import os

os.chdir(r'D:\Users\Fujiilab\Desktop\part1\part1')

from pylab import \*

from numpy import \*

def get\_measurement(count):

lines=file\_y.readlines()

y=[]

H=[]

Measurement Model:

To get

1. y\_measure

(measurement from the GPS)

1. H

(projected matrix)

times=60/minute

for i in range(0,len(lines)):

linearray=lines[i].split(',')

if (int(linearray[1])/100>=(count/times\*100+700)+count\*minute%times):

if (int(linearray[1])/100>(count/times\*100+700)+(count+1)\*minute%times):

break

y.append(int(linearray[2]))

Hcount =[0,0]

Hcount[int(linearray[0])-1]=1

H.append(Hcount)

return y, H

class Model():

Give initial parameter to the space state model:

1, initial speed mean and variance

2. maximum speed, density critical density and speed

Backward wave speed

def \_\_init\_\_(self,Q,R):

self.Q = Q

self.R = R

self.vmax =100.0

self.rhomax=11.0

self.rhoc=0.225\*self.rhomax

self.wf=0.225\*self.vmax

self.vc = self.vmax \* (1-self.rhoc/self.rhomax)

Give initial distribution

Normal (50, sqrt(10))

def pi\_0(self):# prior on x

return normal(50,sqrt(10))

def get\_rho(self,vt):

if vt>=self.vc: rho=self.rhomax\*(1-vt/self.vmax)

else: rho=self.rhomax\*(1/((1+vt/self.wf)))

return rho

def get\_flux(self,v1,v2):

if v2>=v1:

if self.vc>=v2: flux=v2\*self.rhomax\*(1/(1+v2/self.wf))

return flux

else:

if self.vc>=v1: v=self.vc

else: v=v1

Space State Model:

X[t+1]=F(X[t]).

X is velocity vector

In this process,

we need calculate density : get\_rho

flux: get\_flux

velocity: get\_velocity

flux=v\*self.rhomax\*(1-v/self.vmax)

else: flux=min(self.get\_rho(v1)\*v1,self.get\_rho(v2)\*v2)

return flux

def get\_velocity(self,rho):

if rho<=self.rhoc: v=self.vmax\*(1-rho/self.rhomax)

else: v=self.wf\*(self.rhomax/rho-1)

return v

def f(self,x,new\_x=None):

mean=[]

deltaT=minute/60

length=[433.0,453.0]

for i in range(2):

rho=self.get\_rho(x[i])

if i==0: deltaflux=self.get\_flux(x[i],x[i+1])-self.get\_flux(self.pi\_0(),x[i])

else: deltaflux=self.get\_flux(x[i],self.pi\_0())-self.get\_flux(x[i-1],x[i])

mean.append(self.get\_velocity(rho-deltaT/(length[i]/1000)\*deltaflux))

return [mean[0]+normal(0,sqrt(self.Q)),mean[1]+normal(0,sqrt(self.Q))]

def g(self,x,y,H,N):

leng=len(y)

Get y\_measure and projected matrix H

Ht=matrix(H).reshape(leng,2)

Y\_measure=matrix(y).reshape(leng,1)

print '1'

delta=[]

tr\_delta=[]

qweight=[]

for xi in x:

Get estimated Y(t) according to the state X(t)

print xi

xt=matrix(xi).reshape(2,1)

yt = Ht\*xt

delta\_y=y\_measure-yt

tr\_delta\_y =errori.reshape(1,-1)

Get the difference Delta Y between y\_measure and Y(t)

delta.append(delta\_y)

tr\_delta.append(tr\_delta\_y)

for i in range(N):

print delta[i].shape, tr\_delta[i].shape

if i==0:

mul=(delta[i]\*tr\_delta[i])

else:

Calculate Variance of Delta Y

mul=mul+(delta[i]\*tr\_delta[i])

print mul

Omega=(1.0/(N-1))\*mul

print 'Omega',linalg.det(Omega)

Omegainv=linalg.inv(Omega)

for i in range(N):

Calculate and return

weight

qweight.append(1/sqrt(linalg.det(Omega\*(pi\*\*N))) \* exp(-0.5\*tr\_delta[i]\*Omegainv\*delta[i]))

print 'weightchange', qweight

return qweight #calculate weight

class Bootstrap():

def \_\_init\_\_(self,model,N):

self.model = model

self.q = model.f

self.N = N

def resample(self,x,w):

N = len(w)

Ninv = 1 / float(N)

new\_x = empty(N)

c = cumsum(w)

Use the new weight and particle to estimate real state X(t)

u = rand()\*Ninv

i = 0

for j in range(N):

uj = u + Ninv\*j

while uj > c[i]:

i += 1

new\_x[j] = x[i]

new\_w = ones(self.N,dtype=float)/self.N

return new\_x, new\_w

def filtering(self):

T = 180/minute

t = 0

xhat = []

# distributions

f, pi\_0, g = self.q, self.model.pi\_0, self.model.g

# initial state

x0 =[pi\_0(),pi\_0()]

#innitial particles

Initial particle and equal weight

x\_part = [f(x0) for i in range(self.N)]

w=ones(self.N,float)/self.N

for t in range(1,T):

# importance sampling

y,H=get\_measurement(t)

print 'x\_part',x\_part, 't' ,t

Filtering process

Go to the function g(x,y,N) to calculate new weight for each particle

x\_part = [f(xi) for xi in x\_part]

w = w\*[g(x\_part,y,H,self.N)]

print w,sum(w)

w =w / sum(w) # normalise

# # selection

x\_part,w = self.resample(x\_part,w)

xhat.append(sum(array(x\_part)\*w))

return xhat

if \_\_name\_\_ == "\_\_main\_\_":

file\_y=open('y\_measurement.csv','r')

model = Model(Q=10,R=1)

global minute

minute=2

T = 180/minute

N = 10

xhat=[]

bs = Bootstrap(model,N)

xhat=bs.filtering()

plot(xhat,label='estimated state')

legend()

xlabel('t')

show()

file\_y.close()