function HW07SInf(A,B,Q,R,SBC,T,guess)

Rinv = R^-1;

interval = linspace(0,T,10\*T);

solinit = bvpinit(interval,guess);

options = bvpset('RelTol', 1e-5);

SinfSol = bvp4c(@(x,s)SOdeFunc(x,s,A,B,Q,Rinv),...

@(ya,yb)SBCFunc(ya,yb,SBC), solinit,options);

%Get Sinf and define Kinf

Sinf = SinfSol.y(:,1);

Sinf = [Sinf(1:2),Sinf(3:4)];

Kinf = Rinv\*B'\*Sinf;

%Get S(t) for K(t), T=1, 10

T1 = 10;

Tinterval = linspace(0,T1,100\*T1);

solinit = bvpinit(Tinterval,guess);

SSol = bvp4c(@(x,s)SOdeFunc(x,s,A,B,Q,Rinv),...

@(ya,yb)SBCFunc(ya,yb,SBC), solinit,options);

Kt = deval(Tinterval,SSol);

%Get Trajectories for the two systems; this will allow for cost calculation and

solinit = bvpinit(Tinterval,[2,4]');

xSolKinf = bvp4c(@(t,y)KinfOdeFunc(t,y,A,B,Kinf),...

@(ya,yb)TrajBC(ya,yb), solinit,options);

xSolKvar = bvp4c(@(t,y)KvarOdeFunc(t,y,A,B,Rinv,SSol),...

@(ya,yb)TrajBC(ya,yb), solinit,options);

%Calculate cost of solution.

Kt = [Kt(1,:)+Kt(3,:); Kt(2,:)+Kt(4,:)] / 2;

xinf = deval(Tinterval,xSolKinf);

xt = deval(Tinterval,xSolKvar);

CostInf = 0;

CostT = 0;

parts = length(Tinterval);

for ind = 1:length(Tinterval)

CostT = CostT + xt(:,ind)'\*(Q + 2\*Kt(:,ind)\*Kt(:,ind)')\*xt(:,ind)\*(T1/parts);

CostInf = CostInf + xinf(:,ind)'\*(Q + 2\*Kinf(:)\*Kinf(:)')\*xinf(:,ind)\*(T1/parts);

end

CostT = (CostT + xt(:,end)'\*SBC\*xt(:,end)) / 2;

CostInf = (CostInf + xinf(:,end)'\*SBC\*xinf(:,end)) / 2;

%Functions for finding Sinf from given S(T)

%Update S

function ds = SOdeFunc(~,s,A,B,Q,Rinv)

n = length(A);

S = zeros(n);

%Wrap vector s into matrix S

for index = 1:n

S(:,index) = s((index-1)\*n+1:index\*n);

end

dS = -A'\*S - S\*A - Q + S\*B\*Rinv\*B'\*S;

ds = zeros(size(s));

%Unwrap matrix dS into vector ds

for index = 1:n

ds((index-1)\*n+1:index\*n) = dS(:,index);

end

%Define BC, i.e. S(T), for Sinf

function res = SBCFunc(~, yb,SBC)

n = length(SBC);

sBC = zeros(n^2,1);

for index = 1:n

sBC((index-1)\*n+1:index\*n,1) = SBC(:,index);

end

res = yb(:) - sBC;

%Function for otp system solving

function dy = KvarOdeFunc(t,y,A,B,Rinv,SSol)

y = y(:);

S = deval(SSol, t);

S = [S(1:2),S(3:4)];

dy = (A - B\*Rinv\*B'\*S)\*y;

dy = dy(:);

%Function for Kinf system

function dy = KinfOdeFunc(~,y,A,B,Kinf)

y = y(:);

dy = (A - B\*Kinf)\*y;

dy = dy(:);

%Boundary condition for traj; same for both systems

function res = TrajBC(ya,~)

res = ya - [1;-1];