VIETNAM NATIONAL UNIVERSITY HO CHI MINH
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

–oo0oo–

LINEAR ALGEBRA

PROJECT 7: NORM, ANGLES AND YOUR MOVIE CHOICES

Semester: HK242 **Class:**

Lecturer: Dr.Đậu Thế Phiệt

Group:

Full Name	Student ID	Contribution
Nguyễn Hữu Sang	2114636	100%
Nguyễn Thụy Khánh Linh	2052576	100%
Hồ Huỳnh Minh Khoa	2352558	100%
Nguyễn Đỗ Thịnh Phát	2352886	100%

Tp.HCM, . . . / . . . /20. . .

Contents

I. INTRODUCTION

The ability to discern and leverage user preferences has become a commercially significant endeavor, catalyzing substantial advancements in recommendation systems across prominent digital platforms such as Spotify, YouTube, and Shopee. These platforms invest heavily in sophisticated algorithms to meticulously analyze user behavior, thereby enabling the delivery of highly personalized product or content suggestions. For instance, Spotify curates custom playlists based on individual listening patterns, while YouTube utilizes machine learning to recommend videos aligned with a user's prior viewing history. This report aims to explore the underlying patterns in user preferences by applying linear algebra techniques to the extensive MovieLens dataset, which contains millions of ratings from a diverse user base. Through a comparative analysis of these ratings, this report will develop a recommendation framework designed to suggest items to users based on the preferences of those with similar tastes, employing established similarity metrics such as Euclidean distance, scalar product and Pearson correlation.

II. THEORY

1. Norm

Concept: A norm is a measure of a vector's "magnitude" that always returns a non-negative value. The zero vector has a norm of 0. Norms are widely used in data analysis and optimization.

Types of Norms:

- **Euclidean Norm (L_2):** Measures geometric distance between points in space:

$$||x||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- **Manhattan Norm (L_1):** Sum of absolute values, useful for optimization:

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

- **Maximum Norm ($L\infty$):** The largest absolute component value:

$$||x||_\infty = \max_{1\leq i\leq n} |x_i|$$

- **p-Norm:** Generalization of norms with parameter p:

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \quad (p \geq 1)$$

2. Angle Between Vectors

Purpose: The angle between two vectors indicates their similarity:

- Small angle → similar direction.

- Large angle → different direction.

Formulas:

- Dot Product:

$$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

- Vector Norm:

$$||u|| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}$$

- Angle Calculation:

$$\cos\phi = \frac{u \cdot v}{\|u\|\|v\|}$$

3. Pearson Correlation Coefficient

Interpretation: Measures linear correlation between datasets (-1 to 1):

- **-1:** Perfect negative correlation.
- **0:** No correlation.
- **1:** Perfect positive correlation.

Formula:

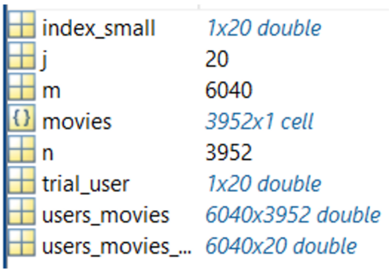
$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

III. PROCEDURE

1. Load data

```
% Clear the workspace
clear;
% Load the user_movies.mat file
load('users_movies.mat', 'movies', 'users_movies', 'users_movies_sort', '
index_small', 'trial_user')
```

First, we load the data from the file "users_movies.mat" using the "load" command. "load" helps us to load actual data stored in those variables shown below while "clear" is used to clear all variables in MATLAB workspace to ensure that no old data interferes. The matrix "users_movies" should have dimensions 6040 × 3952, with integer values ranging from 0 to 5. A rating of 1 represents "strongly dislike", while 5 represents "strongly like". A rating of 0 indicates that the user did not rate the movie. The array "movies" contains the titles of all the movies. Moreover, the matrix "users movies sort" is a subset of "users_movies", containing ratings for the 20 most popular movies.



index_small	1x20 double
j	20
m	6040
movies	3952x1 cell
n	3952
trial_user	1x20 double
users_movies	6040x3952 double
users_movies_...	6040x20 double

Also, the indexes of these popular movies are stored in the array "index_small". Finally, the vector "trial_user" contains ratings of these popular movies by another user who is not part of the database. It is recommended to view all variables and their dimensions using the "Workspace" window in the MATLAB environment.

2. Print the titles

The figure below shows the 20 most popular movies:

```
The rating is determined based on the 20 most popular movies:
1. E.T. the Extra-Terrestrial (1982)
2. Star Wars Episode IV - A New Hope (1977)
3. Star Wars Episode V - The Empire Strikes Back (1980)
4. Star Wars Episode VI - Return of the Jedi (1983)
5. Jurassic Park (1993)
6. Saving Private Ryan (1998)
7. Terminator 2: Judgment Day (1991)
8. The Matrix (1999)
9. Back to the Future (1985)
10. The Silence of the Lambs (1991)
```

```
11. Star Wars Episode I - The Phantom Menace (1999)
12. Raiders of the Lost Ark (1981)
13. Fargo (1996)
14. The Sixth Sense (1999)
15. Braveheart (1995)
16. Shakespeare in Love (1998)
17. The Princess Bride (1987)
18. Schindler's List (1993)
19. The Shawshank Redemption (1994)
20. Groundhog Day (1993)
```

The code below is use to print top 20 most popular movies.

```
% Get the dimensions
[m, n] = size(users_movies);

% Print a header indicating the movies to be listed
fprintf('The rating is determined based on the %d most popular movies: \n', length(index_small))

% Loop through of the popular movies
for j = 1:length(index_small)
   % print the movies title correponding to the current index
   fprintf('%d. %s \n', j, movies(index_small(j)));
end
fprintf('\n);
```

The ”`fprintf()`” function is a versatile tool used to display formatted text directly in the Command Window. When constructing the output string, special format specifiers are used to indicate how different types of data should be displayed. For instance, ”`%d`” serves as a placeholder that allows for the printing of an integer value. To control the layout of the output, the newline character, represented as ”\`n`”, is used to move the cursor to the beginning of the next line, effectively adding a line break.

Generally, ”`length(index_small)`” is used to determine the total number of elements within an array named ”`index_small`”. This numerical result can then be incorporated into a formatted string using ”`fprintf()`” and the ”`%d`” specifier to display this count.

Loops are fundamental for repetitive tasks, and the ”`for`” loop is commonly used to execute a block of code multiple times. Within such a loop, “`fprintf()`” can be employed again to print details for each iteration.

For example, the format string ”`%d. %s` \`n`” instructs the program to print an integer (using `%d`), followed by a period and a space, then a string of characters (using `%s`), and finally, to add a new blank line (\`n`). In short:

- `%d`: Prints the index interger.
- `%s`: Prints the title movies.
- \`n`: Adds new blank line.

This is useful for creating numbered lists, such as displaying an index number alongside a movie title. To further improve the visual separation and readability of the output, an additional "`fprintf('\n')`" can be used to print an extra blank line, creating better spacing in the Command Window.

3. Select people

The following code selects people who rated all of the 20 movies under consideration:

```
% Select the users to compare to
[m1, n1] = size(users_movies_sort);
rating = [];
for j = 1:m1
   if prod(users_movies_sort(j, :)) ~= 0
      rating = [rating; users_movies_sort(j, :)];
   end;
end;
```

Explain:

Initially, the code determines the dimensions of a matrix named "`users_movies_sort`" by using the "`size()`" function. This function call assigns the number of rows (representing users) to the variable m1 and the number of columns (representing movies) to `n1`. Following this, an empty matrix called ratings is created, which will be used to store selected user data:

- `[m1, n1] = size(users_movies_sort)`: Measure the size of the users_movies_sort matrix. m1 is the number of users (rows), n1 is the number of movies (columns).

- `ratings = []`: Create an empty matrix called ratings.

The core of this code segment is a for loop that iterates through each user, from the first user (index 1) up to m1 (the total number of users). Inside this loop, a conditional check is performed using an if statement. This condition evaluates the product of all ratings for the current user `j` (accessed via `users_movies_sort(j, :)`). If this product is not equal to zero (`~= 0`), it implies that the user has provided a rating for every movie (assuming no rating is represented by a zero). When this condition is true, that user's entire row of ratings from `users_movies_sort` is appended to the ratings matrix.

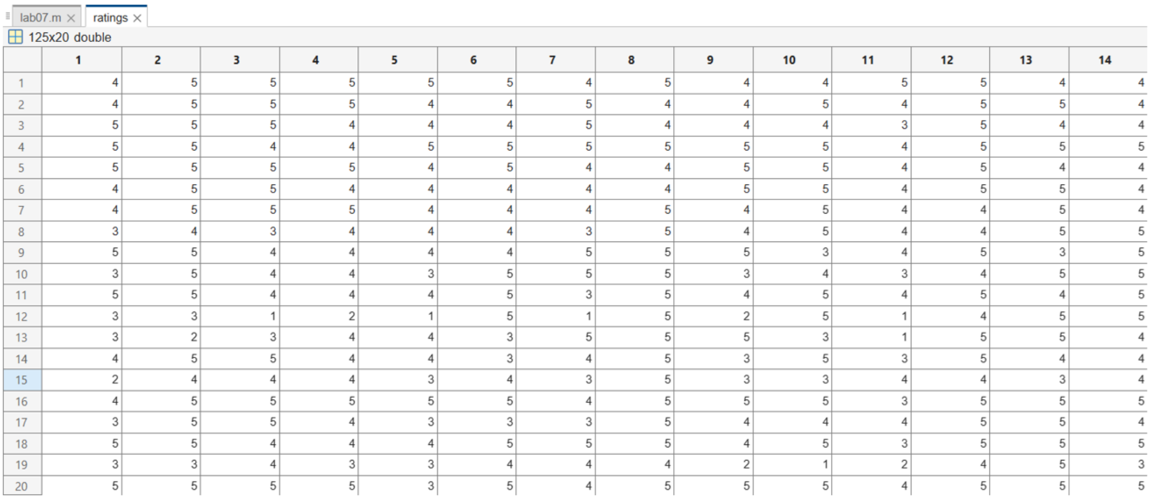
```
for j = 1:m1
        if prod(users_movies_sort(j, :)) ~= 0
        ratings = [ratings; users_movies_sort(j, :)];
end;
```

The whole point of this is to sort out users that rate all the film they watched and add their ratings to a set.

Question 1: What does the command `ratings=[]` do?

This initializes an empty array named ratings that ratings start with no elements. This setup is commonly used to collect or build up data in subsequent operations, specially within loops. The code then iterates over `users_movies_sort` array and adds rows to ratings for users who have rated all the movies they watched.

The below table shows a part of the result of ratings after the execution of the source code:



125x20 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	5	5	5	5	5	4	5	4	4	5	5	4	4
2	4	5	5	5	4	4	5	4	4	5	4	5	5	4
3	5	5	5	4	4	4	5	4	4	4	3	5	4	4
4	5	5	4	4	5	5	5	5	5	5	4	5	5	5
5	5	5	5	5	4	5	4	4	5	5	4	5	4	4
6	4	5	5	4	4	4	4	4	5	5	4	5	5	4
7	4	5	5	5	4	4	4	5	4	5	4	4	5	4
8	3	4	3	4	4	4	3	5	4	5	4	4	5	5
9	5	5	4	4	4	4	5	5	5	3	4	5	3	5
10	3	5	4	4	3	5	5	5	3	4	3	4	5	5
11	5	5	4	4	4	5	3	5	4	5	4	5	4	5
12	3	3	1	2	1	5	1	5	2	5	1	4	5	5
13	3	2	3	4	4	3	5	5	5	3	1	5	5	4
14	4	5	5	4	4	3	4	5	3	5	3	5	4	4
15	2	4	4	4	3	4	3	5	3	3	4	4	3	4
16	4	5	5	5	5	5	4	5	5	5	3	5	5	5
17	3	5	5	4	3	3	3	5	4	4	4	5	5	4
18	5	5	4	4	4	5	5	5	4	5	3	5	5	5
19	3	3	4	3	3	4	4	4	2	1	2	4	5	3
20	5	5	5	5	3	5	4	5	5	5	4	5	5	5

4. Find the Euclidean distance

The following code calculates the Euclidean distance:

```
% Find the Euclidean distance
[m2, n2] = size(ratings);
for i = 1:m2
    eucl(i) = norm(ratings(i,:)-trial_users);
end;
```

In this part, we would calculate the Euclidean distance to measure the similarity between a specific "`trial_user`" and every other user within the ratings dataset.

Where:

- `ratings(i, :)`: is the vector of movie ratings for i-th user.

- `trial_user`: is the vector of movie ratings for the trial user (the user whose similarities we want to find).

The first step in the comparison involves element-wise subtraction, `ratings(i,:) - trial_user`.

- `ratings(i,:)-trial_users`: computes the difference between the i-th user's ratings and the trial user's ratings for each movie.

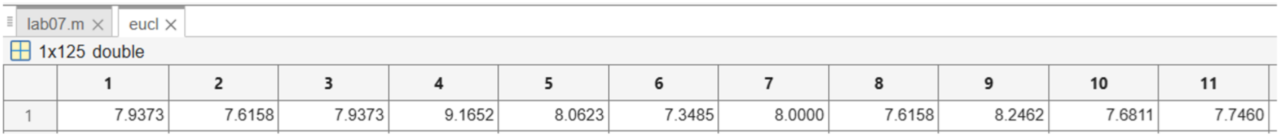
Subsequently, the "`norm()`" function is applied to this difference vector specifically,

- `norm(ratings(i,:)-trial_users)`: calculates the Euclidean distance between these two vectors.

The resulting distance for each comparison is stored in "`eucl(i)`",

- `eucl(i)`: represents the Euclidean distance between the `trial_user` and the i-th user in the ratings array (or matrix). The smaller the value of `eucl(i)`, the more similar the i-th user's ratings are to the trial user's ratings, this means that the i-th user has comparable tastes in movies to the `trial_user`.

Moreover, the eucl vector contains a series of Euclidean distances, each representing the similarity between a trial user and other users in a dataset. A portion of this vector is displayed in the table, showing various distance values calculated:



lab07.m × eucl ×

1x125 double

	1	2	3	4	5	6	7	8	9	10	11
1	7.9373	7.6158	7.9373	9.1652	8.0623	7.3485	8.0000	7.6158	8.2462	7.6811	7.7460

The command "`[smallest_number, position] = min(eucl);`" processes the eucl vector to find its minimum value and the index at which this minimum occurs. The minimum value is stored in the variable "smallest_number", while its corresponding index (or position within the vector) is stored in the variable position. Following this computation, the code uses `disp()` commands to output these findings:

```
% Find the smallest Euclidean distance
[smallest\_number, position] = min(eucl);
disp(['Smallest Eucl: ', num2str(smallest_number)]);
disp(['Position: ', num2str(position)]);
```

Upon running the code, the results indicate that the smallest Euclidean distance found is 5.6569, and this value corresponds to position 14 in the eucl vector. This means that the 14th person in the ratings data has rating patterns most closely matching the trial user, with a calculated Euclidean distance of 5.6569, which is shown in the figure below:

```
Smallest Eucl: 5.6569
Position: 14
```

5. Find the Pearson correlation coefficient

Initially, we create a zero vector to save the coefficient. This vector will eventually store the calculated Pearson correlation coefficient for each user in the ratings matrix when compared against the `trial_user`. Concurrently, the trial_user's rating vector is reshaped into a column vector using `trial_user(:);` to ensure it is correctly oriented for the subsequent matrix operations, as shown in the figure below.

```
    % Compute the Pearson correlation coefficient
    pearson = zeros(size(ratings, 1), 1);
    trial_user = trial_user(:);

    % Compute correlation along the ratings vectors
    for i = 1:size(ratings, 1)
    current_rating = ratings(i, :);

    % Compute mean
    trial_user_mean = mean(trial_user);
    current_ratings_mean = mean(current_rating);

    % Subtract with the mean
    trial_user_centered = trial_user - trial_user_mean;
    current_ratings_centered = current_rating - current_ratings_mean;

    pearson_numerator = sum(trial_user_centered .* current_ratings_centered);
    pearson_denominator = sqrt(sum(trial_user_centered .^ 2) .* sum(current_ratings_centered .^ 2));

    if pearson_denominator == 0
    comr = pearson_numerator / pearson_denominator;
    else
    comr = 0;
    end

    pearson(i) = comr;
    end

    % Find the highest Pearson correlation coefficient
    [max_pearson, pearson_index] = max(pearson);
    disp(['Highest Pearson correlation coefficient: ', num2str(max_pearson)]);
    disp(['Position: ', num2str(pearson_index)]);

```

- `pearson = zeros(size(ratings, 1), 1);`: The line initializes a column vector ‘pearson’ of zeros, with a length equal to the number of users (rows) in the ‘ratings’ matrix. This will store the Pearson correlation coefficients for each user compared to the trial user.

- `trial_user = trial_user(:)';`:’`trial_user(:)`’ reshapes the ‘`trial_user`’ vector into a column vector. ‘” (transpose) converts it back into a row vector.

This ensures that ‘`trial_user`’ is properly oriented for matrix operations that follow.

We then create a loop to traverse along the rows of rating vectors which is the rating of each user in database to compute the Pearson correlation coefficient with the trial user rating by the formula below:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

The process to compute the Pearson correlation coefficient:

Step 1: Compute the mean of 2 sets:

$$\bar{x} = \frac{\sum x_i}{nx}; \bar{y} = \frac{\sum y_i}{ny}$$

Step 2: Compute the subtracted vectors from 2 sets by their means:

- For set 'x', the centered vector x_s would have elements: $x_{si} = x_i - \bar{x}$.
- For set 'y', the centered vector y_s would have elements: $y_{si} = y_i - \bar{y}$.

Step 3: Compute the numeration: $\sum(x_{si} \cdot y_{si})$.

$$\sqrt{\sum x_{si}^2} \cdot \sqrt{\sum y_{si}^2}$$

Step 4: Compute the denominator.

Step 5: Divide the numerator from step 3 with the denominator in step 4.

```
for i = 1:size(ratings, 1)
current_rating = ratings(i, :);
```

This '`for`' loop iterates over each user's ratings in the '`ratings`' matrix. '`current_rating`' stores the ratings of the i-th user (row) for all items.

```
trial_user_mean = mean(trial_user);
current_ratings_mean = mean(current_rating);
```

'`trial_user_mean`' calculates the `mean` (average) rating of the trial user. '`current_ratings_mean`' calculates the mean rating of the i-th user. These means are used to center the ratings by subtracting the mean from each rating.

```
trial_user_centered = trial_user - trial_user_mean;
current_ratings_centered = current_rating - current_ratings_mean;
```

These lines subtract the mean rating from each user's individual ratings, centering the data. '`trial_user_centered`' contains the trial user's centered ratings. '`current_ratings_centered`' contains the centered ratings of the i-th user. Centering helps in comparing the patterns in the ratings, ignoring the absolute values.

```
pearson_numerator = sum(trial_user_centered.* current_ratings_centered);
```

This calculates the numerator of the Pearson correlation coefficient formula. '`.*`' performs element-wise multiplication of the centered ratings. '`sum`' adds up the results of these multiplications.

```
pearson_denominator = sqrt(sum(trial_user_centered .^2) .* sum(
current_ratings_centered .^2));
```

This calculates the denominator of the Pearson correlation coefficient formula. ‘.2̂’ squares each element in the centered rating vectors. ‘sum’ adds up these squared values. ‘sqrt’ takes the square root of the product of these sums.

```
if pearson_denominator ~= 0
    corr = pearson_numerator / pearson_denominator;
else
    corr = 0;
end;
```

This conditional statement checks if the denominator is non-zero to avoid division by zero.

If the denominator is non-zero, ‘corr’ is calculated as the Pearson correlation coefficient.

If the denominator is zero (which can happen if one of the users has constant ratings), ‘corr’ is set to 0.

6. Find the highest Pearson coefficient

```
% Compute the Pearson correlation coefficient
pearson = zeros(size(ratings, 1), 1);
trial_user = trial_user(:)';

% Compute correlation along the ratings vectors
for i = 1:size(ratings,1)
    current_rating = ratings(i, :);

    % Compute mean
    trial_user_mean = mean(trial_user);
    current_rating_mean = mean(current_rating);

    % Subtract the mean
    trial_user_centered = trial_user - trial_user_mean;
    current_rating_centered = current_rating - current_rating_mean;

    % Compute numerator and denominator
    pearson_numerator = sum(trial_user_centered .* current_rating_centered
);
    pearson_denominator = sqrt(sum(trial_user_centered .^ 2) * sum(
current_rating_centered .^ 2));

    % Avoid division by zero
    if pearson_denominator ~= 0
        corr = pearson_numerator / pearson_denominator;
    else
        corr = 0;
    end

    pearson(i) = corr;
end;
```

```

% Find the highest Pearson correlation coefficient
[max_pearson, pearson_index] = max(pearson);
disp(['Highest Pearson correlation coefficient: ', num2str(max_pearson)]);
disp(['Position: ', num2str(pearson_index)]);
```

- `pearson(i) = corr`: The Pearson correlation value 'corr' is stored at position 'i' in the 'pearson' array for each user analyzed in the loop. This process continues until all users have been evaluated for their similarity to the trial user.

- `[max_pearson, pearson_index] = max(pearson);`: This function locates both the maximum Pearson correlation value in the array and identifies its position. The '`max_pearson`' variable holds the highest correlation coefficient (indicating the user with the most similar rating patterns). While '`pearson_index`' records this user's position within the '`ratings`' matrix.

- `disp(['Highest Pearson correlation coefficient: ', num2str(max_pearson)]);` and `disp(['Position: ', num2str(pearson_index)]);`: These commands display the highest calculated correlation coefficient and the position of the most similar user. '`num2str`' converts the numeric values to strings for display purposes. '`disp`' prints out the text and the computed values.

After execution, the analysis shows:

```
Highest Pearson correlation coefficient: 0.62896
Position: 88
```

7. Compare the elements

This section examines the difference between users identified as similar through two distinct methods: Euclidean distance and Pearson correlation.

```
% Compare the elements of the vectors DistIndex, PearsonIndex
% Use the position of the smallest Euclidean distance
closest_user_Dist = position;

% Use the index of the highest Pearson correlation coefficient
closest_user_Pearson = pearson_index;

% Compare if the closest user by Euclidean distance is the same as the closest user by Pearson correlation
if closest_user_Dist == closest_user_Pearson
    disp(['The closest user based on Euclidean distance is the same as the closest user based on Pearson correlation.']);
else
```

```
        disp(['The closest user based on Euclidean distance is different from the closest user baesd on Pearson correlation.']);
    end

    % Display the top 5 users for both distance and Pearson correlatin for comparison
    fprintf('\nTop closeest users baesed on Euclidean distance:\n');
    if length(position) >= 5
        disp(position(1:5)');
    else
        disp(position');
    end

    fprintf('Top closest users based on Pearson correlation:\n');
    if length(pearson_index) >= 5
        disp(pearson_index(1:5)');
    else
        disp(pearson_index');
    end
```

The DistIndex vector identifies users with preferences closest to the trial user based on the Euclidean distance metric. This method calculates the ”straight-line” distance between the trial user's ratings and each other user's ratings.

The PearsonIndex vector identifies users with preferences closest to the trial user based on the Pearson correlation coefficient. This method measures the linear correlation between the trial user's ratings and each other user's ratings, accounting for how well the users' preferences align in terms of direction (both users like/dislike similar movies)

```
The closest user based on Euclidean distance is different from the closest
  user baesd on Pearson correlation.
Top closeest users baesed on Euclidean distance:
  14
Top closest users based on Pearson correlation:
  88
```

The output reveals that user 14 has the closest match by Euclidean distance, while user 88 has the highest Pearson correlation. This divergence demonstrates that these metrics capture different aspects of similarity - Euclidean distance identifies users with numerically similar ratings, while Pearson correlation finds users with similar rating patterns even if their absolute rating scales differ.

In conclusion, the closest user based on Euclidean distance differs from the closest user based on Pearson correlation, highlighting that these two methods capture different aspects of user similarity.

So, we can answer the question. No, the variables `closest_user_Pearson` and `closest_user_Dist` are distinct and serve different purposes. They are identified using two different methods for measuring similarity between the trial user (trial user) and other users in the dataset.

8. Diplay the recommendations

This section generates personalized movie recommendations based on the closest users identified by both similarity metrics. Additionally, a list of movies already liked by the trial user is presented.

```
% Use the position of the smallest Euclidean distance
closest_user_Dist = position;

% Use the index of the highest Pearson correlation coefficient
closest_user_Pearson = pearson_index;

recommend_dist = [];
for k = 1:n
   if (users_movies(closest_user_Dist, k) == 5)
      recommend_dist = [recommend_dist; k];
   end
end

recommend_Pearson = [];
for k = 1:n
   if (users_movies(closest_user_Pearson, k) == 5)
      recommand_Pearson = [recommand_Pearson; k];
   end
end

liked = [];
for k = 1:20
   if (trial_user(k) == 5)
      liked = [liked; index_small(k)];
   end
end

% Print out the recommendations based on Euclidean distance
fprintf('Recommendations based on Euclidean distance:\n');
for i = 1:length(recommend_dist)
   fprintf('%d. %s\n', i, movies{recommend_dist(i)});
end

% Print out the recommendations based on Pearson correlation
fprintf('\nRecommendations based on Pearson correlation:\n');
for i = 1:length(recommend_Pearson)
   fprintf('%d. %s\n', i, movies{recommend_Pearson(i)});
end

% Print out the movies liked by the trial user
fprintf('\nMovies liked by the trial user:\n');
for i = 1:length(liked)
   fprintf('%d. %s\n', i, movies{liked(i)});
```

```
  end
```

The result,

```
Recommendations based on Euclidean distance:
1. Pulp Fiction (1994)
2. Deer Hunter, The (1878)
3. Red Violin, The (Le Violon rouge) (1998)
4. Sixth Sense, The (1999)
5. Children of Paraidse (Les enfants du paradis) (1945)
6. Being John Malkovick (1999)

Recommendations based on Pearson correlation:
1. Taxi Driver (19776)
2. Schindler's List (1993)
3. Fargo (1996)
4. Godfather, The (1972)
5. North by Norhwest (1959)
6. Casablance (1942)
7. Ccitizen Kane (1941)
8. Mr. Smth Goes to Washington (1939)
9. Bonnie and Clyde (1967)
10. Bob Roberts (1992)
11. Paris Is Burning (1990)
12. 12 Angry men (1957)
13. To Kill a Mockingbird (1962)
14. Title not available
15. Frand Day Out, A (1992)
16. Ragin Bull (1980)
17. Annie Hall (1977)
18. Stand by Me (1986)
19. Killing Fields, The (1984)
20. My Life as a Dog (Mitt liv som hund) (1985)
21. Tickle in the Heart, A (1996)
22. Boys, Les (1997)
23. There's Something About Mary (1998)
24. On the Waterfront (1954)
25. Ordinary People (1980)
26. Chariots of Fire (1981)
27. Rain Man (1988)
28. Saving Private Ryan (1998)
29. Life Iis Beautiful (La Vita e bella) (1997)
30. Risky Business (1983)
31. Brief Encounter (1946)
32. Shower (Xizhao) (1999)

Movies liked by the trial user:
1. Star Wars Episode IV - A New Hope (1977)
2. Star Wars Episode V - The Empire Strikes Back (1980)
3. Star Wars Episode VI - Return of the Jedi (1983)
```

```
4. Matric, The (1999)
5. Silence of the Lambs, The (1991)
6. Raiders of the Lost Ark (1981)
7. Groundhog Day (1993)
```

The results show that the two recommendation approaches yield different movie suggestions. The Euclidean-based recommendations include films like ”Pulp Fiction,” ”Deer Hunter,” and ”Red Violin,” while the Pearson-based recommendations include classics like ”Taxi Driver,” ”Schindler’s List,” and ”The Godfather.”, . . .

This variance underscores the importance of selecting the right metric for recommendation systems depending on whether the goal is to match the intensity of preferences (Euclidean) or the overall pattern of preferences (Pearson). The trial user’s liked movies are mostly high-grossing, popular films with a strong fan base, suggesting that recommendations should ideally include films with similar mass appeal.

9. Create a personal rating vector called Myratings

9.1. Rating random movies form 1 to 5

This vector establishes personalized ratings for the 20 most popular movies in the dataset. Each value represents a preference rating on a scale of $1 - 5$, where 1 indicates strong dislike and 5 indicates strong preference.

```
myratings = [5, 4, 3, 2, 1, 5, 3, 4, 2, 1, 5, 4, 3, 2, 5, 1, 3, 4, 2, 5];
```

Then assign random ratings unseen movies:

```
unseen_indices = randi([1 20], 1 5); % Assuming 5 movies are unseen
myratins(unseen_indices) = randi([1, 5], 1, length(unseen_indices));
```

The first line generates a random selection of 5 movies (represented by indices from 1-20) to simulate movies the user hasn’t yet watched.

The second line assigns random ratings between 1-5 to these previously unseen movies. This approach simulates a realistic scenario where a user has only rated a subset of all available movies, allowing the system to test how it handles incomplete rating profiles.

Then we have the result:

```
My ratings for the 20 popular movies:
Movie: E.T. the Extra-Terrestrial (1982), Rating: 5
Movie: Star Wars Episode IV - A New Hope (1977), Rating: 4
Movie: Star Wars Episode V - The Empire Strike Back (1980), Rating: 3
Movie: Star Wars Episode VI - Return of the Jedi (1983), Rating: 2
Movie: Jurassic Park (1993), Rating: 1
Movie: Saving Private Ryan (1998), Rating: 5
Movie: Matrix, The (1999), Rating: 4
Movie: Back to the Future (1985), Rating: 1
Movie: Sience of the Lambs, the (1991), Rating: 1
Movie: Star Wars Episode I - The Phantom Menace (1999), Rating: 5
Movie: Raiders of the Lost Ark (1981), Raing: 4
Movie: Fargo (1996), Rating: 3
```

```
Movie: Sizth Sense, The (1999), Rating: 2
Movie: Braveheart (1995), Rating: 5
Movie: Shakespeare in Love (1998), Rating: 1
Movie: Princess Bride, The (1998), Rating: 3
Movie: Schindler's List (1993), Rating: 4
Movie: Shawshank Redemption, The (1994), Rating: 3
Movie: Groundhog Day (1993), Rating: 5
```

The resulting output displays the complete rating profile, showing movies and their corresponding ratings from the personalized vector. This data serves as the foundation for the personalized recommendation process that follows in subsequent sections.

9.2. ating Movies by using Myratings

a. Ensure myratings is a Row Vector

```
        % Ensure myratins is a row vector
        myratins = myratings(:)';

```

This code ensures that the myratings vector is formatted as a row vector to maintain compatibility with subsequent matrix operations. Since follow-up calculations (such as Euclidean distance or Pearson correlation) require input data to have consistent dimensions, this step prevents size mismatch errors during vector/matrix computations.

b. Filter Users with Complete Ratings

```
        % Personal Recommendations using myratins
        % Ensure myratings is a row vector
        myratings = myratins(:)';
        % Select the users to compare to
        [m1, n1] = size(users_movies_sort);
        ratings = [];
        for j = 1:m1
           if prod(users_movies_sort(j, :)) ~= 0
              ratings = [ratings; users_movies_sort(j, :)];
           end
        end

```

This script ensures data dimension consistency and removes users with missing ratings for more accurate calculations to prepare personal rating data and filter users with complete ratings for comparison.

c. Calculate Euclidean Distances

```
        % Find the Euclidean distance
        [m2, n2] = size(ratings);
        eucl = zeros(m2, 1);
```

```
       for i = 1:m2
          encl(i) = norm(ratings(i, :) - myratings);
       end
       [MinDist, DistIndex] = sort(eucl, 'ascend');
       closest_user_Dist = DistIndex(1);

```

Computes distances between personal rating vector (myratings) and all users in ratings matrix. Then sorts results in ascending order and stores the position of minimally distant user. The user with the smallest distance is identified via Euclidean distance measurement.

d. Center the Ratings

```
       % Find the Euclidean distance
       [m2, n2] = size(ratings);
       eucl = zeros(m2, 1);
       for i = 1:m2
          eucl(i) = norm(ratings(i, :) - myratings);
       end
       [MinDist, DisIndex] = sort(eucl, 'ascend');
       closest_user_Dist = DistIndex(1);

       % Centering the ratings
       ratigns_cent = ratings - mean(ratings, 2) * ones(1, n2);
       myratins_cent = myratings - mean(myratings);

```

Data is nomalized by mean-centering. We computes covariance (numerator) and standard deviations product (denominator) then stores correlation results in pearson vector. The purpose is measuring preference correlation between personal ratings and other users via Pearson coefficient.

e. Compute Pearson Correlation Coefficients

```
       pearson = zeros(m2, 1);

       for i = 1:m2
          % Extract the current row from ratings
          current_user = ratings(i, :);
          % Compute the means
          mean_current_user = mean(current_user);
          mean_myratings = mean(myratings);
          % Subtract the means from the vectors
          centered_current_user = current_user - mean_current_user;
          centered_myratings = myratins - mean_myratings;

          %compute the numerator and the denomination for the Pearson
    correlation coefficient
```

```
       numerator = sum(centerd_current_user .* centered_myratings);

       denominator = sqrt(sum(centered_current .^ 2) * sum(centered_myratings .^ 2));

       % compute the Pearson correlation coefficient
       if denominator ~= 0
          correlation = numerator / denominator;
       else
          correlation = 0;
       end

       % Store the result in teh pearson vector
       pearson(i) = correlation;
    end

    % Sort the pearson vector in pearson order
    [MaxPearson, PearsonIndex] = sort(pearson, 'descend');
    % Find the user with the highest Pearson correlation (index of the first element in PearsonIndex)
    closest_user_Pearson  = PearsonIndex(1);

```

This section identify linear relationships between ratings and generate recommendation list. Operation:

- Completes correlation coefficient calculation.
- Ensures stability with invalid data cases.
- Identifies users with most similar rating patterns.
- Generates recommendations from highest ratings.

9.3. Generate Recommendations

a. Based on Euclidean Distance

```
    % Recommendations based on myratings
    recommend_dist = [];
    for k = 1:n
       if users_movies(closest_user_Dist, k) == 5
          recommend_dist = [recommend_dist; k];
       end
    end

    recommend_Pearson = [];
    for k = 1:n
       if users_movies(closest_user_Pearson, k) == 5
          recommend_Pearson = [recommend_Pearson; k];
```

```
            end
        end

```

This loop generate movie recommendations based on two similarity measurement methods. To do this, we iterate through all movies (n movies). For each method (Euclidean and Pearson): filter movies rated 5 stars by the most similar user and store qualifying movie indices in recommendation arrays.

b. Identify Movies Liked by You

```
        liked = [];
        for k = 1:20
            if myratings(k) == 5
                liked = [liked; index_small(k)];
            end
        end

```

Generate and display recommended movies along with liked movies, operation:

- Create recommendation lists:
  - + `recommend_Pearson`: Movies rated 5 stars by Pearson-similar user.
  - + `liked`: Movies personally rated 5 stars.
- Display results:
  - + Personally liked movies list.
  - + Recommendation lists from both methods (Euclidean and Pearson).

9.4. Display the result

Finally, we assign topic labels to the rated movies:

- Movies liked by users (rated 5 star).
- Recommendation based on Euclidean distance.
- Recommendation based on Pearson correlation.

```
    % Display recommendations and Liked Moives
    disp('Movies liked by user (myratings): ');
    for i = 1:length(liked)
        fprintf('%s \n', movies{liked(i)});
    end

    disp('Recommendations based on Eculidean distance (myratings): ');
    for i = 1:length(recommend_dist)
```

```
        fprintf('%s \n', movies{recommend_dist(i)});
    end

    disp('Recommendations based on Pearson correlation (myratins): ');
    for i = 1:length(recommend_Pearson)
        fprintf('%s \n', movies{recommend_Pearson(i)});
    end
```

The list shown below includes movies that user likes and the recommendations for new users who have not seen these movies. Additionally, they can view the movie rankings from 1 to 5.

```
Movies liked by user (myratings):
E.T. the Extra-Terrestrial (1982)
Saving Private Ryan (1998)
Terminator 2
Star Wars Episode I - The Phantom Menace (1999)
Braveheart (1995)
Groundhog Day (1993)

Recommendations based on Eculidean distance (myratings):
Braveheart (1995)
Frequency (2000)
Gladiator (2000)
Me, Myself and Irene (2000)

Recommendations based on Pearson correlation (myratins):
Manhattan Murder Mystery (1993)
Close Shave, A (1995)
For The Moment (1994)
Monty Python's Life of Brian (1979)
Monty Python and the Holy Grail (1974)
Princess Bride, The (1987)
Rosencrantz and Guildenstren Are Dead (1990)
West Side Story (1961)
Metropolitan (1961)
Metropolitan (1990)
Roger & Me (1989)
Say Anything... (1989)
Producers, The (1968)
Life Is Beautiful (La Vita e bella) (1997)
Shaarkespeare in Love (1998)
Ideal Husband, An (1999)
Radio Days (1987)
American Beayty (1999)
Boys Don't Cry (1999)
Being John Malkovich (1999)
'Night Mother (1986)
Muppet Movie, The (1979)
Auntie Mame (1958)
```

```
Boricua's Bond (2000)
```

IV. SUMMARY AND CONCLUSION

In this project, we used linear algebra techniques to build a basic movie recommendation system based on the MovieLens dataset. The dataset included a matrix of user ratings, movie titles, and a ”`trial_user`” who rated the 20 most popular movies.

We first filtered users who had rated all 20 popular movies by using “rating” array, then selecting users who gave non-zero ratings to all movies and calculated the similarity between each user and the trial user using two different approaches: Euclidean distance and Pearson correlation coefficient. The Euclidean method identified the user with the most similar rating magnitudes, while the Pearson method found the user whose rating pattern most closely matched the trial user’s preferences.

In the Euclidean distance between the ”`trial_user`” and these users was calculated. User 14 had the closest ratings based on this method. Next, the Pearson correlation coefficient was calculated to measure how similar the rating patterns were between the ”`trial_user`” and others. User 88 had the closest match using this method.

Since the results were different, it showed that while user 14 had similar rating magnitudes (Euclidean), user 88 had more similar preferences (Pearson). Based on these closest users, two sets of movie recommendations were made, along with a list of movies already liked by the ”`trial_user`.”

The results showed that the two approaches selected different users as the most similar, emphasizing how each metric captures a different aspect of similarity: Euclidean distance reflects the absolute differences in rating values, whereas Pearson correlation considers the overall trend or pattern in preferences.

Based on the closest users found by each method, we generated two sets of movie recommendations. Additionally, we created a personalized rating vector (”Myratings”) to simulate a new user and repeated the recommendation process.

In the end, three lists were provided: movies liked by the user, recommendation based on Euclidean distance and recommendation beased on Pearson correlation.

In conclusion, this project demonstrates the importance of choosing an appropriate similarity metric in recommendation systems. While Euclidean distance may be suitable when matching the intensity of preferences, Pearson correlation is more effective for identifying users with similar taste trends. Selecting the right method depends on the recommendation goal—whether to match how much users like something or how similarly they rate different items.

V. REFERENCES

[1] Dynamsoft, “Image Processing 101: Spatial Filters and Convolution,” [Online]. Available: https://www.dynamsoft.com

[2] L. Garcia and C. Penland, *MATLAB Projects for Scientists and Engineers.* Upper Saddle River, NJ: Prentice Hall, 1996.

[3] University of South Florida, “Chapter 5: Edge Detection,” [Online]. Available: https://www.cse.usf.edu

[4] Demofox, “Image Sharpening – Convolution Kernels,” [Online]. Available: https://blog.demofox.org

[5] The University of Edinburgh, “Spatial Filters – Mean Filters,” [Online]. Available: https://homepages.inf.ed.ac.uk

[6] “Edge Detection Using Laplacian,” YouTube, [Online]. Available: https://www.youtube.com

[7] G. Strang, *Linear Algebra and Its Applications.* 4th ed., Brooks/Cole, 2006.