

تمرین شماره 6

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

جداسازی کور منابع

دکتر اخوان

بهار 1402

فهرست مطالب

4	۱-۱- بخش اول.....
12	۱-۲- بخش دوم.....
15	۱-۳- بخش سوم :
16	2-3-1- پیاده سازی MOD.....
19	۴-۱- پیاده سازی K-SVD.....

فهرست اشکال

4	شکل (۱-۱) لود کردن داده ها.....
4	شکل (۲-۱) Set up the linear programming problem.....
4	شکل (۳-۱) حل مسئله برنامه ریزی خطی را با استفاده از تابع 'linprog'.....
4	شکل (۴-۱) کد متلب استخراج بردار S.....
5	شکل Vector S (۵-۱) استخراج شده با روش BP.....
5	شکل (۶-۱) کد متلب پیاده سازی روش MP.....
6	شکل (۷-۱) Vector S استخراج شده با روش MP.....
7	شکل (۸-۱) کد متلب محاسبه BP برای x_noisy.....
8	شکل (۹-۱) sBP_noisy استخراج شده.....
9	شکل (۱۰-۱) کد متلب فرموله کردن lasso و حل به ازای لاندا های مختلف.....
12	شکل (۱۱-۱) فریم 20*2 (لازم به ذکر است مدل های دیگر نیز rotate همین میتواند باشد).....
13	شکل (۱۲-۱) کد متلب الف بخش 2.....
13	شکل (۱۳-۱) منحنی MC بر حسب N.....
14	شکل (۱۴-۱) کد متلب پیاده سازی فریم 10*3.....
15	شکل (۱۵-۱) فریم 10*3.....
16	شکل (۱۶-۱) محاسبه mutual Coherence.....
17	شکل (۱۷-۱) کد پیاده سازی MOD.....
18	شکل (۱۸-۱) منحنی همگرایی MOD.....
18	شکل (۱۹-۱) کد پیدا کردن successful recovery rate.....
19	شکل (۲۰-۱) محاسبه E_MOD.....
20	شکل (۲۱-۱) تابع K-SVD.....
20	شکل (۲۲-۱) منحنی همگرایی K-SVD.....

۱-۱- بخش اول

ابتدا داده ها را لود میکنیم.

```
load('hw6-part1.mat');  
% Access the variables:  
% D: Dictionary matrix  
% x: Noiseless observation
```

شکل (۱-۱) لود کردن داده ها

الف) بررسی روش BP:

```
N = size(D, 2);  
f = ones(2 * N, 1);  
Aeq = [D, -D];  
beq = x;  
lb = zeros(2 * N, 1);
```

شکل (۱-۲) Set up the linear programming problem

```
options = optimoptions('linprog', 'Display', 'none');  
yhat = linprog(f, [], [], Aeq, beq, lb, [], [], options);
```

شکل (۱-۳) حل مسئله برنامه ریزی خطی را با استفاده از تابع 'linprog'

```
yhat = linprog(f, [], [], Aeq, beq, lb, [], [], options);  
splus = yhat(1:N);  
sminus = yhat(N+1:end);  
sBP = splus - sminus;
```

شکل (۱-۴) کد متلب استخراج بردار S

sBP	
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

شکل (۵-۱) Vector S استخراج شده با روش BP

ب) به کمک کد زیر روش MP را پیاده سازی میکنیم.

```
%% MP
N0 = 2; % Desired sparsity level
N = size(D, 2);
residual = x; % Set the initial residual as the observation x
sMP = zeros(N, 1); % Initialize the sparse vector sMP
posMP = zeros(1, N0); % Initialize the positions of selected atoms

for i = 1:N0
    correlation = D' * residual; % Calculate the correlations between dictionary atoms and residual
    [~, pos] = max(abs(correlation)); % Find the atom with the maximum correlation
    posMP(i) = pos; % Store the selected atom position
    atom = D(:, pos); % Retrieve the selected atom
    projection = atom' * residual; % Project the residual onto the selected atom
    sMP(pos) = sMP(pos) + projection; % Update the sparse vector
    residual = residual - projection * atom; % Update the residual
end
```

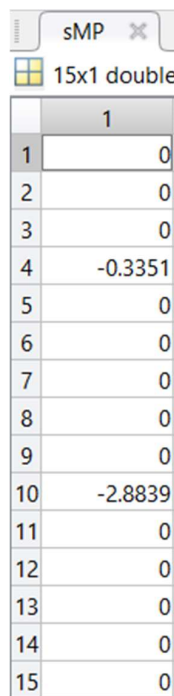
شکل (۶-۱) کد متلب پیاده سازی روش MP

به نظر می رسد که روش MP الگوی پراکندگی سیگنال واقعی را به طور کامل دریافت نکرده است. مقادیر

غیر صفر را به موقعیت هایی اختصاص داد که طبق داده های بدون نویز باید صفر باشند. از سوی دیگر، روش BP

موقعیت های غیر صفر را به دقت شناسایی و مقادیر متناظر آنها را تخمین زد.

تفاوت اصلی بین این دو روش در هدف بهینه سازی نهفته است. هدف روش MP انتخاب مکرر اتم های دیکشنری است که به بهترین وجه با باقیمانده مطابقت دارند، بدون اینکه صریحاً پراکندگی اعمال شود. در مقابل، روش BP یک مشکل بهینه سازی را حل می کند که با به حداقل رساندن ℓ_1 norm بردار پراکنده تحت محدودیت برازش مشاهده، پراکندگی را ترویج می کند.



	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

شکل (۷-۱) Vector S استخراج شده با روش MP

ج) هنگام در نظر گرفتن مشاهدات نویزی و استفاده از روش BP (Basis Pursuit) ممکن است پاسخ به دست آمده به دلیل وجود نویز دقیقاً با پاسخ واقعی یکسان نباشد. هدف روش BP یافتن یک راه حل پراکنده است که ℓ_1 norm را تحت محدودیت برازش مشاهدات به حداقل می رساند. با این حال، نویز می تواند خطا ایجاد کند و بر دقت بردار پراکنده تخمین زده شده تأثیر بگذارد.

در صورت وجود نویز، بردار پراکنده تخمین زده شده از روش BP ممکن است مقادیر غیر صفر در موقعیت هایی داشته باشد که سیگنال واقعی صفر است یا برعکس. دقت تخمین به نسبت سیگنال به نویز و سطح نویز موجود در مشاهدات بستگی دارد.

برای ارزیابی شباهت بین پاسخ به دست آمده و پاسخ واقعی، می توان بردار پراکنده تخمینی sBP را با بردار پراکنده واقعی s (S_BP) مقایسه کرد. اگر روش BP به خوبی عمل کند، sBP تخمین زده باید دارای مقادیر غیر صفر در موقعیت هایی باشد که با مقادیر غیر صفر در s واقعی مطابقت دارد، در حالی که مقادیر کوچک یا صفر در موقعیت های مربوط به مقادیر صفر در s داشته باشد.

```
%% BP X_noisy :
% Set the parameters
N = size(D, 2); % Number of columns in the dictionary
M = size(D, 1); % Number of rows in the dictionary
eps = 0.01; % Threshold for non-zero elements in sBP_noisy

% Solve the Basis Pursuit problem using linear programming
f = ones(2*N, 1);
Aeq = [D, -D];
beq = x_noisy;
lb = zeros(2*N, 1);
yhat = linprog(f, [], [], Aeq, beq, lb, []);

% Extract the positive and negative parts of the solution
splus = yhat(1:N);
sminus = yhat(N+1:end);

% Calculate sBP_noisy
sBP_noisy = splus - sminus;

% Find the positions with non-zero elements in sBP_noisy
posBP_noisy = find(abs(sBP_noisy) > eps)';

% Display sBP_noisy
disp('sBP_noisy:')
[ posBP_noisy; sBP_noisy(posBP_noisy) ]

% Calculate the loss (error) between sBP_noisy and the true s
loss_BP_noisy = norm(sBP_noisy - sBP);

% Display the loss
fprintf('Loss (BP_noisy): %.2f\n', loss_BP_noisy);
```

شکل (۸-۱) کد متلب محاسبه BP برای x_noisy

sBP_noisy	
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

شکل (۹-۱) sBP_noisy استخراج شده

مقدار فاصله با مقدار واقعی S در این حالت برابر با : 0.27 می باشد.

د) در این مساله فرمولی که باید حل شود به صورت زیر می باشد.

$$\text{minimize } \|Ds - x_noisy\|^2 + \lambda \|s\|_1$$

که در آن D ماتریس دیکشنری است، s بردار Sparse است که می خواهیم تخمین بزنیم، x_noisy

مشاهدات نویزی است، و λ پارامتر منظم سازی (regularization) است.

برای حل مسئله LASSO برای مقادیر مختلف λ و یافتن مقادیری که منجر به پاسخ صحیح می شوند،

می توانیم از یک الگوریتم بهینه سازی تکراری مانند الگوریتم حداقل مربعات منظم L1-norm (همچنین به عنوان

الگوریتم LASSO شناخته می شود) استفاده کنیم.


```

%% Lasso with different Lambda
% Set the range of lambda values
lambda_values = [0.001, 0.01, 0.1, 1, 10, 100];

% Initialize the solution matrix
s_lasso = zeros(N, numel(lambda_values));

% Solve the LASSO problem for each lambda value
for i = 1:numel(lambda_values)
    lambda = lambda_values(i);
    [s_lasso(:, i), ~] = lasso(D, x_noisy, 'Lambda', lambda, 'Standardize', false);
end

% Compare the estimated sparse vectors with the true sparse vector
for i = 1:numel(lambda_values)
    pos_lasso = find(abs(s_lasso(:, i)) > 0);
    fprintf('Lambda = %.3f:\n', lambda_values(i));
    fprintf('Non-zero elements of S:\n');
    fprintf('%d ', pos_lasso);
    fprintf('\n');
    fprintf('Values of non-zero elements of S:\n');
    fprintf('%0.2f ', s_lasso(pos_lasso, i));
    fprintf('\n');
    fprintf('Loss (LASSO): %.2f\n', norm(D*s_lasso(:, i) - x_noisy));
    fprintf('-----\n');
end

```

شکل (۱۰-۱) کد متلب فرموله کردن lasso و حل به ازای لاندا های مختلف

که خروجی گزارش به شرح زیر می باشد :

Lambda = 0.001:

Non-zero elements of S:

13 10 5 2

Values of non-zero elements of S:

0.12- 2.07- 0.15- 0.80

Loss (LASSO): 0.10

Lambda = 0.010:

Non-zero elements of S:

13 10 5 2

Values of non-zero elements of S:

0.09- 2.05- 0.13- 0.80

Loss (LASSO): 0.13

Lambda = 0.100:

Non-zero elements of S:

10 2

Values of non-zero elements of S:

2.05- 0.39

Loss (LASSO): 0.65

Lambda = 1.000:

Non-zero elements of S:

Values of non-zero elements of S:

Loss (LASSO): 2.96

Lambda = 10.000:

Non-zero elements of S:

Values of non-zero elements of S:

Loss (LASSO): 2.96

Lambda = 100.000:

Non-zero elements of S:

Values of non-zero elements of S:

Loss (LASSO): 2.96

آنالیز نتایج فوق :

روش LASSO با مقادیر مختلف لاندا توانست بردار پراکنده s را بر اساس مشاهدات نویزدار x_{noisy}

تخمین بزند. انتخاب پارامتر لاندا نقش مهمی در کنترل پراکندگی بردار برآورد شده ایفا می کند.

برای مقادیر کوچک لاندا (0.001 و 0.010)، روش LASSO به طور موثر عناصر غیر صفر s (موقعیت‌های

2، 5، 10، و 13) را شناسایی کرد و تخمین‌های دقیقی را برای مقادیر متناظر آنها ارائه کرد. از دست دادن (اندازه

گیری شده به عنوان تفاوت بین سیگنال بازسازی شده و مشاهده نویزدار) برای این مقادیر لاندا نسبتاً کم بود، که

نشان دهنده تقریب خوبی از s است.

با افزایش مقدار لاندا به 0.100، روش LASSO در انتخاب عناصر غیر صفر محافظه کارتر شد، و در نتیجه

یک تخمین پراکنده‌تر با تنها موقعیت‌های 2 و 10 دارای مقادیر غیر صفر بود. با این حال، ضرر افزایش یافت، که

نشان دهنده یک مبادله بین پراکندگی و دقت است.

برای مقادیر لاندا بزرگتر (1.000، 10.000 و 100.000)، روش LASSO هیچ عنصر غیر صفر را در s

شناسایی نکرد، که منجر به loss 2.96 برای همه مقادیر لاندا شد. این نشان می‌دهد که اثر منظم‌سازی بر عبارت

غالب بوده و منجر به تخمین بسیار پراکنده‌ای می‌شود که نتوانست عناصر غیر صفر واقعی را ثبت کند.

۱-۲- بخش دوم

الف) الگوریتم مورد استفاده برای تولید یک فریم با ابعاد $2 \times N$ ، که در آن N تعداد بردارها است، به شرح زیر است:

محاسبه زاویه $\theta = \arccos(\pi/N)$

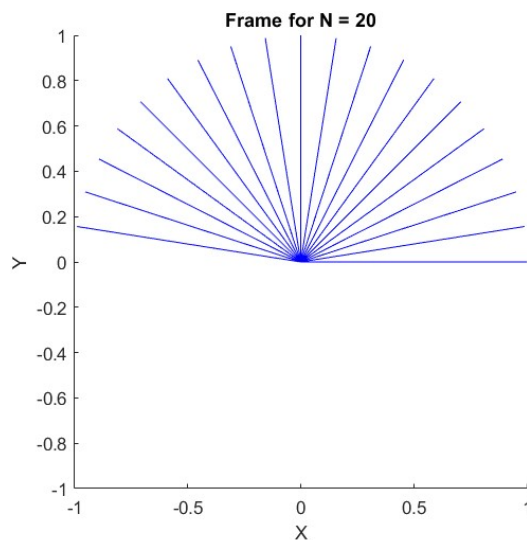
ایجاد یک ماتریس خالی برای ذخیره فریم ها: $\text{vectors} = \text{zeros}(2, N)$

برای هر i از 1 تا N :

- Calculate the angle for the current index: $\text{angle} = \theta * (i-1)$.
- Assign the x-coordinate of the vector at index i as $\cos(\text{angle})$.
- Assign the y-coordinate of the vector at index i as $\sin(\text{angle})$.

بردارهای ماتریس حاصل شامل بردارهای N است که به طور یکنواخت با زوایای متناوب بین آنها مرتب شده اند.

الگوریتم فوق در کد زیر توضیح داده شده است.



شکل (۱-۱۱) فریم 2×20 (لازم به ذکر است مدل های دیگر نیز rotate همین میتواند باشد)

```

%% A
N = 20; % Set the value of N

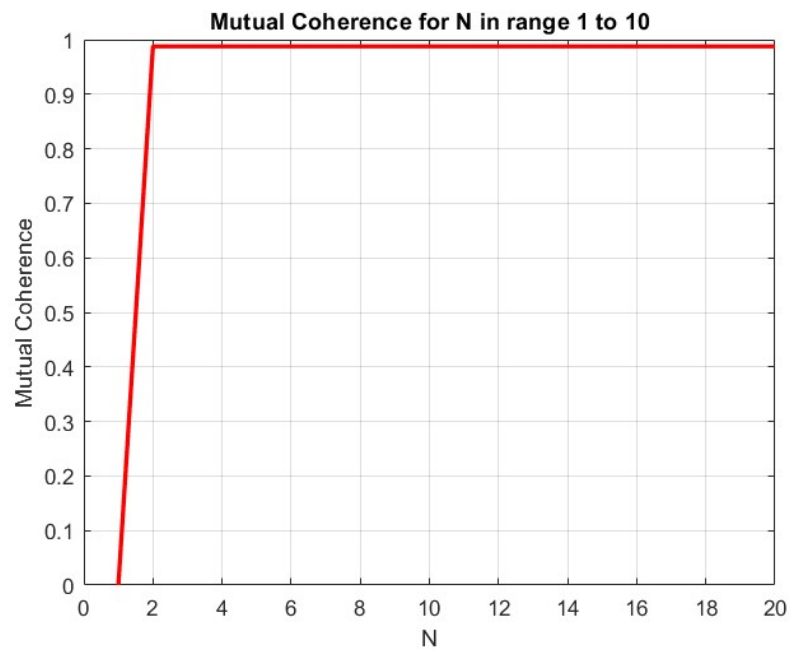
% Generate the frame
theta = pi/N;
vectors = zeros(2, N);
for i = 1:N
    angle = theta * (i-1);
    vectors(:, i) = [cos(angle); sin(angle)];
end

% Plot the frame
figure;
hold on;
for i = 1:N
    plot([0, vectors(1,i)], [0, vectors(2,i)], 'b');
end
axis equal;
axis([-1 1 -1 1]); % Set the axis limits
title('Frame for N = 20');
xlabel('X');
ylabel('Y');
hold off;

% Calculate mutual coherence for N in range 1 to 10
MC = zeros(1, N);
for n = 1:N
    G = vectors(:, 1:n)' * vectors(:, 1:n);
    G = abs(G) - eye(n);
    MC(n) = max(max(G));
end

```

شکل (۱۲-۱) کد متلب الف بخش 2



شکل (۱۳-۱) منحنی MC بر حسب N

ب) ابتدا با استفاده از تابع randn یک فریم ماتریس تصادفی به اندازه 3×10 ایجاد می کنیم. هر عنصر ماتریس از یک توزیع نرمال استاندارد گرفته شده است.

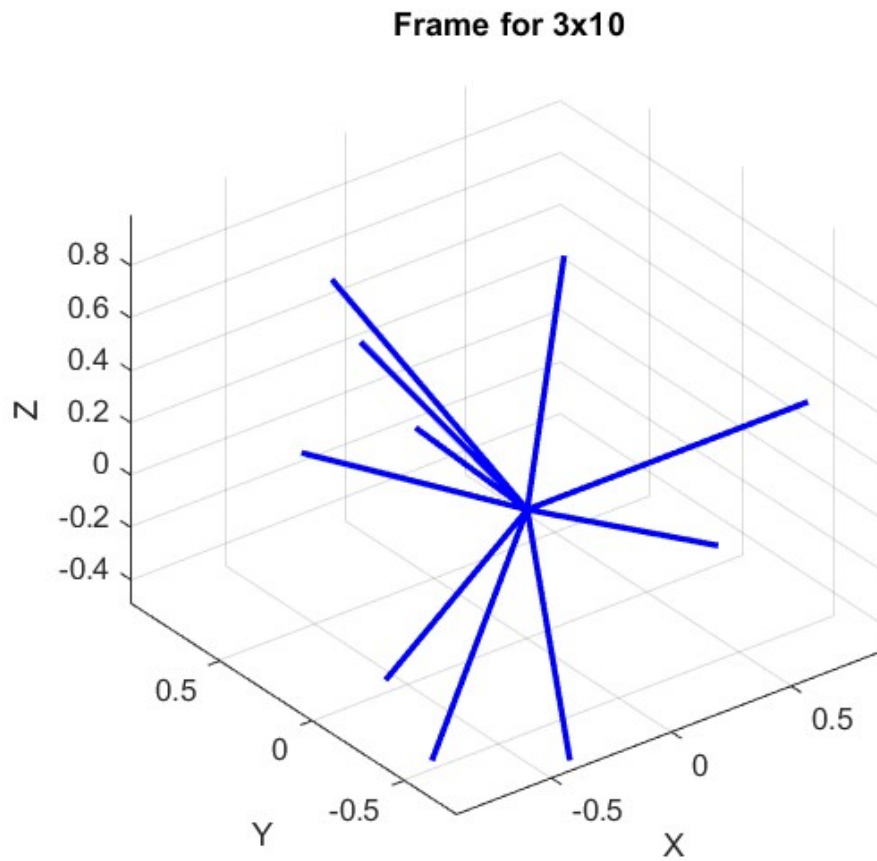
در مرحله بعد، هر ستون فریم را با استفاده از عملگر تقسیم element-wise / و تابع vecnorm نرمال می کنیم. تابع vecnorm (قدر) هر بردار ستون در کادر را محاسبه می کند و بردار ستونی با همان اندازه را با بردارهای طول واحد برمی گرداند.

در نهایت، ماتریس فریم حاصل را نمایش می دهیم که نشان دهنده قاب 3×10 است که هر ستون یک بردار را در فضای سه بعدی نشان می دهد.

```
%% B
N = 10; % Set the value of N
% Generate the frame
frame = randn(3, N);
frame = frame ./ vecnorm(frame);
% Plot the frame
figure;
hold on;
for i = 1:N
    plot3([0, frame(1,i)], [0, frame(2,i)], [0, frame(3,i)], 'b', 'Linewidth', 2);
    view(3)
end
axis equal;
title('Frame for 3x10');
xlabel('X');
ylabel('Y');
zlabel('Z');
grid on;
hold off;
% Calculate mutual coherence
mutual_coherence = calculate_mutual_coherence(frame);
disp("Mutual Coherence: " + mutual_coherence);
function mutual_coherence = calculate_mutual_coherence(frame)
    % Calculate the mutual coherence
    G = frame' * frame;
    G = abs(G) - eye(size(frame, 2));
    mutual_coherence = max(max(G));
end
```

شکل (۱۴-۱) کد متلب پیاده سازی فریم 3×10

مقدار MC در این حالت برابر: 0.98279 میباشد.



شکل (۱-۱۵) فریم 3×10

۱-۳- بخش سوم :

برای محاسبه mutual Coherence به صورت زیر عمل میکنیم.

این مقدار در ماتریس Dictionary برابر 0.86 میباشد.

```
5 %% A :
6 % Step 1: Load matrices from file
7 load('hw6-part3.mat', 'D');
8
9 % Step 2: Compute coherence matrix
10 C = abs(D.' * D);
11
12 % Step 3: Compute maximum absolute inner product
13 max_inner_product = max(max(triu(C, 1)));
14
15 % Step 4: Report mutual coherence measure
16 mutual_coherence = max_inner_product;
17 disp(['Mutual Coherence Measure: ' num2str(mutual_coherence)]);
18
```

Command Window

Mutual Coherence Measure: 0.86803

شکل (۱۶-۱) محاسبه mutual Coherence

2-3-1- پیاده سازی MOD

روش جهت گیری بهینه (MOD) یک الگوریتم قدرتمند است که در نمایش سیگنال پراکنده و یادگیری دیکشنری استفاده می شود. هدف آن تجزیه یک سیگنال داده شده به یک ترکیب خطی پراکنده از اتم ها از یک دیکشنری آموخته شده است. الگوریتم MOD به طور مکرر دیکشنری و ضرایب پراکنده را تا زمان همگرایی به روز می کند، که منجر به تخمینی از دیکشنری و منبع می شود.

نمای کلی الگوریتم:

Initialization: ماتریس دیکشنری D مقداردهی و حداکثر تعداد تکرارها تنظیم میشود

تکرارهای MOD:

آ. بازیابی پراکنده: از یک الگوریتم بازیابی پراکنده، مانند تعقیب تطبیق متعامد (OMP)، برای تخمین ضرایب پراکنده \hat{S} با توجه به دیکشنری D و ماتریس مشاهده X استفاده میشود. OMP به طور مکرر اتم هایی را انتخاب می کند که سیگنال باقیمانده را به بهترین شکل نشان می دهند.

ب به روز رسانی دیکشنری: ماتریس دیکشنری D را با حل یک مسئله حداقل مربعات به روز کنید، که در آن دیکشنری به عنوان حاصلضرب ماتریس مشاهداتی X و معکوس ماتریس ضریب پراکنده S_hat تقریبی شده است.

ج. بررسی همگرایی: خطای نمایش (تابع هدف) را با محاسبه هنجار فروبنیوس تفاوت بین ماتریس مشاهداتی X و حاصلضرب ماتریس دیکشنری D و ماتریس ضریب پراکنده S_hat محاسبه میشود. اگر معیار همگرایی برآورده شد، الگوریتم خاتمه میابد. در غیر این صورت، به تکرار بعدی میرویم.

بخش های مختلف دستورات فوق در کد زیر پیاده سازی شده است :

```
% Set the parameters
No = 2; % Sparsity level
MaxIter = 100; % Maximum number of iterations

% Initialize the dictionary matrix D using random values
D = randn(10, 40);

% Perform MOD algorithm
tic;
for iter = 1:MaxIter
    % Sparse recovery using OMP
    S_hat = zeros(size(D, 2), size(X, 2));
    for i = 1:size(X, 2)
        r = X(:, i);
        Omega = [];
        res_norm = norm(r);
        j = 0;

        while j < No
            j = j + 1;

            % Find the index of the atom that best represents the residual
            proj = abs(D' * r);
            [~, idx] = max(proj);

            % Add the index to the support set
            Omega = union(Omega, idx);

            % Compute the least squares solution on the support set
            S_hat(Omega, i) = pinv(D(:, Omega)) * X(:, i);

            % Update the residual
            r = X(:, i) - D(:, Omega) * S_hat(Omega, i);

            % Calculate the norm of the residual
            res_norm = norm(r);

            % If the residual is small enough, terminate
            if res_norm < 1e-6
                break;
            end
        end
    end

    % Update the dictionary
    D = X * pinv(S_hat);

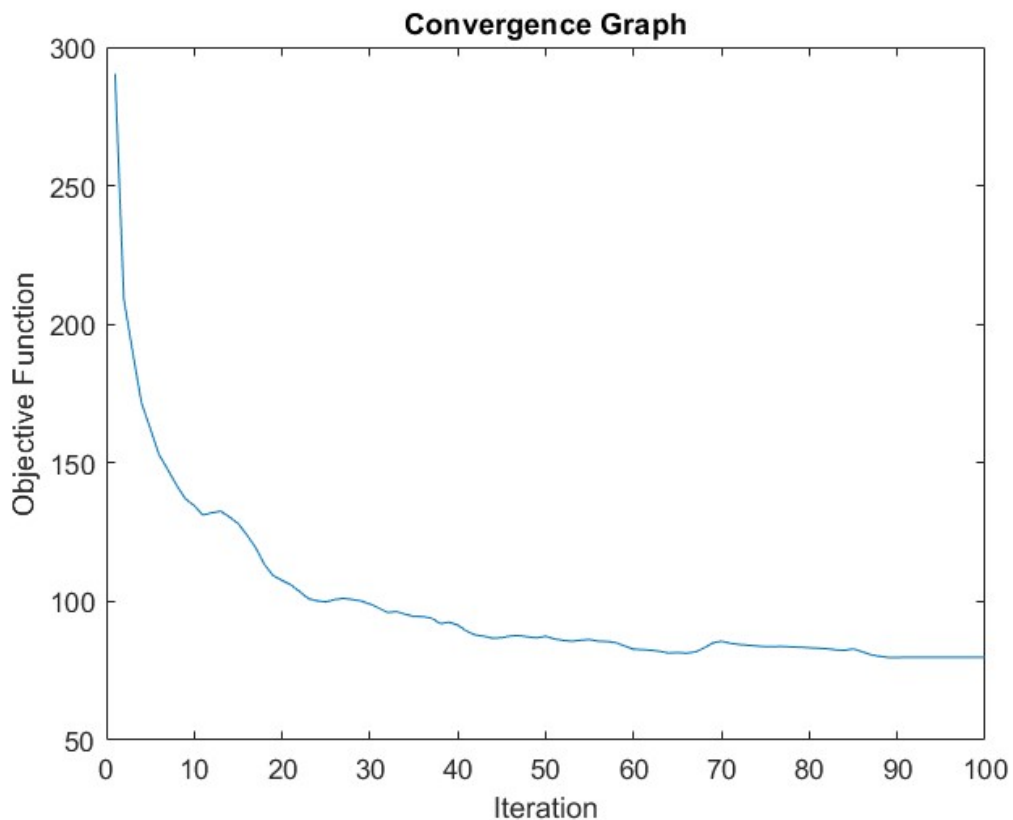
    % Calculate the objective function (representation error)
    error = norm(X - D * S_hat, 'fro')^2;

    % Store the objective function value for the convergence graph
    objective(iter) = error;
end
mod_time = toc;

% Plot the convergence graph
figure;
plot(1:MaxIter, objective);
title('Convergence Graph');
xlabel('Iteration');
ylabel('Objective Function');

% Report the convergence time
disp('Convergence time (MOD): ' + mod_time + ' seconds');
```

شکل (۱۷-۱) کد پیاده سازی MOD



شکل (۱-۱۸) منحنی همگرایی MOD

در این حالت : Convergence time (MOD): 13.2342 seconds

برای پیدا کردن SSR از کد زیر استفاده میکنیم :

```
% Calculate successful recovery rate
recovered_atoms = [];
num_atoms = size(D, 2);
for i = 1:num_atoms
    corr_values = abs(D(:, i)' * D);
    corr_values(i) = 0; % Exclude self-correlation
    max_corr = max(corr_values);

    if max_corr > 0.98
        recovered_atoms = [recovered_atoms i];
        D(:, i) = zeros(size(D, 1), 1); % Remove the recovered atom
    end
end
```

شکل (۱-۱۹) کد پیدا کردن successful recovery rate

در این حالت : 75% Successful Recovery Rate (MOD):

به کمک کد زیر E را محاسبه میکنیم . (چون ماتریس S را در کد برای تخمین استفاده کرده بودیم مجددا

ماتریس S لود شده است.)

```
load('hw6-part3.mat', 'S');  
diff_S = S_hat - S;  
norm_diff_S = norm(diff_S, 'fro');  
norm_S = norm(S, 'fro');  
E_MOD = norm_diff_S / norm_S;  
disp("E_MOD: " + E_MOD);
```

شکل (۲۰-۱) محاسبه E_MOD

که در این حالت این مقدار : E_MOD: 1.0367 میباشد.

۱-۴- پیاده سازی K-SVD

برای پیاده سازی K-SVD به صورت زیر function K-SVD را مینویسیم :

```

function [D_hat, S_hat, convergence] = ksvd(X, D, No, maxIterations)
% X: Observation matrix
% D: Initial dictionary matrix
% No: Sparsity level
% maxIterations: Maximum number of iterations

N = size(D, 2);      % Dictionary size
T = size(X, 2);      % Number of signals

S_hat = zeros(N, T); % Sparse representations
convergence = zeros(1, maxIterations); % Convergence values

for iter = 1:maxIterations
    % Sparse coding step (using OMP)
    S_hat = omp(D, X, No);

    % Dictionary update step
    for k = 1:N
        indices = find(S_hat(k, :) ~= 0); % Find signals that use atom k

        if ~isempty(indices)
            % Update the k-th atom
            E = X(:, indices) - D * S_hat(:, indices) + D(:, k) * S_hat(k, indices);
            [U, ~, V] = svd(E, 'econ');
            D(:, k) = U(:, 1);
            S_hat(k, indices) = V(:, 1); % Update with compatible dimensions
        end
    end

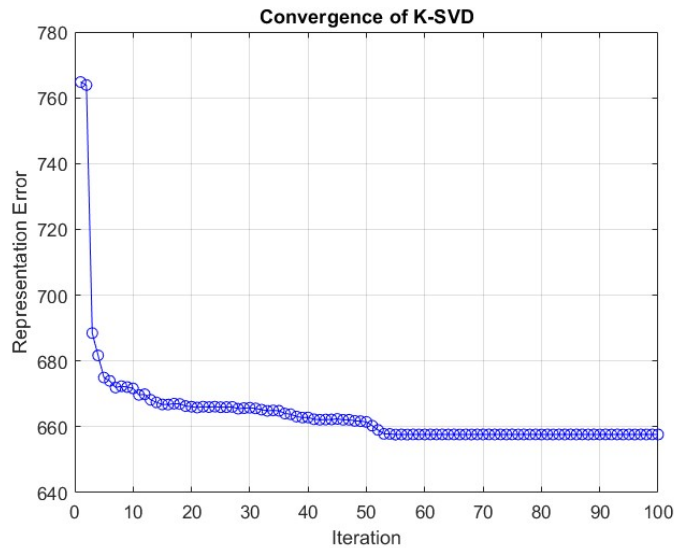
    % Calculate convergence value (Representation Error)
    convergence(iter) = norm(X - D * S_hat, 'fro')^2;
end

D_hat = D; % Estimated dictionary
end

```

شکل (۱-۲۱) تابع K-SVD

در این جا منحنی همگرایی به صورت زیر می باشد :



شکل (۱-۲۲) منحنی همگرایی K-SVD

مشاهده میشود که تا $iter$ حدود 50 هم کافی بود.

زمان ران نیز برابر با: 4.237919 ثانیه میباشد. که به مراتب کمتر از MOD بوده و نشان از این دارد که این

الگوریتم سریع تر همگرا میشود

همانند قبل SSR را نیز محاسبه میکنیم که برای K-SVD این مقدار برابر با: 83 درصد میباشد و نشان از این

دارد که K-SVD برتری بالاتری دارد.

به مانند قبل E_K-SVD را هم محاسبه کرده که در اینجا این مقدار: E_K-SVD: 1.0203 میباشد که نشان

میدهد خطای حاصل از k-SVD کمتر است.