به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

BSS

گزارش تمرین ۶

سالار صفردوست
۸۱۰۱۹۹۴۵۰
۱۴۰۲/۰۲/۲۷

بخش اوّل

الف

Variables - s

15x1 double

	1
1	0
2	1.0000
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	-2.0000
11	0
12	0
13	0
14	0
15	0

جواب به دست آمده صحیح است، چرا که به ازای $D * s$ به بردار مشاهدات می‌رسیم.

ب

```
%% Question 2

N0 = 2;
s = MP(D,x,N0);
```

Variables - s

15x1 double

	1
1	0
2	0
3	0
4	-0.3351
5	0
6	0
7	0
8	0
9	0
10	-2.8839
11	0
12	0
13	0
14	0
15	0

خیر، این جواب، جواب درستی برای مسئله نیست چرا که به ازای $D * s$ به بردار مشاهدات نمی‌رسیم.

ج

Variables - s

15x1 double

	1
1	0
2	0.9013
3	0.1304
4	0
5	-0.2066
6	0
7	0
8	0
9	0
10	-1.9396
11	0.0148
12	0
13	0
14	0
15	0

همانطور که دیده می‌شود مقداری که به دست آمده با مقدار صحیح ناشی از حل بدون نویز تفاوت دارد. (با وجود اینکه Ds برابر x_{noisy} می‌باشد.)

د

```
function s = LASSO(D,x,iterations,lambda)

    [~,N] = size(D);
    s = rand(N,1);
    N_list = 1:N;

    for i = 1:iterations
        for n = N_list
            rho = (x - D(:,N_list~=n) * s(N_list~=n)).' * D(:,n);

            if rho>lambda/2
                s(n) = rho - lambda/2;
            elseif rho<-lambda/2
                s(n) = rho + lambda/2;
            else
                s(n) = 0;
            end

        end
    end

end
```

```
26      %% Question 4
27
28 -    lambda = 0.6663;
29 -    iterations = 1000;
30 -    s1 = LASSO(D,x_noisy,iterations,lambda);
31
32 -    lambda = 4.3001;
33 -    iterations = 1000;
34 -    s2 = LASSO(D,x_noisy,iterations,lambda);
35
```

Variables - s1

15x1 double

	1
1	0
2	0.9644
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	-1.7248
11	0
12	0
13	0
14	0
15	0

Variables - s2

15x1 double

	1
1	0
2	8.8339e-06
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	-0.7604
11	0
12	0
13	0
14	0
15	0

دو مقدار s به دست آمده به ازای دو مقدار مرزی لامبدا هستند که اسپارسیتی با $N_0 = 2$ را محقق می‌کردند، علّت این موضوع این است که کاهش لامبدا معادل کاهش اهمیّت به اسپارسیتی و افزایش آن معادل افزایش اهمیّت اسپارسیتی می‌باشد.

نزدیک‌ترین جوابی که به الف ممکن است، در همان مقدار مرزی $\lambda = 0.6663$ به دست می‌آمد.

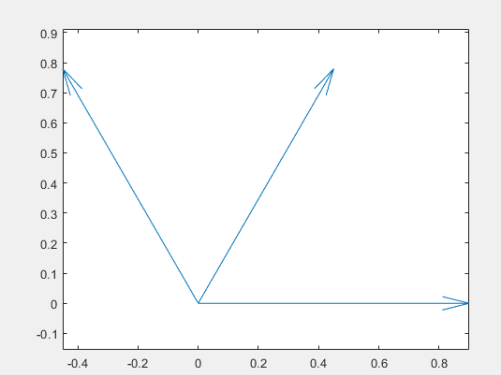
بخش دوم

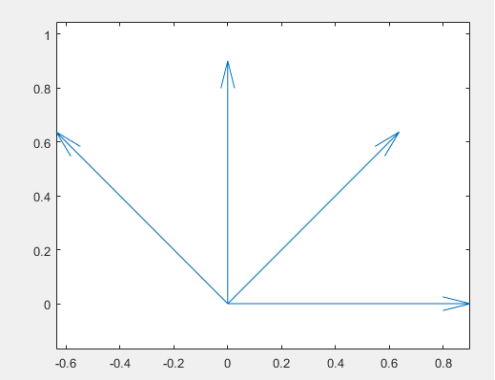
روش مورد اشاره در سوال ۲ برای هر تعداد N و در هر ابعاد دلخواهی توانایی به دست آوردن فریم نسبتاً بهینه را دارد، با این حال برای سوال ۱ از یک حل بهینه که در ۲ بعد نسبتاً راحت به دست می‌آید نیز به عنوان راه حل دوم سوال ۱ قرار داده شده است و نمودار μ بر حسب N نیز برای آن ترسیم شده است.

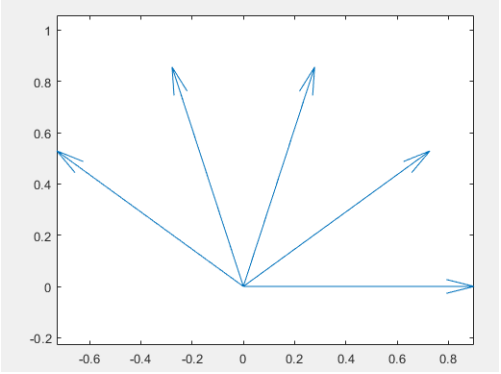
ابتدا راه‌حل دوم نمایش داده می‌شود و سپس به راه‌حل کلی مسئله که هر دو سؤال را حل می‌کند می‌پردازیم.

۱

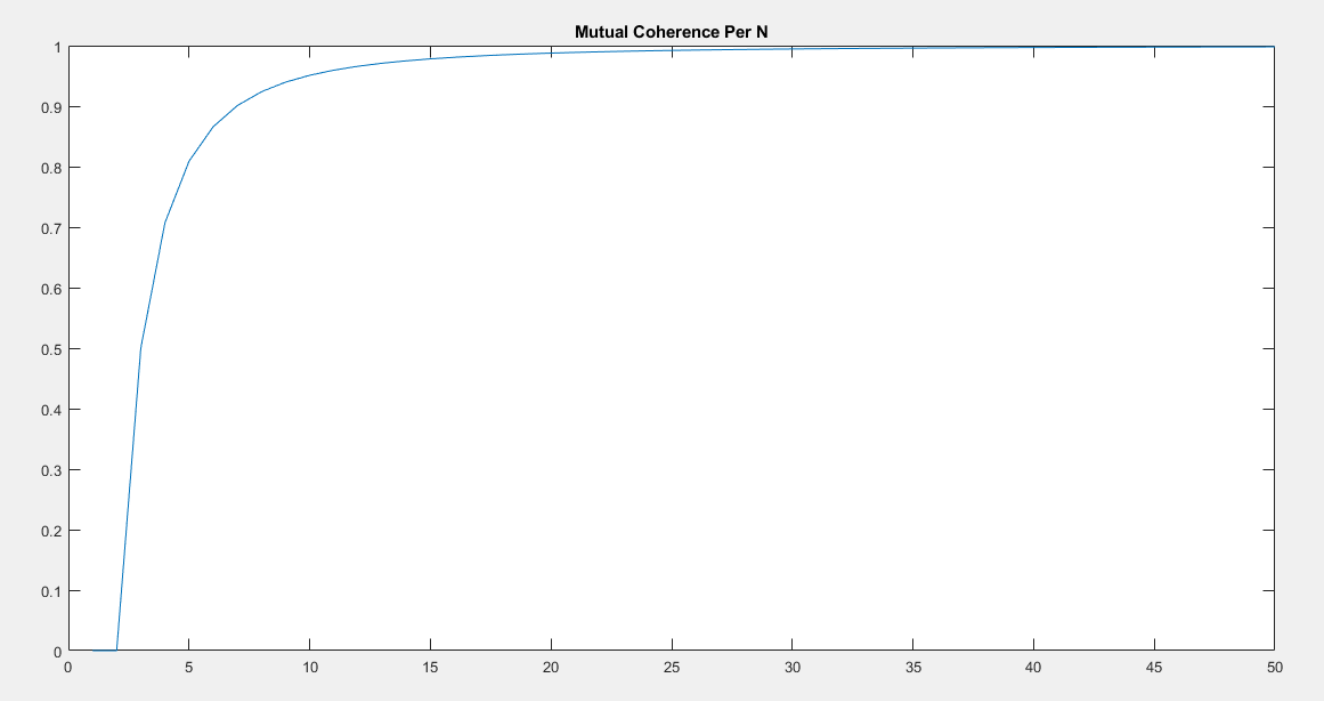
با توجه به اینکه ماکسیمم قدر مطلق ضرب داخلی دو به دوی بردارها نیاز است که مینیمم شود، بهینه‌ترین حالت ممکن برای آن پخش کردن N بردار به صورت یکنواخت در نیم‌دایره‌ای از فضا می‌باشد.







```
%% Second Solution

K = 50;
for k = 1:K
    phi = linspace(0,pi,k+1);
    phi(end) = [];
    D = [cos(phi) ; sin(phi)];
    u(k) = MutCoh(D);
end
plot(u)
title('Mutual Coherence Per N')
```



۲

برای محقق شدن حدودی خواسته‌ی مسئله یک تابع هدف برای هر اتم D فرض شد:

$$\begin{cases} f(d_i) = \sum_{i \neq j} \left(d_i^T d_j\right)^2 = \sum_{i \neq j} (d_i^T d_j)(d_j^T d_i) = d_i^T \left(\sum_{i \neq j} d_j d_j^T \right) d_i = d_i^T D_r d_i \\ s.t.\ d_i^T d_i = 0 \end{cases}$$

حال سعی می‌کنیم با روش $Alternation\ Minimization$ با مینیمم کردن تابع فوق به مینیمم تابع کلی که می‌تواند به صورت جمع تمام این fها تعریف شود برسیم.

دو نکته لازم است که ذکر شود:

۱* تابع هدف تعریف شده لزوماً با تعریف ما از بهینه کردن یک فریم مفهوم یکسانی ندارد. (یا حداقل اثبات نشده است.)

۲* دیده می‌شود که این تابع هدف با روش اشاره شده تعداد بسیاری مینیمم محلی دارد که برای رفع این مشکل از تعداد نسبتاً زیادی D اوّلیه متفاوت استفاده می‌شود تا بهینه‌ترین آن‌ها انتخاب شود.

حال با اضافه کردن شرط نرمال به معادله تابع به روش لاگرانژ خواهیم داشت:

$$g(d_i) = d_i^T D_r d_i + \lambda(1 - d_i^T d_i)$$

با گرادیان گرفتن و مساوی صفر قرار دادن تابع بالا به این نتیجه می‌رسیم که:

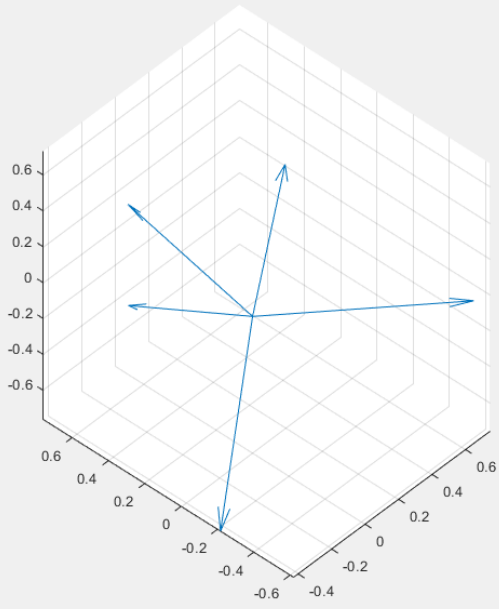
$$[D, \Lambda] = eig(D_r)$$

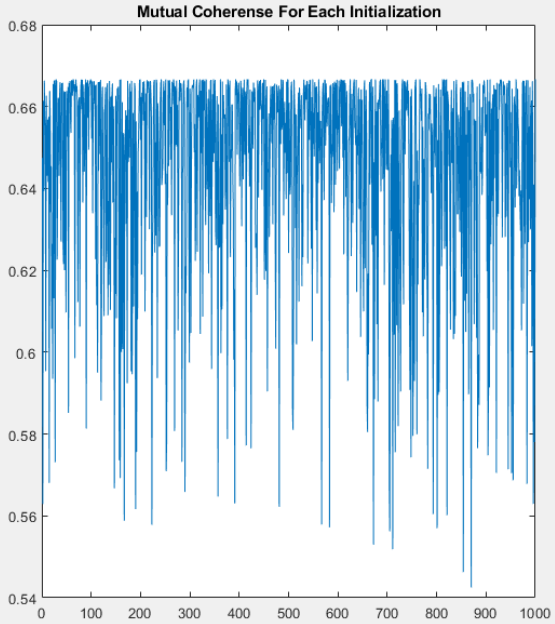
که با انتخاب ستونی از D که مربوط به کمترین درایه‌ی Λ می‌باشد، به مینیمم تابع خواهیم رسید.

```
function [D,u] = FrameDesigner(N,M,iterations)

    K = 1000;
    N_list = 1:N;
    u = zeros(1,K);
    U = zeros(M,N,K);

    for k = 1:K
        D = normc(rand(M,N).*2-1);
        for i = 1:iterations
            for n = N_list
                R = D(:,N_list~=n) * D(:,N_list~=n).';
                [V,L] = eig(R);
                [~,index] = min(diag(L));
                D(:,n) = V(:,index);
            end

        end
        u(k) = MutCoh(D);
        U(:,:,k) = D;
    end

    [~,index] = min(u);
    D = U(:,:,index);

end
```

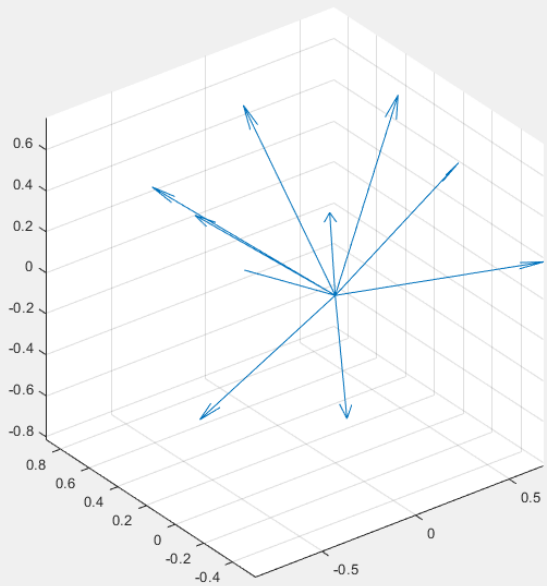
```
%% Question 2

iterations = 40;
[D,u] = FrameDesigner(N,3,iterations);

figure

subplot(1,2,1)
quiver3(zeros(1,N),zeros(1,N),zeros(1,N),D(1,:),D(2,:),D(3,:));
axis equal

subplot(1,2,2)
plot(u);

disp(min(u))

```

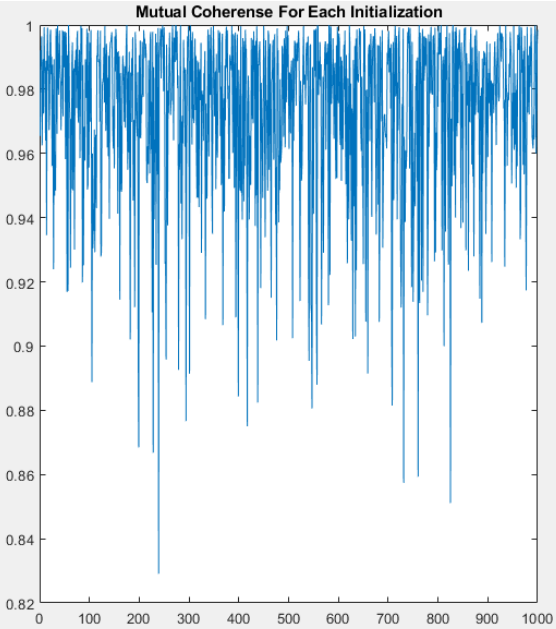
عملکرد این تابع را به ازای $N = 5$ و $N = 10$ مشاهده می‌کنیم.





0.5425





0.8291

مقادیر نوشته شده زیر هر عکس μ هر کدام از فریم‌ها می‌باشد.

بخش سوم

۱

u 0.8680

۲

در هر دو روش می‌توان تعداد مقادیر اوّلیه‌ی داده شده به D را برای فرار از مینیمم محلی به عنوان ورودی به توابع داد، ولی برای مقایسه دو روش هر کدام تنها یکبار مسئله را حل می‌کنند.

```
%% Question 2&4

    %% MOD
iterations = 50;
number_of_initialiations = 1;
[D_hat1,S_hat1,Error1] = MOD(X,size(S,1),iterations,number_of_initialiations);

    %% KSVD

iterations = 50;
number_of_initialiations = 1;
[D_hat2,S_hat2,Error2] = KSVD(X,size(S,1),iterations,number_of_initialiations);

```

MOD:

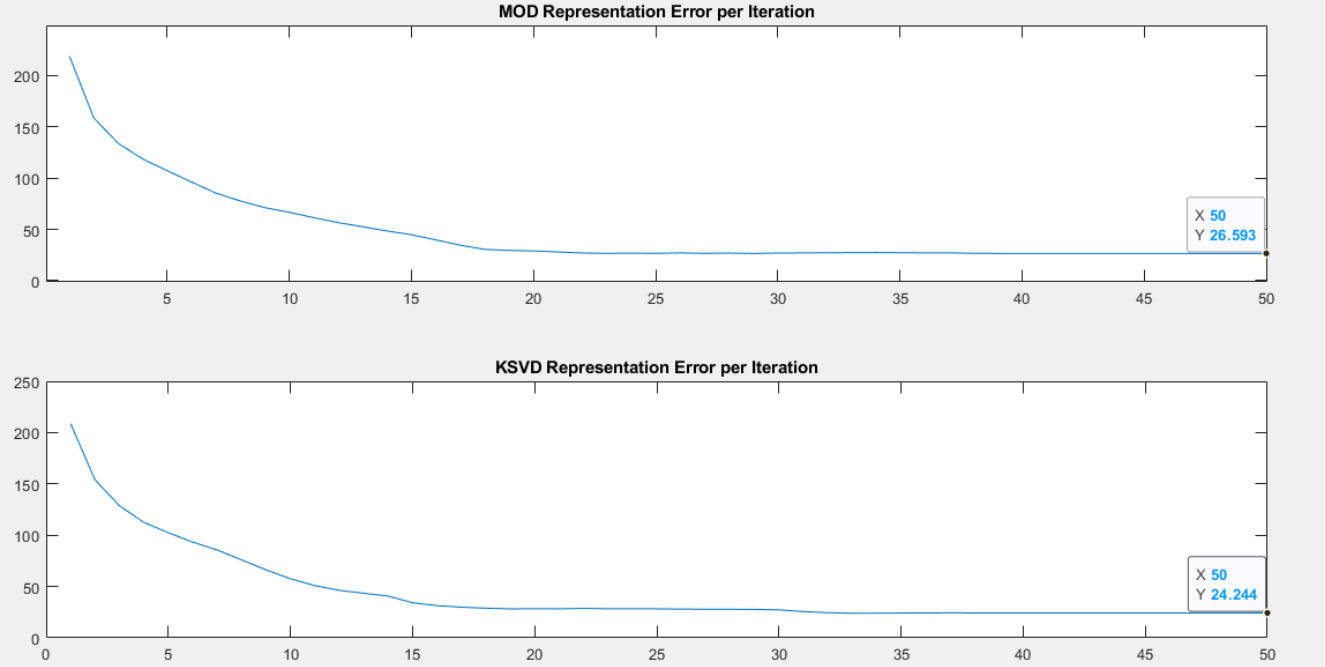
```
function [D,S,Error] = MOD(X,N,iterations,number_of_initialiations)

    tic
    [M,T] = size(X);
    K = number_of_initialiations;
    D = zeros(M,N,K);
    S = zeros(N,T,K);
    Error = zeros(iterations,K);

    for k = 1:K
        D(:,:,k) = normc(rand(M,N)*2-1);
        for i = 1:iterations
            for t = 1:T
                S(:,t,k) = OMP(D(:,:,k),X(:,t),2);
            end
            D(:,:,k) = X*pinv(S(:,:,k));
            D(:,:,k) = normc(D(:,:,k));
            Error(i,k) = (norm(X-D(:,:,k)*S(:,:,k),'fro')).^2;
        end
    end

    [~,index] = min(Error(end,:));

    Error = Error(:,index);
    D = D(:,:,index);
    S = S(:,:,index);
    toc

end
```

KSVD:

```
function [D,S,Error] = KSVD(X,N,iterations,number_of_initialiations)

    tic
    [M,T] = size(X);
    K = number_of_initialiations;
    D = zeros(M,N,K);
    S = zeros(N,T,K);
    Error = zeros(iterations,K);
    N_list = 1:N;

    for k = 1:K
        D(:,:,k) = normc(rand(M,N)*2-1);
        for i = 1:iterations
            for t = 1:T
                S(:,t,k) = OMP(D(:,:,k),X(:,t),2);
            end
            for n = N_list
                Xr = X - D(:,N_list~=n,k) * S(N_list~=n,:,k);
                mXr = Xr(:,S(n,:,k)~=0);
                [U,sig,V] = svd(mXr);
                V = V.';
                [~,index] = max(diag(sig));
                D(:,n,k) = U(:,index);
                S(n,S(n,:,k)~=0,k) = sig(index,index) * V(index,:);
            end
            Error(i,k) = (norm(X-D(:,:,k)*S(:,:,k),'fro')).^2;
        end
    end

    [~,index] = min(Error(end,:));
    Error = Error(:,index);
    D = D(:,:,index);
    S = S(:,:,index);
    toc
```

۳

```
%% Question 3

figure
subplot(2,1,1);
plot(Error1)
title('MOD Representation Error per Iteration')
subplot(2,1,2);
plot(Error2)
title('KSVD Representation Error per Iteration')

```



دیده می‌شود که روش $KSVD$ به مقدار کمتری همگرا شده است.

۴

```
Elapsed time is 2.908963 seconds.
Elapsed time is 3.736534 seconds.
```

اولین عدد برای متد MOD و دومین عدد برای متد $KSVD$ می‌باشد.

۵

```
%% Question 5

srr1 = SRR(D,D_hat1,0.98);
srr2 = SRR(D,D_hat2,0.98);
```

```
function srr = SRR(D,D_hat,T)

    N = size(D,2);
    srr = 0;
    for i = 1:N
        corr_list = abs(D_hat(:,i).'*D);
        [m,j] = max(corr_list);
        if m>T
            srr = srr+1;
            D(:,j) = [];
        end
    end
    srr = srr/N;

end
```

Variables - srr1

1x1 double

	1
1	0.8500

Variables - srr2

1x1 double

	1
1	0.9000

۶

```
%% Question 6

[D_hat1,S_hat1] = Scale_Permutation_Recovery(D,S,S_hat1,D_hat1);
E1 = (norm(S-S_hat1,'fro')^2)/(norm(S,'fro')^2);

[D_hat2,S_hat2] = Scale_Permutation_Recovery(D,S,S_hat2,D_hat2);
E2 = (norm(S-S_hat2,'fro')^2)/(norm(S,'fro')^2);
```

```
function [D_hat,S_hat] = Scale_Permutation_Recovery(D,S,S_hat,D_hat)

    D_temp = D_hat*0;
    S_temp = S_hat*0;

    N = size(D,2);
    for i = 1:N
        corr_list = (D_hat(:,i).'*D);
        [~,j] = max(abs(corr_list));
        if corr_list(j)<0
            D_temp(:,j) = -D_hat(:,i);
            S_temp(j,:) = -S_hat(i,:);
        else
            D_temp(:,j) = D_hat(:,i);
            S_temp(j,:) = S_hat(i,:);
        end
        D(:,j) = 0;
        S(j,:) = 0;
    end

    D_hat = D_temp;
    S_hat = S_temp;

end
```

Variables - E1

1x1 double

	1
1	0.5324

Variables - E2

1x1 double

	1
1	0.4520

* با انجام مقداردهی اوّلیه به تعداد ۲۰ بار به خطاهای کمتر و بهتری نیز می‌رسیم:

Variables - E1

1x1 double

	1
1	0.5055

Variables - E2

1x1 double

	1
1	0.2655