1 (20 points) Basic Calculus

1.1 Derivatives with Scalars

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

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

2. $f(x) = \ln(1 - e^x)$, derive $\dfrac{\partial f(x)}{\partial x}$.

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

1.2 Derivatives with Vectors

Several particular vector derivatives are useful for the course. For matrix $\mathbf{A}$, column vector $\mathbf{x}$ and $\mathbf{a}$, we have

- $\frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \mathbf{a}$,
- $\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$. If $\mathbf{A}$ is symmetric, $\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{A}\mathbf{x}$.

The above rules adopt a *denominator-layout* notation. For more rules, you can refer to this Wikipedia page. Please apply the above rules and calculate following derivatives:

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

$$\frac{\partial f(x)}{\partial x} = -\lambda \frac{\partial a^T x}{\partial x} = -\lambda a$$

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

$$\frac{\partial f(x)}{\partial x} = -\lambda \frac{\partial (x^T A x)}{\partial x} = -\lambda (A + A^T) x = -2\lambda A x$$

↑ A is symmetric

2 (10 points) Solving Equations

1. Given a system of linear equations:

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

(a) Solve for (w_1, w_2).

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

$(w_1, w_2) = (1, 1)$

(b) Once you obtain (w_1, w_2) in (a), please calculate final result y from $y = w_1x_1 + w_2x_2$ when $(x_1, x_2) = (2, 3)$.

$y = 2 \times 1 + 3 \times 1 = 5$

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).

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

$-w_1 = -b$

$w_1 = b$

$w_2 = b$

$w_1^2 + w_2^2 = 2b^2 = 1$

$w_1 = w_2 = b = \frac{1}{\sqrt{2}}$

$(w_1, w_2, b) = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$

3 (5 points) One-Hot Encoding

A dataset S is denoted as $S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$, where each sample refers to the specification of a car.

Represent this dataset S using a matrix and show how your one-hot encoding is derived. **Hint**: (1) For the categorical features, you may use the one-hot encoding strategy. (2) You may choose either a row vector or a column vector to represent each data sample in your result. If you use a row vector to represent each data sample, the shape of the resulting matrix should be 5×9.

	Length (inch)	Height (inch)	Make	Color
$\mathbf{x}_1$	183	62	Toyota	Blue
$\mathbf{x}_2$	181	65	BMW	Silver
$\mathbf{x}_3$	182	59	BMW	Red
$\mathbf{x}_4$	179	68	Ford	Blue
$\mathbf{x}_5$	182	53	Toyota	Black

toyota	BMW	Ford
1	0	0
0	1	0
0	0	1

Blue	Red	Silver	Black
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$$S = \begin{bmatrix} 183 & 62 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 181 & 65 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 182 & 59 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 179 & 68 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 182 & 53 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

5×9

HW2_part2_COGS_118A_2021_Winter (1)

January 10, 2021

1 COGS 118A: Supervised Machine Learning Algorithms

1.1 Homework Assignment 2

1.1.1 Coding Part

Instructions: This Jupyter notebook contains starter codes for the coding questions of the assignment. Use this notebook as a starting point to code the answers to the coding questions. Then generate a PDF file of the notebook and combine it with the PDF file of your solution to the written questions and submit a **single PDF file** to Gradescope.

1.2 4 (10 points) Basic NumPy Operations

[31]:
```
import numpy as np
a = np.array([2,1,7])
b = np.array([-2,7,-5])
A = np.array([[2,4], [1,3], [7,9]])
B = np.array([[2,-2], [-3,7], [6,-5]])
# a ∘ b
np.multiply(a,b)
```

[31]:
```
array([ -4,   7, -35])
```

[32]:
```
# a * b
np.dot(a,b)
```

[32]:
```
-32
```

[33]:
```
# A ∘ B
np.multiply(A,B)
```

[33]:
```
array([[  4,  -8],
       [ -3,  21],
       [ 42, -45]])
```

[34]:
```
# AT*B
np.matmul(np.transpose(A),B)
```

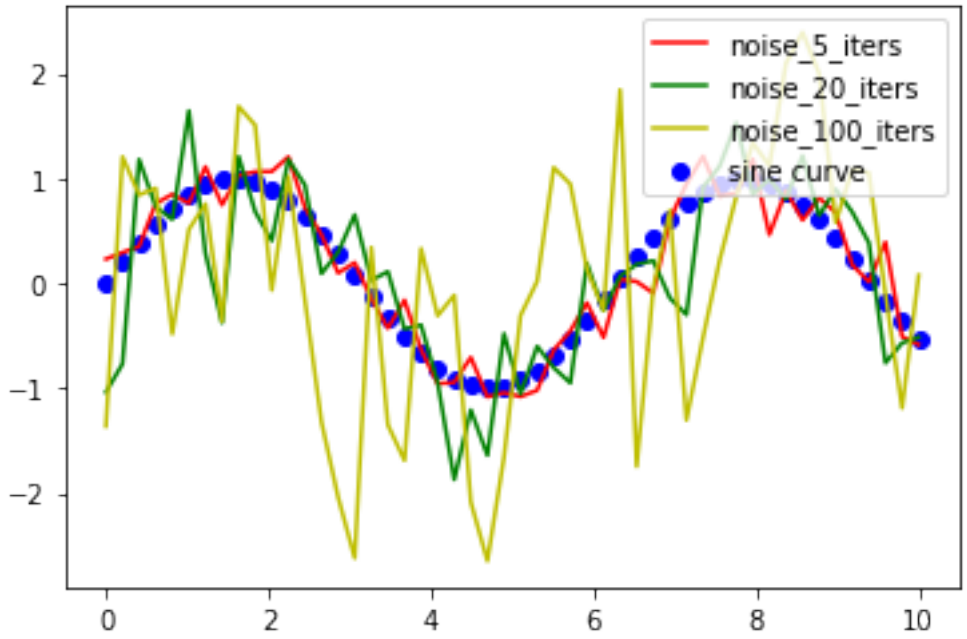
[34]:
```
array([[ 43, -32],
       [ 53, -32]])
```

[35]:
```
# A*BT
np.matmul(A,np.transpose(B))
```

[35]:
```
array([[-4, 22, -8],
       [-4, 18, -9],
       [-4, 42, -3]])
```

1.3 5 (10 points) Basic Plots Using Matplotlib

[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_5 = sine
sine_20 = sine
sine_100 = sine

for i in range(5):
    sine_5 = sine_5 + np.random.normal(scale = 0.1, size = 50)

for j in range(20):
    sine_20 = sine_20 + np.random.normal(scale = 0.1, size = 50)

for k in range(100):
    sine_100 = sine_100 + np.random.normal(scale = 0.1, size = 50)

plt.scatter(space, sine, color = 'b', label = 'sine curve')
plt.plot(space, sine_5, color = 'r', label = 'noise_5_iters')
plt.plot(space, sine_20, color = 'g', label = 'noise_20_iters')
plt.plot(space, sine_100, color = 'y', label = 'noise_100_iters')
plt.legend(loc = 'upper right')
plt.show()
```

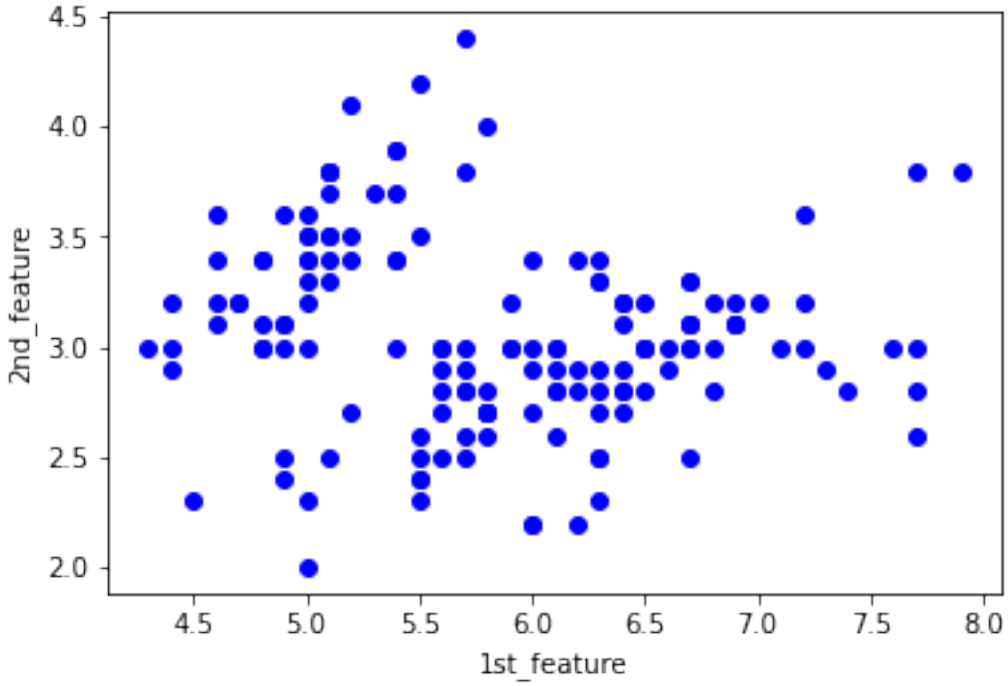


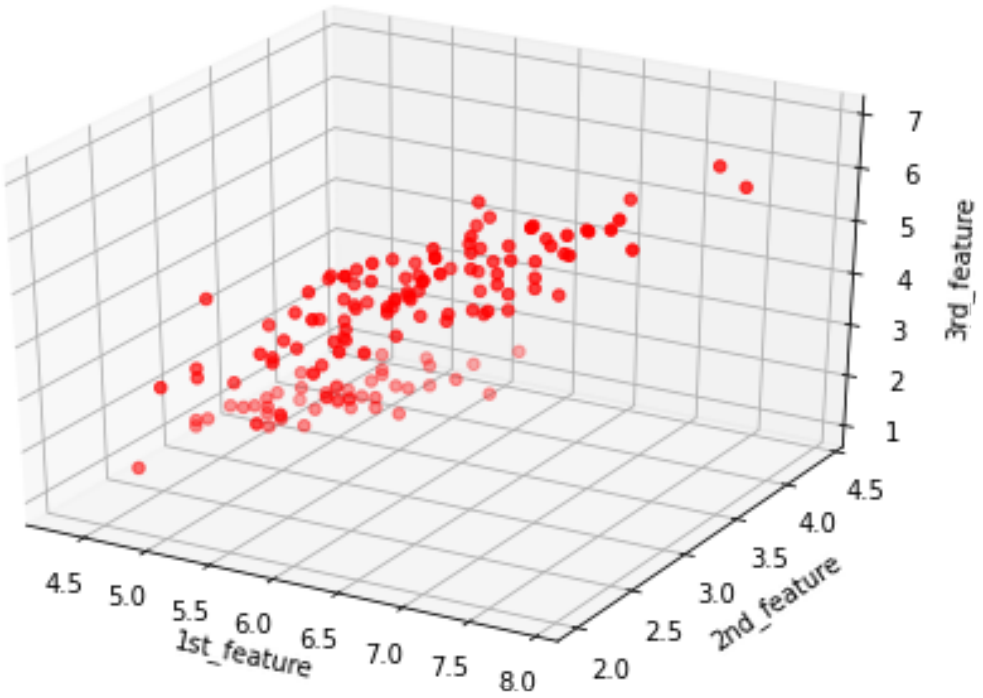
1.4 6 (10 points) Data Visualization

[1]:
```
import matplotlib.pyplot as plt
from sklearn import datasets
from mpl_toolkits.mplot3d import Axes3D
```

Load Iris dataset into Python:

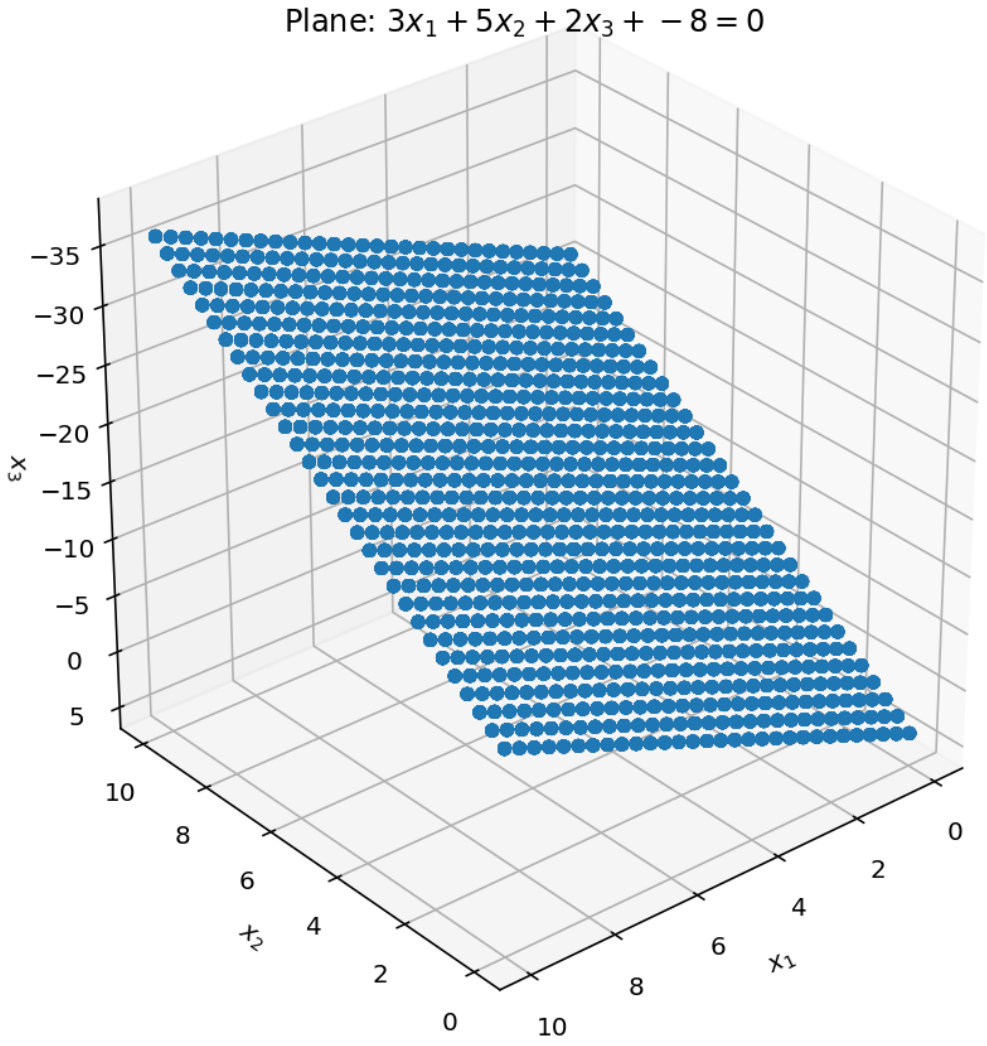
[13]:
```
iris = datasets.load_iris()
X = iris.data
Y = iris.target
x_ax = X[:,0]
y_ax = X[:,1]
z_ax = X[:,2]
plt.scatter(x_ax, y_ax, color = 'b', label = '2_feature')
plt.xlabel('1st_feature')
plt.ylabel('2nd_feature')
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(x_ax, y_ax, z_ax, color = 'r', label = '3_feature')
ax.set_xlabel('1st_feature')
ax.set_ylabel('2nd_feature')
ax.set_zlabel('3rd_feature')
plt.show()
```


4.5
4.0
3.5
3.0
2.5
2.0
2nd_feature
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
1st_feature


7
6
5
4
3
2
1
3rd_feature
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
1st_feature
4.5
4.0
3.5
3.0
2.5
2.0
2nd_feature

1.5 7 (10 points) Ploting Decision Boundaries

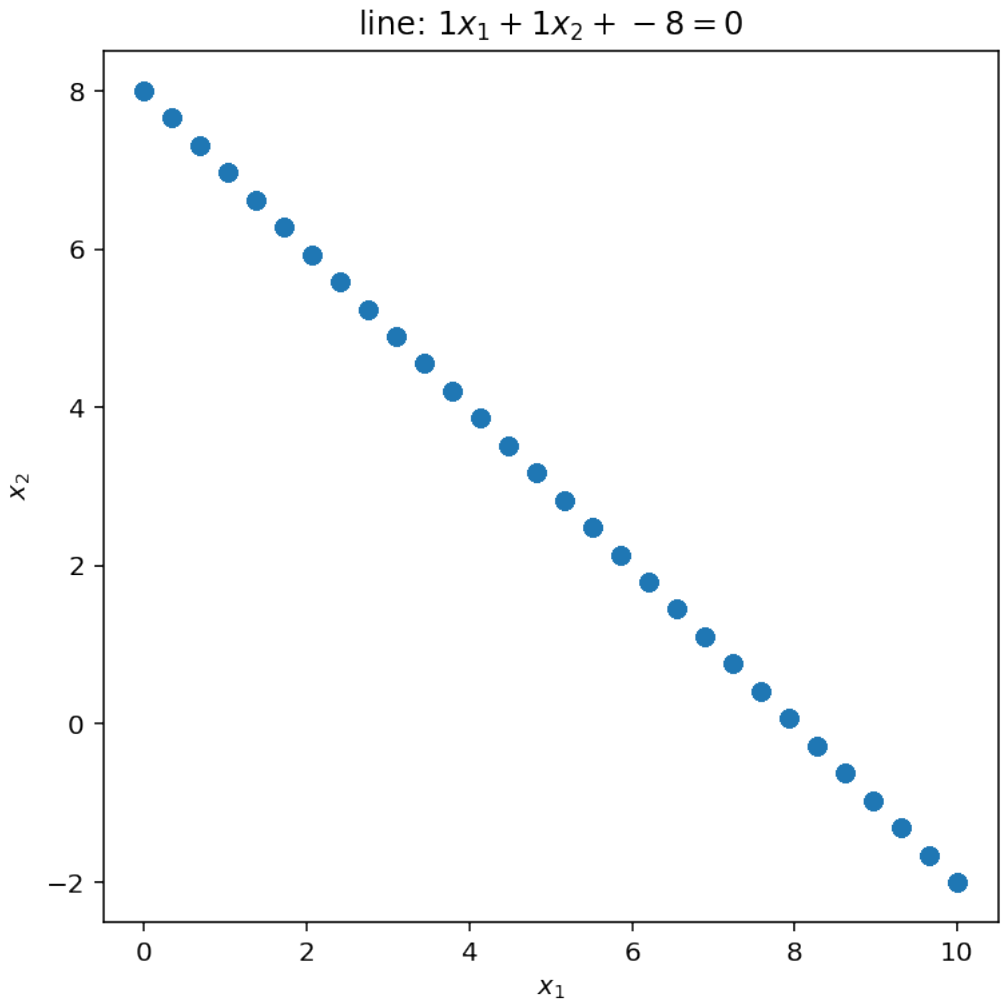
[14]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%config InlineBackend.figure_format ='retina'

def plot_plane(w1, w2, w3, b, num = 30):
  '''Helper function for 7.2; modify this example using plt.plot()instead of␣
 ↪Axes3D() to handle 7.1 & 7.3'''

  X_range = np.linspace(0,10,num)

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

  # 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()

plot_plane(3,5,2,-8)
```



To plot lines and curves in 2-D space, use function `plt.plot()` in place of `Axes3d()`.

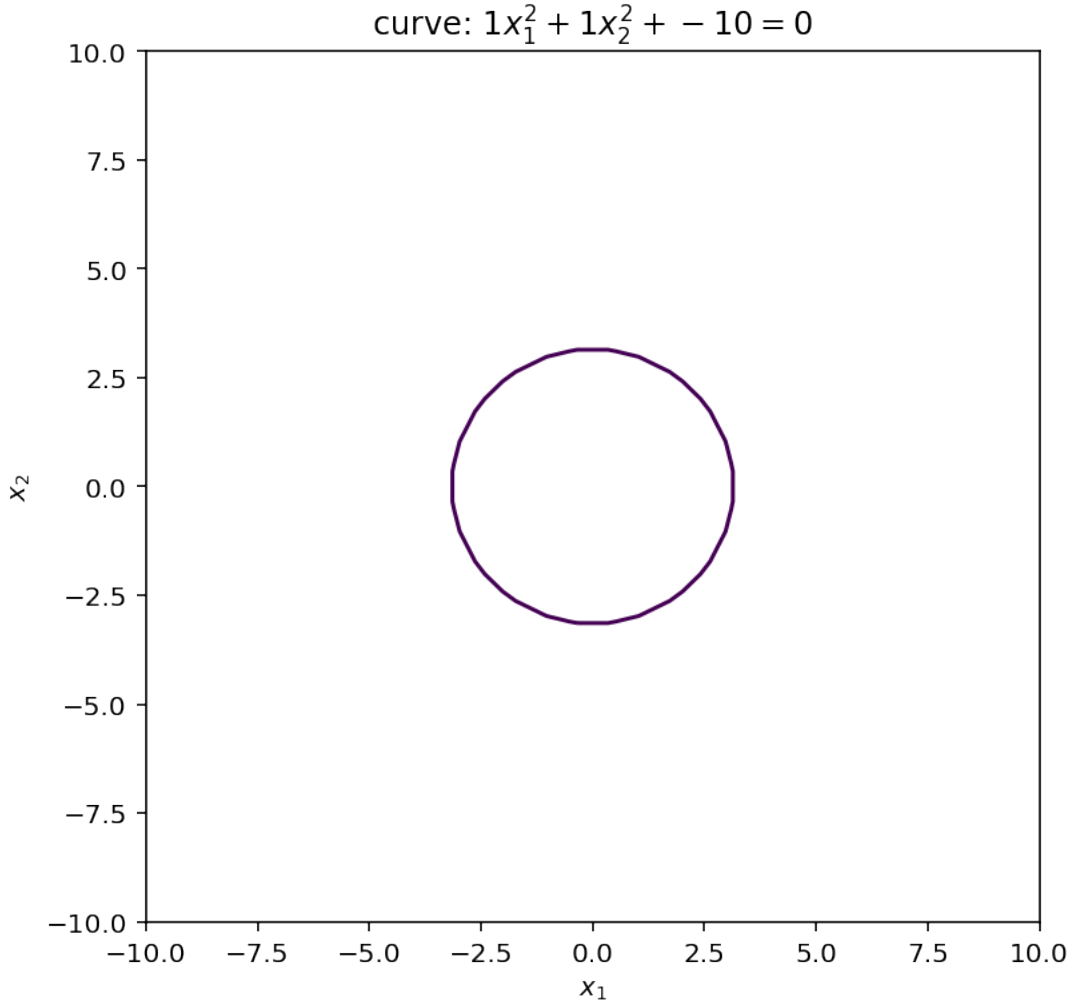
[17]:
```
def plot_plane(w1, w2, b, num = 30):
  '''Helper function for 7.2; modify this example using plt.plot()instead of␣
 ↪Axes3D() to handle 7.1 & 7.3'''

  X_range = np.linspace(0,10,num)

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

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

```
  # does plotting
  fig = plt.figure(figsize = (6, 6))
  plt.scatter(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()

plot_plane(1,1,-8)
```



[29]:
```
def plot_plane(w1, w2, b, num = 30):
  '''Helper function for 7.2; modify this example using plt.plot()instead of␣
↪Axes3D() to handle 7.1 & 7.3'''
```

```
  X_range = np.linspace(-10,10,num)

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

  # this defines the equation to plot
  f = w1*X1_plane**2 + w2*X2_plane**2 + b

  # does plotting
  fig = plt.figure(figsize = (6, 6))
  plt.contour(X1_plane, X2_plane, f, [0])
  plt.xlabel('$x_1$')
  plt.ylabel('$x_2$')
  plt.title("curve: ${}x_1^2+{}x_2^2+{}=0$".format(w1,w2,b))
  plt.show()

plot_plane(1,1,-10)
```



1.6 8 (25 points) Data Manipulation

Get a row or a column of the array X:

[ ]:
```
print(X[0]) # Print the first row of array X.
print(X[:, 0]) # Print the first column of array X.
# ':' here means all rows and '0' means column 0.
```

Get part of the array:

[ ]:
```
print(X[3:5, 1:3]) # Print 4th and 5th rows, 2nd and 3rd columns.
print(X[:3, :2]) # Print first 3 rows, first 2 columns.
```

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).

[30]:
```
print(X[: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. Hint: np.mean(), np.var().

[36]:
```
sec_col_X = X[:,1]
print(np.mean(sec_col_X))
print(np.var(sec_col_X))
```

```
3.0573333333333337
0.1887128888888889
```

3. Perform a linear projection on the 1st 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 1 st 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.

[42]:
```
three_col_X = X[:, :3]
w = np.array([[3],[2],[1]])
a = np.dot(three_col_X, w)
print(np.mean(a))
```

```
27.402666666666665
```

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

[46]:
```
ind_1 = np.random.randint(1,150)
ind_2 = np.random.randint(1,150)
ind_3 = np.random.randint(1,150)
print(X[ind_1-1])
print(X[ind_2-1])
print(X[ind_3-1])
```

```
[5.9 3.2 4.8 1.8]
[6.8 3.  5.5 2.1]
[5.8 2.7 5.1 1.9]
```

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

[49]:
```
new = np.ones((150,1))
new_X = np.hstack((X,new))
print(new_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

[52]:
```
new = X[0]
ne_X = np.vstack((X,new))
print(ne_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]
```