  for i=1:N
    w(t,i) = (det(S(:,:,i))^(-0.5))*  ...
             exp(-0.5*(y(:,t)-y_pred(:,t,i))'*pinv(S(:,:,i))*(y(:,t)-
...
                                    y_pred(:,t,i))) + 1e-99;
  end
%  w(t,:) = exp(log_w(t,:))+ 1e-99*ones(size(w(t,:)));
  w(t,:) = w(t,:)./sum(w(t,:));       % Normalise the weights.


  % Selection Step:
    if resamplingScheme == 1
    outIndex = residualR(1:N,w(t,:)');
```

```
  elseif resamplingScheme == 2
    outIndex = deterministicR(1:N,w(t,:)');
  else
    outIndex = multinomialR(1:N,w(t,:)');
  end
  z_rbpf(1,t,:) = z_rbpf_pred(1,t,outIndex);
  mu_pred = mu_pred(:,outIndex);
  Sigma_pred = Sigma_pred(:,:,outIndex);
  S = S(:,:,outIndex);
  y_pred(:,t,:) = y_pred(:,t,outIndex);


  % Kalman Update:
    for i=1:N
    % Kalman update:
    K =
Sigma_pred(:,:,i)*parameter.C(:,:,z_rbpf(1,t,i))'*pinv(S(:,:,i));
    mu(:,t,i) = mu_pred(:,i) + K*(y(:,t)-y_pred(:,t,i));
    Sigma(:,:,i) = Sigma_pred(:,:,i) -
K*parameter.C(:,:,z_rbpf(1,t,i))*Sigma_pred(:,:,i);
  end
  xest(:,t) = mean(squeeze(mu(:,t,:)), 2); % KPM - unweighted mean
  zest(:,t) = normalize(hist(squeeze(z_rbpf(1,t,:)), 1:n_z)); % KPM
end   % End of t loop.

zsamples = squeeze(z_rbpf); % (1,t,:) -> (t,:)

end

function [zest, xest, zsamples, xsamples,  w] = pfSlds(N, par, y, u,
resamplingScheme)
if nargin < 5, resamplingScheme = 2; end


 [nz,T] = size(y);
 [n_y, n_x, n_z] = size(parameter.C);

z_pf = ones(1,T,N);            z_pf_pred = ones(1,T,N);       %
One-step-ahead predicted values of z.
x_pf = 10*randn(n_x,T,N);
x_pf_pred = x_pf;
y_pred = 10*randn(n_y,T,N);
w = ones(T,N);                 % Importance weights.
```

```
initz = 1/n_z*ones(1,n_z);
xest = zeros(n_x,T); %KPM
zest = zeros(n_z,T); % KPM

for i=1:N
  z_pf(:,1,i) = length(find(cumsum(initz')<rand))+1;
end


for t=2:T
  %fprintf('PF :  t = %i / %i  \r',t,T); fprintf('\n');

  % Sequential Importance Sampling Step:

  for i=1:N
    % sample z(t)~p(z(t)|z(t-1))
    z_pf_pred(1,t,i) =
length(find(cumsum(parameter.T(z_pf(1,t-1,i),:)')<rand))+1;
    % sample x(t)~p(x(t)|z(t|t-1),x(t-1))
    x_pf_pred(:,t,i) = parameter.A(:,:,z_pf_pred(1,t,i)) *
x_pf(:,t-1,i) + ...
                       parameter.B(:,:,z_pf_pred(1,t,i))*randn(n_x,1)
+ ...
                       parameter.F(:,:,z_pf_pred(1,t,i))*u(:,t);
  end
  % Evaluate importance weights.
  for i=1:N
    y_pred(:,t,i) =  parameter.C(:,:,z_pf_pred(1,t,i)) *
x_pf_pred(:,t,i) + ...
                     parameter.G(:,:,z_pf_pred(1,t,i))*u(:,t);
    Cov =
parameter.D(:,:,z_pf_pred(1,t,i))*parameter.D(:,:,z_pf_pred(1,t,i))';
    w(t,i) =  (det(Cov)^(-0.5))*exp(-0.5*(y(:,t)-y_pred(:,t,i))'* ...
                         pinv(Cov)*(y(:,t)-y_pred(:,t,i))) + 1e-99;
  end
  w(t,:) = w(t,:)./sum(w(t,:));       % Normalise the weights.


  % Selection Step:

  if resamplingScheme == 1
    outIndex = residualR(1:N,w(t,:)');
  elseif resamplingScheme == 2
```

```
    outIndex = deterministicR(1:N,w(t,:)');
  else
    outIndex = multinomialR(1:N,w(t,:)');
  end
  z_pf(1,t,:) = z_pf_pred(1,t,outIndex);
  x_pf(:,t,:) = x_pf_pred(:,t,outIndex);
  xest(:,t) = mean(squeeze(x_pf(:,t,:)), 2); % KPM - unweighted mean
  zest(:,t) = normalize(hist(squeeze(z_pf(1,t,:)), 1:n_z)); % KPM
end   % End of t loop.

zsamples = squeeze(z_pf); % (1,t,:) -> (t,:)
xsamples = x_pf;

end

function [X_ind, X3d] = dummyEncoding(X, nStates)
[N, D] = size(X);
if nargin < 2
    nStates = nunique(X);
end

%{
offset = cumsum(nStates);
offset = [0, offset(1:end-1)];
X = bsxfun(@plus, X, offset)';
I = repmat(1:N, D, 1);
K = max(sum(nStates), max(X(:)));
ndx = sub2ind([N, K], I(:), X(:));
X_ind = false(N, K);
X_ind(ndx) = true;
%}

K = max(nStates);
X_ind = zeros(N, sum(nStates));
X3d = zeros(N, D, K);
for d = 1:D
  idx = sum(nStates(1:d-1))+1:sum(nStates(1:d));
  miss = isnan(X(:,d));
  X_ind(~miss,idx) = bsxfun(@eq, X(~miss,d), [1:nStates(d)]);
  X_ind(miss,idx) = NaN;
  X3d(~miss,d,1:nStates(d)) = reshape(X_ind(~miss, idx), [sum(~miss)
1 nStates(d)]);
end
```

```
test = false;
if test
    X = bsxfun(@minus, X', offset); % return X back to original state
    tic
    if nargin < 2
        nStates = zeros(1,D);
        for j=1:D
            nStates(j) = length(unique(X(:,j)));
        end
    end
    X_ind2 = zeros(N,sum(nStates));
    offset = 0;
    for s = 1:length(nStates)
        for i = 1:N
            X_ind2(i,offset+X(i,s)) = 1;
        end
        offset = offset+nStates(s);
    end
    toc
    assert(isequal(X_ind, X_ind2));
end
end

function [zest, xest, zsamples] = ekf(N, par, y, u)
[nz,T] = size(y);
 [n_y, n_x, n_z] = size(par.C);
 z_ekf = ones(1,T,N);

z_ekf_pred = ones(1,T,N);
mu = 0.01*randn(n_x,T,N);
mu_pred = 0.01*randn(n_x,N);
Sigma = zeros(n_x,n_x,N);
Sigma_pred = zeros(n_x,n_x,N);
S = zeros(n_y,n_y,N);
y_pred = 0.01*randn(n_y,T,N);
w = ones(T,N);
initz = 1/n_z*ones(1,n_z);
xest = zeros(n_x,T); % KPM
zest = zeros(n_z,T); % KPM
for i=1:N
  Sigma(:,:,i) = 1*eye(n_x,n_x);
  Sigma_pred(:,:,i) = Sigma(:,:,i);
  z_ekf(:,1,i) = length(find(cumsum(initz')<rand))+1;
```

```
  S(:,:,i) =
par.C(:,:,z_ekf(1,1,i))*Sigma_pred(:,:,i)*par.C(:,:,z_ekf(1,1,i))' + ...
                par.D(:,:,z_ekf(1,1,i))*par.D(:,:,z_ekf(1,1,i))';
end



    for i=1:N
    % sample z(t)~p(z(t)|z(t-1))
    z_ekf(1,t,i) = length(find(cumsum(par.T(z_ekf(1,t-1,i),:)')<rand))+1;

% Kalman prediction:
    mu_pred(:,i) = par.A(:,:,z_ekf_pred(1,t,i))*mu(:,t-1,i) + ...
                   par.F(:,:,z_ekf_pred(1,t,i))*u(:,t);

Sigma_pred(:,:,i)=par.A(:,:,z_ekf_pred(1,t,i))*Sigma(:,:,i)*par.A(:,:,z_ek
f_pred(1,t,i))'...
                        +
par.B(:,:,z_ekf_pred(1,t,i))*par.B(:,:,z_ekf_pred(1,t,i))';
    S(:,:,i)=
par.C(:,:,z_ekf_pred(1,t,i))*Sigma_pred(:,:,i)*par.C(:,:,z_ekf_pred(1,t,i)
)' + ...
               par.D(:,:,z_ekf_pred(1,t,i))*par.D(:,:,z_ekf_pred(1,t,i))';
    y_pred(:,t,i) = par.C(:,:,z_ekf_pred(1,t,i))*mu_pred(:,i) + ...
                      par.G(:,:,z_ekf_pred(1,t,i))*u(:,t);
  end;
  % Evaluate importance weights.
  for i=1:N,
    w(t,i) = (det(S(:,:,i))^(-0.5))*  ...
             exp(-0.5*(y(:,t)-y_pred(:,t,i))'*pinv(S(:,:,i))*(y(:,t)- ...
                                    y_pred(:,t,i))) + 1e-99;
  end;
%  w(t,:) = exp(log_w(t,:))+ 1e-99*ones(size(w(t,:)));
  w(t,:) = w(t,:)./sum(w(t,:));       % Normalise the weights.

   for i=1:N,
    % Kalman update:
    K = Sigma_pred(:,:,i)*par.C(:,:,z_ekf(1,t,i))'*pinv(S(:,:,i));
    mu(:,t,i) = mu_pred(:,i) + K*(y(:,t)-y_pred(:,t,i));
    Sigma(:,:,i) = Sigma_pred(:,:,i) -
K*par.C(:,:,z_ekf(1,t,i))*Sigma_pred(:,:,i);
  end;
  xest(:,t) = mean(squeeze(mu(:,t,:)), 2); % KPM - unweighted mean
  zest(:,t) = normalize(hist(squeeze(z_ekf(1,t,:)), 1:n_z)); % KPM
end;   % End of t loop.
```

```
zsamples = squeeze(z_ekf); % (1,t,:) -> (t,:)
end

function [styles, colors, symbols, str] =  plotColors()
% Use plot(x,y,str{i}) to print in i'th style
% Use colors and linestyles, not markers

colors =  ['b' 'r' 'k' 'g' 'c' 'y' 'm' ...
        'r' 'b' 'k' 'g' 'c' 'y' 'm'];
symbols = ['o' 'x' '*' '>' '<' '^' 'v' ...
        '+' 'p' 'h' 's' 'd' 'o' 'x'];
styles = {'-', ':', '-.', '--', '-', ':', '-.', '--', ...
  '-', ':', '-.', '--', '-', ':', '-.', '--'};

for i=1:length(colors)
  str{i} = sprintf('-%s%s', colors(i), symbols(i));
  %str{i} = sprintf('%s%s', colors(i), styles{i});
end

end

function outIndex = deterministicR(inIndex,q)
if nargin < 2, error('Not enough input arguments.'); end

[S,arb] = size(q);  % S = Number of particles.

% Residual Resampling:
N_off= zeros(1,S);
u=zeros(1,S);


cumDist = cumsum(q');
aux=rand(1);
u=aux:1:(S-1+aux);
u=u./S;
j=1;
for i=1:S
   while (u(1,i)>cumDist(1,j))
      j=j+1;
   end
   N_off(1,j)=N_off(1,j)+1;
end
```

```
%figure;
%plot(cumDist,'-b'); hold on;
%plot(u, '-r');

%figure;
%bar(N_off);


index=1;
for i=1:S
  if (N_off(1,i)>0)
    for j=index:index+N_off(1,i)-1
      outIndex(j) = inIndex(i);
    end
  end
  index= index+N_off(1,i);
end
end

function outIndex = multinomialR(inIndex,q)
if nargin < 2, error('Not enough input arguments.'); end

[S,arb] = size(q);  % S = Number of particles.

% Multinomial Sampling:
N_off= zeros(1,S);
cumDist= cumsum(q');
% generate S ordered random variables uniformly distributed in [0,1]

u = fliplr(cumprod(rand(1,S).^(1./(S:-1:1))));
j=1;
for i=1:S
  while (u(1,i)>cumDist(1,j))
    j=j+1;
  end
  N_off(1,j)=N_off(1,j)+1;
end

index=1;
for i=1:S
  if (N_off(1,i)>0)
    for j=index:index+N_off(1,i)-1
      outIndex(j) = inIndex(i);
```

```
    end
  end
  index= index+N_off(1,i);
end
end
function outIndex = residualR(inIndex,q)
    if nargin < 2, error('Not enough input arguments.'); end

[S,arb] = size(q);  % S = Number of particles.

% vanilla re-sample:

N_off= zeros(1,S);
% first integer part
q_res = S.*q'; %'
N_off = fix(q_res);
% residual number of particles to sample
N_res=S-sum(N_off);
if (N_res~=0)
  q_res=(q_res-N_off)/N_res;
  cumDist= cumsum(q_res);
  % generate N_res ordered random variables uniformly distributed in
[0,1]
  u = fliplr(cumprod(rand(1,N_res).^(1./(N_res:-1:1))));
  j=1;
  for i=1:N_res
    while (u(1,i)>cumDist(1,j))
      j=j+1;
    end
    N_off(1,j)=N_off(1,j)+1;
  end
end

%copy traj
index=1;
for i=1:S
  if (N_off(1,i)>0)
    for j=index:index+N_off(1,i)-1
      outIndex(j) = inIndex(i);
    end
  end
  index= index+N_off(1,i);
end
```

end

(Particle filter has been implemented close to Dunham which is then extended to rbpf and ekf)