به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

جداسازی کور منابع (BSS)

تمرین سری هشتم

نام و نام خانوادگی	علیرضا فداکار
شماره دانشجویی	۸۱۰۱۹۹۳۵۳
تاریخ ارسال گزارش	

فهرست گزارش سوالات

سوال ۱: تولید داده‌های مسئله

در این قسمت نحوه تولیدکردن داده‌ها را توضیح می‌دهیم. ابتدا به کمک قطعه کد زیر پارامترهای اصلی مسئله و ماتریس دیکشنری D را تولید می‌کنیم.

```
% defining parameters
M = 3;
N = 6;
T = 1e3;
mu = 1;    % Mutual coherence

while(mu > 0.9)
    % Generating Dictionary matrix
    D = randn(M, N);

    % Normalizing columns of D
    column_norms = vecnorm(D);
    D = D * diag(1./column_norms);

    % obtaining mutual coherence
    A = D'*D;
    idx = eye(N);
    A = A .* (1-idx);
    A = A(~idx);
    mu = max(A);
end
```

در حلقه while ماتریس D را (با ابعاد $M \times N$) ابتدا توسط تابع randn با استفاده از توزیع گوسی استاندارد تولید می‌کنیم و سپس ستونهای آن را نرمالیزه می‌کنیم ، در نهایت معیار mutual coherence را که طبق تعریف برابر است با $\mu = \max_{i \neq j} |d_i^T d_j|$ محاسبه می‌کنیم اگر مقدار بدست بیشتر از 0.9 بود با توجه به شرطی که برای حلقه while در نظر گرفتیم دوباره مراحل تکرار می‌شود تا اینکه این معیار زیر 0.9 شود.

در ادامه به کمک قطعه کد زیر ماتریس منابع با ابعاد $N \times T$ را تولید می‌کنیم. برای اینکار چون sparsity level برابر $N_0 = 1$ است ابتدا به کمک تابع rand یک ماتریس تصادفی با ابعاد $N \times T$ با درایه‌های یکنواخت در بازه $[-5, 5]$ تولید می‌کنیم. سپس یک ماتریس تصادفی با ابعاد $N \times T$ تولید می‌کنیم به طوری که در هر ستون آن دقیقا یک درایه 1 وجود داشته باشد و بقیه درایه‌ها صفر باشند. در

نهایت دو ماتریس بدست آمده را ضرب درایه به درایه می‌کنیم تا ماتریس منابع مطابق صورت سوال تولید شود.

در مرحله بعد ، به کمک کد زیر ماتریس نویز را با ابعاد $M \times T$ و. با توزیع گوسی با میانگین صفر و واریانس 0.01 تولید کرده و در نهایت ماتریس مشاهدات را نیز بدست می‌آوریم.

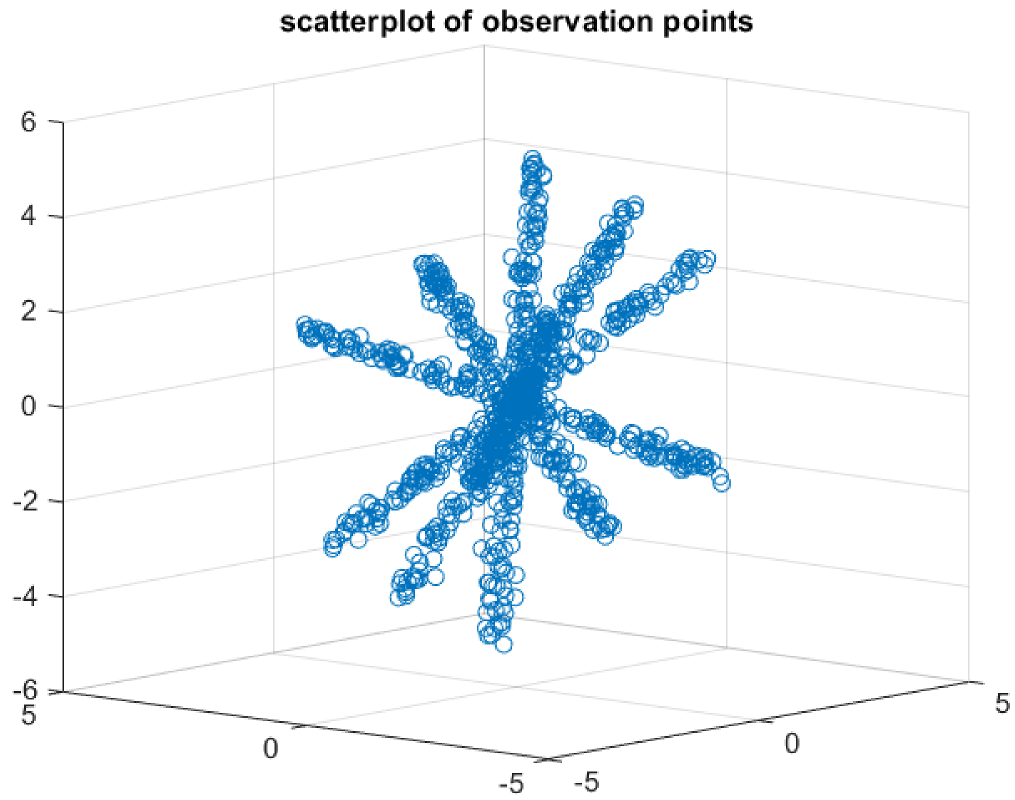
```
%% Generating Noise matrix
Noise_std = 1e-1;
Noise = Noise_std * randn(M, T);
%% Observation Matrix
X = D*S + Noise;
```

سوال ۱- قسمت الف: scatterplot مشاهدات

در این قسمت به کمک تابع scatter3 نمودار scatterplot مشاهدات را رسم می‌کنیم.

```
%% part a
scatter3(X(1,:), X(2,:), X(3,:));
```

نمودار بدست آمده در شکل ۱ قابل مشاهده است:



شکل ۱: نمودار **scatterplot** سه بعدی مشاهدات

همانطور که انتظار می‌رفت چون $N_0 = 1$ و $N = 6$ با توجه به رابطه زیر داده‌ها در نزدیکی یکی از 6 بردار ستونهای D قرار دارند. بنابرین به صورت چشمی می‌توان جهت ستونهای D را مشاهده کرد. در لحظه دلخواه $1 \leq t \leq T$:

$$X^t = DS^t = \sum_{i=1}^{6} d_i S_{it}$$

با توجه به رابطه بالا چون دقیقا یکی از درایه‌های ستون t ام S ناصفر است ستون t ام بردار مشاهدات مضربی (که یکنواخت در بازه $[-5,5]$ است) از یکی ستونهای D است که البته با یک نویز گوسی با واریانس 0.01 نیز جمع می‌شود.

توجه داشته باشید واضح است که ما درباره ترتیب ستونهای D و همچنین جهت آنها ابهام داریم و از روی نمودار نمی‌توان جهت و یا ترتیب آنها را مشخص کرد. همانطور که در ابتدای ویدیوهای درس بحث شد این ابهام جایگشت یا permutation در مسئله BSS اهمتی برای ما ندارد. دلیل آن این است که در مسئله BSS هر ترتیبی که برای ستون‌های D بدست آید همان ترتیب برای سطرهای ماتریس منابع نیز پس از تخمین ، وجود دارد. و اگر ستونی از D در -1 ضرب شود سطر متناظر با آن نیز در ماتریس منابع در -1 ضرب می‌شود. بنابرین این ابهام اهمیتی برای ما ندارد.

سوال ۱- قسمت ب: اعمال روش MOD روی مشاهدات

در این قسمت روش MOD را پیاده سازی کرده و روی ماتریس مشاهدات اعمال می‌کنیم. مطابق درس از روش alternation minimization استفاده می‌کنیم. این روش را به صورت یک تابع در متلب پیاده‌سازی می‌کنیم که در ادامه قابل مشاهده است:

```
function [Dhat, Shat] = MOD(X, N, N0, maxIter, sp_alg, thre)
    [M, ~] = size(X);
    Dhat = randn(M, N);
    Dhat = Dhat./vecnorm(Dhat);
    objDiff = inf;
    objVals = zeros(1, maxIter);
    normX = norm(X(:), 2);
    numIter = 1;

    while(objDiff > thre && numIter < maxIter)
        % Fist we assume D is fixed and known
        switch sp_alg
            case "MP"
                Shat = MP(X, Dhat, N0);
            case "OMP"
                Shat = OMP(X, Dhat, N0);
            otherwise
                disp("Invalid Algorithm!!");
        end

        % Now assume S is fixed and known
        Dhat = X*pinv(Shat);
```

```
        Dhat = Dhat./vecnorm(Dhat);

        curObj = X-Dhat*Shat;
        objVals(numIter) = norm(curObj(:), 2)/normX;
        if numIter > 1
            objDiff = abs(objVals(numIter) -
objVals(numIter-1));
        end
        numIter = numIter + 1;
    end

    fprintf("MOD algorithm finished in %d
iterations!\n", numIter);
    figure
    plt_range = 1:(numIter-1);
    plot(plt_range, objVals(plt_range));
    grid on
    xlabel("Iteration");
    ylabel("objective");
    title("normalized objective value for MOD:
$\frac{||X-\hat{D}\hat{S}||}{||X||}$" , 'interpreter',
'latex');
end
```

مطابق کد بالا فرض می‌کنیم ابتدا ماتریس D معلوم و ثابت است. سپس به ازای هر لحظه $T \geq t \geq 1$ تعداد T مسئله sparse recovery را برای تخمین ستون‌های S حل می‌کنیم. در این پروژه فقط کافیست دو الگوریتم MP و OMP را برای حل مسئله sparse recovery problem پیاده‌سازی کنیم. این دو الگوریتم را به صورت دو تابع جداگانه به صورت زیر پیاده‌سازی می‌کنیم:

```
function Shat = MP(X, D, N0)
    [~, N] = size(D);
    [~, T] = size(X);
    Shat = zeros(N, T);

    % iterating over columns of S
    for t = 1:T
       x = X(:, t);
       for i = 1:N0
           ro = x'*D;
           [~, idx] = max(abs(ro));
           Shat(idx, t) = ro(idx);
           x = x - ro(idx)*D(:, idx);
       end
```

```
    end
end


function Shat = OMP(X, D, N0)
    [~, N] = size(D);
    [~, T] = size(X);
    Shat = zeros(N, T);

    % iterating over columns of S
    for t = 1:T
       x = X(:, t);
       posOMP = zeros(1,N0);
       for i=1:N0
           ro = x'*D;
           [~, posOMP(i)] = max(abs(ro));
           if i>1
               Dsub = D(:,posOMP(1:i));
               Shat(posOMP(1:i), t) = pinv(Dsub)*X(:,
t);
               x = X(:, t)-D*Shat(:, t);
           else
               Shat(posOMP(1), t) = ro(posOMP(1));
               x = x-Shat(posOMP(1), t)*D(:,posOMP(1));
           end
       end
    end
```

در گام دوم الگوریتم فرض می‌کنیم ماتریس منابع Shat معلوم و ثابت باشد و به وسیله دستور pinv (برای بدست آورن ماتریس pseudo inverse) ماتریس Dhat را بروزرسانی کرده و سپس ستون‌های آن را نرمالیزه می‌کنیم.

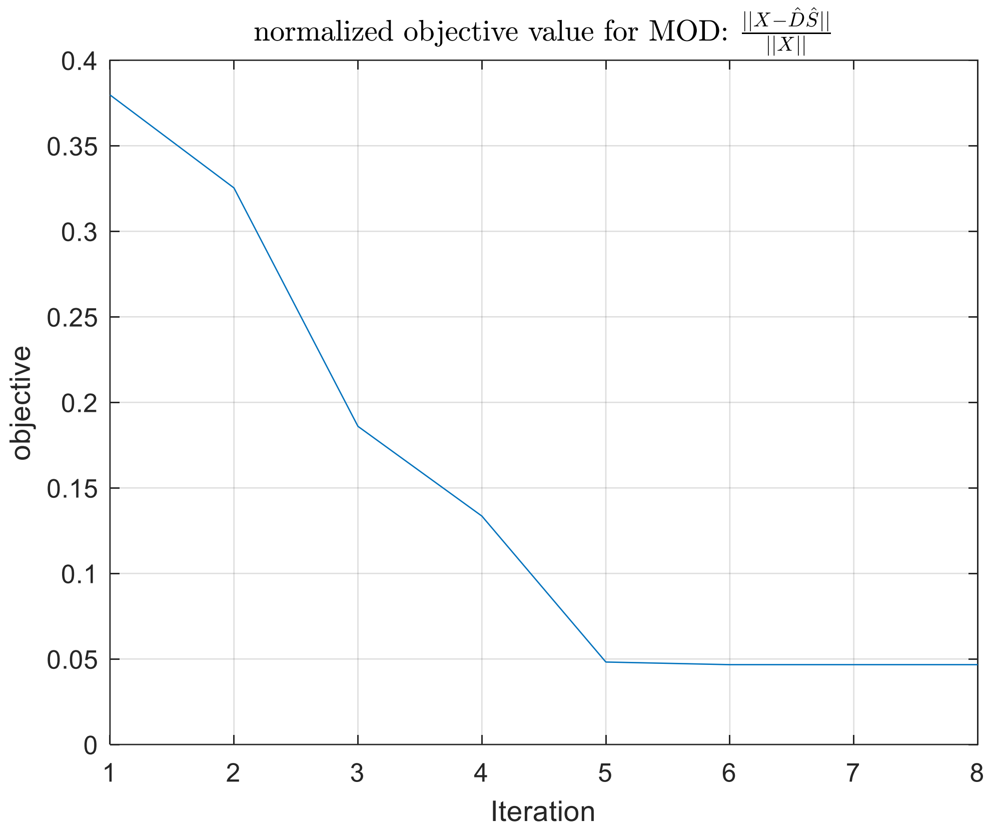
تا الان یک iteration از الگوریتم MOD را انجام داده‌ایم. این الگوریتم را تا جایی تکرار می‌کنیم که شرط خاتمه الگوریتم برقرار شود.

سپس به کمک تابع زیر که نام آن را Evaluation گذاشته ایم successful recovery rate را محاسبه کرده و ابهام ترتیب ستونهای $Dhat$ و سطرهای $Shat$ و همچنین جهت آنها را نیز برطرف می‌کنیم. از این تابع در قسمت‌های بعد نیز استفاده می‌کنیم (به صورت تعمیم یافته این کد را پیاده‌سازی کرده‌ایم).

```
function [suc_re_rate, num_recover, Dcorrect, Scorrect]
= Evaluation(D, Dhat, Shat, thre)
    [~, N] = size(D);
    recover_index = (-1)*ones(1,N);
    correct_perm = (-1)*ones(1,N);
    dir_uncertainty = ones(1, N);

    for i = 1:N
        valid_perm_ind = setdiff(1:N, correct_perm);
        valid_rec_ind = setdiff(1:N, recover_index);

        ro_perm = D(:, i)'*Dhat(:, valid_perm_ind);
        [~, idx_perm] = max(abs(ro_perm));
        correct_perm(i) = valid_perm_ind(idx_perm);

        if ro_perm(idx_perm) < 0
            dir_uncertainty(i) = -1;
        end

        ro_rec = Dhat(:, i)'*D(:, valid_rec_ind);
        [val, idx_rec] = max(abs(ro_rec));
        if val >= thre
            recover_index(i) = valid_rec_ind(idx_rec);
        end
    end

    num_recover = sum(recover_index > 0);
    suc_re_rate = (num_recover/N)*100;
    Dcorrect = Dhat*diag(dir_uncertainty);
    Dcorrect = Dcorrect(:, correct_perm);
    Scorrect = diag(dir_uncertainty)*Shat;
    Scorrect = Scorrect(correct_perm, :);
end
```

به طور خلاصه در کد بالا ابتدا حاصل $d_i^T Dhat$ را بدست آوردیم و در یک حلقه for روی ستونهای D ، به ازای هر ستون ماتریس D ستونی از $Dhat$ که با آن همبستگی بیشتری دارد انتخاب کردیم و به طور همزمان ابهام دامنه و جهت منفی و مثبت نیز رفع کردیم.

پس از اجرای کد نمودار تابع هدف به صورت شکل ۲ خواهد شد:



شکل ۲: نمودار تابع هدف نرمالیزه شده بر حسب **iteration**

مقدار successful recovery نیز به صورت زیر بدست آمد:

```
MOD algorithm finished in 9 iterations!
MOD method:
number of recovered atoms = 6 of 6
Successful recovery rate = 100.00
```

بنابرین تمام 6 ستون D با ترشولد همبستگی بزرگتر از 0.99 ریکاور شدند. ماتریس اصلی D و ماتریس تخمین زده شده $Dhat$ نیز پس از رفع ابهام ترتیب ستونها در شکل‌های زیر مشاهده می‌شود:

1	2	3	4	5	6
-0.6713	-0.8795	0.1257	0.0392	0.0966	-0.5501
-0.2082	0.0114	0.9816	-0.4415	-0.0090	-0.6306
0.7113	-0.4758	0.1440	0.8964	0.9953	0.5474

شکل ۳: ماتریس **D**

1	2	3	4	5	6
-0.6706	-0.8798	-0.1314	-0.0365	0.0927	-0.5488
-0.2086	0.0075	-0.9811	0.4460	-0.0049	-0.6325
0.7119	-0.4753	-0.1418	-0.8943	0.9957	0.5466

شکل ۴: ماتریس ***Dhat*** در روش **MOD**

سوال ۱- قسمت ج: اعمال روش K-SVD روی مشاهدات

در این قسمت ، تمام مراحل قسمت قبل را برای الگوریتم K-SVD تکرار می‌کنیم. گام اول این الگوریتم همانند الگوریتم MOD قسمت الف تعداد T مسئله sparse recovery را برای تخمین ماتریس S با روش MP حل می‌کنیم (با فرض ثابت بودن $Dhat$).

در گام دوم الگوریتم در یک حلقه for ستونهای $Dhat$ و سطرهای $Shat$ متناظر را مطابق درس بروزرسانی می‌کنیم. در مرحله i ام ماتریس X_r^i را به صورت زیر بدست می‌آوریم:

$$X_r^i = X - \sum_{n\neq i} d_n s_n^T$$

سپس سطر s_i^T را در نظر می‌گیریم و به کمک دستور find اندیس درایه‌های ناصفر آن را بدست می‌آوریم و با توجه به اندیس‌های بدست آمده ستونهای متناظر ماتریس X_r^i را در یک ماتریس جدید قرار داده تا ماتریس ${X_r^i}_{modify}$ بدست آید. در نهایت اگر سطر s_i^T تمام صفر نبود (اگر تمام صفر بود این مرحله را ignore می‌کنیم تا در مراحل بعدی درایه‌های ناصفر در این سطر ظاهر شود) ، تجزیه SVD را روی ${X_r^i}_{modify}$ اعمال می‌کنیم:

$${X_r^i}_{modify} = U\Sigma V^T$$

حال بردار d_i و سطر s_i^T را به این صورت بروزرسانی می‌کنیم:

$$\begin{cases} d_1 = U(:,1) \\ s_i^T = \Sigma(1,1)V(1,:) \end{cases}$$

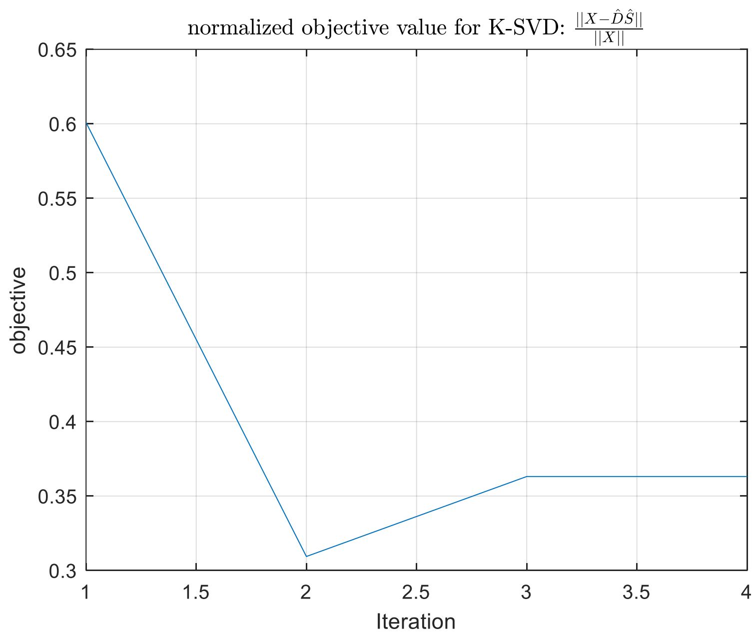
تمام مراحل مذکور را به صورت تابع زیر در متلب پیاده سازی می‌کنیم:

```
function [Dhat, Shat] = K_SVD(X, N, N0, maxIter, sp_alg, thre)
    [M, ~] = size(X);
    Dhat = randn(M, N);
    Dhat = Dhat./vecnorm(Dhat);
    objDiff = inf;
    objVals = zeros(1, maxIter);
```

```
    normX = norm(X(:), 2);
    numIter = 1;

    while(objDiff > thre && numIter < maxIter)
        % Fist we assume D is fixed and known
        switch sp_alg
            case "MP"
                Shat = MP(X, Dhat, N0);
            case "OMP"
                Shat = OMP(X, Dhat, N0);
            otherwise
                disp("Invalid Algorithm!!");
                Dhat = nan;
                Shat = nan;
        end

        % Now assume S is fixed and known
        R = X - Dhat*Shat;
        for i = 1:N
            Ri = R + Dhat(:, i)*Shat(i, :);
            if sum(Shat(i,:) ~= 0) > 0
                nonzero_index = find(Shat(i,:));
                Ri_modify = Ri(:, nonzero_index);

                % perform svd
                [U, Sigma, V] = svd(Ri_modify);

                % Update
                Dhat(:, i) = U(:, 1);
                Shat(i, nonzero_index) =
Sigma(1,1)*V(1,:);
            end
        end
        curObj = X-Dhat*Shat;
        objVals(numIter) = norm(curObj(:), 2)/normX;
        if numIter > 1
            objDiff = abs(objVals(numIter) -
objVals(numIter-1));
        end
        numIter = numIter + 1;
    end

    fprintf("K-SVD algorithm finished in %d
iterations!\n", numIter);
    figure
    plt_range = 1:(numIter-1);
```

```
    plot(plt_range, objVals(plt_range));
    grid on
    xlabel("Iteration");
    ylabel("objective");
    title("normalized objective value for K-SVD:
$\frac{||X-\hat{D}\hat{S}||}{||X||}$" , 'interpreter',
'latex');
end
```

پس از اجرای کد ، نمودار تابع هدف نرمالیزه شده و همچنین مقدار successful recovery rate به صورت زیر بدست آمد:



شکل ۵: نمودار تابع هدف نرمالیزه شده بر حسب **iteration**

```
K-SVD algorithm finished in 5 iterations!
K-SVD method:
number of recovered atoms = 6 of 6
Successful recovery rate = 100.00
```

بنابرین در این روش نیز تمام 6 ستون از D با ترشولد همبستگی بزرگتر از 0.99 به درستی ریکاور شدند. در نهایت ماتریس تخمین زده شده $Dhat$ در شکل ۶ مشاهده می‌شود (ماتریس دقیق D در شکل ۳ قابل مشاهده است):

	1	2	3	4	5	6
1	0.6706	-0.8798	0.1314	0.0365	-0.0927	-0.5488
2	0.2086	0.0075	0.9811	-0.4460	0.0049	-0.6325
3	-0.7119	-0.4753	0.1418	0.8943	-0.9957	0.5466

شکل ۶: ماتریس $\boldsymbol{Dhat}$ تخمین زده شده با روش **K-SVD**

سوال ۱- قسمت د: محاسبه E_mod و E_ksvd

توجه داشته باشید در قسمت‌های قبل ابهام ترتیب ستونهای $Dhat$ و سطرهای $Shat$ را برطرف کردیم.

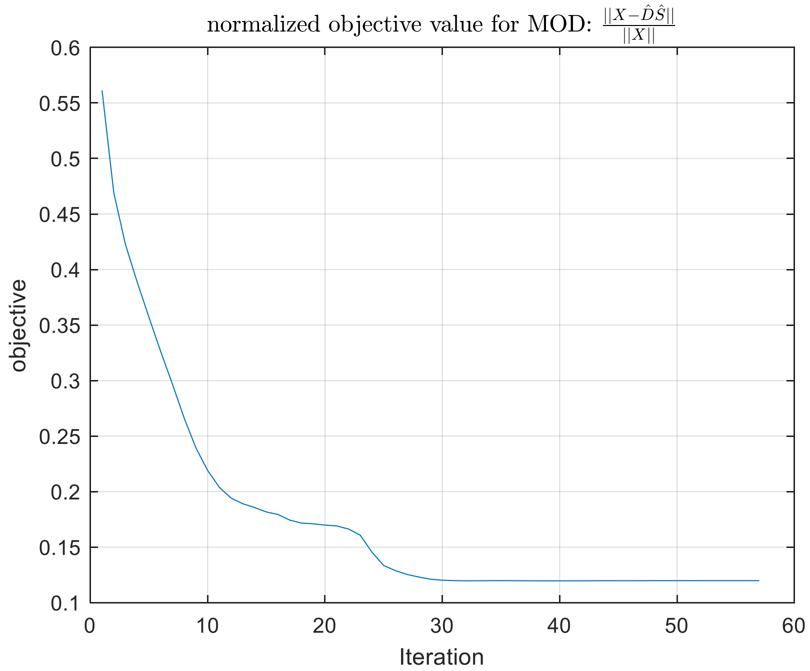
مقادیر E_MOD و E_KSVD به صورت زیر بدست می‌آید:

```
E_MOD:
     0.0024

E_KSVD:
     1.3614
```

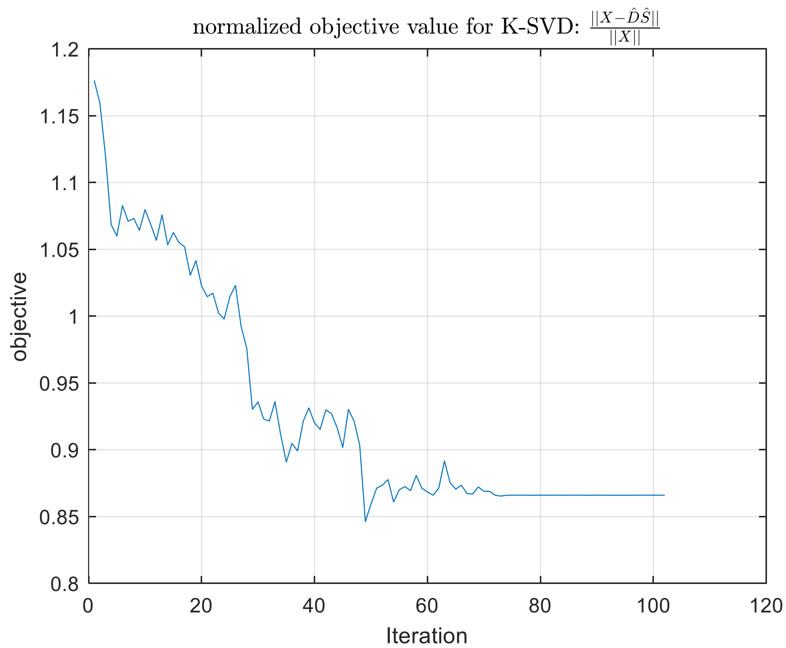
سوال ۲-

در این سوال الگوریتم‌های MOD و K-VSD را روی داده‌های hw8.mat اعمال می‌کنیم. اکثر کد مانند سوال اول است با این تفاوت که در گام اول هر دو روش از OMP برای بروزرسانی S استفاده می‌کنیم.

پس از اجرای کد نمودار objective value به صورت دو شکل ۷ و ۸ خواهد شد.



شکل ۷: روش **MOD**



شکل ۸: روش **K-SVD**

مقادیر successful recovery rate و تعداد iteration هایی که دو الگوریتم به پایان رسیدند در شکل۹ مشاهده می‌شوند:

```
MOD algorithm finished in 58 iterations!
MOD method:
number of recovered atoms = 46 of 50
Successful recovery rate = 92.00

K-SVD algorithm finished in 103 iterations!
K-SVD method:
number of recovered atoms = 47 of 50
Successful recovery rate = 94.00
```

شکل ۹: نتایج

همانطور که مشاهده دقت دو روش به ترتیب 92% و 94% بدست آمد. البته این نتایج به مقدار دهی اولیه وابسته است و با اجرای کد در دفعات مختلف مقادیر successful recovery rate دو روش معمولا در بازه 90% تا 100% تغییر می‌کند (مقدار ترشولد را برابر 10^{-11} در نظر گرفته‌ایم). در نتیجه از لحاظ successful recovery rate هر دو الگوریتم تقریبا عملکرد یکسان و مناسب دارند.

از لحاظ سرعت همگرایی الگوریتم ksvd کندتر از MOD است و اجرای کد آن به خاطر گام دوم آن زمان بیشتری نسبت به MOD نیاز دارد.