VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE


---

## Assignment Report

# LINEAR ALGEBRA

---

| | | |
|---|---|---|
| Advisor: | Dau The Phiet | |
| Members: | Tran Dinh Dang Khoa | 2211649 |
| | Chau Vinh Ky | 2211785 |
| | Pham Tan Khoa | 2252359 |
| | Pham Tan Phong | 2252614 |
| | Nguyen Khac Viet | 2252904 |
| | Nguyen Hoang Ngoc Tuan | 2252873 |

HO CHI MINH CITY, MAY 2023

# Contents

# I Problem 1.

A code breaker intercepted the encoded message below.
45 -35 38 -30 18 -18 35 -30 81 -60 42 -28 75 -55 2 -2 22 -21 15 -10

Let the inverse of the encoding matrix be $A^{-1} = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$

(a) Given that $[45 \quad -35]A^{-1} = [10 \quad 15]$ and $[38 \quad -30]A^{-1} = [8 \quad 14]$. Write and solve two systems of equations to find $w, x, y$, and $z$.

(b) Decode the message.

**Theory:**

This problem involves decoding a message that has been encoded using a matrix transformation.
To decode the message, we need to multiply each encoded vector by the inverse of A. The inverse of A can be computed using standard methods of matrix inversion, such as Gaussian elimination.

**Handwritten solution:**

We have:

$$\begin{bmatrix} 45 & -35 \end{bmatrix} \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} 45w - 35y \\ 45x - 35z \end{bmatrix}^T = \begin{bmatrix} 10 \\ 15 \end{bmatrix}^T$$

$$\begin{bmatrix} 38 & -30 \end{bmatrix} \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} 38w - 30y \\ 38x - 30z \end{bmatrix}^T = \begin{bmatrix} 8 \\ 14 \end{bmatrix}^T$$

From the above equations, we can write the following system of equations:

$$\begin{cases} 45w - 35y = 10 \\ 45x - 35z = 15 \\ 38w - 30y = 8 \\ 38x - 30z = 14 \end{cases}$$

Solve the system of equations using Gaussian Elimination, we get:

$$\begin{cases} w = 1 \\ x = -2 \\ y = 1 \\ z = -3 \end{cases}$$

Therefore, the inverse of A is:

$$A^{-1} = \begin{bmatrix} 1 & -2 \\ 1 & -3 \end{bmatrix}$$

After that, we can decode the message by multiplying each encoded vector by the inverse of A.

**Python code:**

```
import numpy as np

# Define the encoded message as a numpy array
encodedMSG = np.array ( [[45, -35, 38, -30, 18, -18, 35, -30, 81, -60,
    42, -28, 75, -55, 2, -2, 22, -21, 15, -10]] )

# Define the known vectors [45 -35]A^-1 and [38 -30]A^-1 as numpy arrays
v1 = np.array ( [[10, 15],  [8, 14]] )
v2 = np.array ( [[45, -35], [38, -30]] )

# Solve the systems of equations to find w, x, y, and z
[[w, x], [y, z]] = np.linalg.solve (v2, v1)

# Round w, x, y, and z to integers and print them
w, x, y, z = map (int, np.round([w, x, y, z]))
print("w = ", w)
print("x = ", x)
print("y = ", y)
print("z = ", z)

#Shape the encoded message into a 2x10 matrix
encodedMSG = encodedMSG.reshape(10,2)

# Define the encoding matrix A as a numpy array
A_inv = np.array([[w, x], [y, z]])
```

```
25
26 # Decode the message using the inverse of the encoding matrix A^-1
27 decodedMSG = np.dot(encodedMSG,A_inv).astype(int)
28 #mod 26 to get the correct values for the message
29 decodedMSG = decodedMSG % 26
30
31
32 # Turn the decoded message into a string
33 result = ''
34 for row in decodedMSG:
35     for val in row:
36         result += chr (val + 64) # Convert the integer value to its
    corresponding ASCII character
37 result = result.replace ('@',' ') # Replace any '@' with ' ' (space)
38
39 # Print the decoded message
40 print ("Decoded Message: ", result)
```

**Result:**

```
40  print ("Decoded Message: ", result)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\Documents\VS CODE\LaTeX\Report\Linear Algebra\code> python problem1.py
w =  1
x =  -2
y =  1
z =  -3
Decoded Message:  JOHN RETURN TO BASE
PS D:\Documents\VS CODE\LaTeX\Report\Linear Algebra\code>
```

Console output of *problem1.py*

# II Problem 2.

Construct an inner product in $\mathbb{R}^n$. In that inner product, write a program to input any number of vectors in $\mathbb{R}^n$ and return the orthogonal basis and orthonormal basis of the subspace spanned by these vectors. (Use Gram - Schmidt process). From that, given any vector in $\mathbb{R}^n$, find the coordinates in that basis and find the length of the vector.

**Theory:**

An inner product in $\mathbb{R}^n$ is a function that takes two vectors as input and returns a scalar. Mathematically, the inner product of two vectors $\vec{u}$ and $\vec{v}$ in $\mathbb{R}^n$ is defined as:

$$\langle u, v \rangle = u_1 v_1 + u_2 v_2 + ... + u_n v_n$$

An orthogonal basis for a vector space is a set of vectors that are mutually orthogonal, meaning that every pair of vectors in the set is orthogonal (their dot product is zero).

An orthonormal basis for a vector space is a set of vectors that are both orthogonal and normalized. In other words, an orthonormal basis is an orthogonal basis where all vectors have unit length.

The **Gram-Schmidt process** is a method for orthonormalizing a set of vectors in an inner product space, most commonly the Euclidean space Rn equipped with the standard inner product. The Gram-Schmidt process takes a finite, linearly independent set of vectors $S = \{v_1, ..., v_k\}$ for $k \leq n$ and generates an orthogonal set $S' = \{u_1, ..., u_k\}$ that spans the same $k$-dimensional subspace of $\mathbb{R}^n$ as $S$.

In order to solve this problem, we need to first prompt the user to enter the dimension of the vector space and the vectors. Then, we need to implement the Gram-Schmidt process to find the orthogonal basis and orthonormal basis of the subspace spanned by the input vectors.

The simplest and most common inner product for any subspace is the Dot Product. Therefore, we will use the Dot Product as the inner product for this problem.

**Python code:**

```python
import numpy as np

def main():
    # Prompt user for input
    n           = int(input("Enter the dimension of the vector space: "))
    num_vectors = int(input("Enter the number of vectors: "))
    vectors     = np.zeros((num_vectors, n))

    # Read vectors from user input
    for i in range(num_vectors):
        print(f"Enter vector {i+1}:")
        vectors[i] = np.array(input().split(), dtype=float)

    # Perform Gram-Schmidt orthogonalization
    orthogonal_basis, orthonormal_basis = gram_schmidt(vectors)

    # Print the orthogonal basis
    print("\nOrthogonal Basis:")
    for i in range(num_vectors):
        print(f"Vector {i+1}: {orthogonal_basis[i]}")

    # Print the orthonormal basis
    print("\nOrthonormal Basis:")
    for i in range(num_vectors):
        print(f"Vector {i+1}: {orthonormal_basis[i]}")

    # Find the coordinates of a vector in the orthonormal basis
    input_vector = np.array(input("\nEnter a vector in R^n to find its coordinates in the basis: ").split(), dtype=float)
    coordinates  = find_coordinates(orthonormal_basis, input_vector)

    # Print the coordinates of the vector in the orthonormal basis
    print("\nCoordinates of the vector in the orthonormal basis:")
    for i in range(num_vectors):
        print(f"Coordinate {i+1}: {coordinates[i]}")

    # Calculate the length of the input vector
    vector_length = np.linalg.norm(input_vector)
    print(f"\nLength of the vector: {vector_length}")

def gram_schmidt(vectors):
```

```
    num_vectors, n    = vectors.shape
    orthogonal_basis  = np.zeros_like(vectors)
    orthonormal_basis = np.zeros_like(vectors)

    for i in range(num_vectors):
        orthogonal_basis[i] = vectors[i]
        for j in range(i):
            # Calculate the projection of vectors[i] onto orthogonal_basis[j]
            projection = np.dot(vectors[i], orthogonal_basis[j]) / np.dot(orthogonal_basis[j], orthogonal_basis[j])
            orthogonal_basis[i] -= projection * orthogonal_basis[j]
        # Normalize the orthogonal basis vector to obtain the orthonormal basis vector
        orthonormal_basis[i] = orthogonal_basis[i] / np.linalg.norm(orthogonal_basis[i])

    return orthogonal_basis, orthonormal_basis

def find_coordinates(orthonormal_basis, vector):
    num_vectors, n = orthonormal_basis.shape
    coordinates    = np.zeros(num_vectors)

    for i in range(num_vectors):
        # Calculate the dot product of the input vector with each orthonormal basis vector
        coordinates[i] = np.dot(vector, orthonormal_basis[i])

    return coordinates

main()
```

**Result:**

```
PS D:\Documents\VS CODE\LaTeX\Report\Linear Algebra\code> python problem2.py
Enter the dimension of the vector space: 3
Enter the number of vectors: 3
Enter vector 1:
1 0 0
Enter vector 2:
1 1 0
Enter vector 3:
1 1 1

Orthogonal Basis:
Vector 1: [1. 0. 0.]
Vector 2: [0. 1. 0.]
Vector 3: [0. 0. 1.]

Orthonormal Basis:
Vector 1: [1. 0. 0.]
Vector 2: [0. 1. 0.]
Vector 3: [0. 0. 1.]

Enter a vector in R^n to find its coordinates in the basis: 1 2 3

Coordinates of the vector in the orthonormal basis:
Coordinate 1: 1.0
Coordinate 2: 2.0
Coordinate 3: 3.0

Length of the vector: 3.7416573867739413
PS D:\Documents\VS CODE\LaTeX\Report\Linear Algebra\code>
```

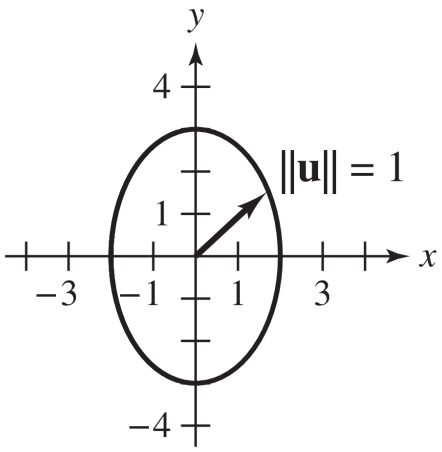
Console output of *problem2.py*

# III Problem 3.

In $\mathbb{R}^2$, the weighted inner product is given by

$$\langle x, y\rangle = ax_1y_1 + bx_2y_2$$

where $a$ and $b$ are positive. Find a weighted inner product such that the graph represents a unit circle as




In that inner product space, reflect that unit circle about an input plane.

**Theory:**

Weighted inner products have exactly the same algebraic properties as the "ordinary" inner product. But they introduce weights or importance factors that modify the calculations and geometric interpretations.

Let's consider a vector space $V$ over a field $F$. A weighted inner product on $V$ is a function that assigns a scalar value to each pair of vectors in $V$, incorporating weights or importance factors. Formally, a weighted inner product is defined as:

$$\langle \cdot, \cdot \rangle_w : V \times V \to F$$

where $\langle \cdot, \cdot \rangle_w$ represents the weighted inner product, and $V \times V$ denotes the Cartesian product of $V$ with itself.

In a weighted inner product, each component of the vector is multiplied by a corresponding weight or importance factor before the dot product is calculated. This allows for a

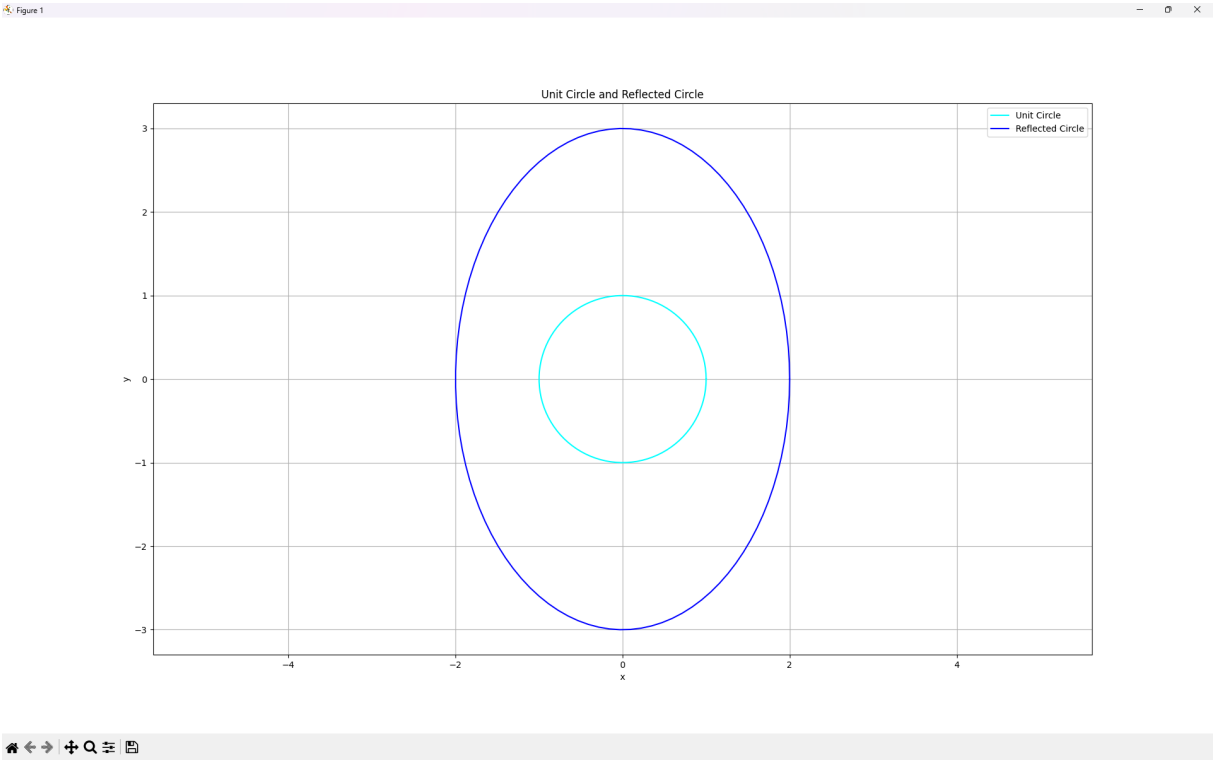
more flexible and nuanced treatment of vector spaces, where certain components or dimensions may carry more significance or contribute differently to the overall computation.

In order to solve this problem, we need to first find the weighted inner product such that the graph represents a unit circle. Then, we need to reflect that unit circle about an input plane.

**Python code:**

```
import numpy as np
import matplotlib.pyplot as plt

theta = np.linspace(0, 2*np.pi, 100)

# Define the unit circle
x = np.cos(theta)
y = np.sin(theta)

# Define the reflection constants
a = 2
b = 3

# Define the refl inner product
x_refl = a * np.cos(theta)
y_refl = b * np.sin(theta)

# Plot the original and refl unit circles
plt.figure(figsize=(8, 8))
plt.plot(x, y, color='cyan', label='Unit Circle')
plt.plot(x_refl, y_refl, color='blue', label='Refl Circle')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Unit Circle and Reflected Circle')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

**Result:**




Unit Circle and its reflection about the input plane

# IV Conclusion

This assignment has been instrumental in our exploration of the fundamental principles of Linear Algebra. It has offered us a platform to delve into various key concepts, including matrices, vectors, systems of linear equations, and inner product spaces, all within the context of utilizing Python as a powerful computational tool.

Throughout the assignment, we have actively engaged with exercises and Python problems, allowing us to solidify our understanding of these foundational concepts. By applying these concepts to real-world scenarios, we have honed our ability to solve practical problems using the tools and techniques of Linear Algebra.

This assignment has not only expanded our knowledge base but has also contributed to the development of our mathematical skills. By working through the exercises and Python problems, we have sharpened our ability to analyze and interpret mathematical structures and relationships. The hands-on nature of using Python has further strengthened our computational thinking and problem-solving abilities.

Overall, this assignment has provided us with a valuable opportunity to deepen our understanding of Linear Algebra, develop proficiency in utilizing Python for mathematical computations, and enhance our mathematical skills in a practical context.

# References

[1] Howard Anton and Chris Rorres. *Elementary Linear Algebra*. John Wiley & Sons, New York, 2013.

[2] Ron Larson and David C. Falvo. *Elementary Linear Algebra*. Brooks Cole, Boston, 2008.

[3] Dang Van Vinh. *DAI SO TUYEN TINH*. NXB DHQG, 2019.