Homework 1

Kyle Hadley

In [2]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
import warnings
warnings.simplefilter('ignore')
```

1. Projections

(a)

To compute the projection of $x = [1, 2, 3]^T$ onto $y = [1, 1, -2]^T$, we will use the following equation:

$$proj_y x = y(y^T y)^{-1} y^T x$$

We can perform this calculation using Numpy and built-in matrix multiplication, transpose, and inverse functions.

In [4]:
```
x = np.array([[1], [2], [3]])
y = np.array([[1], [1], [-2]])

#print(x)
#print(y)

proj_yx = y.dot(np.linalg.inv(np.transpose(y).dot(y)).dot(np.transpose(y).dot(x)))

print(proj_yx)
```

```
[[-0.5]
 [-0.5]
 [ 1. ]]
```

The result is $proj_y x = [-0.5, -0.5, 1]^T$.

(b)

To compute the projection of $x = [1, 2, 3]^T$ onto the range $Y = \begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 0 & 1 \end{bmatrix}$ we will use the following equation:

$$proj_Y x = Y(Y^T Y)^{-1} Y^T x$$

We can perform this calculation using Numpy and built-in matrix multiplication, transpose, and inverse functions.

In [5]:
```
x = np.array([[1], [2], [3]])
y = np.array([[1, 1], [-1, 0], [0, 1]])

#print(x)
#print(y)

proj_yx = y.dot(np.linalg.inv(np.transpose(y).dot(y)).dot(np.transpose(y).dot(x)))

print(proj_yx)
```

```
[[1.]
 [2.]
 [3.]]
```

The result is $proj_y x = [1, 2, 3]^T$.

2. Block Matrix Computations

(a)

$$AB = \begin{bmatrix} A_{11}B_{11} + \ldots + A_{1N}B_{N1} & \ldots & A_{11}B_{1K} + \ldots + A_{1N}B_{NK} \\ \vdots & & \vdots \\ A_{M1}B_{11} + \ldots + A_{MN}B_{N1} & \ldots & A_{M1}B_{1K} + \ldots + A_{1N}B_{NK} \end{bmatrix}.$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_{11} \in \mathbb{R}^{n_1 \times k_1}$, $B_{1K} \in \mathbb{R}^{n_1 \times k_K}$, $B_{N1} \in \mathbb{R}^{n_N \times k_1}$, and $B_{NK} \in \mathbb{R}^{n_N \times k_K}$.

(b)

$$AB = \begin{bmatrix} A_1B_1 & \ldots & A_1B_k \\ \vdots & & \vdots \\ A_mB_1 & \ldots & A_mB_k \end{bmatrix}.$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_1 \in \mathbb{R}^{n \times 1}$ and $B_k \in \mathbb{R}^{n \times 1}$.

(c)

$$AB = \begin{bmatrix} | \\ A_1 \\ | \end{bmatrix} [- \quad B_1 \quad -] + \ldots + \begin{bmatrix} | \\ A_n \\ | \end{bmatrix} [- \quad B_n \quad -].$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_1 \in \mathbb{R}^{1 \times k}$ and $B_n \in \mathbb{R}^{1 \times k}$.

(d)

$$ADB = \begin{bmatrix} A_1DB_1 & \dots & A_1DB_k \\ \vdots & & \vdots \\ A_mDB_1 & \dots & A_mDB_k \end{bmatrix}.$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_1 \in \mathbb{R}^{n\times 1}$ and $B_k \in \mathbb{R}^{n\times 1}$.

(e)

$$ADB = \sum_{x=1}^{n}\sum_{y=1}^{n} \begin{bmatrix} | \\ A_x \\ | \end{bmatrix} D_{xy} \begin{bmatrix} - & B_y & - \end{bmatrix}.$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_1 \in \mathbb{R}^{1\times k}$ and $B_k \in \mathbb{R}^{1\times k}$.

(f)

$AB = \begin{bmatrix} AB_1 & \dots & AB_k \end{bmatrix}.$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B_1 \in \mathbb{R}^{n\times 1}$ and $B_k \in \mathbb{R}^{n\times 1}$.

(g)

$$AB = \begin{bmatrix} A_1B \\ \vdots \\ A_mB \end{bmatrix}.$$

Given the resulting matrix, we know the required dimensions of the sub-blocks of B are $B \in \mathbb{R}^{n\times k}$ (since there are no sub-blocks of B).

3. Linear Transformations of Sets

(a) Affine Sets

Given $\mathcal{X}_1 = \{x | x_1 + x_2 = 1, x \in \mathbb{R}^2\}$ and $\mathcal{X}_2 = \{x | x_1 - x_2 = 1, x \in \mathbb{R}^2\}$, we can draw the set of points for Ax for $x \in \mathcal{X}_1$ and $x \in \mathcal{X}_2$.

For the condition where $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, we can solve for Ax such that

$$Ax = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We can now draw the set of points by finding two points and then drawing the line through these two points.

We can now define the set of points for when $x \in \mathcal{X}_1$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 1$$

thus our first point is $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

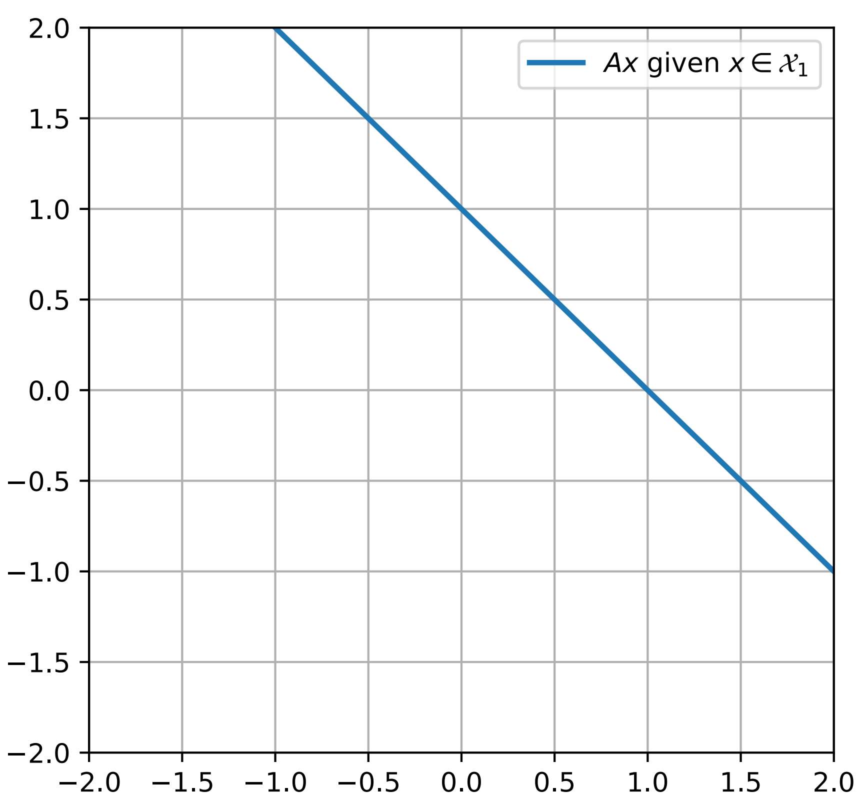
When $x_1 = 1$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

In [6]:

```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [0, 1]
x_2 = [1, 0]

x = np.linspace(-5, 5, num=100)
y = (x_2[1] - x_2[0])/(x_1[1] - x_1[0]) * x + (x_1[0]*x_2[1] - x_1[1]*x_2[0]) / (x_1[0]

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$Ax$ given $x \in \mathcal{X}_1$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



We can now define the set of points for when $x \in \mathcal{X}_2$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = -1$$

thus our first point is $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$.

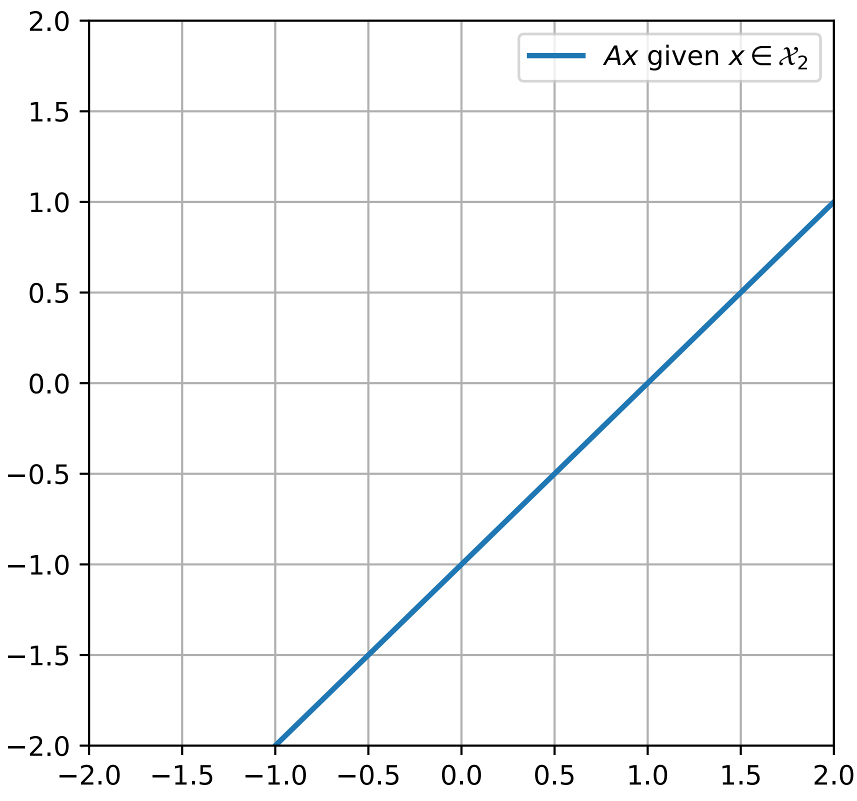
When $x_1 = 1$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

In [8]:

```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [0, 1]
x_2 = [-1, 0]

x = np.linspace(-5, 5, num=100)
y = (x_2[1] - x_2[0])/(x_1[1] - x_1[0]) * x + (x_1[0]*x_2[1] - x_1[1]*x_2[0]) / (x_1[0]

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$Ax$ given $x \in \mathcal{X}_2$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



For the condition where $A = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix}$, we can solve for Ax such that

$$Ax = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ -x_2 \end{bmatrix}$$

We can now draw the set of points by finding two points and then drawing the line through these two points.

We can now define the set of points for when $x \in \mathcal{X}_1$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 1$$

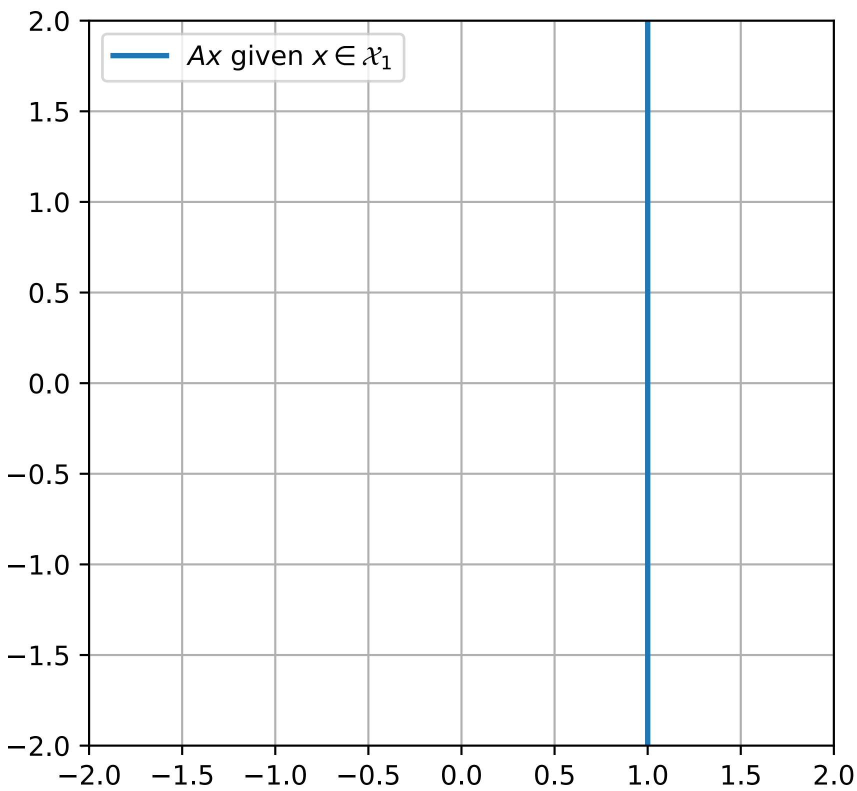
thus our first point is $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

When $x_1 = 1$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

In [9]:
```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [1, 1]
x_2 = [-1, 0]

# By inspection, we see from the coordinates that Ax is a vertical line @ x_1 = 1, thus

fig, ax = plt.subplots(figsize=(5, 5))
ax.axvline(1, label='$Ax$ given $x \in \mathcal{X}_1$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



We can now define the set of points for when $x \in \mathcal{X}_2$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = -1$$

thus our first point is $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$.

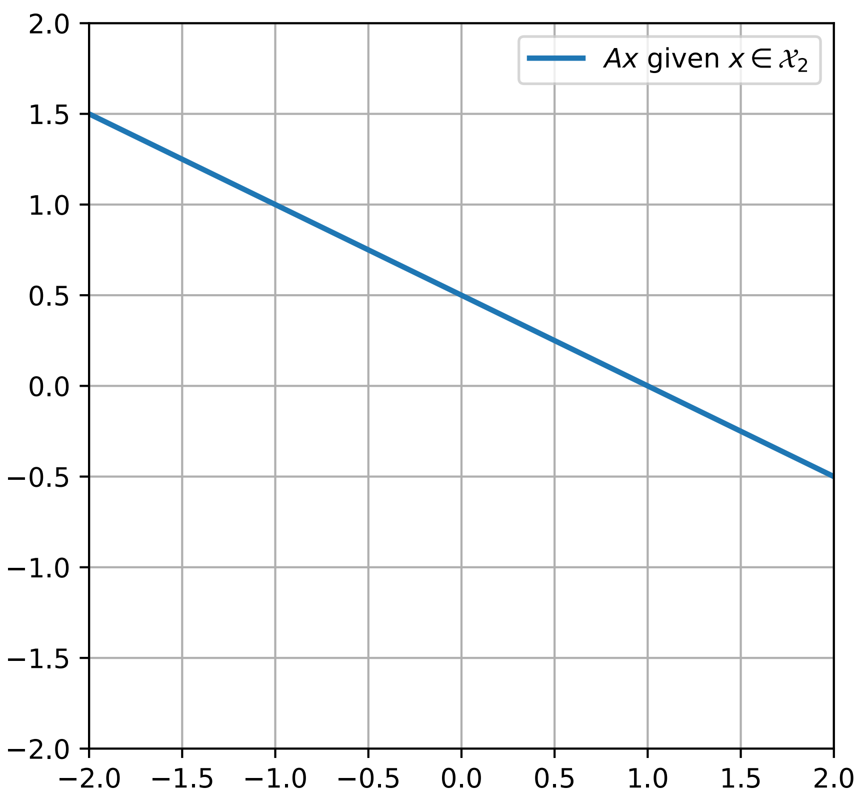
When $x_1 = 1$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

In [10]:

```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [-1, 1]
x_2 = [1, 0]

x = np.linspace(-5, 5, num=100)
y = (x_2[1] - x_2[0])/(x_1[1] - x_1[0]) * x + (x_1[0]*x_2[1] - x_1[1]*x_2[0]) / (x_1[0]

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$Ax$ given $x \in \mathcal{X}_2$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



For the condition where $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$, we can solve for Ax such that

$$Ax = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ x_1 + x_2 \end{bmatrix}$$

We can now draw the set of points by finding two points and then drawing the line through these two points.

We can now define the set of points for when $x \in \mathcal{X}_1$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 1$$

thus our first point is $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$.

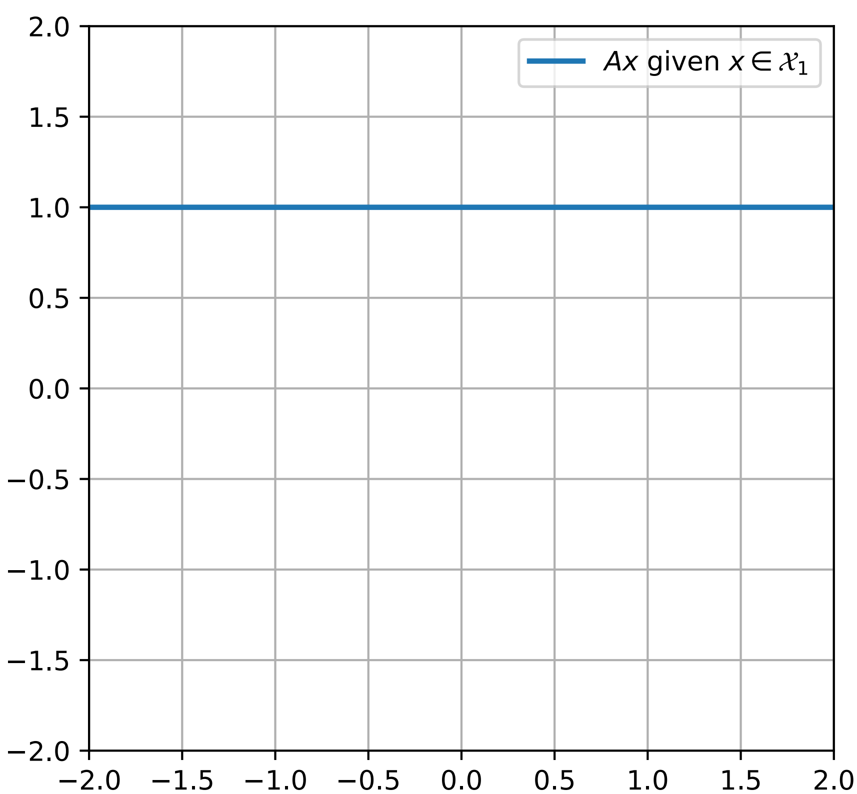
When $x_1 = 1$, we find that given $x \in \mathcal{X}_1$,

$$x_2 = 1 - x_1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

In [11]:

```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [-1, 1]
x_2 = [1, 1]

x = np.linspace(-5, 5, num=100)
y = (x_2[1] - x_2[0])/(x_1[1] - x_1[0]) * x + (x_1[0]*x_2[1] - x_1[1]*x_2[0]) / (x_1[0]

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$Ax$ given $x \in \mathcal{X}_1$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



We can now define the set of points for when $x \in \mathcal{X}_2$. When $x_1 = 0$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = -1$$

thus our first point is $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

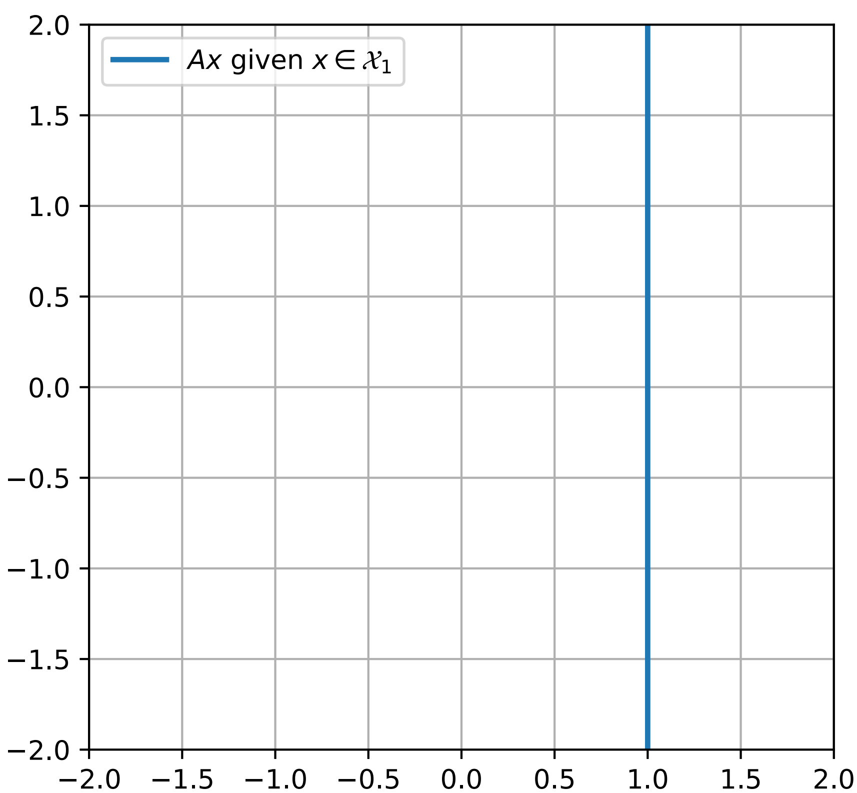
When $x_1 = 1$, we find that given $x \in \mathcal{X}_2$,

$$x_2 = x_1 - 1 = 0$$

thus our second point is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

In [12]:

```
# Given coordinates defined above, define a line y that is the set of points Ax.
x_1 = [1, 1]
x_2 = [-1, 1]

# By inspection, we see from the coordinates that Ax is a vertical line @ x_1 = 1, thus

fig, ax = plt.subplots(figsize=(5, 5))
ax.axvline(1, label='$Ax$ given $x \in \mathcal{X}_1$', linewidth=2)
ax.set_xlim([-2, 2])
ax.set_ylim([-2, 2])
ax.grid()
ax.legend()
plt.show()
```



(b) Unit Balls

Given $\mathcal{X}_1 = \{x \mid |x|_1 \leq 1, x \in \mathbb{R}^2\}$, $\mathcal{X}_2 = \{x \mid |x|_2 \leq 1, x \in \mathbb{R}^2\}$, and $\mathcal{X}_\infty = \{x \mid |x|_\infty \leq 1, x \in \mathbb{R}^2\}$, we can draw the set of points for Ax for $x \in \mathcal{X}_1$, $x \in \mathcal{X}_2$, and $x \in \mathcal{X}_\infty$.

We can generate an initial T-chart of points within the defined sets X_1, X_2, and X_∞.

For X_1,

x_1	x_2
1	0
1/2	1/2
0	1
-1/2	1/2
-1	0
-1/2	-1/2
0	-1
1/2	-1/2

which implies that the resulting initial set is a diamond shape.

For X_2,

x_1	x_2
1	0
0.707	0.707
0	1
-0.707	0.707
-1	0
-0.707	-0.707
0	-1
0.707	-0.707

which implies that the resulting initial set is a circular shape.

For X_∞,

x_1	x_2
1	0
1	1
0	1
-1	1

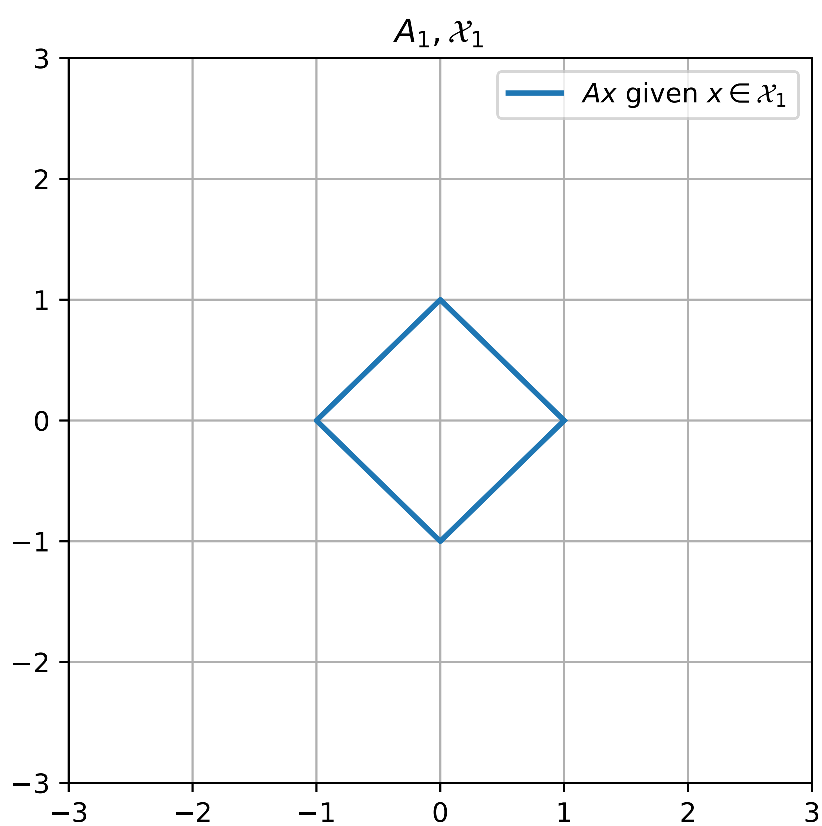
x_1	x_2
-1	0
-1	-1
0	-1
1	-1

which implies that the resulting initial set is a square shape.

For the condition where $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, the resulting set of points is identical to the initial set of points (because $A = I$).

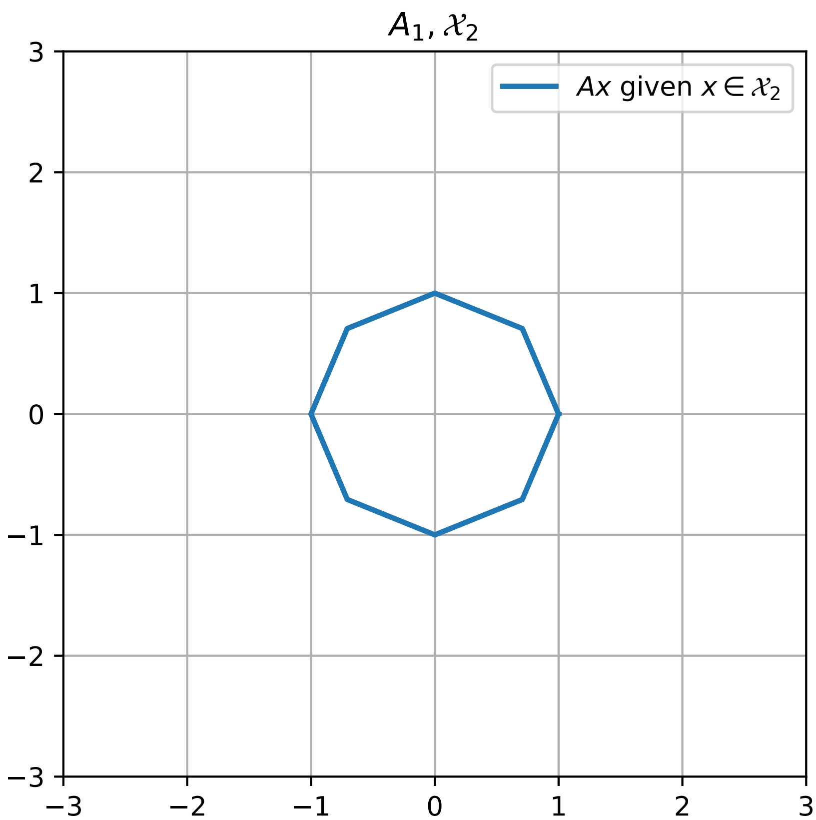
In [13]:

```
A = np.array([[1, 0], [0, 1]])
x_1 = np.array([1, 1/2, 0, -1/2, -1, -1/2, 0, 1/2, 1])
x_2 = np.array([0, 1/2, 1, 1/2, 0, -1/2, -1, -1/2, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_1, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```



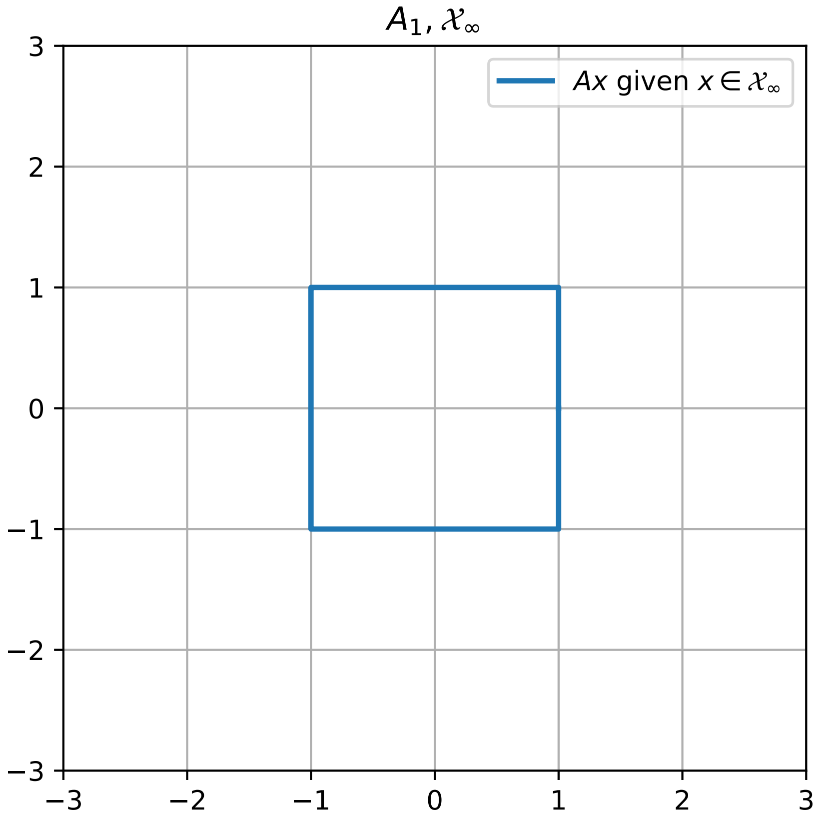
In [14]:

```
A = np.array([[1, 0], [0, 1]])
x_1 = np.array([1, .707, 0, -.707, -1, -.707, 0, .707, 1])
x_2 = np.array([0, .707, 1, .707, 0, -.707, -1, -.707, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_2$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_1, \mathcal{X}_{2}$')
ax.legend()
ax.grid()
plt.show()
```



In [15]:

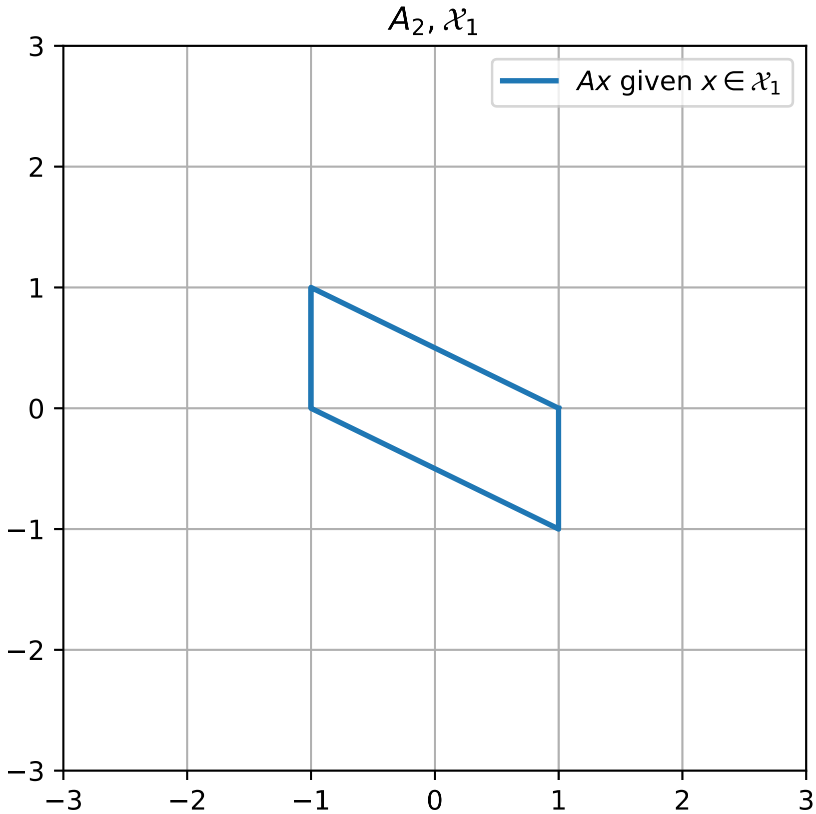
```
A = np.array([[1, 0], [0, 1]])
x_1 = np.array([1, 1, 0, -1, -1, -1, 0, 1, 1])
x_2 = np.array([0, 1, 1, 1, 0, -1, -1, -1, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_{\infty}$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_1, \mathcal{X}_{\infty}$')
ax.legend()
ax.grid()
plt.show()
```



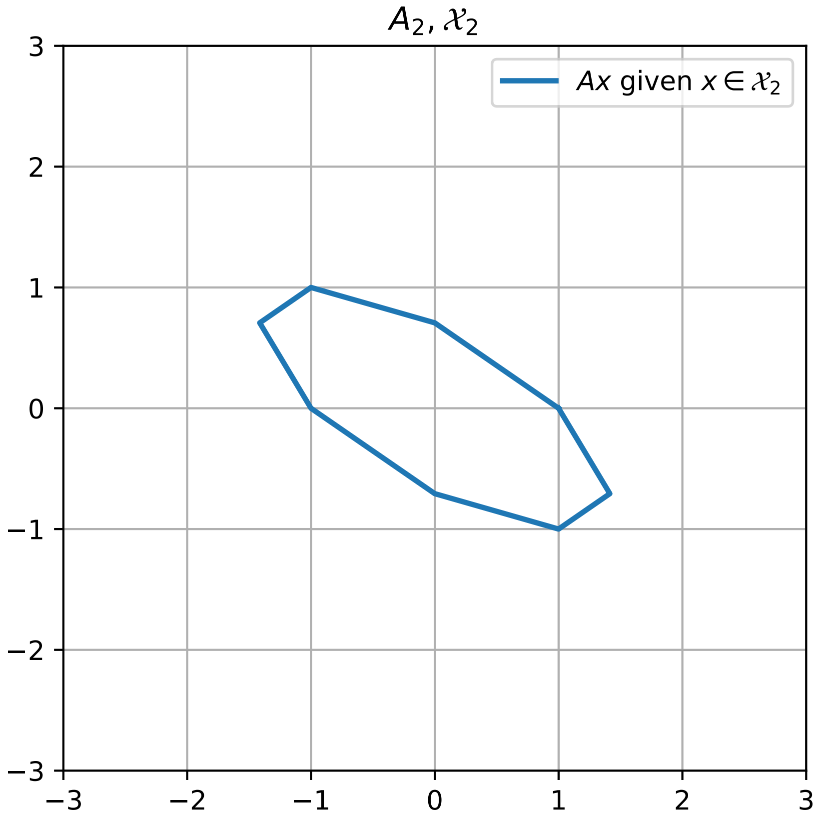
For the condition where $A = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix}$, we can calculate our new T-tables in-code to produce the plots as shown below.

In [16]:

```
A = np.array([[1, 1], [0, -1]])
x_1 = np.array([1, 1/2, 0, -1/2, -1, -1/2, 0, 1/2, 1])
x_2 = np.array([0, 1/2, 1, 1/2, 0, -1/2, -1, -1/2, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_2, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```

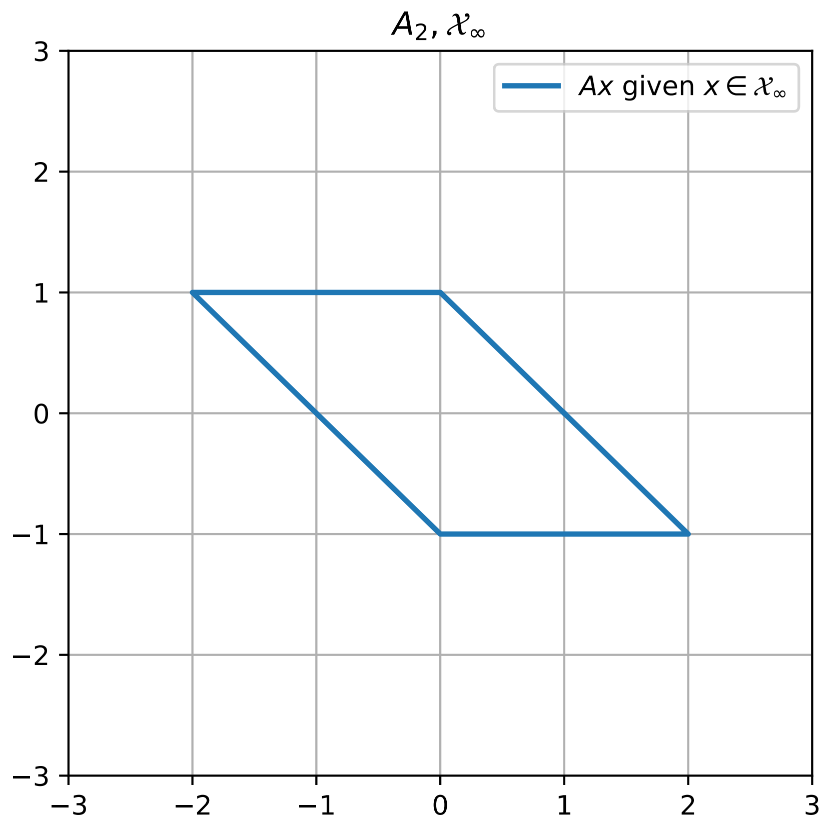


In [17]:
```
A = np.array([[1, 1], [0, -1]])
x_1 = np.array([1, .707, 0, -.707, -1, -.707, 0, .707, 1])
x_2 = np.array([0, .707, 1, .707, 0, -.707, -1, -.707, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_2$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_2, \mathcal{X}_{2}$')
ax.legend()
ax.grid()
plt.show()
```



In [18]:

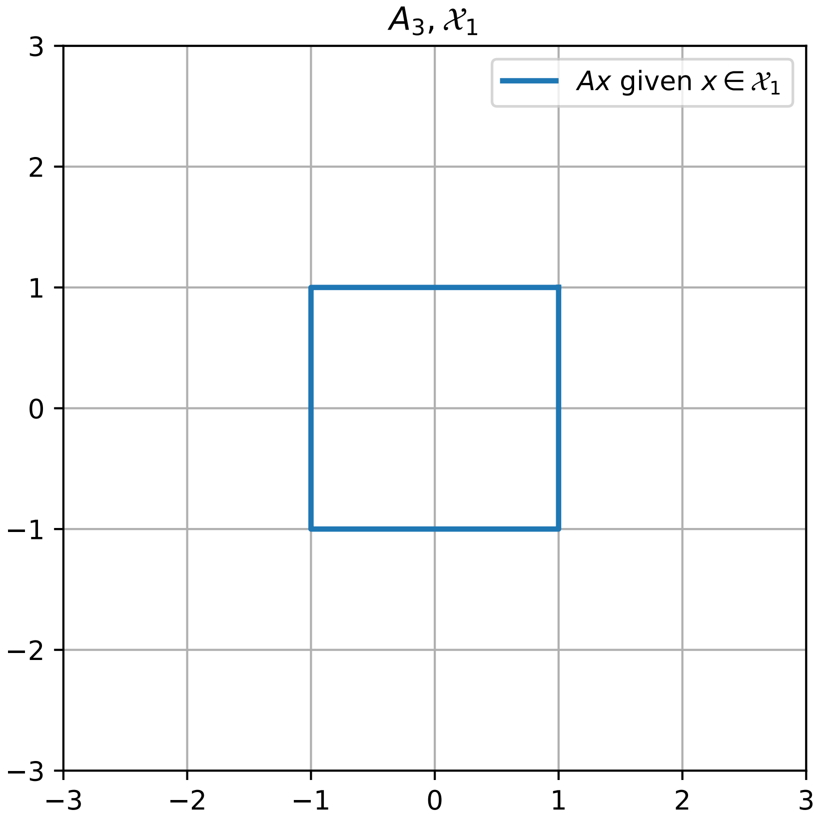
```
A = np.array([[1, 1], [0, -1]])
x_1 = np.array([1, 1, 0, -1, -1, -1, 0, 1, 1])
x_2 = np.array([0, 1, 1, 1, 0, -1, -1, -1, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_{\infty}$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_2, \mathcal{X}_{\infty}$')
ax.legend()
ax.grid()
plt.show()
```



For the condition where $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$, we can calculate our new T-tables in-code to produce the plots as shown below.

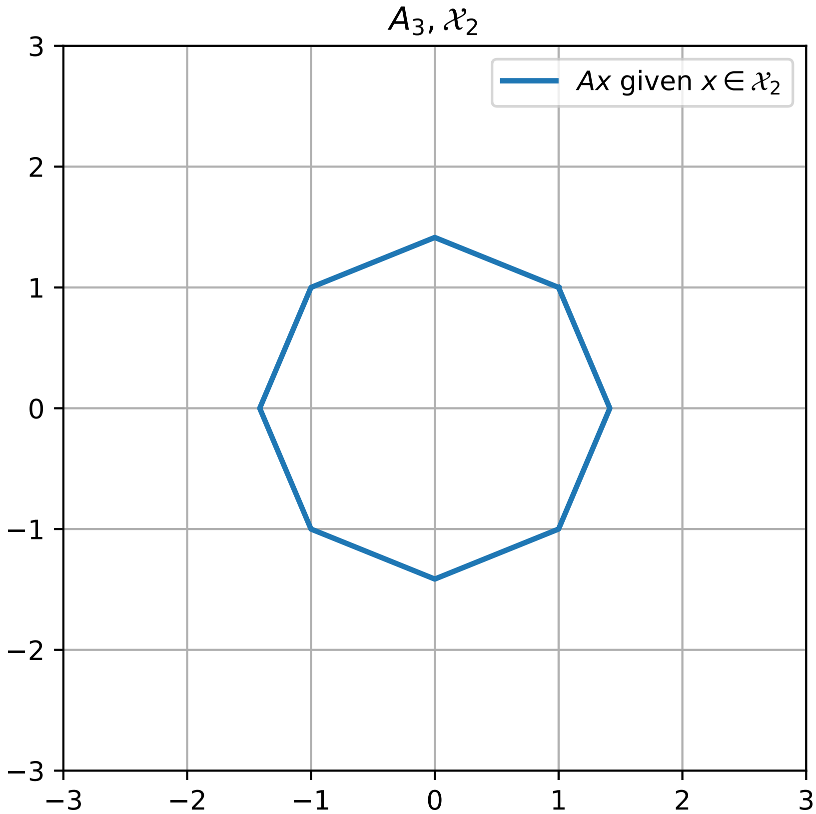
In [19]:

```
A = np.array([[1, -1], [1, 1]])
x_1 = np.array([1, 1/2, 0, -1/2, -1, -1/2, 0, 1/2, 1])
x_2 = np.array([0, 1/2, 1, 1/2, 0, -1/2, -1, -1/2, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```



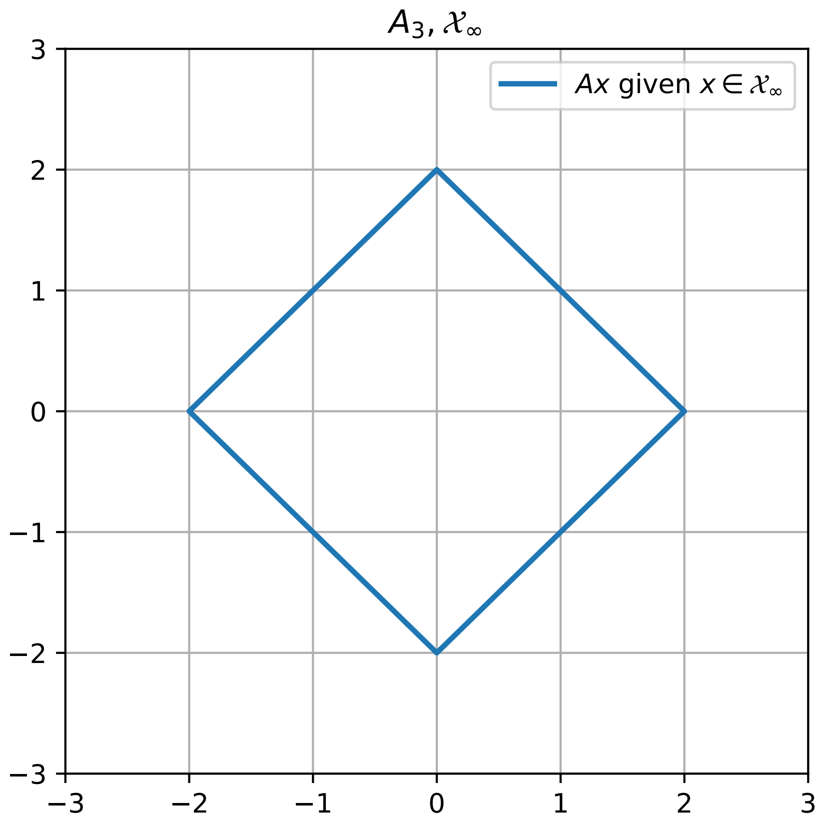
In [20]:

```
A = np.array([[1, -1], [1, 1]])
x_1 = np.array([1, .707, 0, -.707, -1, -.707, 0, .707, 1])
x_2 = np.array([0, .707, 1, .707, 0, -.707, -1, -.707, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_2$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{2}$')
ax.legend()
ax.grid()
plt.show()
```



In [21]:

```
A = np.array([[1, -1], [1, 1]])
x_1 = np.array([1, 1, 0, -1, -1, -1, 0, 1, 1])
x_2 = np.array([0, 1, 1, 1, 0, -1, -1, -1, 0])

x = np.stack((x_1, x_2))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_{\infty}$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{\infty}$')
ax.legend()
ax.grid()
plt.show()
```



(c) Convex Hulls

Given $\Delta_2 = \{x \mid \mathbf{1}^T x = 1, x \geq 0 x \in \mathbb{R}^2\}$, $\Delta_3 = \{x \mid \mathbf{1}^T x = 1, x \geq 0 x \in \mathbb{R}^3\}$, and $\Delta_4 = \{x \mid \mathbf{1}^T x = 1, x \geq 0 x \in \mathbb{R}^4\}$, we can draw the set of points for Ax for $x \in \Delta_2$, $x \in \Delta_3$, and $x \in \Delta_3$.

We can generate an initial T-chart of values for x within the defined sets.

For Δ_2, given that $\mathbf{1}^T x = 1$ we know that

$$x_1 + x_2 = 1 \text{ where } x \geq 0$$

Thus, we can generate a T-chart defined as,

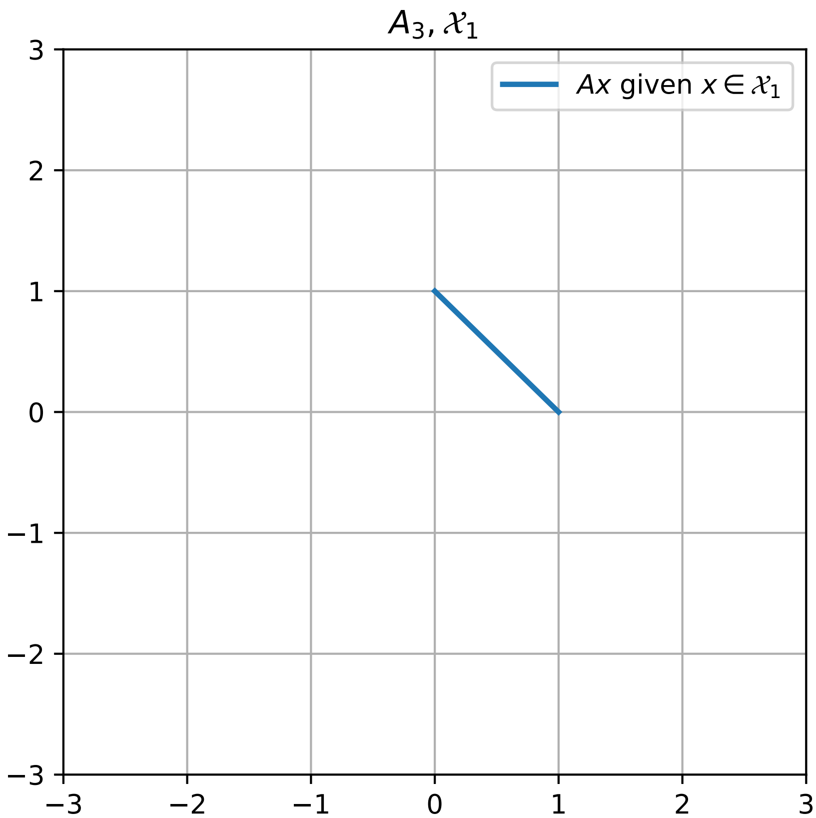
x_1	x_2
1	0
0	1
1/2	1/2

For the condition where $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, we can plot the results of Ax as shown below.

In [7]:

```
A = np.array([[1, 0], [0, 1]])
x_1 = np.array([1, 1/2, 0])
x_2 = np.array([0, 1/2, 1])

x = np.stack((x_1, x_2))
```

```
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```



For Δ_3, given that $\mathbf{1}^T x = 1$ we know that

$$x_1 + x_2 + x_3 = 1 \text{ where } x \geq 0$$

Thus, we can generate a T-chart defined as,

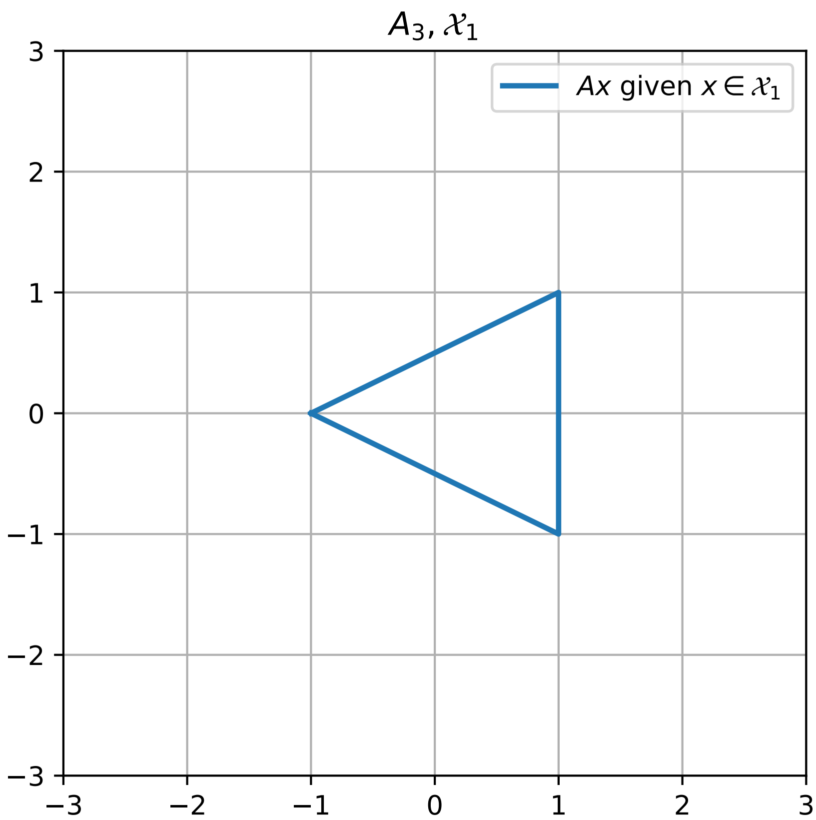
x_1	x_2	x_3
1	0	0
0	1	0
0	0	1

For the condition where $A = \begin{bmatrix} -1 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$, we can plot the results of Ax as shown below.

In [28]:

```
A = np.array([[-1, 1, 1], [0, 1, -1]])
x_1 = np.array([1, 0, 0, 1])
x_2 = np.array([0, 1, 0, 0])
x_3 = np.array([0, 0, 1, 0])
```

```
x = np.stack((x_1, x_2, x_3))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```



For Δ_4, given that $\mathbf{1}^T x = 1$ we know that

$$x_1 + x_2 + x_3 + x_4 = 1 \text{ where } x \geq 0$$

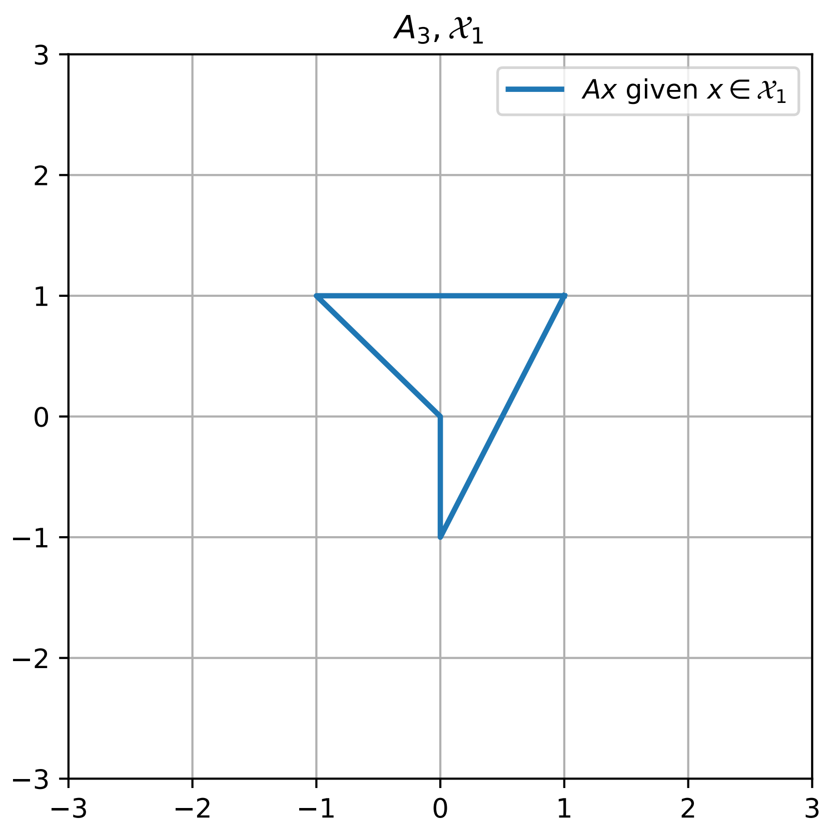
Thus, we can generate a T-chart defined as,

x_1	x_2	x_3	x_4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

For the condition where $A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$, we can plot the results of Ax as shown below.

In [31]:
```
A = np.array([[1, -1, 0, 0], [1, 1, 0, -1]])
```

```
x_1 = np.array([1, 0, 0, 0, 1])
x_2 = np.array([0, 1, 0, 0, 0])
x_3 = np.array([0, 0, 1, 0, 0])
x_4 = np.array([0, 0, 0, 1, 0])

x = np.stack((x_1, x_2, x_3, x_4))
Ax = A.dot(x)

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(Ax[0,:], Ax[1,:], linewidth=2, label='$Ax$ given $x \in \mathcal{X}_1$')
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_title('$A_3, \mathcal{X}_{1}$')
ax.legend()
ax.grid()
plt.show()
```



4. Affine and Half Spaces

(a)

For $a^T = \begin{bmatrix} 1 & -1 \end{bmatrix}$ and $X = \{x \in \mathbb{R}^2 | a^T x = 0\}$, the set is defined as:

$$a^T x = 0$$

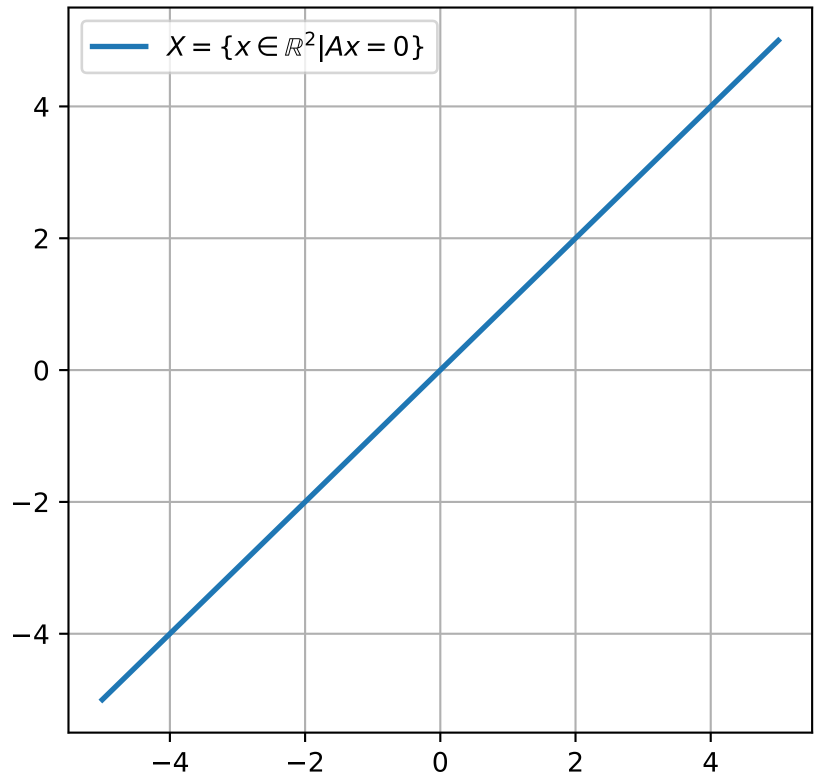
$$\begin{bmatrix} 1 & -1 \end{bmatrix} x = 0$$

$$x_1 - x_2 = 0$$

$$x_2 = x_1$$

This is space is a *subspace* but **not** a *affine space* nor a *half space*.

In [20]:
```
x = np.linspace(-5, 5, num=100)
y = x

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$X = \{ x \in \mathbb{R}^{2} | Ax = 0 \}$', linewidth=2)
ax.legend()
ax.grid()
plt.show()
```



For $a^T = \begin{bmatrix} 1 & -1 \end{bmatrix}$ and $X = \{x \in \mathbb{R}^2 | a^T x = 1\}$, the set is defined as:

$$a^T x = 1$$

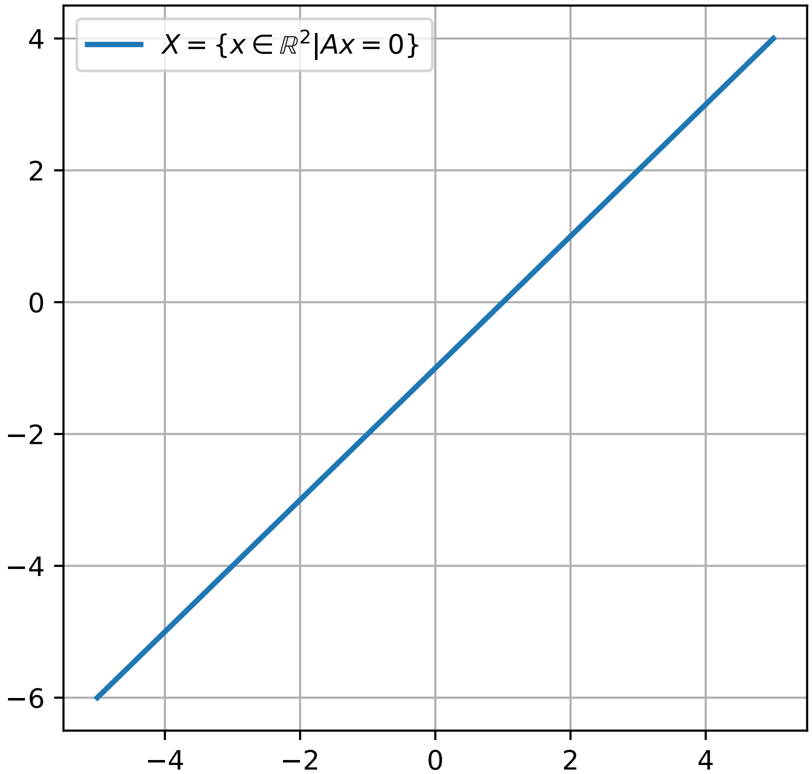
$$\begin{bmatrix} 1 & -1 \end{bmatrix} x = 1$$

$$x_1 - x_2 = 1$$

$$x_2 = x_1 - 1$$

This is space is a *affine space* but **not** a *subspace* nor a *half space*.

In [21]:
```
x = np.linspace(-5, 5, num=100)
y = x - 1

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$X = \{ x \in \mathbb{R}^{2} | Ax = 0 \}$', linewidth=2)
ax.legend()
ax.grid()
plt.show()
```



For $a^T = [1 - 1]$ and $X = \{x \in \mathbb{R}^2 | a^T x \leq 1\}$, the set is defined as:

$$a^T x \leq 1$$

$$[1 - 1]x \leq 1$$

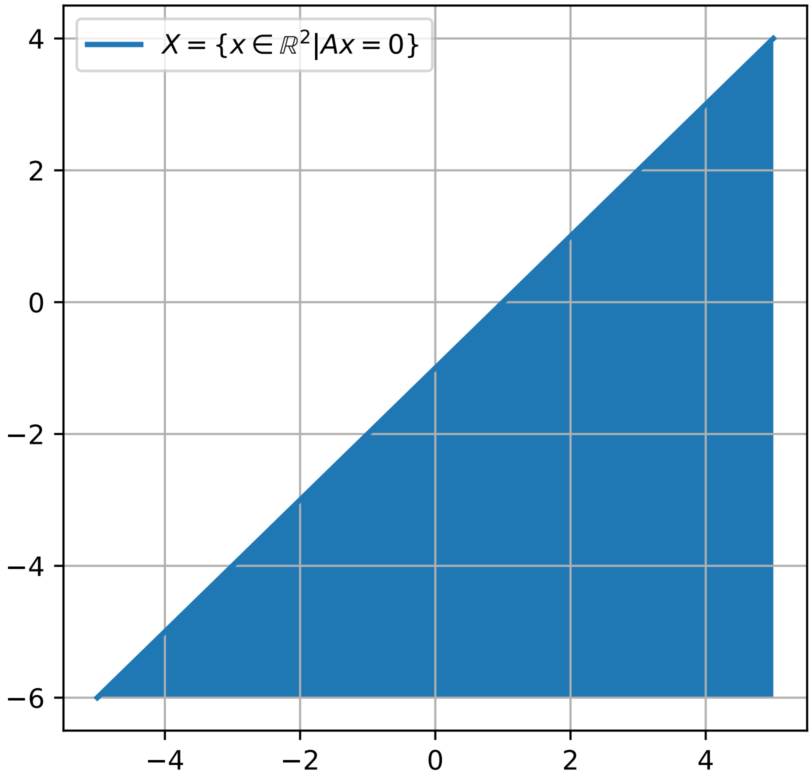
$$x_1 - x_2 \leq 1$$

$$x_2 \geq x_1 - 1$$

This is space is a *half space* but **not** an *affine space* nor a *subspace*.

In [22]:

```
x = np.linspace(-5, 5, num=100)
y = x - 1
y2 = -6 + x*0

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x, y, label='$X = \{ x \in \mathbb{R}^{2} | Ax = 0 \}$', linewidth=2)
ax.fill_between(x, y, y2)
ax.legend()
ax.grid()
plt.show()
```



(b)

For $a^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ and $X = \{x \in \mathbb{R}^2 | a^T x = 0\}$, the set is defined as:

$$a^T x = 0$$

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} x = 0$$

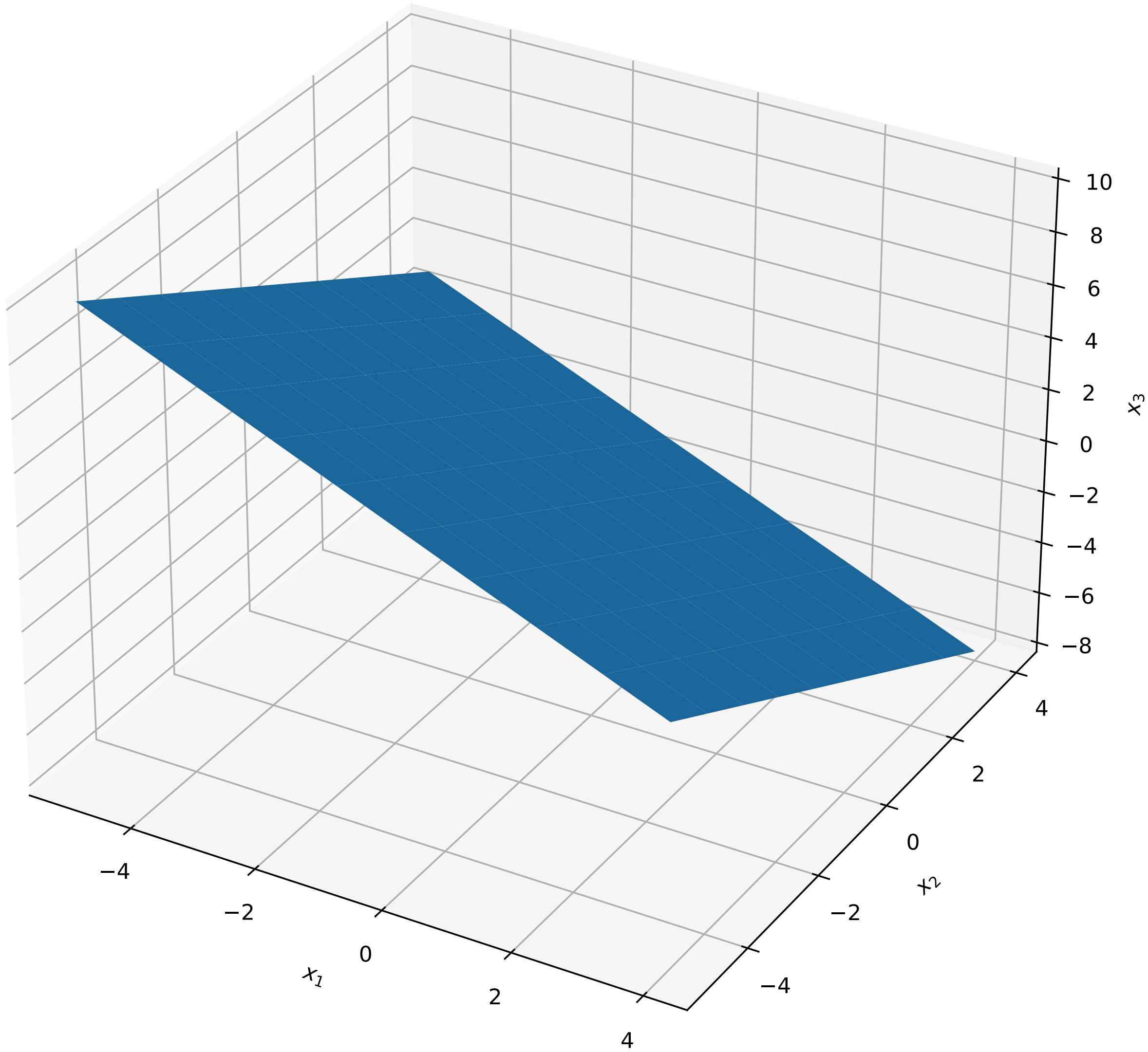
$$x_1 + x_2 + x_3 = 0$$

$$x_3 = -x_1 - x_2$$

This is space is a *subspace* but **not** an *affine space* nor a *half space*.

In [23]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data.
x_1 = np.arange(-5, 5, 1)
x_2 = np.arange(-5, 5, 1)
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = -x_1 - x_2

# Plot the surface.
surf = ax.plot_surface(x_1, x_2, x_3)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



For $a^T = [1 \quad 1 \quad 1]$ and $X = \{x \in \mathbb{R}^2 | a^T x = 1\}$, the set is defined as:

$$a^T x = 0$$

$$[1 \quad 1 \quad 1]\, x = 1$$

$$x_1 + x_2 + x_3 = 1$$

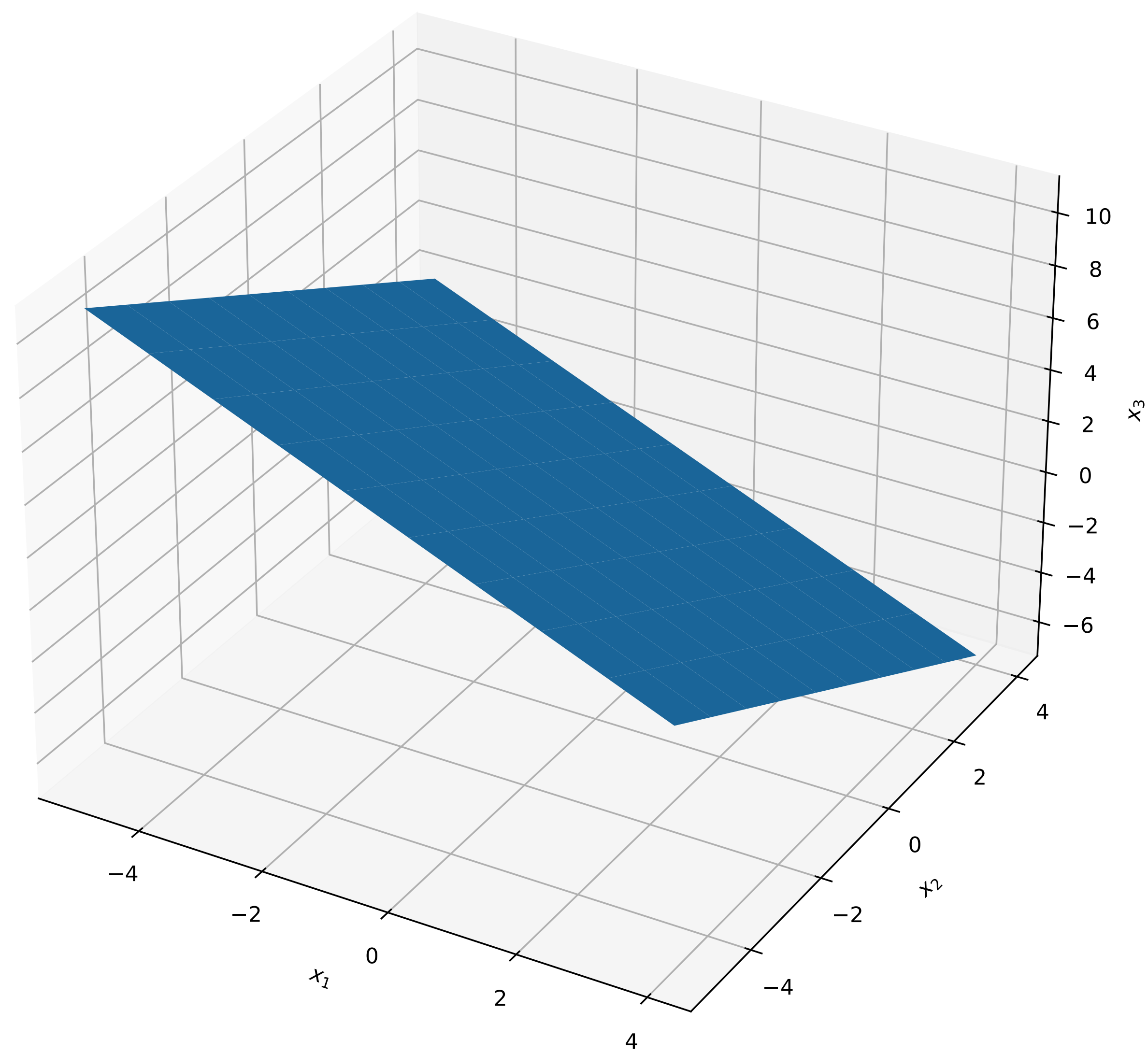
$$x_3 = 1 - x_1 - x_2$$

This is space is an *affine space* but **not** a *subspace* nor a *half space*.

In [24]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data.
x_1 = np.arange(-5, 5, 1)
x_2 = np.arange(-5, 5, 1)
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = 1 - x_1 - x_2
```

```python
# Plot the surface.
surf = ax.plot_surface(x_1, x_2, x_3)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



For $a^T = [1 \quad 1 \quad 1]$ and $X = \{x \in \mathbb{R}^2 | a^T x \leq 1\}$, the set is defined as:

$$a^T x \leq 0$$

$$[1 \quad 1 \quad 1]\, x \leq 1$$

$$x_1 + x_2 + x_3 \leq 1$$

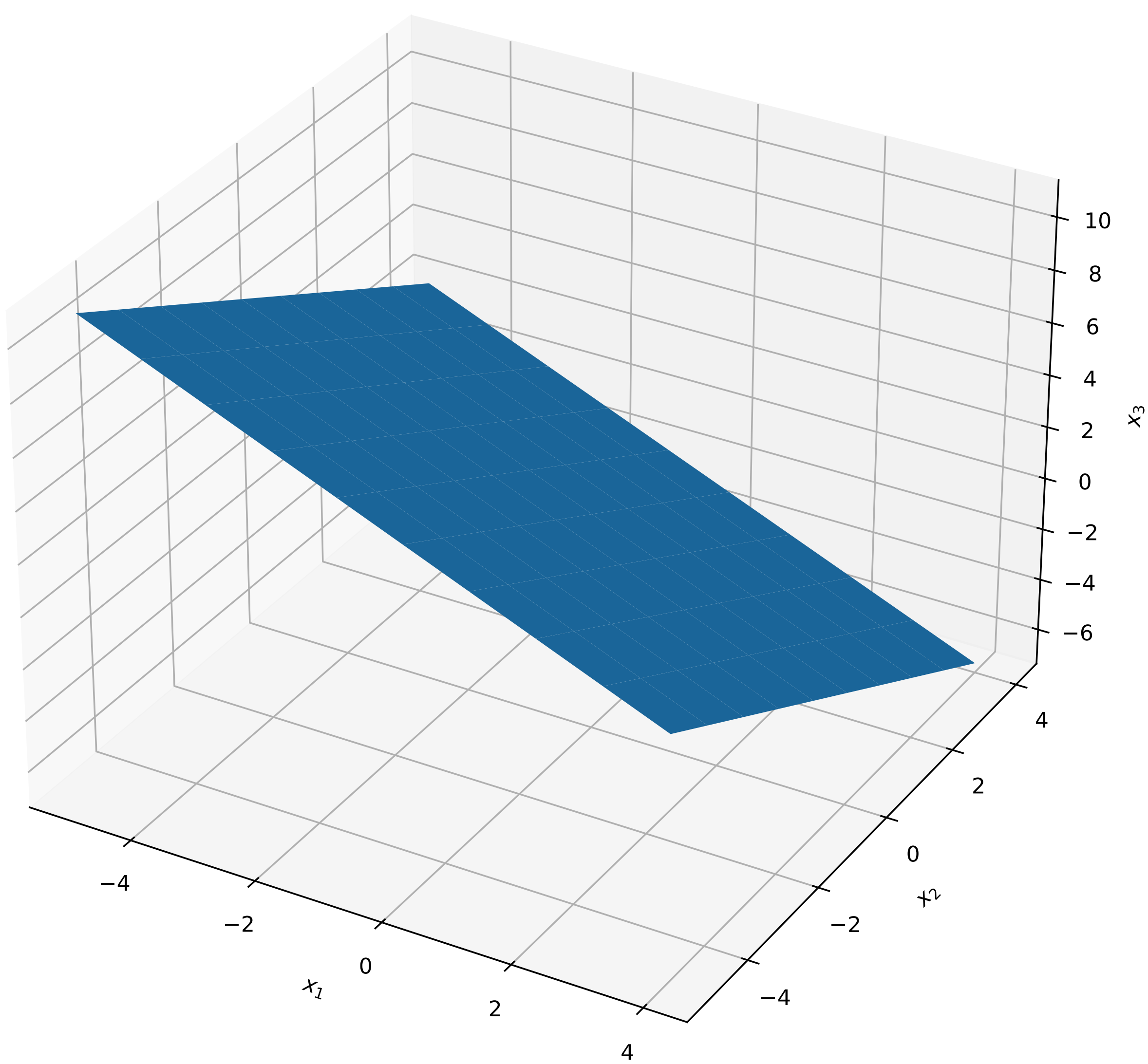
$$x_3 \leq 1 - x_1 - x_2$$

This is space is a *half space* but **not** a *subspace* nor an *affine space*.

Note: Due to my limited knowledge of 3D plots in matplotlib, I was unable to generate a 'fill-in' above the surface as shown below. A correct plot would encompass the points on the surface and any value above the surface.

In [25]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data.
x_1 = np.arange(-5, 5, 1)
x_2 = np.arange(-5, 5, 1)
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = 1 - x_1 - x_2

# Plot the surface.
surf = ax.plot_surface(x_1, x_2, x_3)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



(c)

For $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$ and $X = \{x \in \mathbb{R}^2 | Ax = 0\}$, the set is defined as:

$$Ax = 0$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} x = 0$$

$$\begin{bmatrix} x_1 + x_2 + x_3 \\ x_1 - x_2 \end{bmatrix} = 0$$

From this, we have two equations. We can solve one equation for x_2 with respect to x_1 such that,

$$x_1 - x_2 = 0$$
$$x_2 = x_1$$

Subsiting this in our other equation we find,

$$x_1 + x_2 + x_3 = 0$$

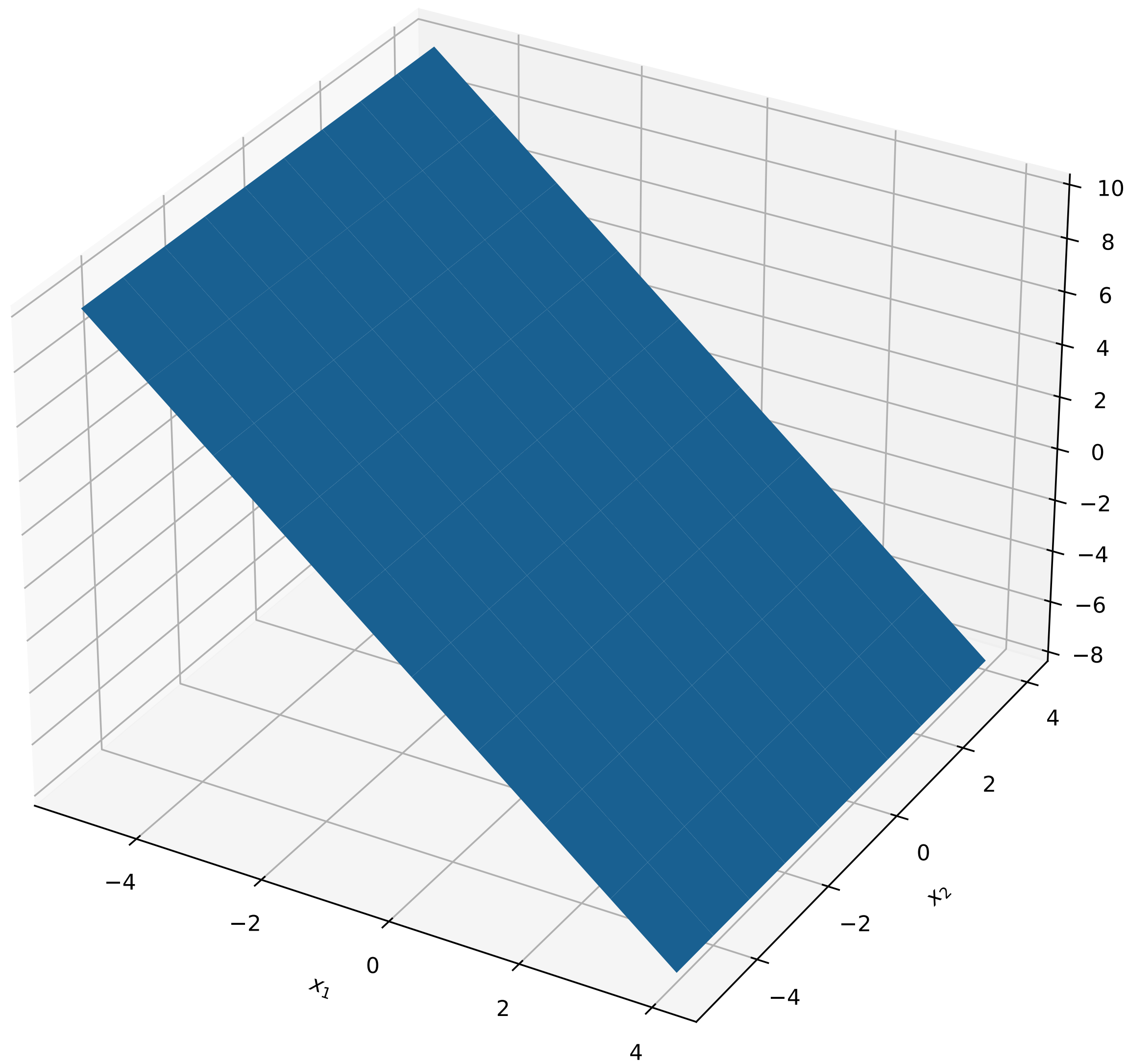
$$x_1 + (x_1) + x_3 = 0$$

$$x_3 = -2x_1$$

This is space is a *subspace* but **not** an *affine space* nor a *half space*.

In [26]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data.
x_1 = np.arange(-5, 5, 1)
x_2 = x_1
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = -2 * x_1

# Plot the surface.
surf = ax.plot_surface(x_1, x_2, x_3)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



For $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $X = \{x \in \mathbb{R}^2 | Ax = b\}$, the set is defined as:

$$Ax = b$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 + x_2 + x_3 \\ x_1 - x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

From this, we have two equations. We can solve one equation for x_2 with respect to x_1 such that,

$$x_1 - x_2 = 1$$

$$x_2 = x_1 - 1$$

Subsiting this in our other equation we find,

$$x_1 + x_2 + x_3 = 1$$

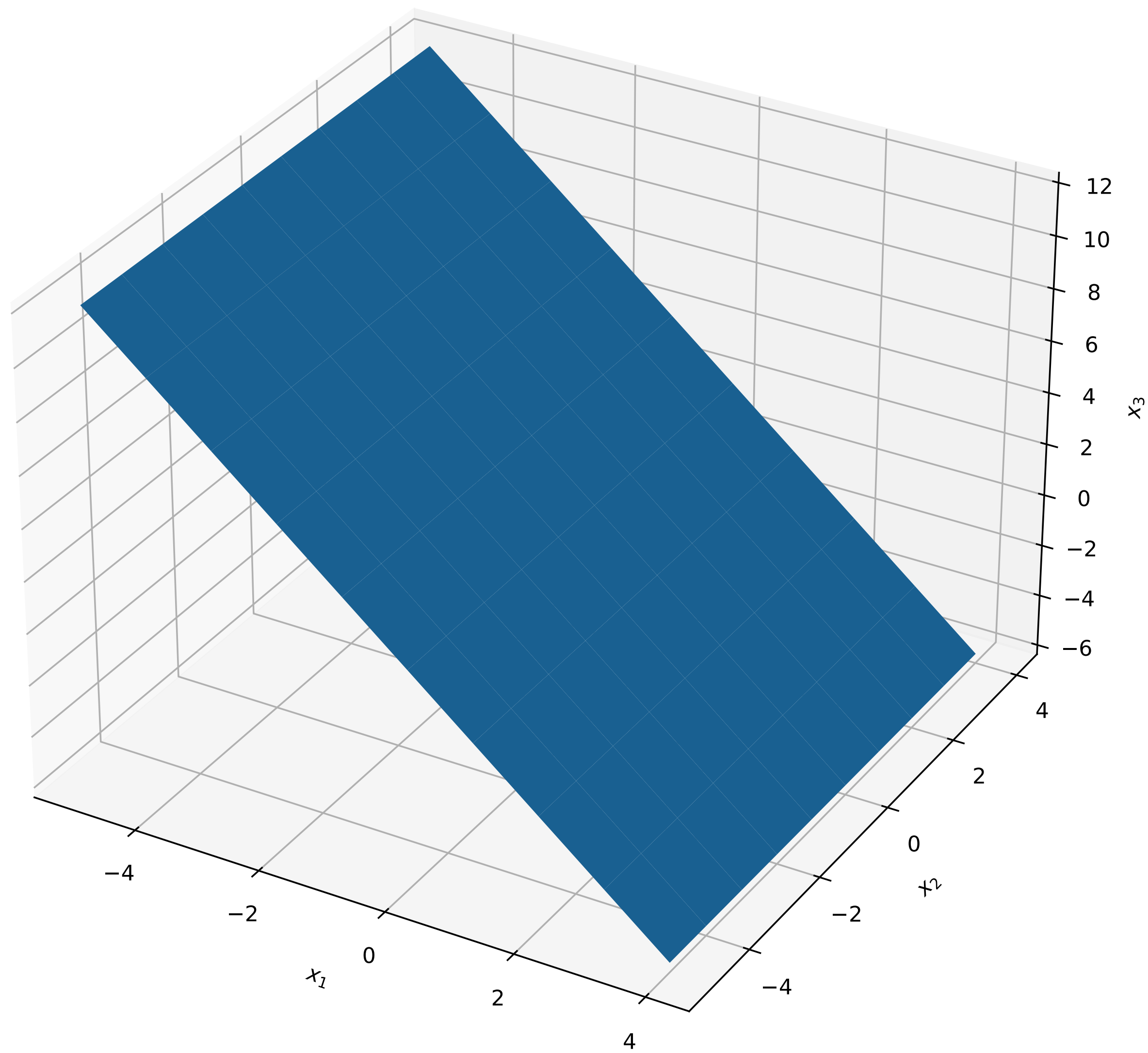
$$x_1 + (x_1 - 1) + x_3 = 1$$

$$x_3 = 2 - 2x_1$$

This is space is an *affine space* but **not** a *subspace* nor a *half space*.

In [27]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data.
x_1 = np.arange(-5, 5, 1)
x_2 = x_1
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = 2 - 2 * x_1

# Plot the surface.
surf = ax.plot_surface(x_1, x_2, x_3)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



For $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $X = \{x \in \mathbb{R}^2 | Ax \leq b\}$, the set is defined as:

$$Ax \leq b$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 + x_2 + x_3 \\ x_1 - x_2 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

From this, we have two equations. We can plot both equations on the graph and identify the region that satisfies both equations. We first solve the bottom row,

$$x_1 - x_2 \leq 1$$

$$x_2 \geq x_1 - 1$$

Solving the top row,

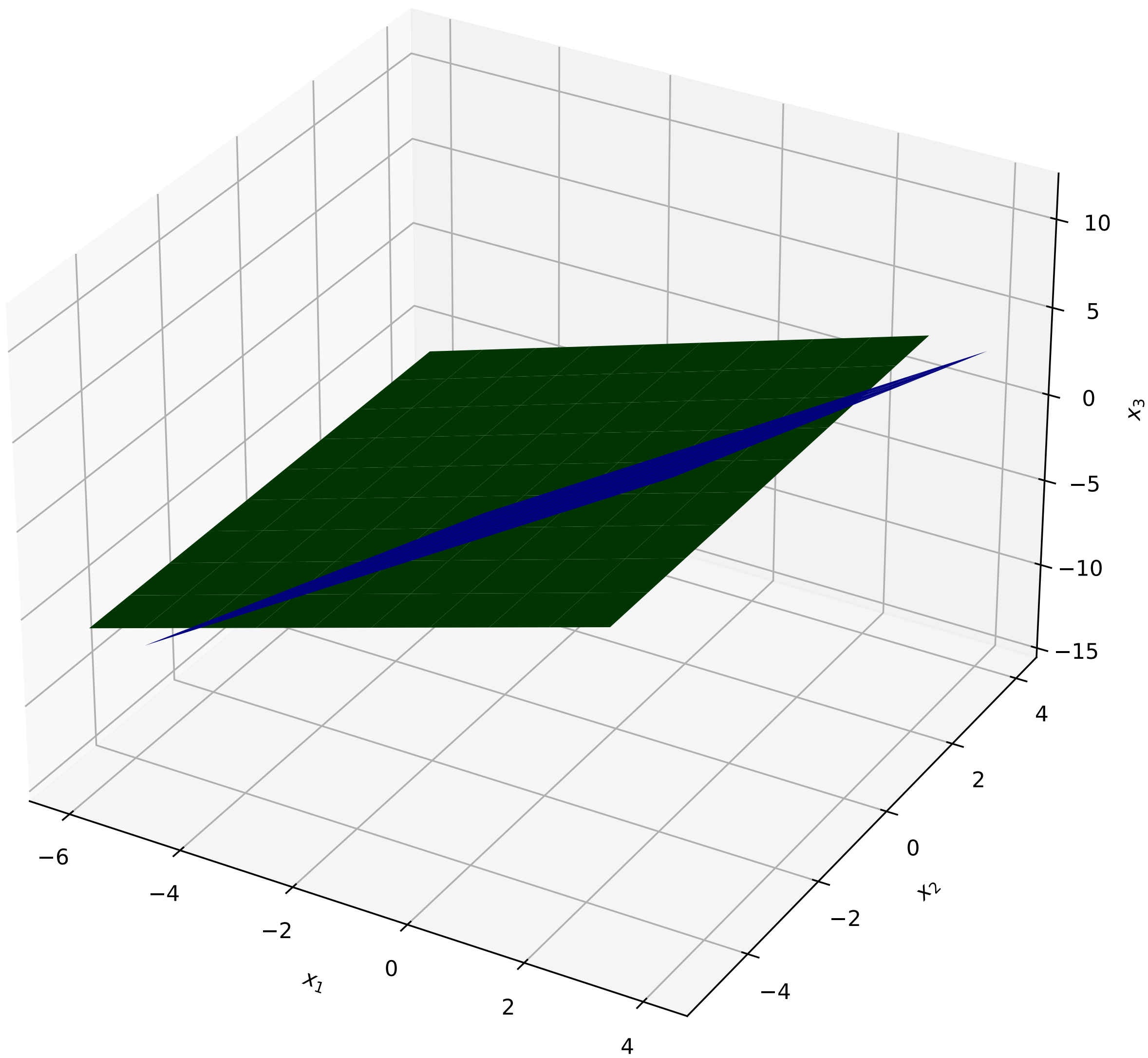
$$x_1 + x_2 + x_3 \leq 1$$

$$x_3 \leq 1 - x_2 - x_3$$

This is space is a *half space* but **not** a *subspace* nor an *affine space*.

In [28]:

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

# Make data for the bottom row
n_1 = np.arange(-5, 5, 1)
n_2 = n_1 - 1
n_2, n_1 = np.meshgrid(n_1, n_2)
n_3 = n_1

# Make data for the top row
x_1 = np.arange(-5, 5, 1)
x_2 = x_1
x_2, x_1 = np.meshgrid(x_1, x_2)
x_3 = 1 - x_2 - x_3

# Plot the surface.
ax.plot_surface(x_1, x_2, x_3, color='blue')
ax.plot_surface(n_1, n_2, n_3, color='green')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```



5. Coordinates

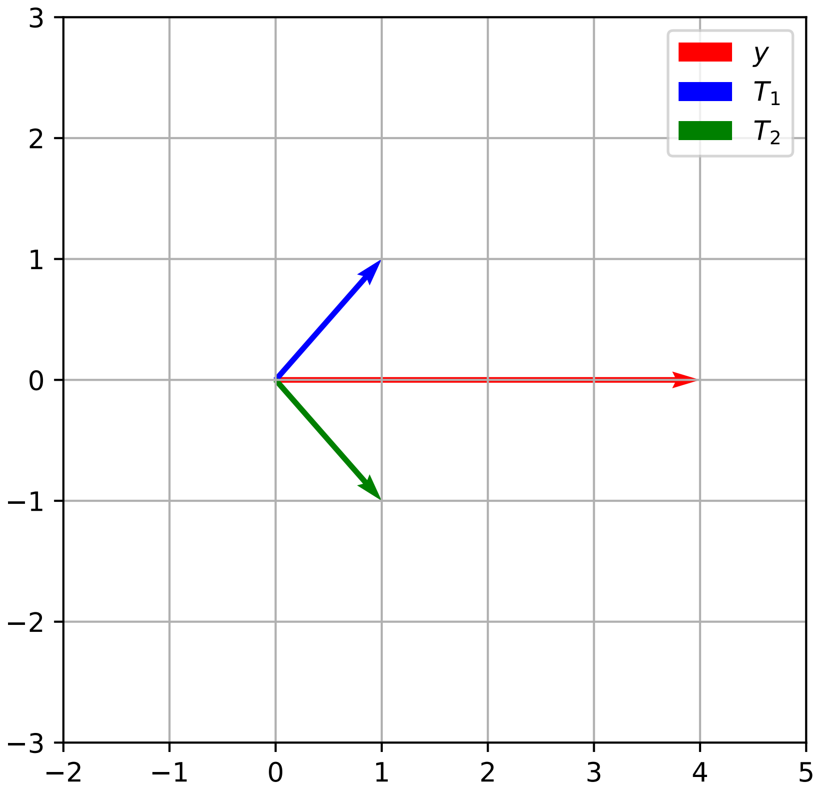
(a)

Given $y = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, we can plot the columns of the matrix T and y to compute the coordinates of the vector y with respect to new basis.

In [29]:

```
y = np.array([[4], [0]])
T = np.array([[1, 1], [1, -1]])

origin = np.array([[0], [0]])

fig, ax = plt.subplots(figsize=(5, 5))
origin = np.array([[0, 0, 0], [0, 0, 0]])
ax.quiver([0], [0], y[0], y[1], angles='xy', color='r', scale_units='xy', scale=1, labe
ax.quiver([0], [0], T[0,0], T[1,0], angles='xy', color='b', scale_units='xy', scale=1,
ax.quiver([0], [0], T[0,1], T[1,1], angles='xy', color='g', scale_units='xy', scale=1,
```

```
ax.set_xlim([-2, 5])
ax.set_ylim([-3, 3])
ax.grid()
ax.legend()
plt.show()
```



By solving for x where $y = Tx$, we find that $x = T^{-1}y$. Solving for x, we find that $x = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$.

In [30]:

```
x = np.linalg.inv(T).dot(y)

print('Coordinates of y with respect to new basis:\n', x)
```

```
Coordinates of y with respect to new basis:
 [[2.]
 [2.]]
```

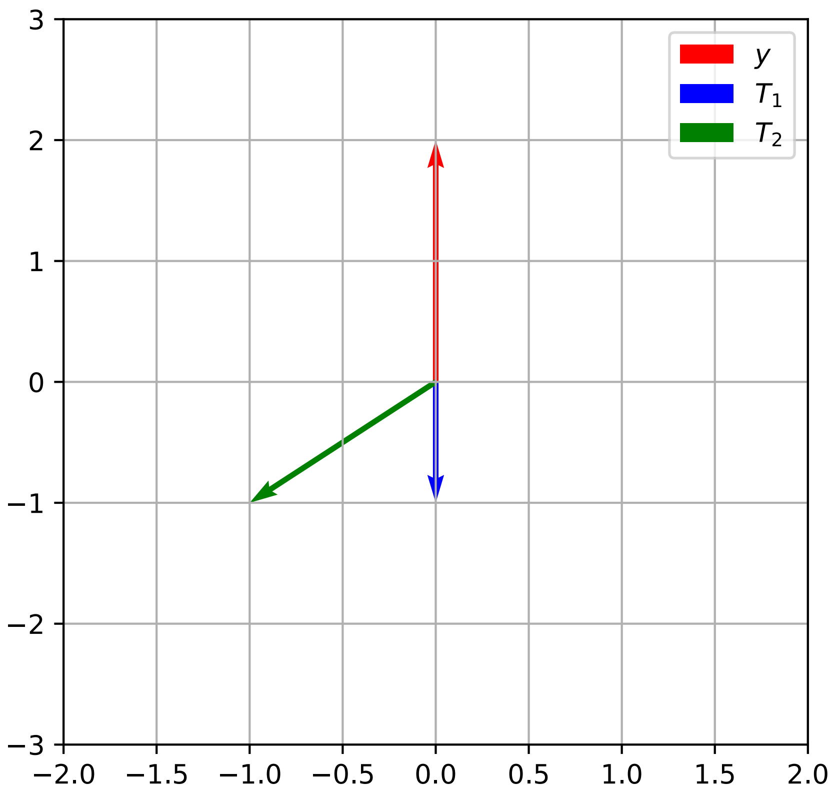
(b)

Given $y = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$ and $T = \begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix}$, we can plot the columns of the matrix T and y to compute the coordinates of the vector y with respect to new basis.

In [31]:

```
y = np.array([[0], [2]])
T = np.array([[0, -1], [-1, -1]])

origin = np.array([[0], [0]])

fig, ax = plt.subplots(figsize=(5, 5))
origin = np.array([[0, 0, 0], [0, 0, 0]])
ax.quiver([0], [0], y[0], y[1], angles='xy', color='r', scale_units='xy', scale=1, labe
```

```
ax.quiver([0], [0], T[0, 0], T[1, 0], angles='xy', color='b', scale_units='xy', scale=1
ax.quiver([0], [0], T[0,1], T[1,1], angles='xy', color='g', scale_units='xy', scale=1,
ax.set_xlim([-2, 2])
ax.set_ylim([-3, 3])
ax.grid()
ax.legend()
plt.show()
```



By solving for x where $y = Tx$, we find that $x = T^{-1}y$. Solving for x, we find that $x = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$.

In [32]:

```
x = np.linalg.inv(T).dot(y)

print('Coordinates of y with respect to new basis:\n', x)
```

```
Coordinates of y with respect to new basis:
 [[-2.]
 [ 0.]]
```

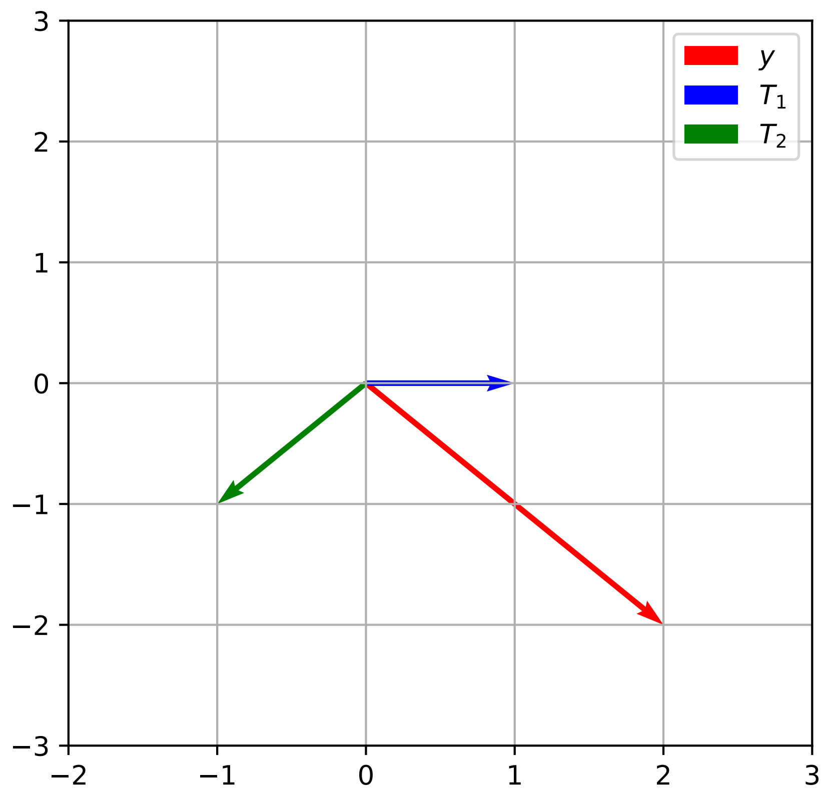
(c)

Given $y = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix}$, we can plot the columns of the matrix T and y to compute the coordinates of the vector y with respect to new basis.

In [33]:

```
y = np.array([[2], [-2]])
T = np.array([[1, -1], [0, -1]])

origin = np.array([[0], [0]])

fig, ax = plt.subplots(figsize=(5, 5))
```

```
origin = np.array([[0, 0, 0], [0, 0, 0]])
ax.quiver([0], [0], y[0], y[1], angles='xy', color='r', scale_units='xy', scale=1, labe
ax.quiver([0], [0], T[0, 0], T[1, 0], angles='xy', color='b', scale_units='xy', scale=1
ax.quiver([0], [0], T[0, 1], T[1, 1], angles='xy', color='g', scale_units='xy', scale=1
ax.set_xlim([-2, 3])
ax.set_ylim([-3, 3])
ax.grid()
ax.legend()
plt.show()
```



By solving for x where $y = Tx$, we find that $x = T^{-1}y$. Solving for x, we find that $x = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$.

In [34]:

```
x = np.linalg.inv(T).dot(y)

print('Coordinates of y with respect to new basis:\n', x)
```

```
Coordinates of y with respect to new basis:
 [[4.]
 [2.]]
```

6. Finding a Nullspace Basis

(a) Basis Derivation

Given that $A \in \mathbb{R}^{m \times n} (m < n)$ and $B = \begin{bmatrix} -A_1^{-1}A_2 \\ I \end{bmatrix}$.

(i)

We assume that $v \in \mathcal{N}(A)$ and thus we know that $Av = 0$, i.e. $\begin{bmatrix} A_1 & A_2 \end{bmatrix} v = 0$.

We let $v = \begin{bmatrix} u \\ w \end{bmatrix}$ where u and w satisifies the relationship $A_1 u + A_2 w = 0$. Given that A_1 is a square matrix, we can multiply the equation by A_1^{-1} such that

$$u + A_1^{-1} A_2 w = 0$$

$$u = -A_1^{-1} A_2 w$$

Given this relationship, we can now consider the product

$$Bw = \begin{bmatrix} -A_1^{-1} A_2 \\ I \end{bmatrix} w$$

$$Bw = \begin{bmatrix} -A_1^{-1} A_2 w \\ Iw \end{bmatrix}$$

$$Bw = \begin{bmatrix} u \\ w \end{bmatrix} = v.$$

Thus, we can see that any vector $v \in \mathcal{N}(A)$ can be written as $v = Bw$ for some $w \in \mathbb{R}^{n-m}$.

(ii)

Let us assume there exists a column vector $c = [\, c_1 c_2 \ldots c_{n-m} \,]^T$ such that $Bc = 0$.

Given $Bc = 0$,

$$\begin{bmatrix} -A_1^{-1} A_2 \\ I \end{bmatrix} c = 0$$

$$\begin{bmatrix} -A_1^{-1} A_2 c \\ c \end{bmatrix} = 0$$

$$c = 0.$$

Thus, the column vector c is the zero vector, which implies the columns B are linearly independent.

(b) Computation

(i)

Given $A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \end{bmatrix}$, we can solve for the basis of the nullspace as follows,

$$Ax = 0$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1 + x_4 - x_6 \\ x_2 + x_5 \\ x_3 + 2x_4 \end{bmatrix} = 0$$

Given that we have 6 variables and 3 equations, there are infinitely many solutions. Thus, we can choose to solve for 3 of the variables - specifically x_1, x_2, and x_3.

$$x_1 = -x_4 + x_6$$

$$x_2 = -x_5$$

$$x_3 = -2x_4$$

Writing this in vector form, we see that

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -2 \\ 1 \\ 0 \\ 0 \end{bmatrix} x_4 + \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} x_5 + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} x_6$$

Therefore, the null space has a basis formed by the set $\left\{ \begin{bmatrix} -1 \\ 0 \\ -2 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}$.

(ii)

Given $A = \begin{bmatrix} 2 & -1 & 1 & 2 \\ 1 & 1 & 3 & 4 \end{bmatrix}$, we can solve for the basis of the nullspace as follows.

First, we must transform the matrix A into reduced row echelon form. This is performed by a series of row operation,

$$A = \begin{bmatrix} 2 & -1 & 1 & 2 \\ 1 & 1 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1/2 & 1/2 & 1 \\ 1 & 1 & 3 & 4 \end{bmatrix} \qquad \text{(multiply first row by 1/2)}$$

$$\begin{bmatrix} 1 & -1/2 & 1/2 & 1 \\ 0 & 3/2 & 5/2 & 3 \end{bmatrix} \qquad (\text{add } -1 \text{ times the 1st row to the 2nd row})$$

$$\begin{bmatrix} 1 & -1/2 & 1/2 & 1 \\ 0 & 1 & 5/3 & 2 \end{bmatrix} \qquad (\text{multiply 2nd row by } 2/3)$$

$$\begin{bmatrix} 1 & 0 & 4/3 & 2 \\ 0 & 1 & 5/3 & 2 \end{bmatrix} \qquad (\text{add } 1/2 \text{times the 2nd row to the 1st row})$$

Now we can solve the equation $Ax = 0$,

$$Ax = 0$$

$$\begin{bmatrix} 1 & 0 & 4/3 & 2 \\ 0 & 1 & 5/3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1 + \frac{4}{3}x_3 + 2x_4 \\ x_2 + \frac{5}{3}x_3 + 2x_4 \end{bmatrix} = 0$$

Given that we have 4 variables and 2 equations, there are infinitely many solutions. Thus, we can choose to solve for 2 of the variables - specifically x_1 and x_2.

$$x_1 = -\frac{4}{3}x_3 + 2x_4$$

$$x_2 = -\frac{5}{3}x_3 - 2x_4$$

Writing this in vector form, we see that

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -\frac{4}{3} \\ -\frac{5}{3} \\ 1 \\ 0 \end{bmatrix} x_3 + \begin{bmatrix} -2 \\ -2 \\ 0 \\ 1 \end{bmatrix} x_4$$

Therefore, the null space has a basis formed by the set $\left\{ \begin{bmatrix} -\frac{4}{3} \\ -\frac{5}{3} \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -2 \\ -2 \\ 0 \\ 1 \end{bmatrix} \right\}$.