Jamie Ash

Homework 2

Question 1. Let V be a vector space over F and U, $W \subset V$ be subspaces. Explain why $U + W$ and $U \cap W$ are subspaces of V. Give an example of a vector space V , and two subspaces W_1 , W_2 such that $W_1 \cup W_2$ is not a subspace. Next, give an example where it is a subspace. Can you figure out a general condition for when W_1∩W_2$ is a subspace?

Let $x, y \in U$ and $z, t \in W$ then $(x + z), (x + t), (y + z), (y + t) \in U + W$.
For $U + W$ to be a subspace, it must meet the two rules of subspaces. So, we need to show that...

> (*i*) $(x + z) + (y + t) \in U + W$
> (*ii*) for every $k \in U + W : ck \in U + W$ where $c \in F$ (this includes $c = 0$)

Take any element $(x + z) + (y + t) \in U + W$. We can rearange the parentheses to produce different two-element sums...

> (x + t) + (y + z)
> (x + y) + (t + z)

I would like to give special attention to the second line. We can always rearange the parenthasis to make the two-element sums be in either U or W. So those two-element summations can be carried out, and the result will be in either U or W. This is because we know addition is defined in U and W. Lets call those two-element sums k and j, given by $(x + y) \in U$ and $(t + z) \in W$ respectivly. Then ($k + j$) will be in $U + W$, because $k \in U$ and $j \in W$. So no matter what two elements we take from $U + W$, their addition will be defined.

For scaler multiplication, we take a similar aprouce. Take any element $(x + z) + (y + t) \in U + W$.
We can first distribute the scalear, $c \in F$, then we can rearange the parentheses to produce different two-element sums...

> $c * ((x + z) + (y + t))$ =
> $(cx + ct) + (cy + cz) = c(x + t) + c(y + z)$
> or
> $(cx + cy) + (ct + cz) = c(x + y) + c(t + z)$

From here the logic follows the same path as defining addition in the above paragraph.
So then $U + W$ is a subspace.
■

Now for $U \cap W$. Let $x, y \in U$ and $x, z \in W$ so that $x \in U \cap W$.
For $U \cap W$, again we need to show that $U \cap W$ meets the two rules of subspaces...

> (*i*) for all $x, y \in U \cap W$: $y + x \in U \cap W$
> (*ii*) for every $x \in U \cap W$: $cx \in U \cap W$ where $c \in F$ (this includes $c = 0$)

For any $x \in U \cap W$ means $x \in U$. Because U is a subspace, all elements in U meet the two rules of subspaces. All element in $U \cap W$ being in U means $U \cap W \subset U$, and all the elemets in $U \cap W$ must also meet the two rules of vector subspaces. So then $U \cap W$ is a vector subspace.
■

As an example of two substaces W_1 , $W_2 \subset V$ where $W_1 \cup W_2$ is not a subspace, take V to be $\mathbb{R}^3$, W_1 to be a line in $\mathbb{R}$ (that goes through the origin), and W_2 to be a plane in $\mathbb{R}^2$ (that goes through the origin). Here W_1, $W_2 \subset V$, but addition is not defined in $W_1 \cup W_2$ becasue we cannot add vectors of different dimensions.

As an example where $W_1 \cup W_2$ is a subspace, take W_1 to be a a line in $\mathbb{R}$ and W_2 to be an line in $\mathbb{R}$ that is orthogonal to W_1. Make it so they both pass through the origin in V. I think as long as both subspaces are the same dimension, and pass through the origin, then their union will be a subspace.

Question 2. Show that the solution set to $A\overline{x} = \overline{b}$ is never a subspace when $\overline{b} \neq 0$.

Suppose the solution set to $A\overline{x} = \overline{b}$ were a subspace. Call it U over F,
So then U meets the second rule of subspaces. That is...

> (*ii*) for every $x \in U$: $cx \in U$ where $c \in F$

F being a field means $0 \in F$. So $\overline{0} \in U$, because when c is 0, $0 * \overline{x} = \overline{0}$.

Then $\overline{x} = \overline{0}$ in $A\overline{x} = \overline{b}$, making $b = \overline{0}$. But $b \neq \overline{0}$. This is a contradiction.

So the solution set to $A\overline{x} = \overline{b}$ is only a subspace when $b = \overline{0}$

Question 3. In this problem we will go over an important construction of vector spaces. Let $W \subset V$ be a subspace. We say that vectors $x, y \in V$ are congruent mod(W) (and write $x = y(\text{mod } W))$ if $x - y \in W$.

> (a) Show that for every $x \in V$, $x = x(\text{mod} W)$.

For any element $\overline{x} \in V$, $\overline{x} - \overline{x} = \overline{0}$
$\overline{0} \in W$ because W is a subspace.
So $\overline{x} - \overline{x} \in W$. Then $\overline{x} = \overline{x}(modW)$
■

> (b) Show that for every $x, y \in V$, if $x = y(modW)$ then $y = x(modW)$.

Suppose $\overline{x} = \overline{y}(modW)$. This means $w|(\overline{x} - \overline{y})$ where $w \in W$.
By the definition of divisibility $cw = \overline{x} - \overline{y}$ where $c \in \mathbb{Z}$.
Multiply both sides by -1.

$$-cw = -1 * (\overline{x} - \overline{y})$$
$$kw = (\overline{y} - \overline{x})$$

Where $k \in \mathbb{Z}$, from the multiplication of two integers -1 and c

So then $w|\overline{x} - \overline{y} \implies w|\overline{y} - \overline{x}$
and $x = y(modW) \implies y = x(modW)$
■

> (c) Show that if $x = y(modW)$ and $y = z(modW)$ then $x = z(modW)$.

Proof.
Suppose $x = y(modW)$ and $y = z(modW)$.
Then $\overline{w}|(\overline{x} - \overline{y})$ and $w|(\overline{y} - \overline{z})$ where $w \in W$.
Similarly $c\overline{w} = \overline{x} - \overline{y}$ and $lw = \overline{y} - \overline{z}$ where $c, l \in \mathbb{Z}$.
Isolating $\overline{y}$ and substituting we get...

$$\overline{y} = l\overline{w} + \overline{z}$$
$$c\overline{w} = x - (l\overline{w} + \overline{z})$$
$$c\overline{w} + l\overline{w} = x - \overline{z}$$
$$\overline{w}(c + l) = x - \overline{z}$$
$$k\overline{w} = x - \overline{z}$$

Where $k \in \mathbb{Z}$ from the adition of two integers l and c.

So then $k\overline{w} = x - \overline{z}$ means $\overline{w}|(\overline{x} - \overline{z})$ and $\overline{x} = \overline{z}(modW)$

■

> (d) Let $V = R^2$, and W be the y-axis. What is the dimension of V/W?

Answer. Two dimensional?

Question 4. Let $T : V \to W$ be a linear transformation.

> (a) Let $U \subset V$ be a subspace of V. Show that $T(U)$ is a subspace of W.

We need to show that $T(U)$ satisfies the two rules of vector subspaces, and that $T(U) \subset W$.

Proof
First s howing that for any elements $z, t \in T(U)$, $z + t \in T(U)$.
Take any elements $z, t \in T(U)$. They are linear transformations of some element in U.

$$z = T(x) \text{ where } x \in U$$
$$t = T(y) \text{ where } y \in U$$

Since U is subspace, and $x, y \in U$ then $x + y \in U$.
$x + y \in U$ means $T(x + y) \in T(U)$.
By the first rule of linear transformations (first being the first one listed in class)...

$$T(x + y) = T(x) + T(y)$$

Then $T(x + y) \in T(U)$ means $T(x) + T(y) \in T(U)$

Now we show that for any elements $z \in T(U)$, $cz \in T(U)$ where $c \in F$.
Again, let $z \in T(U)$ such that...

$$z = T(x) \text{ where } x \in U$$

U being a subspace means for and $x \in U$, $cx \in U$.
$cx \in U$ means $T(cx) \in T(U)$. By the second rule of linear transformations (second one listed in class)...

$$T(cx) = cT(x) = cz$$

So for any $z \in T(U)$, $cz \in T(U)$.

With that $T(U)$ satisfies the two rules of vector subspaces.

Lastly I need to show that $T(U) \subset W$.
T sends any element of V to W written as $T : V \to W$.
So $T(V) \subset W$.
As stated $U \subset V$, so any element $x \in U$ means $x \in V$.
Then any element $T(x) \in T(U)$ is also $T(x) \in V$.
$T(U) \subset T(V)$ and $T(V) \subset W$ means $T(U) \subset W$.

Now we've shown that $T(U)$ satisfies the two rules of subspaces, and that $T(U) \subset W$.
Therefore $T(U)$ is a subspace of W.
■

> (b) Let $U \subset W$ be a subspace of W . Show that
> $T^{-1}(U) = \{x \in V | T(x) \in U\}$ is a subspace of V.

Answer. We need to show that $T^{-1}(U)$ follows the two rules of vector subspaces, and that $T^{-1}(U)$ is a subset of V.

Proof
Showing $T^{-1}(U)$ obides by the first rule of subspaces.
Given any $x, y \in T^{-1}(U)$ we have a $T(x), T(y) \in U$.
Given any $T(x), T(y) \in U$ we have a $x, y \in T^{-1}(U)$.
U is a subspace so for any $T(x), T(y) \in U$ there exists $T(x) + T(y) \in U$.
T is a linear transformation so $T(x) + T(y) = T(x + y)$ meaning $T(x + y) \in U$.
When $T(x + y) \in U$ there is a $x + y \in T^{-1}(U)$.
Then for any $x, y \in T^{-1}(U)$ there exists $x + y \in T^{-1}(U)$.

Showing $T^{-1}(U)$ obides by the second rule of subspaces.
Again, given any $T(x) \in U$ we have a $x \in T^{-1}(U)$.
U is a subspace, so for any $T(x) \in U$ there exists $cT(x) \in U$.
T is a linear transformation, so $cT(x) = T(cx)$.
$T(cx) \in U$ means there is some $cx \in T^{-1}(U)$.
Then for any $x \in T^{-1}(U)$ there exists some $cx \in T^{-1}(U)$.

Showing $T^{-1}(U) \subset V$. Look at the left most side of the set builder notation in $T^{-1}(U)$, that is $\{x \in V | T(x) \in U\}$.
All elements of $T^{-1}(U)$ are in V so $T^{-1}(U) \subset V$.

We've shown that $T^{-1}(U)$ follows the two rules of vector subspaces, and that $T^{-1}(U)$ is a subset of V.
Therefore $T^{-1}(U)$ is a subspace of V.
■

Question 5. Suppose we're trying to send a message composed of 1's and 0's. The line we're trying to send the message across has some noise in it which causes some of the bits we send to come out wrong on the other side. If we want to make sure that the person we are sending the message to gets the correct information, we can use something called an error correcting code. This is a method in which we can convert our original message to a longer one which is more resistant to noise - the recipient can recover the correct message, even if a certain percentage of the bits is wrong. In this problem, you will explore two simple error correcting codes. In each one of them, the original message is cut up into small chunks of a fixed size. Each chunk is considered as a vector over F_2 . A linear transformation is then applied to this vector to produce a new vector which is the one that is actually sent down the line.

> (a) Consider the map $T : F_2 \to F_2^5$ given by $T(x) = (x, x, x, x, x)$. Show that this is a linear transformation.

I need to show that $T(x + y) = T(x) + T(y)$ where $x, y \in F_2$.
I need to show that $T(cx) = cT(x)$ where $c \in K$. K being the fields F_2 is over.

It does not matter if you add then copy the elements in T or copy then add. I'm not sure how to show this apropriatly I'm sure it's really fast. I'm going to play a board game now though, I've run out of time thank you.

> (b) We can create an error correcting code by taking our original message,cutting it into chunks of size 1, and sending T of each chunk down the line. If our original message was 1010, we would send 1111000011110000 down the line. What would we send down the line if our original message was 0010?

Answer. If out original message was 0010 we would send 0000000011110000 down the line.

> (c) The sender used the error correcting code outlined above. The recipient received the following message: 110111101000. What was the sender's original message?

Answer. The original code was probably 110.

> (d) How much longer is the error correcting code version of a message than the original message? What percentage of the bits can be received incorrectly and still allow the receiver to decode the message?

The error correcting code is four times longer. Only 25% of the bits can be recieved incorrectly and still allow a reciever to decode the message.

> (e) The following error correcting code is called the Hamming (7, 4) code. It breaks up the original message into chunks of 4 bits, and replaces each 4 bit sequence with a 7 bit message using the linear transformation: $T(x, y, z, w) = (x, y, z, w, x + y + w, x + z + w, y + z + w)$. If we wanted to send the message 01001110, what would we send down the line?

Answer. Defining a hamming function so I don't have to add ones and zeros. Then applying the function to the message we would like to send.

```
In [23]: def ham(x, y, z, w):
             return [x, y, z, w, (x+y+w) % 2, (x+z+w) % 2, (y+z+w) % 2]

         bit1 = ham(0, 1, 0, 0)
         bit2 = ham(1, 1, 1, 0)

         [bit1, bit2]
```

```
Out[23]: [[0, 1, 0, 0, 1, 0, 1], [1, 1, 1, 0, 0, 0, 0]]
```

So I would send 01001011110000 down the line.

> (f) Is this linear transformation injective? Why is this question important in decoding?

I will assume it is, because if not it would be difficult to decode. Injective being one-to-one, and meaning the recieved error corrected message can be mapped back to a single original message (assuming no errors in the message). If it was not injective then the error corected code would map back to more than one original message and that would be confounding.

> (g) To decode a message, find the element of the image of T that has the most bits in common with the message you received. Decode 1101001.

I found the original message to be 1001. I goofed up and wrote the original code in R, but I'm using jupyter notebook with a python kernal and cannot run the R script to be out put directly in this document. So here is the code. I will describe what I did.

In [ ]:
```
# hamming functions
ham = function(x, mod = 2){
  c(x[1], x[2], x[3], x[4], (x[1]+x[2]+x[4]) %% mod, (x[1]+x[3]+x[4]) %% mod,
}

# Creating all combinations of [0,1] length 4 vectors.
inputs = expand.grid(x = 0:1, y = 0:1, z = 0:1, w = 0:1)

# the image of T: applying the ham function to each combination of 4 bit input
imgt = apply(inputs, FUN = ham, MARGIN = 1)

# finding which 7 bit output in ImT is the least different from 1101001
dift = abs(imgt - c(1,1,0,1,0,0,1))
dift = apply(dift, FUN = sum, MARGIN = 2)
# indexing least different vector
idx = which.min(dift)

# pulling first 4 bits from the indexed vector to find the original message
original = imgt[1:4,10]
original
```

output: 1 0 0 1

Here's what I did...

(1) I defined a hamming function called `ham`, like I did in question (e) that perfrmes the Hamming calculations in modular 2 on a length 4 vector.

(2) I then created a matrix (mxn = 4x16) with all possible combinations of 0 or 1's as the elements in the length 4 columns called `inputs`.

(3) I applied the `ham` functions to the columns of the 16X4 `inputs` matrix. This produced a 16X7 matrix called `imgt`. It represents all possible outputs of the hamming function on all the possible inputs of length 4 vectors.

(4) I subtracted our vector of interest, 1101001, from each column of the `imgt` matrix. I took the absolute value of this difference, and took the sum. This produced a vector called `dift` of length 16 that holds all the differences of our vector of interest, 1101001, from the image of T.

(5) I indexed the column that contained the smallest difference from 1101001, and found it to be the message 1001.

> (h) Verify that any two vectors in the image of T differ by at least 3 bits.

Again here's R code. I subtracted each vector from the other vectors and found the minimum difference. For each case the minimum was 3.

In [ ]:
```
# Showing that the minium difference is 3 for all vectors
arr = imgt
vects = split(arr, rep(1:ncol(arr), each = nrow(arr)))

for(i in 1:length(vects)){
  idx = 1:length(vects)
  # subsetting the vector to check
  x = subset(t(arr), is.element(idx, i))
  # subletting everything except the vector being checked
  y = subset(t(arr), !is.element(idx, i))
  # data class and shape stuff
  x = unname(x[1,])
  y = t(y)
  # finding the difference between the checking vector
  dift = abs(y - x)
  dift = apply(dift, FUN = sum, MARGIN = 2)
  # finding the least amount of difference
  print(min(dift))
  }
```

output: 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

> (i) if at most 1 out of every 7 bits in a line might be wrong, explain why the (7, 4) Hamming code can always be used to decode messages. What can go wrong if more bits than that are incorrect?

If a bit from the first 4 digits in the error corrected message are wrong, then they are checked by the last 3 digits. If a bit from the last 3 digits in the error corrected message are wrong, then they are checked by the first 4 digits. If a digit is wrong from both, it would be no longer be injective. Maybe it would no longer be injective if any two different bits are worng?