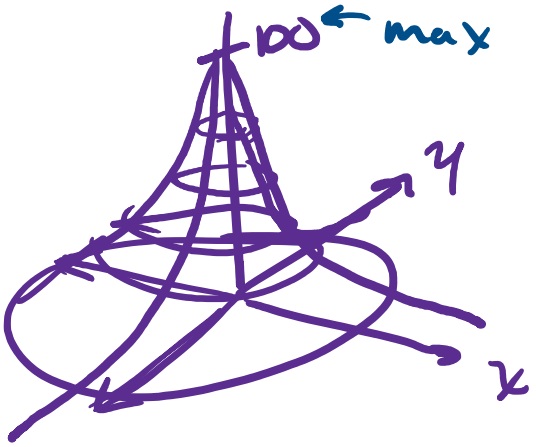
P1

Sunday, April 21, 2024 12:33 AM

A) $T(x,y) = \frac{100}{x^2+y^2+1}$

- Denominator can never be < 1, so T is never > 100.
- x^2+y^2 is a circle around the origin.
- As x and y increase radially, the denominator increases and will make $T(x,y)$ asymptote at 0.

Overall the shape should resemble



with circular level curves.

B)

$T(x,y)$ has its maximum at $T(0,0)$

C)

$$T(x,y) = \frac{100}{x^2+y^2+1}$$

$$\frac{\partial T}{\partial x} = \frac{\partial}{\partial x}\left[\frac{100}{x^2+y^2+1}\right] \qquad a = y^2+1$$

$$100 \cdot \frac{\partial T}{\partial x}\left[\frac{1}{x^2+a^2}\right]$$

$$\frac{\partial T}{\partial x}\left[(x^2+a^2)^{-1}\right]$$

$$-1 \cdot (x^2 + a^2)^{-2} \cdot \frac{\partial}{\partial x}\left[x^2 + a^2\right)$$

$$\left(-1 \cdot (x^2 + a^2)^{-2} \cdot (2x)\right)$$

$$\frac{\partial T}{\partial x} = 100 \cdot -1 \cdot (x^2 + a^2)^{-2} \cdot 2x$$

$$= \frac{-200x}{(x^2 + y^2 + 1)^2}$$

B/C symmetry,

$$\frac{\partial T}{\partial y} = \frac{-200y}{(x^2 + y^2 + 1)^2}$$

$$\nabla T = -200 \begin{bmatrix} \frac{x}{(x^2+y^2+1)^2} \\ \frac{y}{(x^2+y^2+1)^2} \end{bmatrix}$$

$$\nabla T(3,2) = -200 \begin{bmatrix} 3/(x^2+y^2+1)^2 \end{bmatrix}$$

$$\nabla T(3,2) = -200 \begin{bmatrix} 3/(3^2+2^2+1)^2 \\ 2/(3^2+2^2+1)^2 \end{bmatrix}$$

$$= -200 \begin{bmatrix} 3/196 \\ 2/196 \end{bmatrix}$$

$$\nabla T(x,y) = \frac{-200}{196} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\nabla T(x,y) \sim -\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\max\left(\begin{bmatrix} -3 \\ -2 \end{bmatrix} \cdot \vec{v}\right)$$

$$|\vec{v}| = \sqrt{3^2+2^2}$$

$$= \sqrt{4+4}$$

$$= \sqrt{13}$$

$$\vec{v} = \frac{1}{\sqrt{13}}\begin{bmatrix} -3 \\ -2 \end{bmatrix} = \frac{-1}{\sqrt{13}}\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\frac{-1}{\sqrt{13}}\begin{bmatrix} 3 \\ 2 \end{bmatrix} \cdot \frac{-200}{196}\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\frac{200}{196\sqrt{13}} \cdot (3^2+2^2)$$

$\frac{14 \cdot 200}{196 \cdot \sqrt{13}} = \frac{100}{91} \cdot \sqrt{13} \approx 3.96$

Direction of maximum increase will always be $\nabla T(a,b)$,

so the vector, $\vec{v}$, that maximizes temperature increase is

$\vec{v} = \frac{1}{\sqrt{13}} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$. The magnitude of increase is $\hat{v} \cdot \nabla T(3,2)$

which equals $\frac{100}{91} \cdot \sqrt{13}$ or ≈ 3.962.

D) Conceptually, the opposite direction must be the greatest decrease, so $-\vec{v}$.

The direction $\frac{1}{\sqrt{13}} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ is opposite to the gradient, so it will be the greatest decrease.

This decrease is $\frac{-100}{91} \cdot \sqrt{13}$ or ≈ -3.96

E)

$\begin{bmatrix} -3 \\ -2 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = 0$

$-3a - 2b = 0 \qquad a^2 + b^2 = 1$

$-3a = 2b$

$-\frac{3}{2} a = b$

$a^2 + \left(-\frac{3}{2} a\right)^2 = 1$

$a^2 + \frac{9}{4} a^2 = 1$

$\frac{13}{4}a^2 = 1$

$a^2 = \frac{4}{13}$

$a = \pm \frac{2}{\sqrt{13}}$

$b = \frac{-3}{2}a$

$b = \frac{-3}{2}\left(\frac{2}{\sqrt{13}}\right)$ $\qquad$ $b = \frac{-3}{2}\left(\frac{-2}{\sqrt{13}}\right]$

$b = \frac{-3}{\sqrt{13}}$ $\qquad$ $b = \frac{3}{\sqrt{13}}$

so,

$\vec{v} = \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2/\sqrt{13} \\ -3/\sqrt{13} \end{bmatrix} = \frac{1}{\sqrt{13}}\begin{bmatrix} 2 \\ -3 \end{bmatrix}$

The direction, $\vec{v}$, that satisfies $\vec{v} \cdot \nabla T = 0$ is perpendicular to gradient and will be unaffected by the gradient.

$\vec{v} = \frac{1}{\sqrt{13}}\begin{bmatrix} 2 \\ -3 \end{bmatrix}$ or $\frac{1}{\sqrt{13}}\begin{bmatrix} -2 \\ 3 \end{bmatrix}$

P2

Monday, April 22, 2024 1:32 AM

A)

$$f(x,y) = 10x^2y - 5x^2 - 4y^2 - x^4 - 2y^4$$

$$\frac{\partial f}{\partial x} = 20xy - 10x - 4x^3$$

$$\frac{\partial f}{\partial y} = 10x^2 - 8y - 8y^3$$

$$\nabla f = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial x \end{bmatrix}$$

Critical points when $\nabla f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix} = \begin{bmatrix} 20xy - 10x - 4x^3 \\ 10x^2 - 8y - 8y^3 \end{bmatrix}$$

(0,0) is guaranteed critical point

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -2x(2x^2 - 10y + 5) \\ 10x^2 - 8(y^3 + y) \end{bmatrix}$$

$\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$

$0 = -2x(2x^2 - 10y + 5)$

$0 = 2x^2 - 10y - 5$

$2x^2 = 10y - 5$

$x^2 = \frac{10y-5}{2}$

$0 = 10x^2 - 8(y^3 + y)$

$10x^2 = 8(y^3 + y)$

$x^2 = \frac{4}{5}(y^3 + y)$

$\frac{10y-5}{2} = \frac{4}{5}(y^3 + y)$

$5(10y - 5) = 8(y^3 + y)$

$50y - 25 = 8y^3 + 8y$

$0 = 8y^3 - 42y + 25$ will be used to solve for y-coordinates

$x^2 = \frac{10y-5}{2}$

$x = \pm\sqrt{\frac{10y-5}{2}}$ will be used to solve for x-coordinates

$$\nabla^2 f = \begin{bmatrix} \partial^2 f/\partial x^2 & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial x \partial y} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$\nabla^2 f = \begin{bmatrix} 20y - 10 - 12x^2 & 20x \\ 20x & -24y^2 - 8 \end{bmatrix}$$

$$D(x,y) = (20y - 10 - 12x^2)(-24y^2 - 8) - (20x)^2$$

Used to classify points with second derivative test.

Rest of PZ in Jupyter Notebook

From numpy:

critical points are (x,y)	$f(x,y)$
$(-2.644, 1.898)$	8.5
$(-0.856, 0.647)$	-1.48
$(0,0)$	0
$(0.856, 0.647)$	-1.48
$(2.644, 1.898)$	8.5

Part 2: Finding and Classifying Critical Points and their Level Curves (continued from handwritten)

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Getting the coefficients from:

$$0 = 8y^3 - 42y + 25$$

In [ ]:
```
coeff = [8,0, -42, 25]
roots = np.roots(coeff)
roots
```

Out[ ]:
```
array([-2.54515663,  1.89838443,  0.6467722 ])
```

Removing the negative because it will create a NaN and is not a critical point because it cannot be solved.

In [ ]:
```
roots = roots[1:]
roots
```

Out[ ]:
```
array([1.89838443, 0.6467722 ])
```

Solving for the x-coordinates using:

$$x = \pm\sqrt{\frac{10y-5}{2}}$$

In [ ]:
```
pos_f = np.sqrt((10*roots -5)/2)

neg_f = -1* pos_f
```

In [ ]:
```
pos_f
```

Out[ ]:
```
array([2.6442243 , 0.85665687])
```

In [ ]:
```
neg_f
```

Out[ ]:
```
array([-2.6442243 , -0.85665687])
```

Organizing the solved x and y coordinates

In [ ]:
```
y = np.concatenate((roots, roots, np.array([0])))
y
```

Out[ ]:
```
array([1.89838443, 0.6467722 , 1.89838443, 0.6467722 , 0.        ])
```

In [ ]:
```
x = np.concatenate((pos_f, neg_f, np.array([0])))
x
```

Out[ ]:
```
array([ 2.6442243 ,  0.85665687, -2.6442243 , -0.85665687,  0.        ])
```

Inputting the critical points into:

$f(x, y) = 10x^2y - 5x^2 - 4y^2 - x^4 - 2y^4$

In [ ]:
```
func = 10*x**2*y - 5*x**2 - 4*y**2 - x**4 - 2*y**4
func
```

Out[ ]:
```
array([ 8.49585813, -1.48467882,  8.49585813, -1.48467882,  0.        ])
```

Solving for the Second Derivative test by taking

$det(\nabla^2 f)$

In [ ]:
```
d_x_y = (20*y - 10 - 12 * x**2)*(-24*y**2 - 8)  - (20*x)**2
d_x_y
```

Out[ ]:
```
array([2488.7172337 , -187.63626429, 2488.7172337 , -187.63626429,
         80.        ])
```

Solving for f_{xx}

In [ ]:
```
f_xx = (20*y - 10 - 12 * x**2)
f_xx
```

Out[ ]:
```
array([-55.93537725,  -5.87088796, -55.93537725,  -5.87088796,
       -10.        ])
```

In [ ]:
```
crit = pd.DataFrame()
crit["x"] = x
crit["y"] = y
crit["f(x,y)"] = func
crit["D(x,y)"] = d_x_y
crit["f_xx"] = f_xx

crit
```

Out[ ]:

	x	y	f(x,y)	D(x,y)	f_xx
0	2.644224	1.898384	8.495858	2488.717234	-55.935377
1	0.856657	0.646772	-1.484679	-187.636264	-5.870888
2	-2.644224	1.898384	8.495858	2488.717234	-55.935377
3	-0.856657	0.646772	-1.484679	-187.636264	-5.870888
4	0.000000	0.000000	0.000000	80.000000	-10.000000

Classifying the critical points found using the Second Derivative Test

In [ ]:
```
def classify(row: pd.DataFrame):
    if row["D(x,y)"] < 0:
        return "saddle"
    elif row["D(x,y)"] == 0:
        return "unknown"
    else:
        if row["f_xx"] > 0:
            return "minimum"
        else:
            return "maximum"
```

```
crit["classification"] = crit.apply(classify, axis = 1)
crit
```

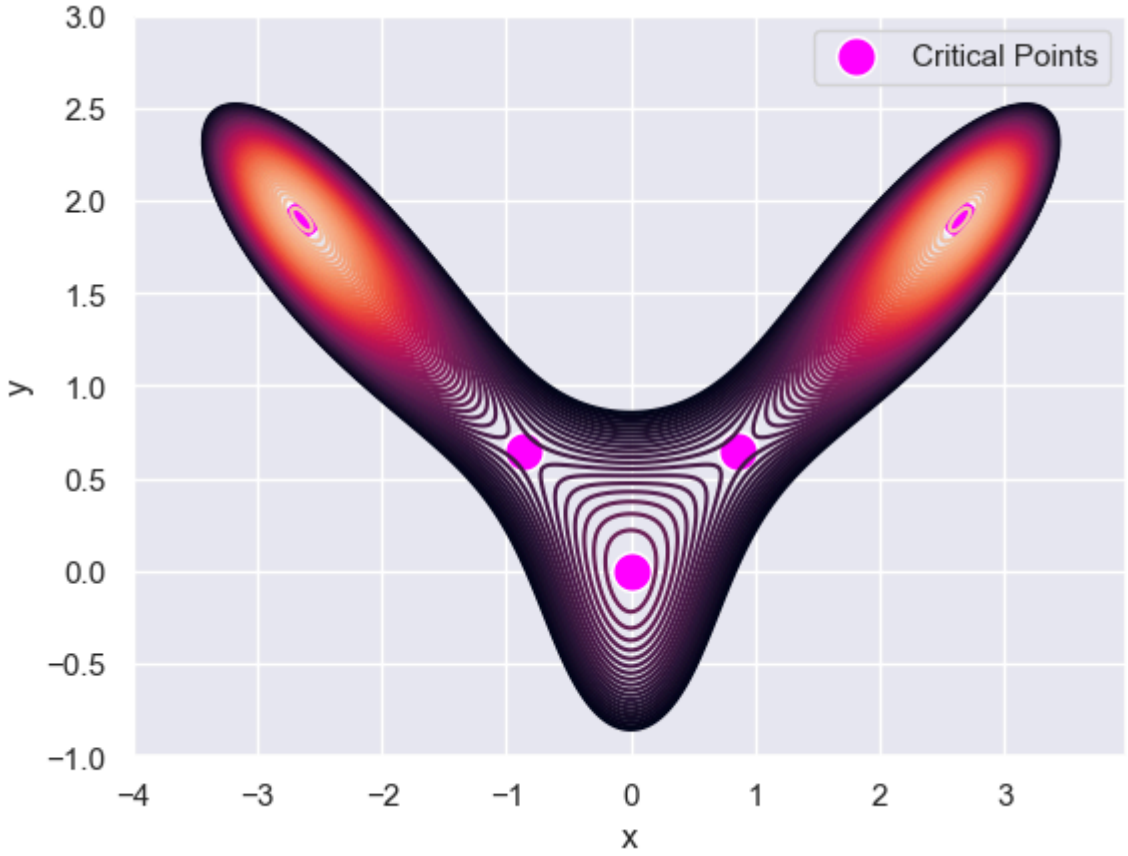
Out[ ]:

	x	y	f(x,y)	D(x,y)	f_xx	classification
0	2.644224	1.898384	8.495858	2488.717234	-55.935377	maximum
1	0.856657	0.646772	-1.484679	-187.636264	-5.870888	saddle
2	-2.644224	1.898384	8.495858	2488.717234	-55.935377	maximum
3	-0.856657	0.646772	-1.484679	-187.636264	-5.870888	saddle
4	0.000000	0.000000	0.000000	80.000000	-10.000000	maximum

B) Plotting the contour and level curves of the function

In [ ]:
```
delta = 0.025
x = np.arange(-4.0, 4.0, delta)
y = np.arange(-4.0, 4.0, delta)
X, Y = np.meshgrid(x, y)
Z = 10*X**2*Y - 5*X**2 - 4*Y**2 - X**4 - 2*Y**4

levels = np.arange(-4,10, 0.2)
```

In [ ]:
```
sns.set_theme()
plt.contour(X,Y,Z, levels = levels)
plt.ylim(-1, 3)
sns.scatterplot(data = crit, x = "x", y="y", s= 200, color = "magenta", label = "Critical Points")
plt.show()
```


Critical Points
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
y
−4
−3
−2
−1
0
1
2
3
x

Part 3: Demonstration of Laplace Expansion

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
A = [[2, 1, 2, 6],
     [6, 4, 3, 5],
     [8, 1, 4, 9],
     [9, 8, 7, 8]]

A = np.array(A)
A
```

Out[ ]:
```
array([[2, 1, 2, 6],
       [6, 4, 3, 5],
       [8, 1, 4, 9],
       [9, 8, 7, 8]])
```

In [ ]:
```
def minor(A, row, col):
    new_A = A
    new_A = np.delete(A, row, axis=0)
    new_A = np.delete(new_A, col, axis =1)
    return new_A
```

In [ ]:
```
minor(A, 1,1)
```

Out[ ]:
```
array([[2, 2, 6],
       [8, 4, 9],
       [9, 7, 8]])
```

In [ ]:
```
def laplace(A, row):
    sums = 0

    coeff = A[row]

    for i in range(len(coeff)):
        sums+= (-1)**(row+i) * coeff[i] * np.linalg.det(minor(A, row, i))
```

```
    return sums
```

In [ ]:
```
laplace(A, 0)
```

Out[ ]: 229.00000000000023

In [ ]:
```
np.linalg.det(A)
```

Out[ ]: 229.00000000000028

P4

Monday, April 22, 2024 4:09 AM

$$A\vec{x} = \lambda\vec{x}$$

$$A\vec{x} - \lambda\vec{x} = 0$$

$$(A-\lambda)\vec{x} = 0$$

$$\begin{matrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & 1 & 2 \end{matrix}$$

$$\begin{matrix} 2-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & 1 & 2-\lambda \end{matrix}$$

$$(2-\lambda)\begin{vmatrix} 2-\lambda & -1 \\ 1 & 2-\lambda \end{vmatrix} - (-1)\begin{vmatrix} -1 & -1 \\ 0 & 2-\lambda \end{vmatrix} + 0$$

$$(2-\lambda)\left[(2-\lambda)^2-(-1)\right]+\left[-(2-\lambda)\right]$$

$$(2-\lambda)\left[(2-\lambda)^2+1\right]+(2-\lambda)$$

$$(2-\lambda)\left[(2-\lambda)^2+1+2-\lambda\right]$$

$$\left[4-4\lambda+\lambda^2+2-\lambda\right]$$

$$(2-\lambda)\left[6-5\lambda+\lambda^2\right]$$

$$(2-\lambda)(-3+\lambda)(-2+\lambda)$$

$$\lambda=\{3,2\}$$

Positive definite b/c all eigenvalues are positive

Condition number: $\frac{3}{2}$

Part 5: Unique Solutions and Least Squares for an Overdetermined Matrix

In [ ]:
```
import pandas as pd
import numpy as np
```

A)

Displaying the singular values for matrix A

In [ ]:
```
A = np.array([[6, 5, 3],
              [4, 1, 5],
              [6, 3, 6],
              [5, 3, 6],
              [6, 6, 3]])

u, s, v = np.linalg.svd(A)

s
```

Out[ ]:
```
array([17.92667464,  4.69961929,  0.74021274])
```

B)

Finding the eigenvalues of A.

In [ ]:
```
# a_square = np.matmul(A.T, A)
a_square = A.T @ A
a_square
```

Out[ ]:
```
array([[149, 103, 122],
       [103,  80,  74],
       [122,  74, 115]])
```

In [ ]:
```
eigenvalues, eigenvectors = np.linalg.eig(a_square)
eigenvalues
```

Out[ ]:
```
array([321.36566366,   0.54791491,  22.08642144])
```

The singular value σ_i is related to the eigenvalue λ_i by:

$$\sigma_i = \sqrt{\lambda_i}$$

C)

Because there are only 3 singular values in A), $rank(A) = 3$

In [ ]:
```
np.linalg.matrix_rank(A)
```

Out[ ]:
```
3
```

D)

Yes, even though the column space and row space do not have the same dimensionality, m > n and each column vector is linearly independent. This means that our input space is $\mathbb{R}^3$ and maps onto $\mathbb{R}^5$ - however they do not span all of $\mathbb{R}^5$. The three column vectors are linearly independent meaning $Span(v_1, v_2, v_3) = 3$, where v_i is a column vector in A. Because the $Span(v_1, v_2, v_3) = dim(Col(A))$, if there is a solution in the solution space, it must be unique.

E)

Finding solution or approximation for

$$b = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 3 \\ 1 \end{bmatrix}$$

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

Out[ ]:
```
array([-1,  2,  0,  3,  1])
```

In [ ]:
```
A, b
```

Out[ ]:
```
(array([[6, 5, 3],
        [4, 1, 5],
        [6, 3, 6],
        [5, 3, 6],
        [6, 6, 3]]),
 array([-1,  2,  0,  3,  1]))
```

Demonstration that $Ax = b$ does not have a solution

In [ ]:
```
np.linalg.solve(A, b)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[8], line 1
----> 1 np.linalg.solve(A, b)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312
\site-packages\numpy\linalg\linalg.py:396, in solve(a, b)
    394 a, _ = _makearray(a)
    395 _assert_stacked_2d(a)
--> 396 _assert_stacked_square(a)
    397 b, wrap = _makearray(b)
    398 t, result_t = _commonType(a, b)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312
\site-packages\numpy\linalg\linalg.py:213, in _assert_stacked_square(*arrays)
    211 m, n = a.shape[-2:]
    212 if m != n:
--> 213     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

Finding the value of x that minimizes $||Ax - b||$.

In [ ]:
```
x, _, _, s = np.linalg.lstsq(A, b, rcond=None)
x
```

Out[ ]:
```
array([-2.06119825,  1.37336076,  1.54641296])
```

P6

Tuesday, April 23, 2024 11:07 PM

P6. (5 points) Suppose A and B are $n \times n$ matrices and $x \in \mathbb{R}^n$ are the variables. What is the gradient of the dot product of Ax with Bx? What about the Hessian?

$$f(\vec{x}) = (A\vec{x}) \cdot (B\vec{x})$$

$$\nabla f(\vec{x}) = \nabla\left[(A\vec{x}) \cdot (B\vec{x})\right]$$

$$= \nabla(A\vec{x}) \cdot B\vec{x} + A\vec{x} \cdot \nabla(B\vec{x})$$

$$= A^T B\vec{x} + A\vec{x} \cdot B^T$$

$$\nabla f(\vec{x}) = A^T B\vec{x} + A\vec{x}B^T$$

$$H_f = \nabla(\nabla f)$$

$$= \nabla\left(A^T B\vec{x} + A\vec{x} \cdot B^T\right)$$

$$= \nabla(A^T B\vec{x}) + \nabla(A\vec{x} \cdot B^T)$$

$$H_f = AB^T + A^T B$$

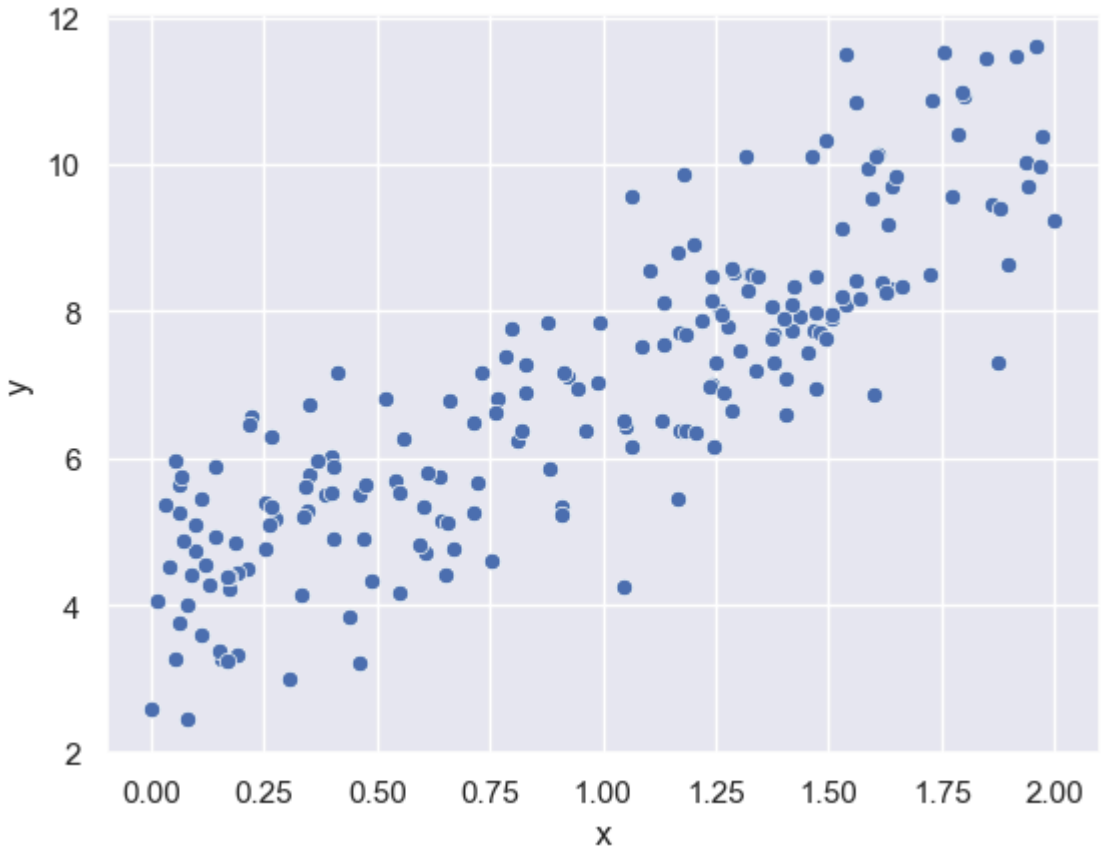
$$H_f = AB^T + A^T B$$

Part 7: Implementing the Mini-Batch Gradient Descent

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
sns.set_theme()
```

In [ ]:
```
data = pd.DataFrame()

data["x"] = pd.read_csv("X.csv")
data["y"] = pd.read_csv("y.csv")

sns.scatterplot(data = data, x= "x" , y = "y")
```

Out[ ]:
```
<Axes: xlabel='x', ylabel='y'>
```



In [ ]:
```
x = data["x"].to_numpy()
y = data["y"].to_numpy()
```

run_mini_batch returns $\theta_k + 1$ given θ_k

In [ ]:
```
def run_mini_batch(theta, learning_rate, batch_size, x, y):
    indices = np.random.choice(len(x), size=batch_size, replace=False)
    x_subset = x[indices]
    y_subset = y[indices]

    x_b = np.c_[np.ones(batch_size), x_subset]

    gradients = 2/batch_size * np.dot(x_b.T, np.dot(x_b, theta) - y_subset)
    theta_n = theta - learning_rate*gradients
```

```
    return theta_n
```

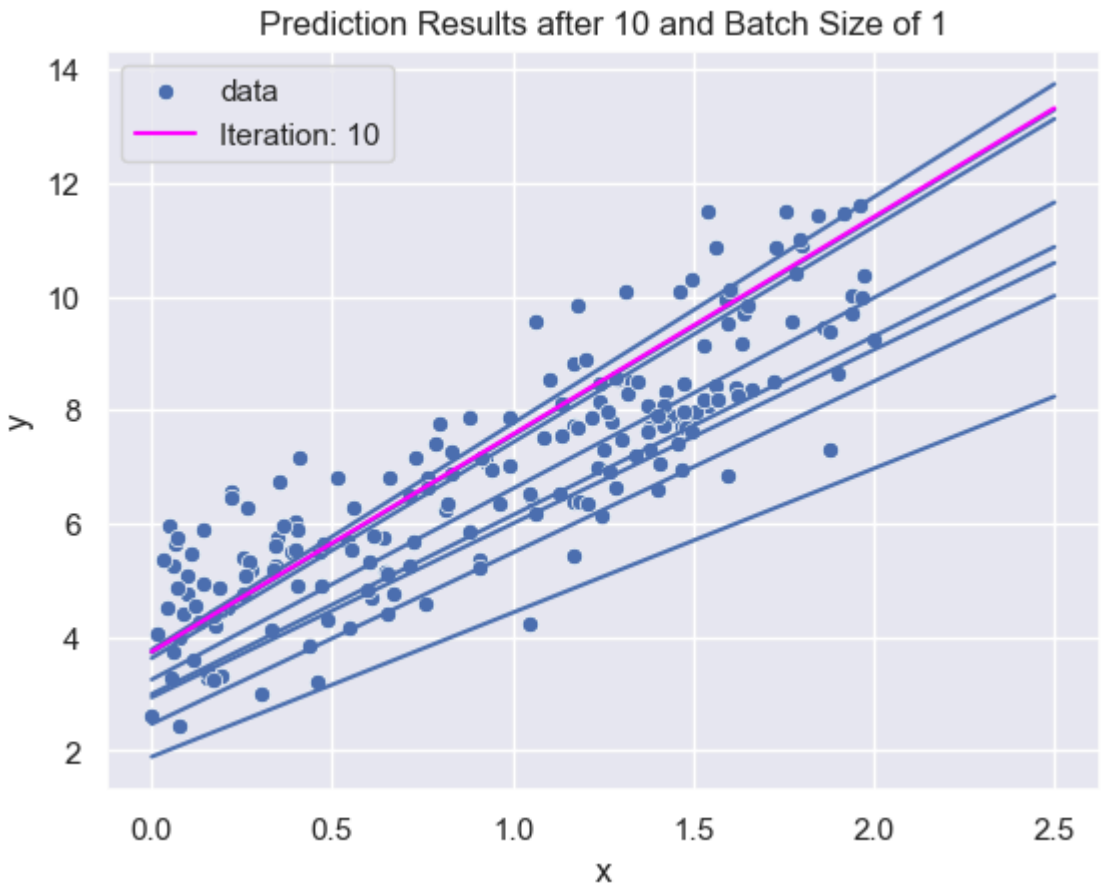
In [ ]:
```
def get_mean_squared_error(theta, x, y):
    y_pred = theta[0] + theta[1]*x
    mse = np.sum((y-y_pred)**2) / len(x)

    return mse
```

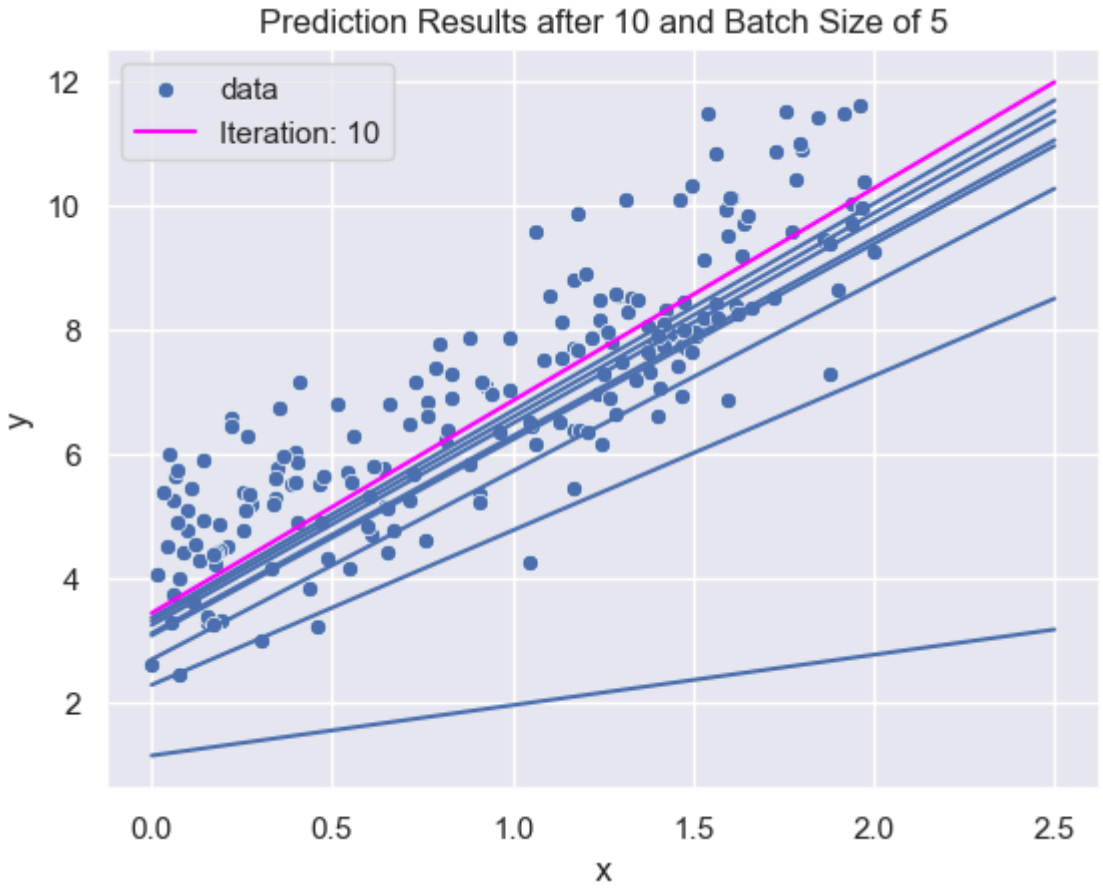
In [ ]:
```
def plot_batch(theta, current_iter, n_iterations):
    x_pred = np.arange(0,3, 0.5)
    y_pred = theta[0] + theta[1]*x_pred
    y_pred

    if current_iter != n_iterations - 1:
        sns.lineplot(x = x_pred, y = y_pred, color = "#4c72b0")
    else:
        sns.lineplot(x = x_pred, y = y_pred, label = f"Iteration: {current_iter+1}", color = "magenta")
```

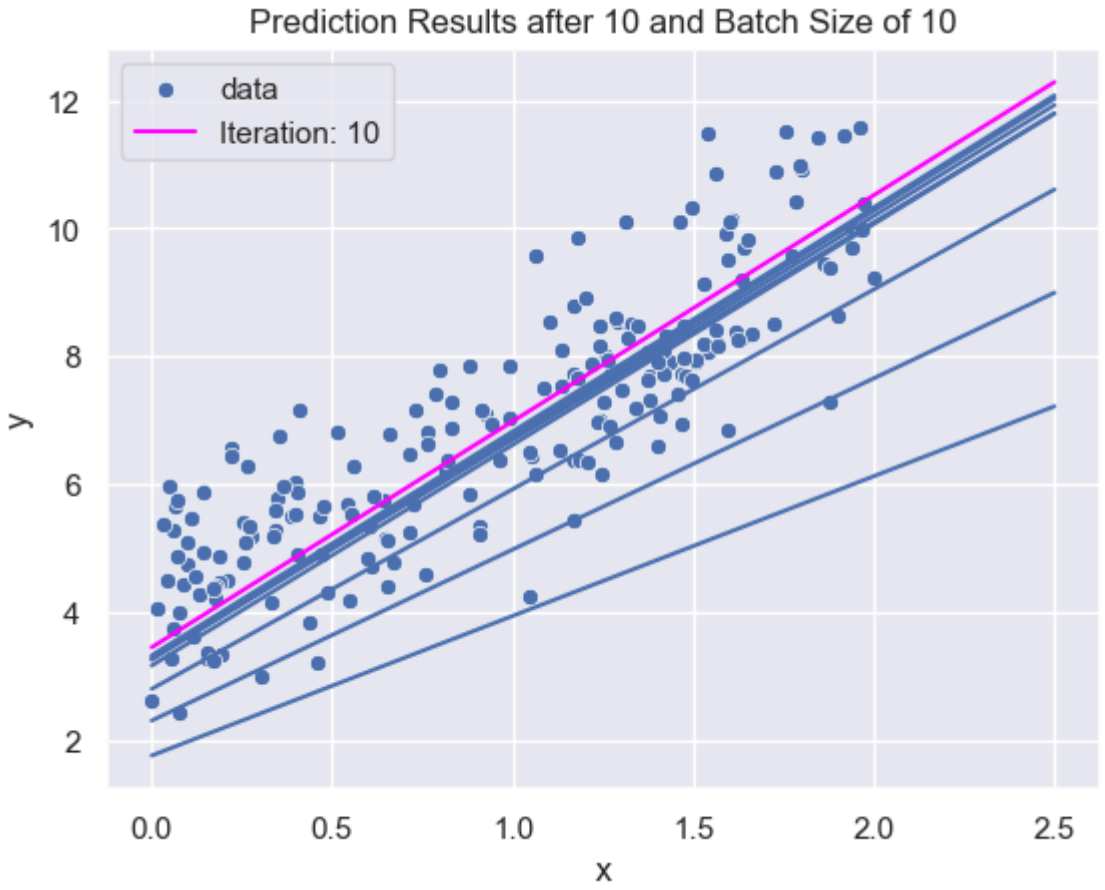
In [ ]:
```
learning_rate = 0.1
batch_sizes = [1, 5, 10]
n_iterations = 10

theta = np.empty((n_iterations, 2, len(batch_sizes)))
mse = np.empty((n_iterations, len(batch_sizes)))
```

In [ ]:
```
for b in range(len(batch_sizes)):
    sns.scatterplot(data=data, x="x", y="y", label="data")

    theta[0, :, b] = np.random.randn(2)
    mse[0, b] = get_mean_squared_error(theta[0, :, b], x, y)


    for i in range(1, n_iterations):


        theta[i, :, b] = run_mini_batch(theta[i-1, :, b], learning_rate, batch_sizes[b], x, y)


        plot_batch(theta[i, :, b], i, n_iterations)
```

```
        mse[i, b] = get_mean_squared_error(theta[i, :, b], x, y)

    plt.title(f"Prediction Results after {n_iterations} and Batch Size of {batch_sizes[b]}")
    plt.show()
```

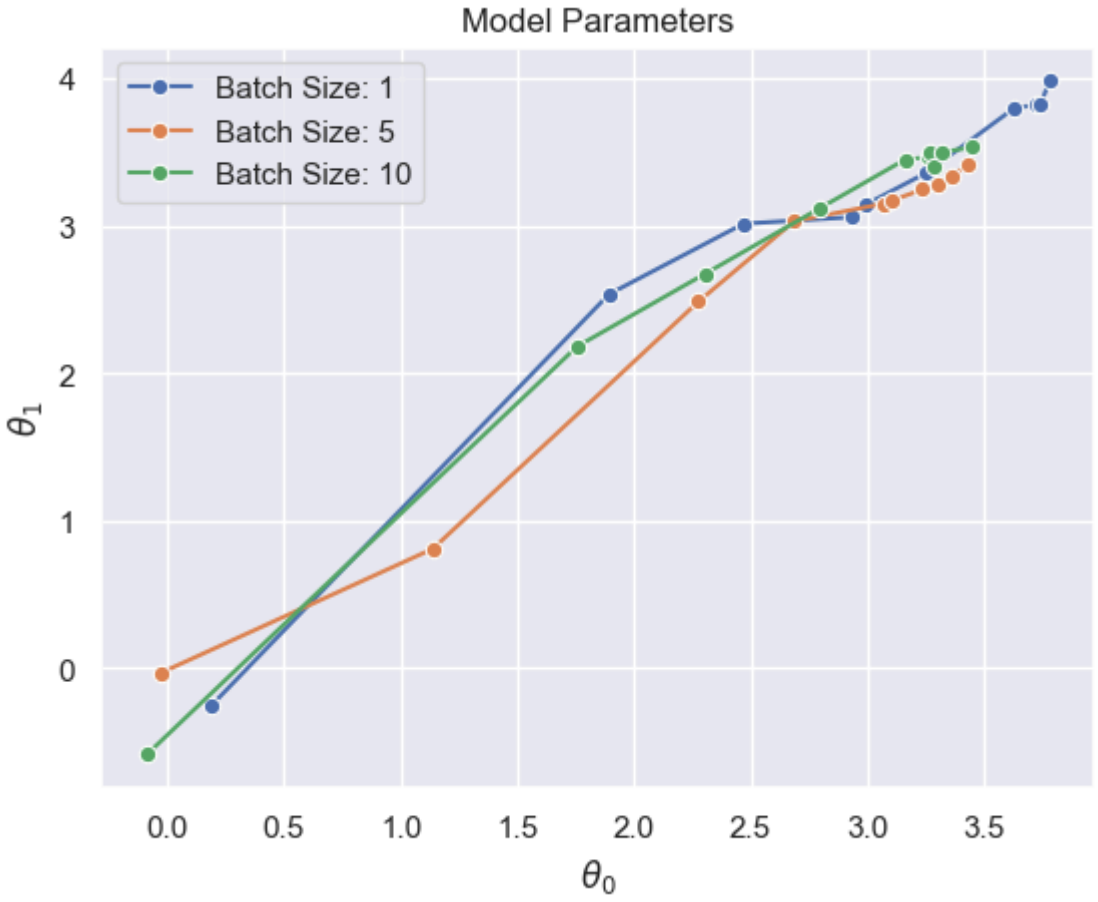



Prediction Results after 10 and Batch Size of 5
data
Iteration: 10
12
10
8
6
4
2
y
0.0
0.5
1.0
1.5
2.0
2.5
x



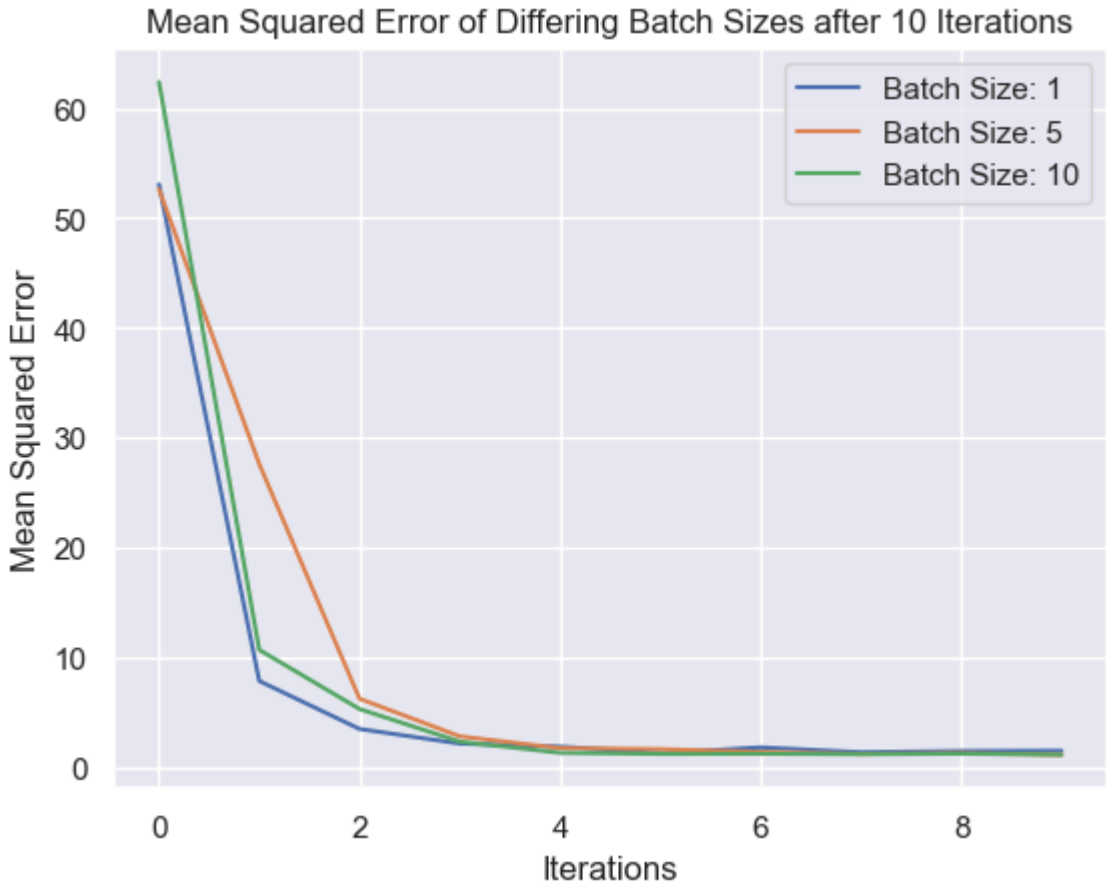
In [ ]:
```
for b in range(len(batch_sizes)):
    theta_0 = theta[:, 0, b]
    theta_1 = theta[:, 1, b]

    sns.lineplot(x = theta_0, y = theta_1, marker = "o", label = f"Batch Size: {batch_sizes[b]}")
plt.xlabel(r"$\theta_0$")
plt.ylabel(r"$\theta_1$")
plt.title("Model Parameters")
```

Out[ ]:
```
Text(0.5, 1.0, 'Model Parameters')
```



In [ ]:
```
for b in range(len(batch_sizes)):
    sns.lineplot(x = np.arange(n_iterations), y = mse[:,b], label = f"Batch Size: {batch_sizes[b]}")

plt.xlabel("Iterations")
plt.ylabel("Mean Squared Error")
plt.title(f"Mean Squared Error of Differing Batch Sizes after {n_iterations} Iterations")
```

Out[ ]:
```
Text(0.5, 1.0, 'Mean Squared Error of Differing Batch Sizes after 10 Iterations')
```


Mean Squared Error of Differing Batch Sizes after 10 Iterations
Batch Size: 1
Batch Size: 5
Batch Size: 10
Mean Squared Error
0
10
20
30
40
50
60
0
2
4
6
8
Iterations