```
% clear all;
% clc;

A = -1;
M = 9;
tau = 0.5;
theta1 = pi/6;
theta = linspace(-90,90,361);

m = 1:1:M;
% QPSK1
% tau_s = [0.25,0.5,0.75,0,0.25,0.5,0.75,0,0.25];
% QPSK2
% tau_s = [0.25,0.5,0.75,0,0.25,0.5,0.75,0,0.25]+0.25;
% QPSK3
% tau_s = [0.25,0.5,0.75,0,0.25,0.5,0.75,0,0.25]+0.5;
% QPSK4
tau_s = [0.25,0.5,0.75,0,0.25,0.5,0.75,0,0.25]-0.25;

F0 = A/sqrt(M)*M*tau;

temp = m*sin(theta1)-2*tau_s;
F1 = A/sqrt(M)*tau*sinc(tau)*exp(-1i*pi*tau)*sum(exp(1i*pi*temp));
abs(F1)
angle(F1)/pi*180

temp1 = 0;
for n = 0:1:(M-1)
    temp1 = exp(1i*pi*n*sind(theta))+temp1;
end
F0_theta = A/sqrt(M)*tau*temp1;
figure(1);
plot(theta,20*log10(abs(F0_theta)),'r');
hold on;
figure(2);
plot(theta,180/pi*(angle(F0_theta)),'r');
hold on;

temp2 = 0;
for n = 0:1:(M-1)
    temp2 = exp(1i*pi*(n*sind(theta)-2*tau_s(n+1))) + temp2;
end
F1_theta = A/sqrt(M)*tau*sinc(tau)*exp(-1i*pi*tau)*temp2;
figure(1);
plot(theta,20*log10(abs(F1_theta)),'b');
hold on;
figure(2);
plot(theta,180/pi*(angle(F1_theta)),'b');
hold on;



function
cost=cost_function(A,B,C,M,theta0,theta1,theta2,mag0,mag1,mag2,tau,tau_s)
```

```
%%
F0 = A/sqrt(M)*sum(tau,2);
err0 = abs(F0 - A*mag0);

%%
temp1 = repmat((0:1:(M-1)),size(tau,1),1);

F1 = A/sqrt(M)*sum(tau.*sinc(tau).*exp(1i*pi*((temp1.*sin(theta1))-
(tau+2*tau_s))),2);
err1 = abs(F1-mag1*B);

%%
F2 = A/sqrt(M)*sum(tau.*sinc(2*tau).*exp(1i*pi*((temp1.*sin(theta2))-
2*(tau+2*tau_s))),2);
err2 = abs(F2-mag2*C);

cost = 100*(err0+err1+err2);


function [bpp,bgp,bpv,bgv]=PSO(cost_function_handle,C,D,B,P)

%
PSO(phase_shifter,receiver_option,BER_target_secured,BER_target_unsecured,W_s
ecured,W_unsecured,...
%
space_resolution,k,d,extra_power_target,desired_direction,do_not_care_bandwid
th,number_element)

% 'cost_function_name': input the cost_function file name here, this function
must return a scalar value
% C: the constant variables for the cost_function use;
% D: the dimention of the problem
% B: the boundary of the search-space, it is D*2, the first colume: lower
boundary, the second colume: upper boundary;
% P: PSO options
% P(1): the number of the particles;
% P(2): the number of the iterations;
% P(3): the option for initial position,
%       '0':random within the boundary
% P(4): Initial inertia weight, default = 0.9
% P(5): Final inertia weight, default = 0.4
% P(6): Epoch when inertial weight at final value, default = 2000
% P(7): acceleration const 1 (local best influence), default = 2
% P(8): acceleration const 2 (global best influence), default = 2
% P(9): the boundary condition,
%       '1': saturation at limit
%       '2': wraparound at limit
%       '3': the real wraparound method
%       '4': bounce off limit
%       '5': the real bounce method



% receiver_option=C(1);
% BER_target_secured=C(2);
% BER_target_unsecured=C(3);
```

```matlab
% W_secured=C(4);
% W_unsecured=C(5);
% space_resolution=C(6);
% k=C(7);
% d=C(8);
% extra_power_target=C(9);
% desired_direction=C(10);
% transition_bandwidth=C(11)&C(12);
% number_element=C(13);

%%
% initial position for each particle P(1)*D

if P(3)==0                          % random within the boundary

particle_position=(ones(P(1),1)*(B(:,1))')+rand(P(1),D).*(ones(P(1),1)*(B(:,2
)-B(:,1))');
elseif P(3)==1                      % specified initial positions
    disp('Error');
end

% best position for each particle, equal to the initial position (best
particle position)
bpp=zeros(P(2)+1,P(1),D).*NaN;
bpp(1,:,:)=particle_position;

% best global position is function value
cost=feval(cost_function_handle,C(1),C(2),C(3),C(4),C(5),C(6),C(7),C(8),C(9),
C(10),particle_position(:,1:1:C(4)),particle_position(:,(C(4)+1):1:end));
[cost_min,index]=min(cost);
bgp=zeros(1,P(2)+1,D).*NaN;
bgp(1,1,:)=particle_position(index,:);

% track the best particle value and best global value
bpv=zeros(P(2)+1,P(1)).*NaN;
bgv=zeros(1,P(2)+1).*NaN;

bpv(1,:)=cost;
bgv(1)=cost_min;

%% optimization
%

vel_coeff=zeros(1,P(2))*NaN;

% generate random initial velocity,P(1)*D
vel=2*(rand(P(1),D)-1).*(ones(P(1),1)*(B(:,2)-B(:,1))');

for n=1:1:P(2)

    % generate coefficient of the the velocity
    if n<=P(6)
        vel_coeff(n)=(P(5)-P(4))/(P(6)-1)*(n-1)+P(4);
    else
        vel_coeff(n)=P(5);
```

```
    end

    % update the particles' velocity
    temp_bpp=reshape(bpp(n,:,:),P(1),D);

    vel=vel_coeff(n)*vel+...
        rand(1)*P(7)*(temp_bpp-particle_position)+...
        rand(1)*P(8)*(repmat(reshape(bgp(1,n,:),1,D),P(1),1)-
particle_position);

    % update the particles' position
    particle_position=particle_position+vel;

    % apply the boundary condition:
    % '1' saturation at limit
    % '2' wraparound at limit
    % '3' the real wraparound method
    % '4' the bounce method
    % '5' the real bounce method

    posmaskmin=repmat((B(:,1))',P(1),1);
    posmaskmax=repmat((B(:,2))',P(1),1);
    posmaskrange=repmat(((B(:,2))'-(B(:,1))'),P(1),1);

    minposmask_throwaway = particle_position <= posmaskmin;
    minposmask_keep      = particle_position >  posmaskmin;
    maxposmask_throwaway = particle_position >= posmaskmax;
    maxposmask_keep      = particle_position <  posmaskmax;

    if P(9)==1
        % this is the saturation method
        particle_position = (minposmask_throwaway .* posmaskmin) +
(minposmask_keep .* particle_position);
        particle_position = (maxposmask_throwaway .* posmaskmax) +
(maxposmask_keep .* particle_position);
    elseif P(9)==2
        % this is the wraparound method
        particle_position = (minposmask_throwaway .* posmaskmax) +
(minposmask_keep .* particle_position);
        particle_position = (maxposmask_throwaway .* posmaskmin) +
(maxposmask_keep .* particle_position);
    elseif P(9)==3
        % this is the real wraparound method
        for m=1:1:5
            minposmask_throwaway = particle_position <= posmaskmin;
            minposmask_keep      = particle_position >  posmaskmin;
            maxposmask_throwaway = particle_position >= posmaskmax;
            maxposmask_keep      = particle_position <  posmaskmax;
            particle_position = (minposmask_throwaway .* (particle_position +
posmaskrange)) +...
                                (minposmask_keep .* particle_position);
            particle_position = (maxposmask_throwaway .* (particle_position -
posmaskrange)) +...
                                (maxposmask_keep .* particle_position);
        end
```

```
    elseif P(9)==4
        % this is the bounce method, particles bounce off the boundaries with
-vel
        particle_position = (minposmask_throwaway .* posmaskmin) +
(minposmask_keep .* particle_position);
        particle_position = (maxposmask_throwaway .* posmaskmax) +
(maxposmask_keep .* particle_position);

        vel = (vel.*minposmask_keep) + (-vel.*minposmask_throwaway);
        vel = (vel.*maxposmask_keep) + (-vel.*maxposmask_throwaway);
    elseif P(9)==5
        % this is the real bounce method, particles bounce off the boundaries
with -vel
        for m=1:1:5
            minposmask_throwaway = particle_position <= posmaskmin;
            minposmask_keep      = particle_position >  posmaskmin;
            maxposmask_throwaway = particle_position >= posmaskmax;
            maxposmask_keep      = particle_position <  posmaskmax;
            particle_position = (minposmask_throwaway .* (2*posmaskmin-
particle_position)) +...
                                (minposmask_keep .* particle_position);
            particle_position = (maxposmask_throwaway .* (2*posmaskmax-
particle_position)) +...
                                (maxposmask_keep .* particle_position);
            vel = (vel.*minposmask_keep) + (-vel.*minposmask_throwaway);
            vel = (vel.*maxposmask_keep) + (-vel.*maxposmask_throwaway);
        end
    end

%    update the bpp bpv bgp and bgv

cost=feval(cost_function_handle,C(1),C(2),C(3),C(4),C(5),C(6),C(7),C(8),C(9),
C(10),particle_position(:,1:1:C(4)),particle_position(:,(C(4)+1):1:end));
    bpv(n+1,:)=cost;

    bpp(n+1,:,:)=repmat((bpv(n+1,:)<bpv(n,:))',1,D) .*particle_position + ...

repmat((bpv(n+1,:)>=bpv(n,:))',1,D).*reshape(bpp(n,:,:),P(1),D);


bpv(n+1,:)=(bpv(n+1,:)<bpv(n,:)).*bpv(n+1,:)+(bpv(n+1,:)>=bpv(n,:)).*bpv(n,:)
;

    [min_bpv,index]=min(bpv(n+1,:));

    if min_bpv<bgv(n)
        bgp(1,n+1,:)=bpp(n+1,index,:);
    else
        bgp(1,n+1,:)=bgp(1,n,:);
    end

    bgv(n+1)=min([min(bpv(n+1,:)) bgv(n)]);

    disp(['Iteration = ',num2str(n)]);
```

```
end

subplot(2,2,1);
semilogy(1:1:P(2)+1,bgv);
grid on;

end
```