

Chapter 5

1. Introduction

In this chapter you will learn how to use MATLAB to work with lengths, angles and projections in subspaces of \mathfrak{R}^n and later in certain linear spaces.

2. Lengths and Angles

Recall from class that the dot product is your key for understanding lengths and angles.

The best way to find the dot product of $\vec{u} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$ in MATLAB, is to do the

following:

```
>> u = [2; 3; 4];  
>> v = [1; 4; 9];  
>> dot(u, v)
```

Recall that the length of \vec{u} is denoted by $\|\vec{u}\|$ calculated by computing $\sqrt{\vec{u} \cdot \vec{u}}$ and if θ is the angle between \vec{u} and \vec{v} , then $\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos \theta$.

Here are some other MATLAB commands that will help you get through this section.

```
>> sqrt(49)      %finds the square root of 49  
>> acos(3/5)    %finds the inverse cosine of 3/5
```

Problems: (Do not use the MATLAB function norm)

2a. 5.1 #2

2b. 5.1 #5

3. Orthogonal Projections

In this section you will use MATLAB to project a vector onto a subspace. For now the subspace will be defined by an orthonormal basis. In section 7, you will see another way to do this even if you do not have an orthonormal basis.

Here is the key idea: Assume V is a subspace of \mathfrak{R}^n , $(\bar{u}_1, \bar{u}_2, \bar{u}_3, \dots, \bar{u}_n)$ is an orthonormal basis for V , and \bar{x} is any vector in \mathfrak{R}^n . Since the orthogonal projection of \bar{x} onto V is in V , and we have a basis for V , we only need to find the correct coefficients to use to build the projection. More specifically, we need to find the correct c_1, c_2, \dots, c_n such that $proj_V(\bar{x}) = c_1\bar{u}_1 + c_2\bar{u}_2 + \dots + c_n\bar{u}_n$. Here is the secret: $c_1 = \bar{x} \cdot \bar{u}_1, c_2 = \bar{x} \cdot \bar{u}_2, \dots, c_n = \bar{x} \cdot \bar{u}_n$

Here is how you would do 5.1 #28

Find the orthogonal projection of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
onto the subspace of \mathfrak{R}^4 spanned by

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

```
>> v1 = [1; 1; 1; 1];
>> v2 = [1; 1; -1; -1];
>> v3 = [1; -1; -1; 1];
>> u1 = (1/norm(v1))*v1;           %Normalize the vectors
>> u2 = (1/norm(v2))*v2;
>> u3 = (1/norm(v3))*v3;
>> x = [1;0;0;0];
>> c1 = dot(x,u1);                 %Calculate the coefficients
>> c2 = dot(x,u2);
>> c3 = dot(x,u3);
>> proj = c1*u1 + c2*u2 + c3*u3; %Calculate the projection
>> rats(proj) %Display projection with rational numbers
```

ans =

```
    3/4
    1/4
   -1/4
    1/4
```

So, the orthogonal projection of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ onto the subspace spanned by $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$, $\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$ is $\begin{bmatrix} 3/4 \\ 1/4 \\ -1/4 \\ 1/4 \end{bmatrix}$.

Problems:

3a. 5.1 #26

3b. 5.1 #29 (Hint: Look at Fact 5.1.9)

4. Roll Your Own Orthonormal Basis

The Gram-Schmidt process is a way to turn any basis in \mathfrak{R}^n into an orthonormal basis. We are going to start with a basis $(\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n)$ and turn it into the orthonormal basis $(\vec{u}_1, \vec{u}_2, \dots, \vec{u}_n)$ that spans the same space. Here is the basic idea:

- 1) If \vec{v}_1 is not a unit vector, divide by its magnitude to turn it into a vector with length 1. Call this new vector \vec{u}_1 .
- 2) \vec{v}_2 has a component parallel to \vec{u}_1 and a component perpendicular to \vec{u}_1 . To get \vec{u}_2 , just normalize the perpendicular component.
- 3) \vec{v}_3 has a component in the subspace spanned by (\vec{u}_1, \vec{u}_2) and a component orthogonal to the subspace. \vec{u}_3 is the normalized perpendicular component.

If you keep doing step 3) on all of your vectors, you will end up with an orthonormal basis that spans the same space.

Here is how you would do this in MATLAB starting with the basis $\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)$.

MATLAB trick: The `subplot` command lets you put several graphs in the same window.

The following is somewhat long, but worth typing in.

```

>> v1 = [1;-2];
>> v2 = [2;3];
>> origin = [0;0];
>> picture_1 = [v1 origin v2];
>> subplot(2,2,1);
>> plot(picture_1(1,:),picture_1(2,:), 'LineWidth',3);
>> axis([-3 3 -3 3]);
>> axis('square');
>> u1 = (1/norm(v1))*v1;
>> picture_2 = [u1 origin v2];
>> subplot(2,2,2);
>> plot(picture_2(1,:),picture_2(2,:), 'LineWidth',3);
>> axis([-3 3 -3 3]);
>> axis('square');
>> v2_parallel = dot(v2,u1)*u1;
>> v2_perp = v2 - v2_parallel;
>> picture_3 = [u1 origin v2_perp];
>> subplot(2,2,3);
>> plot(picture_3(1,:),picture_3(2,:), 'LineWidth',3);
>> axis([-3 3 -3 3]);
>> axis('square');
>> u2 = (1/norm(v2_perp))*v2_perp;
>> picture_4 = [u1 origin u2];
>> subplot(2,2,4);
>> plot(picture_4(1,:),picture_4(2,:), 'LineWidth',3);
>> axis([-3 3 -3 3]);
>> axis('square');>> u1 %see the value of u1

```

```

u1 =

    0.4472
   -0.8944

```

```
>> u2 %see the value of u2
```

```

u2 =

    0.8944
    0.4472

```

The new basis is $\left(\begin{bmatrix} .4472 \\ -.8944 \end{bmatrix}, \begin{bmatrix} .8944 \\ .4472 \end{bmatrix} \right)$.

Problems:

4a. 5.2 #14 (you do not need to plot your vectors for this one)

4b. 5.2 #29

4c. 5.2 #32

5. Transposes and Symmetry

Oftentimes, given a matrix, it is useful to make a new matrix whose columns are the rows of the original matrix. This new matrix is called the transpose of the original matrix.

More formally, the entry in the i^{th} row of the j^{th} column of A is the j^{th} row and i^{th} column of the transpose of A . The `'` command makes this easy to do in MATLAB.

```
>> A = [1 2 3; 9 7 5]
```

```
A =
```

```
     1     2     3
     9     7     5
```

```
>> A'
```

```
ans =
```

```
     1     9
     2     7
     3     5
```

Recall

1) A is a symmetric matrix iff $A = A^T$

2) A is a skew-symmetric matrix iff $A = -A^T$

Problems:

Use symbolic 2x2 matrices in MATLAB to answer the following problems. Then generalize (without MATLAB) to the $n \times n$ case.

Hints: $(AB)^T = B^T A^T$. $(A^T)^T = A$ Here is how to prove that $A^T A$ is symmetric:

$$(A^T A)^T = A^T (A^T)^T = A^T A$$

5a. 5.3 #22

5b. 5.3 #24

6. Orthogonal Matrices

There are several things you know about orthogonal transformations, and you can use all of them to help decide whether a given matrix is orthogonal.

What you know:

1) An orthogonal transformation preserves vector lengths (that is, T is orthogonal iff $\|T(\bar{x})\| = \|\bar{x}\|$ for all \bar{x})

2) An orthogonal transformation preserves angles between vectors.

3) The columns of the matrix form an orthonormal basis, which means that $A^T A = I$. (Do you see why? Look at Fact 5.3.7)

Here is how to use MATLAB to check each of these properties. Note that 1) and 2) can only be used to show that a transformation is not orthogonal, while 3) can be used to show that a matrix is orthogonal and that can be used to show that a matrix is not orthogonal.

1) Use property 1:

```
>> A = [1 2; 2 1];  
>> x = [1;2];  
>> norm(x)
```

ans =

2.2361

```
>> norm(A*x)
```

ans =

6.4031

So, $\|\bar{x}\| \neq \|A\bar{x}\|$, and hence $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ is not orthogonal.

2) Use property 2: (This is 5.3 #4)

```
>> A = (1/7)*[2 6 -3; 6 -3 2; 3 2 6];  
>> v1 = [1;2;3];  
>> v2 = [3;2;1];  
>> acos(dot(v1,v2)/(norm(v1)*norm(v2)))
```

ans =

0.7752

```
>> acos(dot(A*v1,A*v2)/(norm(A*v1)*norm(A*v2)))
```

ans =

0.5244

So, then angle between \bar{v}_1 and \bar{v}_2 is not the same as the angle between $A\bar{v}_1$ and $A\bar{v}_2$.

Hence $\frac{1}{7} \begin{bmatrix} 2 & 6 & -3 \\ 6 & -3 & 2 \\ 3 & 2 & 6 \end{bmatrix}$ is not orthogonal, since $A^T A = I$.

3) Use property 3: (This is 5.3 #3)

```
>> A = (1/3)*[2 -2 1; 1 2 2; 2 1 -2];
```

```
>> A'*A
```

ans =

```
1 0 0
0 1 0
0 0 1
```

So $\frac{1}{3} \begin{bmatrix} 2 & -2 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & -2 \end{bmatrix}$ is orthogonal.

Problems:

6a. 5.3 #1

6b. 5.3 #2

6c. 5.3 #52

7. Least Squares Solutions and Another Way to Calculate Orthogonal Projections.

In a perfect world, every set of equations would be consistent. Unfortunately, our world is not perfect. Luckily, a theory has been developed to come up with the “best” solution to an inconsistent set of equations. More specifically, if $A\bar{x} = \bar{b}$ is inconsistent, we solve for $A\bar{x} =$ the projection of \bar{b} onto $\text{im}(A)$. A way of finding \bar{x} is given below. Once you know \bar{x} , computing $A\bar{x}$ will give you the projection.

We will do 5.4, example 1 in MATLAB. The set of equations we want to solve is

$$\begin{aligned}x + y &= 0 \\x + 2y &= 0 \\x + 3y &= 6\end{aligned}$$

Note that this system is inconsistent.

We will also use the “best” solution to orthogonally project $\begin{bmatrix} 0 \\ 0 \\ 6 \end{bmatrix}$ onto $\text{im}(A)$.

```
>> A = [1 1 ; 1 2; 1 3];
>> b = [0; 0; 6];
>> x = inv(A'*A)*A'*b
```

```
x =
-4.0000
 3.0000
>> A*x
```

```
ans =
-1.0000
 2.0000
 5.0000
```

So the least squares solution is $\bar{x}^* = \begin{bmatrix} -4 \\ 3 \end{bmatrix}$ and the orthogonal projection of $\begin{bmatrix} 0 \\ 0 \\ 6 \end{bmatrix}$ onto the

image of $\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ is $\begin{bmatrix} -1 \\ 2 \\ 5 \end{bmatrix}$

Problems:

7a. 5.4 #20 (You can use MATLAB instead of a paper and pencil)

7b. 5.4 #22

8. Data Fitting

In this section you will use section 7 to find the best equation to fit some given data. Here is 5.4 #32:

We are trying to fit a linear function of the form $f(t) = c_0 + c_1t$ to the data points (0, 3), (1, 3) and (1, 6). In other words, we want to find a solution to the following equations:

$$3 = c_0 + c_1 \cdot 0 \quad (\text{since we want the point } (0, 3) \text{ on the line})$$

$$3 = c_0 + c_1 \cdot 1 \quad (\text{since we want the point } (1, 3) \text{ on the line})$$

$$6 = c_0 + c_1 \cdot 1 \quad (\text{since we want the point } (1, 6) \text{ on the line})$$

In matrix form,
$$\begin{bmatrix} 3 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

```
>> A = [1 0; 1 1; 1 1];  
>> b = [3; 3; 6];  
>> x = inv(A'*A)*A'*b
```

x =

```
3.0000  
1.5000
```

So our linear function is $f(t) = 3 + 1.5t$. We will use MATLAB to see how good it is. Remember that our three points are (0, 3), (1, 3) and (1, 6).

MATLAB trick: The `scatter` command will plot points without connecting them.

```
>> points = [0 1 1; 3 3 6]
```

points =

```
0    1    1  
3    3    6
```

```
>> scatter(points(1,:), points(2,:), 10, 'filled');  
>> axis([-1 2 2 7])    %Set the axis  
>> hold on            %Stop MATLAB from erasing our graph
```

```
>> t = linspace(-1,2); %Set up domain for linear function
>> y = 3+1.5*t;        %Define linear function
>> plot(t,y);         %Be sure to look at the graph
```

More problems:

8a. 5.4 #30

8b. 5.4 #32

Hint: Here is how to plot the quadratic function $y = -2t^2 + 3t + 5$ on the interval $[-2, 2]$

```
>> t = linspace(-2,2);
>> y = -2*t.^2 + 3*t + 5; Note the . in front of the ^
>> plot(t,y);
```

8c. 5.4 #38

9. Inner Product Spaces

This is a good place to look back and see what you've accomplished in the last couple of chapters. Here's a recap:

Chapter 3: Useful concepts for \mathfrak{R}^n , including linear independence, dimension and subspace, among others.

Chapter 4: The Chapter 3 concepts generalized to arbitrary linear spaces.

5.1 – 5.3: dot products, lengths, angles, and orthogonal projections for \mathfrak{R}^n .

Now we are going to look at dot products, lengths, angles, and orthogonal projections for some spaces other than \mathfrak{R}^n . The basic idea is that given a linear space, we make up a rule that has the same properties as the dot product on \mathfrak{R}^n . This rule is called an inner product.

Here is 5.5, example 5: The space is $C[0, 2\pi]$ and we define $\langle f, g \rangle = \int_0^{2\pi} f(t)g(t) dt$.

Some notes about MATLAB before we do this:

- 1) `trapz` is the MATLAB command to use the trapezoid rule of integrating.
- 2) MATLAB is subject to round off errors. If MATLAB reports that something is close to 0, it is probably actually equal to 0 for the applications we are doing.

```
>> t = linspace(0,2*pi);
>> trapz(t,sin(t).*cos(t))
```

ans =

```
4.0332e-017
```

The last line says that $\int_0^{2\pi} f(t)g(t)dt = 4.0332 \times 10^{-17} = -.0000000000000000040332$, which is close enough to 0 for us to call it equal to 0.

Problems:

9a. 5.5 #10

9b. 5.5 #20

9c. 5.5 #24

10. Orthogonal Projections

Once you have an inner product, you can do orthogonal projections like you did in section 3 above. We will go through 5.5, example 8. We are using the formula from Fact

5.5.5: $f_n(t) = a_0 \frac{1}{\sqrt{2}} + b_1 \sin(t) + c_1 \cos(t) + \dots + b_n \sin(nt) + c_n \cos(nt)$

where $a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f \frac{1}{\sqrt{2}} dt$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin(nt) dt$$

$$c_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos(nt) dt$$

```
>> t = linspace(-pi,pi);           %set up the domain
>> f = t;

                                   %define the function

>> %calculate the coefficients
>> a0 = (1/pi)*trapz(t, 1/sqrt(2)*f);
>> b1 = (1/pi)*trapz(t, sin(t).*f);
>> c1 = (1/pi)*trapz(t, cos(t).*f);
>> b2 = (1/pi)*trapz(t, sin(2*t).*f);
>> c2 = (1/pi)*trapz(t, cos(2*t).*f);
>> b3 = (1/pi)*trapz(t, sin(3*t).*f);
>> c3 = (1/pi)*trapz(t, cos(3*t).*f);
>> b4 = (1/pi)*trapz(t, sin(4*t).*f);
>> c4 = (1/pi)*trapz(t, cos(4*t).*f);
```

```

>> %compute the approximations
>> f1 = a0*(1/sqrt(2)) + b1*sin(t) + c1*cos(t)

>> f2 = a0*(1/sqrt(2)) + b1*sin(t) + c1*cos(t) +
      b2*sin(2*t) + c2*cos(2*t);

>> f3 = a0*(1/sqrt(2)) + b1*sin(t) + c1*cos(t) +
      b2*sin(2*t) + c2*cos(2*t) +
      b3*sin(3*t) + c3*cos(3*t);

>> f4 = a0*(1/sqrt(2)) + b1*sin(t) + c1*cos(t) +
      b2*sin(2*t) + c2*cos(2*t) +
      b3*sin(3*t) + c3*cos(3*t) +
      b4*sin(4*t) + c4*cos(4*t);

>>%graph the approximations with the original function
>> subplot(2,2,1); plot(t,f); hold on; plot(t,f1);
>> subplot(2,2,2); plot(t,f); hold on; plot(t,f2);
>> subplot(2,2,3); plot(t,f); hold on; plot(t,f3);
>> subplot(2,2,4); plot(t,f); hold on; plot(t,f4);

```

Problems:

10a. 5.5 #12

10b. 5.5 #26