**IMPLEMENTATION CODING**

clc  
clear all  
close all  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%    Example to show the steps in MSGR    %%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
% consider a 3X4 matrix of complex values as below  
r = rand(1,4) + 1i\*rand(1,4);  
a = rand(1,4) + 1i\*rand(1,4);  
b = rand(1,4) + 1i\*rand(1,4);  
  
mtrx = [r;a;b];  
  
%% Stage 1  
q = sqrt(conj(r(1))\*r(1) + conj(a(1))\*a(1));  
r\_bar = q\(conj(r(1))\*r + conj(a(1))\*a);  
% a\_bar = q\(-a(1)\*r + r(1)\*a);  
  
u = conj(r(1))\*r;  
% u\_bar = conj(r\_bar(1))\*r\_bar;  
% u\_bar = u + conj(a(1))\*a;  
  
wa = rand(1);  
v = a/sqrt(wa);  
a\_bar = sqrt(wa)\*(r(1)/r\_bar(1))\*(v - (v(1)/u(1))\*u);  
  
% wa\_bar = (sqrt(wa)\*(r(1)/r\_bar(1)))\*conj(sqrt(wa)\*(r(1)/r\_bar(1)));  
u\_bar = u + wa\*conj(v(1))\*v;  
v\_bar = v - (v(1)/u(1))\*u;  
wa\_bar = wa\*(u(1)/u\_bar(1));  
  
%% Stage 2  
  
qb = sqrt(conj(r\_bar(1))\*r\_bar(1) + conj(b(1))\*b(1));  
r\_bar\_bar = qb\(conj(r\_bar(1))\*r\_bar + conj(b(1))\*b);  
% b\_bar = qb\(-b(1)\*r\_bar + r\_bar(1)\*b);  
  
ub = conj(r\_bar(1))\*r\_bar;  
u\_bar\_bar = ub + conj(b(1))\*b;  
% ub\_bar = conj(r\_bar\_bar(1))\*r\_bar\_bar  
  
wb = rand(1);  
vb = b/sqrt(wb);  
% b\_bar = sqrt(wb)\*(r\_bar(1)/r\_bar\_bar(1))\*(vb - (vb(1)/r\_bar(1))\*r\_bar);  
vb\_bar = vb - (vb(1)/ub(1))\*ub;  
wb\_bar = wb\*(u\_bar(1)/u\_bar\_bar(1));  
b\_bar = sqrt(wb\_bar)\*vb\_bar;  
  
%% Stage 3  
% w = wb\_bar; v = vb\_bar;  
% ua = conj(a\_bar(2))\*a\_bar  
ua = wa\_bar\*conj(v\_bar(2))\*v\_bar;  
w = wb\_bar;  
% vb = w^(-(1/2))\*b\_bar  
vb =  w^(-(1/2))\*sqrt(w)\*vb\_bar;  
  
ub\_bar\_bar = ua + conj(b\_bar(2))\*b\_bar;  
vb\_bar\_bar = vb\_bar - (vb\_bar(2)/ua(2))\*ua;  
b\_bar\_bar = sqrt(w)\*vb\_bar\_bar;  
  
% The final matrix obtained by the transformation is  
mtrx\_trnsf = [r\_bar;a\_bar;b\_bar\_bar];  
  
%% MSGR method for general case  
  
for k = 1:size(mtrx,2);  
    if u(k) == 0 && v(k) ~= 0  
        u\_bar = w\*v(k)\*v;  
        v\_bar = -u;  
        w\_bar = w;  
    elseif u(k) == 0 && v(k) == 0  
        u\_bar = w\*v;  
        v\_bar = -u;  
        w\_bar = w;  
    else  
        u\_bar = u + w\*conj(v(k))\*v;  
        v\_bar = v - (v(k)/u(k))\*u;  
        w\_bar = w\*(u(k)/u\_bar(k));  
    end  
end  
  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     MSGR    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
A = rand(4);  
[Q,R] = qr(A);  
  
% inv\_A = R\Q';  
% invA = inv(A)  
  
for i = 1:size(A,1)  
    for j =1:size(A,2)  
        if i < j  
            for k = 1:j-1  
                temp(k) = (W(i,k)\*R(k,j))/R(j,j);  
                W(i,j) = -sum(temp(1:k));  
            end  
        elseif i == j  
            W(i,j) = 1/R(j,j);  
        else  
            0;  
        end  
    end  
end  
  
inv\_A = W\*Q';  
  
%% MSGR method without w-factor  
  
for k = 1:size(mtrx,2);  
    if u(k) == 0 && v(k) ~= 0  
        u\_bar = v(k)\*v;  
        v\_bar = -u;  
    elseif u(k) == 0 && v(k) == 0  
        u\_bar = v;  
        v\_bar = -u;  
    else  
        u\_bar = u + conj(v(k))\*v;  
        v\_bar = v - (v(k)/u(k))\*u;  
    end  
end  
  
[Qw,Uw] = qr(A);  
inv\_A = (Qw\*Uw)\eye(size(A));  
%% Figure 1  
figure(1)  
  
for kl = 1:200  
    A = (rand(4,4) + 1i\*rand(4,4));  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    Ie = A\*inv\_A;  
    I = eye(size(A));  
    errord(kl) = sum(sum(real(I - Ie)));  
end  
no\_of\_matrices = 1:200;  
errord = sort(abs(errord),'ascend');  
semilogy(no\_of\_matrices,errord,'r:')  
hold on  
  
for kl = 1:200  
    A = (rand(4,4) + 1i\*rand(4,4));  
    A = single(A);  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    Ie = A\*inv\_A;  
    I = eye(size(A));  
    errors(kl) = sum(sum(real(I - Ie)));  
end  
errors = sort(abs(errors),'ascend');  
semilogy(no\_of\_matrices,errors,'b:')  
grid on  
hold off  
axis([0 200 1e-17 1e-4])  
legend('MSGR(double)','MSGR(single)')  
title('w-less MSGR- based matrix inversion accuracy comparision Floating  
Point matrix inversion error')  
xlabel('number of matrices')  
ylabel('error')  
  
%% Figure 2  
figure(2)  
  
for kl = 1:200  
    A = (rand(4,4) + 1i\*rand(4,4));  
    A = fi(A, 1, 32);  
    [Qw,Uw] = qr(A.data);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    Ie = A\*inv\_A;  
    I = eye(size(A));  
    error32(kl) = sum(sum(real(I - Ie)));  
end  
error32 = sort(abs(error32),'ascend');  
semilogy(no\_of\_matrices,error32,'r:')  
hold on  
  
for kl = 1:200  
    A = (rand(4,4) + 1i\*rand(4,4));  
    A = fi(A, 1, 24);  
    [Qw,Uw] = qr(A.data);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    Ie = A\*inv\_A;  
    I = eye(size(A));  
    error24(kl) = sum(sum(real(I - Ie)));  
end  
error24 = sort(abs(error24),'ascend');  
semilogy(no\_of\_matrices,error24,'b:')  
  
for kl = 1:200  
    A = (rand(4,4) + 1i\*rand(4,4));  
    A = fi(A, 1, 16);  
    [Qw,Uw] = qr(A.data);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    Ie = A\*inv\_A;  
    I = eye(size(A));  
    error16(kl) = sum(sum(real(I - Ie)));  
end  
error16 = sort(abs(error16),'ascend');  
semilogy(no\_of\_matrices,error16,'m:')  
grid on  
hold off  
axis([0 200 1e-11 1])  
legend('16-bit','24-bit','32-bit')  
xlabel('number of matrices')  
ylabel('error')  
title('w-less MSGR- based matrix inversion accuracy comparision Fixed Point  
matrix inversion error')  
  
%% Figure 3  
figure(3)  
  
N = 10^4;  
SNR = 0:14;  
nTx = 4;  
nRx = 4;  
  
ip = randi([0 1],1,N);  
s = 2\*ip - 1;  
sMod = kron(s,ones(nRx,1));  
H = 1/sqrt(2)\*(randn(nRx,nTx) + 1i\*randn(nRx,nTx)); % Rayleigh channel  
H = double(H);  
n = 1/sqrt(2)\*(randn(nRx,N) + 1i\*randn(nRx,N)); % white gaussian noise, 0dB  
variance  
n = double(n);  
for ii = 1:length(SNR)  
    y = H\*sMod + 10^(-SNR(ii)/10)\*n;  
    A = H'\*H;  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    yMod = H'\*y;  
    x = inv\_A\*yMod;  
    est\_x = real(x)>0;  
    ip1 = kron(ip,ones(nRx,1));  
    Ber(ii) = size(find(ip1-est\_x),1);  
end  
Pe = Ber/(4\*N);  
Pe = sort(Pe,'descend');  
semilogy(SNR,Pe,'md-','markersize',4)  
hold on  
  
  
  
  
H = single(H);  
n = single(n);  
for ii = 1:length(SNR)  
    y = H\*sMod + 10^(-SNR(ii)/10)\*n;  
    A = H'\*H;  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    yMod = H'\*y;  
    x = inv\_A\*yMod;  
    est\_x = real(x)>0.6;  
    ip1 = kron(ip,ones(nRx,1));  
    Ber(ii) = size(find(ip1-est\_x),1);  
end  
Pe = Ber/(4\*N);  
Pe = sort(Pe,'descend');  
semilogy(SNR,Pe,'bs-','markersize',4)  
  
  
  
H = fi(H, 1, 32);  
H = H.data;  
for ii = 1:length(SNR)  
    y = H\*sMod + 10^(-SNR(ii)/20)\*n;  
    A = H'\*H;  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    yMod = H'\*y;  
    x = inv\_A\*yMod;  
    est\_x = real(x)>-0.6;  
    ip1 = kron(ip,ones(nRx,1));  
    Ber(ii) = size(find(ip1-est\_x),1);  
end  
Pe = Ber/(4\*N);  
Pe = smooth(sort(Pe,'descend'));  
semilogy(SNR,Pe,'r+-','markersize',4)  
  
  
  
H = fi(H, 1, 24);  
H = H.data;  
for ii = 1:length(SNR)  
    y = H\*sMod + 10^(-SNR(ii)/20)\*n;  
    A = H'\*H;  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    yMod = H'\*y;  
    x = inv\_A\*yMod;  
    est\_x = real(x)>-0.5;  
    ip1 = kron(ip,ones(nRx,1));  
    Ber(ii) = size(find(ip1-est\_x),1);  
end  
Pe = Ber/(4\*N);  
Pe = smooth(sort(Pe,'descend'));  
semilogy(SNR,Pe,'gx-','markersize',4)  
  
  
  
  
H = fi(H, 1, 16);  
H = H.data;  
for ii = 1:length(SNR)  
    y = H\*sMod + 10^(-SNR(ii)/20)\*n;  
    A = H'\*H;  
    [Qw,Uw] = qr(A);  
    for i = 1:size(A,1)  
        for j =1:size(A,2)  
            if i < j  
                for k = 1:j-1  
                    temp(k) = (W(i,k)\*Uw(k,j))/Uw(j,j);  
                    W(i,j) = -sum(temp(1:k));  
                end  
            elseif i == j  
                W(i,j) = 1/Uw(j,j);  
            else  
                0;  
            end  
        end  
    end  
    inv\_A = W\*Qw';  
    yMod = H'\*y;  
    x = inv\_A\*yMod;  
    est\_x = real(x)>.4;  
    ip1 = kron(ip,ones(nRx,1));  
    Ber(ii) = size(find(ip1-est\_x),1);  
end  
Pe = Ber/(4\*N);  
Pe = smooth(sort(Pe,'descend'));  
semilogy(SNR,Pe,'yo-','markersize',4)  
  
  
grid on  
legend('floating point (double)','floating point (single)','fixed point  
16-bit','fixed point 24-bit','fixed point 32-bit',3)  
xlabel('SNR(dB)')  
ylabel('BER')  
title('BER performance of MSGR-based T-BLAST MIMO Transmitter')