Homework Assignment 1

Dillon Ford: A16092047

1 (20 points) Basic Calculus

1.1 Derivatives with Scalars

1. $f(x) = \lambda + e^{\lambda} + e^{\lambda x}$ where λ is a constant scalar, derive $\frac{\partial f(x)}{\partial x}$

Solution:

$\frac{\partial f(x)}{\partial x}$ = 0 + 0 + $e^{\lambda x}\lambda$

= $\boxed{\lambda e^{\lambda x}}$

2. $f(x) = \ln\left(1 + e^x + e^{2x}\right)$, derive $\frac{\partial f(x)}{\partial x}$

Solution:

$\frac{\partial f(x)}{\partial x}$ = $\frac{1}{1+e^x+e^{2x}}$ $\frac{\partial}{\partial x}(1 + e^x + e^{2x})$

= $\frac{1}{1+e^x+e^{2x}}$ $(e^x + 2e^{2x})$

= $\boxed{\dfrac{e^x + 2e^{2x}}{1 + e^x + e^{2x}}}$

1.2 Derivatives with Vectors

1. $f(\mathbf{x}) = \lambda - \mathbf{x}^T\mathbf{A}^T\mathbf{x}$ **where** $\mathbf{A}$ **is a symmetric matrix and** λ **is a constant scalar, derive** $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$

Solution:

Since A is a symmetric matrix, $A^T = A$

We can rewrite $f(\mathbf{x}) = \lambda - \mathbf{x}^T\mathbf{A}\mathbf{x}$

Then,

$\frac{\partial f(x)}{\partial x}$ = 0 - $\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}}$

= $\boxed{-2Ax}$

2. $f(\mathbf{x}) = (\mathbf{a}+\mathbf{x})^T(\mathbf{a}+\lambda\mathbf{x})$ **where** λ **is a constant scalar, derive** $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$

Solution:

Expanding out $f(\mathbf{x})$ we get,

$f(\mathbf{x}) = a^Ta + \lambda a^Tx + x^Ta + \lambda x^Tx$

$\frac{\partial f(x)}{\partial x}$ = $0 + \lambda a + a + \lambda(x\frac{\partial(x)}{\partial x} + x\frac{\partial(x)}{\partial x})$

= $\lambda a + a + \lambda(x + x)$

= $\lambda a + a + \lambda(2x)$

= $\boxed{(\lambda+1)a + 2\lambda x}$

2 (10 points) Solving Equations

1. Given a system of linear equations:

$$\begin{cases} -3w_1 + 4w_2 & = 1 \\ 2w_1 - w_2 & = 2 \end{cases}$$

(a) Solve for (w_1, w_2).

(b) Once you obtain (w_1, w_2) in (a), please calculate final result y from $y = w_1 x_1 + w_2 x_2$ when $(x_1, x_2) = (2, 3)$

Solution:

(a):

$2w_1 - w_2 = 2$

$\Rightarrow w_2 = 2w_1 - 2$

$\Rightarrow -3w_1 + 4(2w_1 - 2) = 1$

$\Rightarrow -3w_1 + 8w_1 - 8 = 1$

$\Rightarrow 5w_1 = 9$

$\Rightarrow w_1 = \frac{9}{5}$

$\Rightarrow w_2 = 2(\frac{9}{5}) - 2$

$\Rightarrow w_2 = \frac{8}{5}$

Hence, $\boxed{(w_1, w_2) = (\frac{9}{5}, \frac{8}{5})}$

(b):

$y = w_1 x_1 + w_2 x_2$, when $(x_1, x_2) = (2, 3)$

$\Rightarrow y = \frac{9}{5}x_1 + \frac{8}{5}x_2$

$\Rightarrow y = \frac{9}{5}(2) + \frac{8}{5}(3)$

$\Rightarrow y = \frac{18}{5} + \frac{24}{5}$

$\Rightarrow \boxed{y = \frac{42}{5}}$

2. Given a system of equations:

$$\begin{cases} -3w_1 + 4w_2 & = b \\ w_1 - 2w_2 & = -b \\ w_1^2 + w_2^2 & = 1 \end{cases}$$

Solve for (w_1, w_2, b)

Solution:

Adding the first 2 equations yields,

$-2w_1 + 2w_2 = 0$

$\Rightarrow 2w_2 = 2w_1$

$\Rightarrow w_2 = w_1$

$\Rightarrow w_1^2 + (w_1)^2 = 1$

$\Rightarrow w_1 = \frac{1}{\sqrt{2}}$

$\Rightarrow w_2 = \frac{1}{\sqrt{2}}$

$w_1 - 2w_2 = -b$

$\Rightarrow \frac{1}{\sqrt{2}} - \frac{2}{\sqrt{2}} = -b$

$\Rightarrow -\frac{1}{\sqrt{2}} = -b$

$\Rightarrow b = \frac{1}{\sqrt{2}}$

Hence, $\boxed{(w_1, w_2, b) = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})}$

3 (10 points) Basic NumPy Operations

In [1]:
```
import numpy as np
a = np.array([3,1,7])
b = np.array([-3,4,-5])
A = np.array([[1,4], [1,3], [7,9]])
B = np.array([[2,-4], [-3,7], [6,-5]])
```

3.1 Vector

1. $\mathbf{a} \circ \mathbf{b}$

In [2]:
```
print(np.multiply(a,b))
```

```
[ -9   4 -35]
```

2. $\mathbf{a} \cdot \mathbf{b}$

In [3]:
```
print(np.dot(a,b))
```

```
-40
```

3. 2 Matrix

1. $A \circ B$

In [4]:
```
print(np.multiply(A,B))
```

```
[[  2 -16]
 [ -3  21]
 [ 42 -45]]
```

2. $A^T B$

In [5]:
```
print(np.transpose(A).dot(B))
```

```
[[ 41 -32]
 [ 53 -40]]
```

3. AB^T

In [6]:
```
print(np.dot(A, B.transpose()))
```

```
[[-14  25 -14]
 [-10  18  -9]
 [-22  42  -3]]
```

4. $A + B$

In [7]:
```
print(np.add(A,B))
```

```
[[ 3  0]
 [-2 10]
 [13  4]]
```

5. $A - B$

In [8]:
```
print(np.subtract(A,B))
```

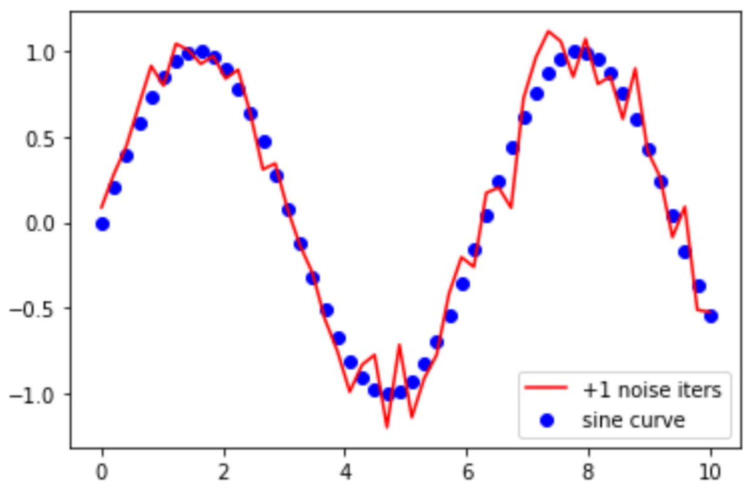
```
[[-1  8]
 [ 4 -4]
 [ 1 14]]
```

4 (15 points) Basic Plots Using Matplotlib

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
```

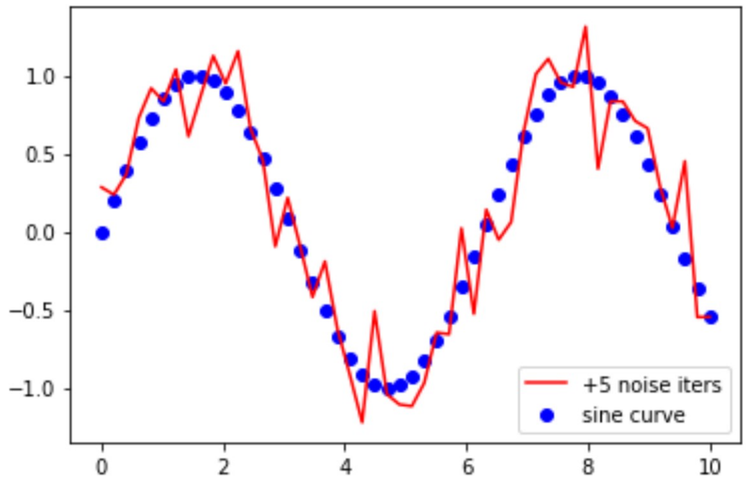
1. apply 1 iteration of the Gaussian noise (0 mean, 0.1 standard deviation) on the original sine curve

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
space = np.linspace(0, 10, num = 50)
sine = np.sin(space)
sine_plusnoise = sine
for i in range(2):
  sine_plusnoise = sine_plusnoise + np.random.normal(scale = 0.1, size
= 50)
  if i in [1]:
    plt.scatter(space, sine, color = 'b', label = 'sine curve')
    plt.plot(space, sine_plusnoise, color = 'r', label = '+{} noise ite
rs'.format(i))
    plt.legend(loc = 'lower right')
    plt.show()
```



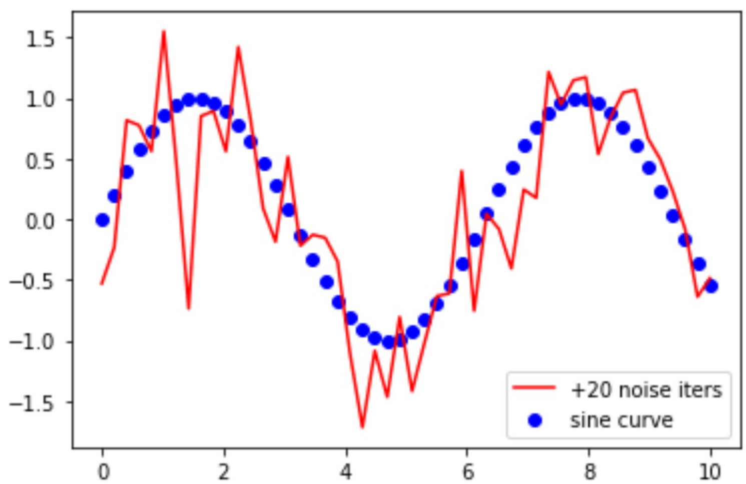
2. apply 5 iterations of the Gaussian noise (0 mean, 0.1 standard deviation) on the original sine curve

In [122]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
space = np.linspace(0, 10, num = 50)
sine = np.sin(space)
sine_plusnoise = sine
for i in range(6):
  sine_plusnoise = sine_plusnoise + np.random.normal(scale = 0.1, size
= 50)
  if i in [5]:
    plt.scatter(space, sine, color = 'b', label = 'sine curve')
    plt.plot(space, sine_plusnoise, color = 'r', label = '+{} noise it
ers'.format(i))
    plt.legend(loc = 'lower right')
    plt.show()
```



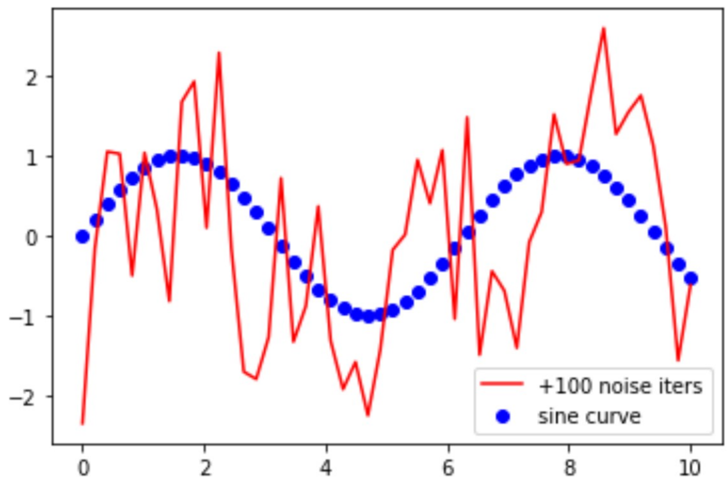
3. apply 20 iterations of the Gaussian noise (0 mean, 0.1 standard deviation) on the original sine curve

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
space = np.linspace(0, 10, num = 50)
sine = np.sin(space)
sine_plusnoise = sine
for i in range(21):
  sine_plusnoise = sine_plusnoise + np.random.normal(scale = 0.1, size
= 50)
  if i in [20]:
    plt.scatter(space, sine, color = 'b', label = 'sine curve')
    plt.plot(space, sine_plusnoise, color = 'r', label = '+{} noise ite
rs'.format(i))
    plt.legend(loc = 'lower right')
    plt.show()
```



4. apply 100 iterations of the Gaussian noise (0 mean, 0.1 standard deviation) on the original sine curve

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
space = np.linspace(0, 10, num = 50)
sine = np.sin(space)
sine_plusnoise = sine
for i in range(101):
  sine_plusnoise = sine_plusnoise + np.random.normal(scale = 0.1, size
= 50)
  if i in [100]:
    plt.scatter(space, sine, color = 'b', label = 'sine curve')
    plt.plot(space, sine_plusnoise, color = 'r', label = '+{} noise ite
rs'.format(i))
    plt.legend(loc = 'lower right')
    plt.show()
```



5 (10 points) Data Visualization

In [14]:
```
import matplotlib.pyplot as plt
from sklearn import datasets
from mpl_toolkits.mplot3d import Axes3D
```

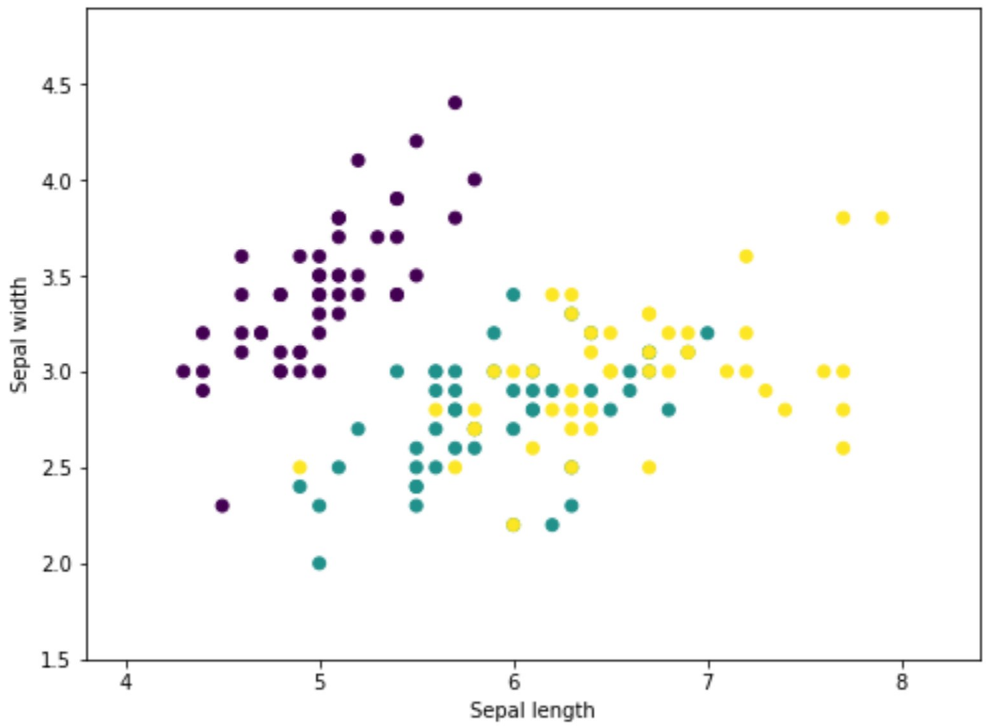
Load Iris dataset into Python:

In [15]:
```
iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

1. Show a scatter plot for the first 2 feature dimensions in 2-D space.

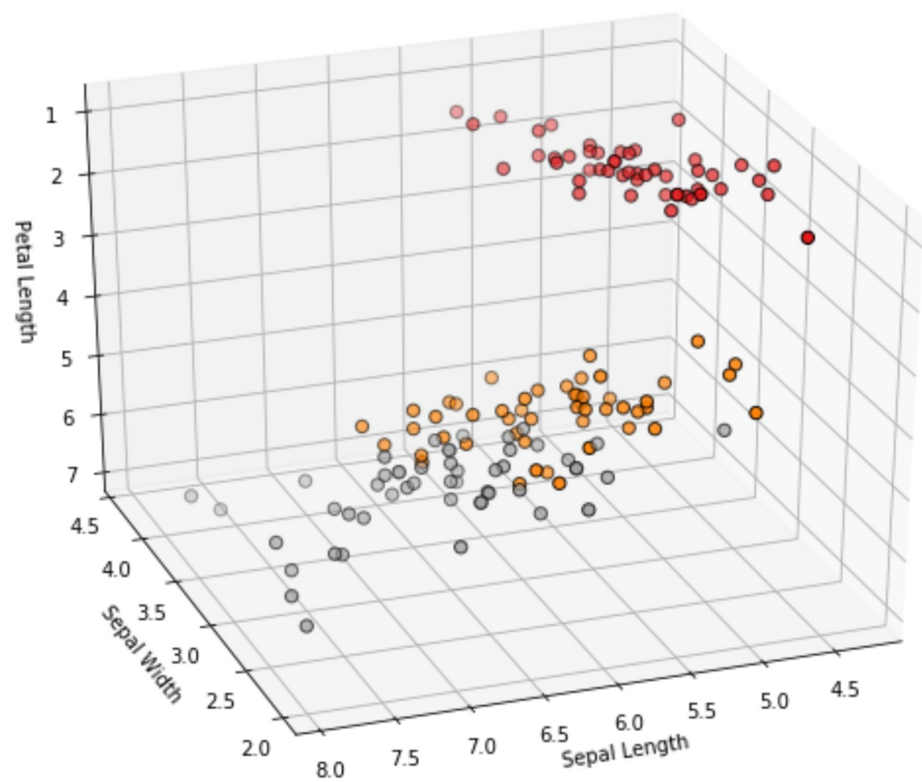
In [16]:
```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c = Y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
```

Out[16]: (1.5, 4.9)



2. Show a scatter plot for the first 3 feature dimensions in 3-D space.

In [17]:
```
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=Y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)

ax.set_xlabel("Sepal Length")
ax.set_ylabel("Sepal Width")
ax.set_zlabel("Petal Length");
```

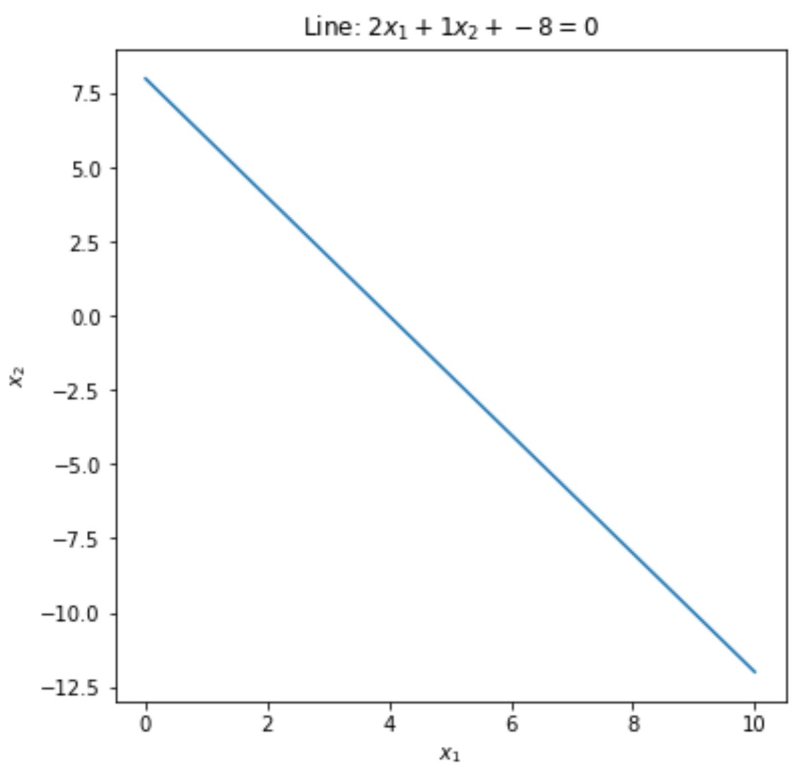


6 (10 points) Ploting Decision Boundaries

1. $w_1 x_1 + w_2 x_2 + b = 0$ **where** $(w_1, w_2, b) = (2, 1, -8)$

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt

def plot_line(w1,w2,b,num=30):
    '''plots a 1d line in 2d space
    across $x_1,x_2 \in [0,10] with num samples per dimension$
    as defined by eqn $w_1 x_1^2 + w_2 x_2^2 + b = 0$ '''

    # sets up space to plot np.linspace() used for 2d plotting
    X1_plane = np.linspace(0, 10, num)
    X2_plane = np.linspace(0, 10, num)

    # this defines the equation to plot
    X2_plane = (b + w1 * X1_plane)/(-w2)

    # does plotting
    plt.figure(figsize = (6, 6))
    plt.plot(X1_plane, X2_plane)
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title("Line: ${}x_1+{}x_2+{}=0$".format(w1,w2,b))
    plt.show()
```

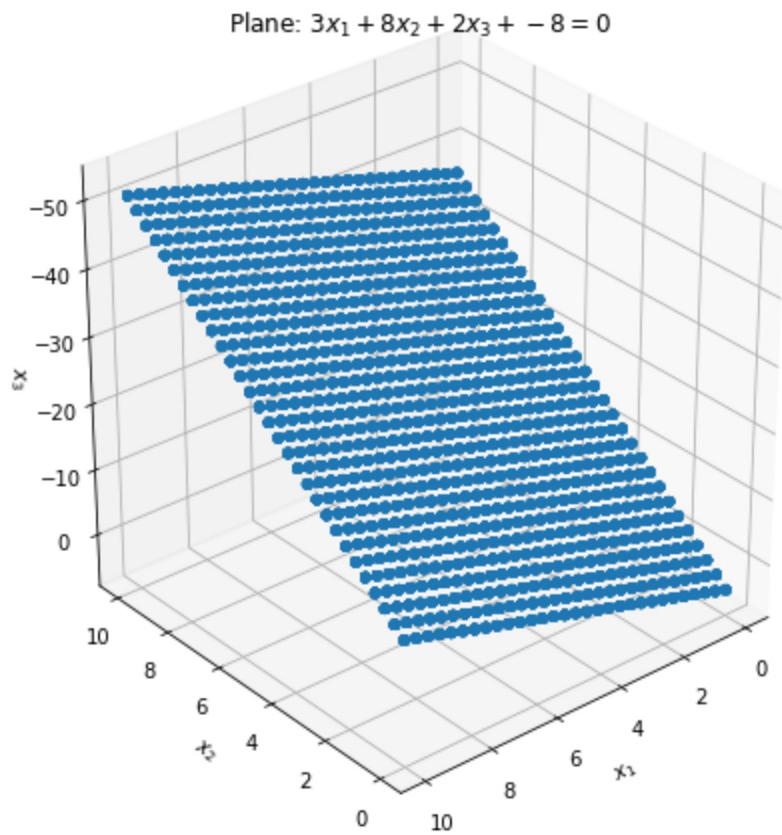
In [19]:
```
plot_line(2,1,-8)
```



2. $w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0$ **where** $(w_1, w_2, w_3, b) = (3, 8, 2, -8)$

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def plot_plane(w1, w2, w3, b, num = 30):
    '''plots a 2d plane in 3d space
    cross $x_1,x_2,x_3 \in [0,10] with num samples per dimension$
    as defined by eqn $w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0$ '''
    X_range = np.linspace(0,10,num)

  # sets up space to plot, 3d equivalent of np.linspace() used for 2d p
lotting
    X1_plane, X2_plane, X3_plane = np.meshgrid(X_range, X_range, X_rang
e)

  # this defines the equation to plot
    X3_plane = (b + w1 * X1_plane + w2 * X2_plane)/(-w3)

  # does plotting
    fig = plt.figure(figsize = (6, 6))
    ax = Axes3D(fig, elev = -150, azim = 130)
    ax.scatter(X1_plane, X2_plane, X3_plane)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('$x_3$')
    plt.title("Plane: ${}x_1+{}x_2+{}x_3+{}=0$".format(w1,w2,w3,b))
    plt.show()
```

In [21]:
```
plot_plane(3,8,2,-8)
```

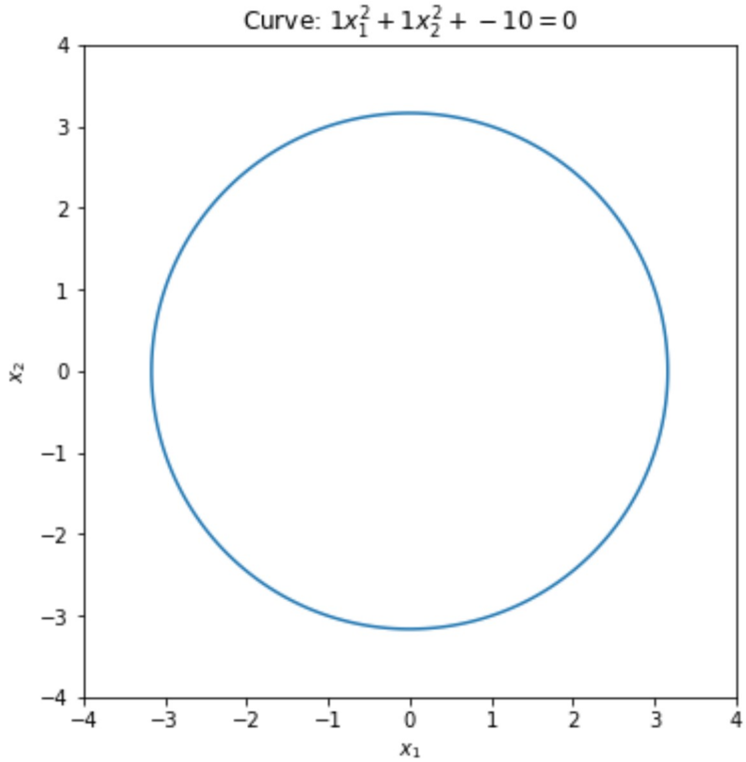


3. $w_1 x_1^2 + w_2 x_2^2 + b = 0$ **where** $(w_1, w_2, b) = (1, 1, -10)$

In [118]:

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

def plot_curve(w1,w2,b,num=300):
    '''plots a 1d curve in 2d space
    across $x_1,x_2 \in [0,10] with num samples per dimension$
    as defined by eqn $w_1 x_1 + w_2 x_2 + b = 0$ '''

    # sets up space to plot np.linspace() used for 2d plotting
    X1_plane = np.linspace(-4, 4, num)
    X2_plane = np.linspace(-4, 4, num)

    # for use of contour plot
    X, Y = np.meshgrid(X1_plane, X2_plane)

    # this defines the equation to plot
    Z = w1*X**2 + w2*Y**2 + b

    # does plotting
    fig, ax = plt.subplots(figsize=(6,6))
    ax.contour(X,Y,Z,[0], colors ='#1f77b4')
    plt.gca().set_aspect('equal')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title("Curve: ${}x_1^2+{}x_2^2+{}=0$".format(w1,w2,b))
    plt.show()
```

In [121]:
```
plot_curve(1,1,-10)
```



7 (25 points) Data Manipulation

1. Show the first 2 features of the first 5 data points (i.e. first 2 columns and first 5 rows) of array X. (You can print the 5×2 array).

In [24]:
```
print(X[0:5,:2])
```

```
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]]
```

2. Calculate the mean and the variance of the 2nd feature (the 2nd column) of array X.

In [25]:
```
print('Variance:',np.var(X[:,1:2]))
print('Mean:',np.mean(X[:,1:2]))
```

```
Variance: 0.18871288888888889
Mean: 3.0573333333333337
```

3. Perform a linear projection on the 3 features of all data points using the weight vector w, where w = (3, 2, 1). You can do so by calculating a dot product between the 3 features of all data points and the weight vector w. The shape of projected data points should be (150, 1) or (150,), depending on the weight vector is regarded as a matrix or a vector. Calculate the mean of the projected data points.
Hint: You can calculate the projection with a single line code using np.dot, but you should be careful with the dimension matching for the dot product.

In [30]:
```
w = np.array([3,2,1])
lin_proj = np.dot(X[:,0:3],np.transpose(w))
print(lin_proj)
print('shape of projected data points:',np.shape(lin_proj))
print('mean of projected data points:',np.mean(lin_proj))
```

```
[23.7 22.1 21.8 21.5 23.6 25.7 22.  23.3 20.4 22.4 25.1 22.8 21.8 20.
 26.6 27.4 25.3 23.7 26.4 24.4 24.7 24.2 22.  23.6 23.1 22.6 23.4 24.
1
 23.8 22.1 22.2 24.5 25.3 26.3 22.4 22.6 24.8 23.3 20.5 23.6 23.3 19.
4
 20.9 23.6 24.8 21.8 24.5 21.6 24.8 23.  32.1 30.1 31.8 25.1 29.7 27.
2
 30.2 22.8 30.2 24.9 22.5 27.9 26.4 28.8 26.2 30.7 27.3 26.9 27.5 25.
7
 28.9 27.9 28.8 28.6 29.3 30.2 30.8 31.1 28.3 25.8 25.1 25.  26.7 28.
5
 26.7 29.3 31.  27.9 26.9 25.5 26.1 28.9 26.6 22.9 26.4 27.3 27.1 28.
7
 23.3 26.8 31.5 27.9 33.2 30.3 31.3 35.4 24.2 34.  30.9 34.9 31.  29.
9
 31.9 27.1 28.1 30.9 31.  37.4 35.2 27.4 32.8 27.3 35.4 29.2 32.4 34.
 29.  29.2 30.4 33.4 33.9 37.7 30.4 29.6 29.1 35.2 31.3 30.9 28.8 32.
3
 31.9 32.  27.9 32.7 32.4 31.3 28.9 30.7 30.8 28.8 23.7]
shape of projected data points: (151,)
mean of projected data points: 27.378145695364243
```

4. Randomly sample 3 data points (rows) of array X by randomly choosing the row indices. Show the indices and the sampled data points.
Hint: np.random.randint

In [27]:
```
print(X[np.random.randint(X.shape[0], size =3), :])
```

```
[[5.8 2.7 4.1 1. ]
 [7.6 3.  6.6 2.1]
 [5.1 3.8 1.6 0.2]]
```

5. Add one more feature (one more column) to the array X after the last feature. The values of the added feature for all data points are constant 1. Show the first data point (first row) of the new array. Hint: np.ones, np.hstack

In [28]:
```
X = np.hstack([X, np.ones([X.shape[0],1])])
print(X[0])
```

```
[5.1 3.5 1.4 0.2 1. ]
```

6. Add one more data point (one more row) to the array X after the last data point. The value of the added data point is the same as the first data point. Show the first feature (first column) of the new array. Hint: np.vstack

In [29]:
```
X = np.vstack([X, X[0]])
print(X[:,0])
```

```
[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9
5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9
5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9
6.1
 6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6
5.5
 5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9
7.3
 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7
7.2
 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8
6.8
 6.7 6.7 6.3 6.5 6.2 5.9 5.1]
```