Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Benjamin Liu"
COLLABORATORS = ""
```

Homework 3: Loss Minimization

Modeling, Estimation and Gradient Descent

Due Date: Tuesday 10/9, 11:59 PM

Course Policies

Here are some important course policies. These are also located at http://www.ds100.org/fa18/ (http://www.ds100.org/fa18/).

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your solution.

This Assignment

In this homework, we explore modeling data, estimating optimal parameters and a numerical estimation method, gradient descent. These concepts are some of the fundamentals of data science and machine learning and will serve as the building blocks for future projects, classes, and work.

After this homework, you should feel comfortable with the following:

- Practice reasoning about a model
- Build some intuition for loss functions and how they behave
- Work through deriving the gradient of a loss with respect to model parameters
- Work through a basic version of gradient descent.

This homework is comprised of completing code, deriving analytic solutions, writing LaTex and visualizing loss.

Submission - IMPORTANT, PLEASE READ

For this assignment and future assignments (homework and projects) you will also submit your free response and plotting questions to gradescope. To do this, you can download as PDF (`File->Download As->PDF via Latex (.pdf)`). You are responsible for submitting and tagging your answers in gradescope. For each free response and plotting question, please include:

1. Relevant code used to generate the plot or inform your insights
2. The written free response or plot

We are doing this to make it easier on our graders and for you, in the case you need to submit a regrade request. Gradescope (as of now) is still better for manual grading.

Score breakdown

Question	Points
Question 1a	1
Question 1b	1
Question 1c	1
Question 1d	1
Question 1e	1
Question 2a	2
Question 2b	1
Question 2c	1
Question 2d	1
Question 2e	1
Question 2f	1
Question 3a	1
Question 3b	3
Question 3c	2
Question 4a	3
Question 4b	1
Question 4c	1
Question 4d	1
Question 4e	1
Question 5a	2
Question 5b	4
Question 5c	0
Question 5d	0
Question 6a	3
Question 6b	3

Question	Points
Question 6c	3
Question 6d	3
Question 6e	3
Question 6f	3
Question 6g	3
Question 7a	1
Question 7b	1
Question 7c	1
Question 7d	1
Question 7e	0
Total	56

Getting Started

```python
In [2]:  # Imports
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import csv
         import re
         import seaborn as sns

         # Set some parameters
         plt.rcParams['figure.figsize'] = (12, 9)
         plt.rcParams['font.size'] = 16
         np.set_printoptions(4)
```

```python
In [3]:  # We will use plot_3d helper function to help us visualize gradient
         from hw3_utils import plot_3d
```

Load Data

Load the data.csv file into a pandas dataframe.
Note that we are reading the data directly from the URL address.

```
In [4]: # Run this cell to load our sample data
        data = pd.read_csv("https://github.com/DS-100/fa18/raw/gh-pages/assets/datasets/hw
        data.head()
```

Out[4]:

	x	y
0	-5.000000	-7.672309
1	-4.966555	-7.779735
2	-4.933110	-7.995938
3	-4.899666	-8.197059
4	-4.866221	-8.183883

1: A Simple Model

Let's start by examining our data and creating a simple model that can represent this data.

Question 1

Question 1a

First, let's visualize the data in a scatter plot. After implementing the `scatter` function below, you should see something like this:
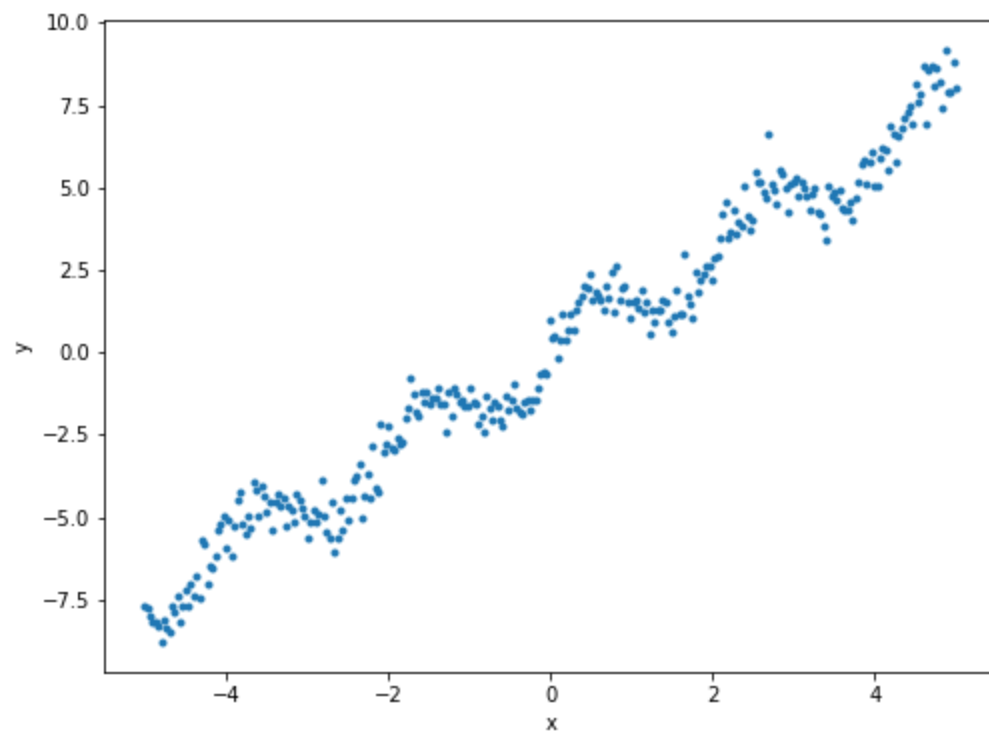
```
In [5]:  def scatter(x, y):
             """
             Generate a scatter plot using x and y

             Keyword arguments:
             x -- the vector of values x
             y -- the vector of values y
             """
             plt.figure(figsize=(8, 6))
             ### BEGIN Solution
             sns.scatterplot(x, y)
             ### END Solution
             # YOUR CODE HERE
     #          raise NotImplementedError()


         x = data['x']
         y = data['y']
         scatter(x,y)
```
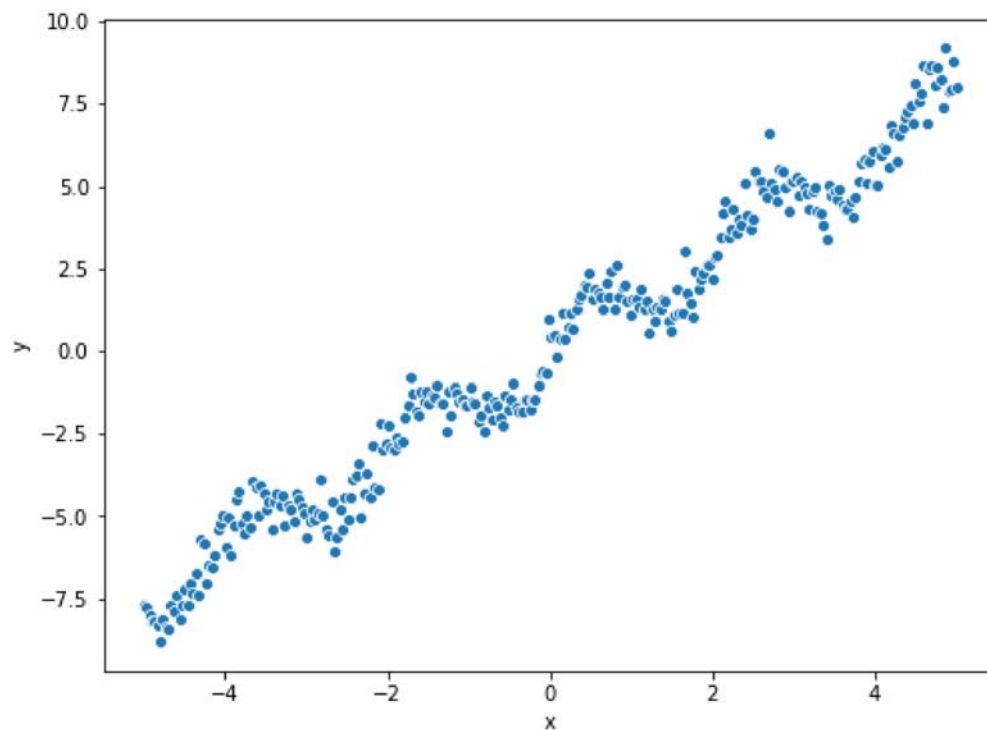
Question 1b

Describe any significant observations about the distribution of the data. How can you describe the relationship between x and y?

Answer

1. x and y are positively correlated
2. after removing the linear trend, there is significant periodic pattern (seasonality) between x and y

Question 1c

The data looks roughly linear, with some extra noise. For now, let's assume that the data follows some underlying linear model. We define the underlying linear model that predicts the value y using the value x as: $f_{\theta^*}(x) = \theta^* \cdot x$

Since we cannot find the value of the population parameter θ^* exactly, we will assume that our dataset approximates our population and use our dataset to estimate θ^*. We denote our estimation with θ, our fitted estimation with $\hat{\theta}$, and our model as:

$$f_\theta(x) = \theta \cdot x$$

Based on this equation, define the linear model function `linear_model` below to estimate **y** (the y-values) given **x** (the x-values) and θ. This model is similar to the model you defined in Lab 5: Modeling and Estimation.

```
In [6]:  def linear_model(x, theta):
             """
             Returns the estimate of y given x and theta

             Keyword arguments:
             x -- the vector of values x
             theta -- the scalar theta
             """
             ### BEGIN Solution
             y = x * theta
             ### END Solution
             # YOUR CODE HERE
         #     raise NotImplementedError()
             return y
```

```
In [7]:  assert linear_model(0, 1) == 0
         assert linear_model(10, 10) == 100
         assert np.sum(linear_model(np.array([3, 5]), 3)) == 24
         assert linear_model(np.array([7, 8]), 4).mean() == 30
```

Question 1d

In class, we learned that the L^2 (or squared) loss function is smooth and continuous. Let's use L^2 loss to evaluate our estimate θ, which we will use later to identify an optimal θ, represented as $\hat{\theta}$. Define the L^2 loss function `l2_loss` below.

```
In [8]:  def l2_loss(y, y_hat):
             """
             Returns the average l2 loss given y and y_hat

             Keyword arguments:
             y -- the vector of true values y
             y_hat -- the vector of predicted values y_hat
             """
             ### BEGIN Solution
             return np.mean((y-y_hat)**2)
             ### END Solution
             # YOUR CODE HERE
         #      raise NotImplementedError()
```
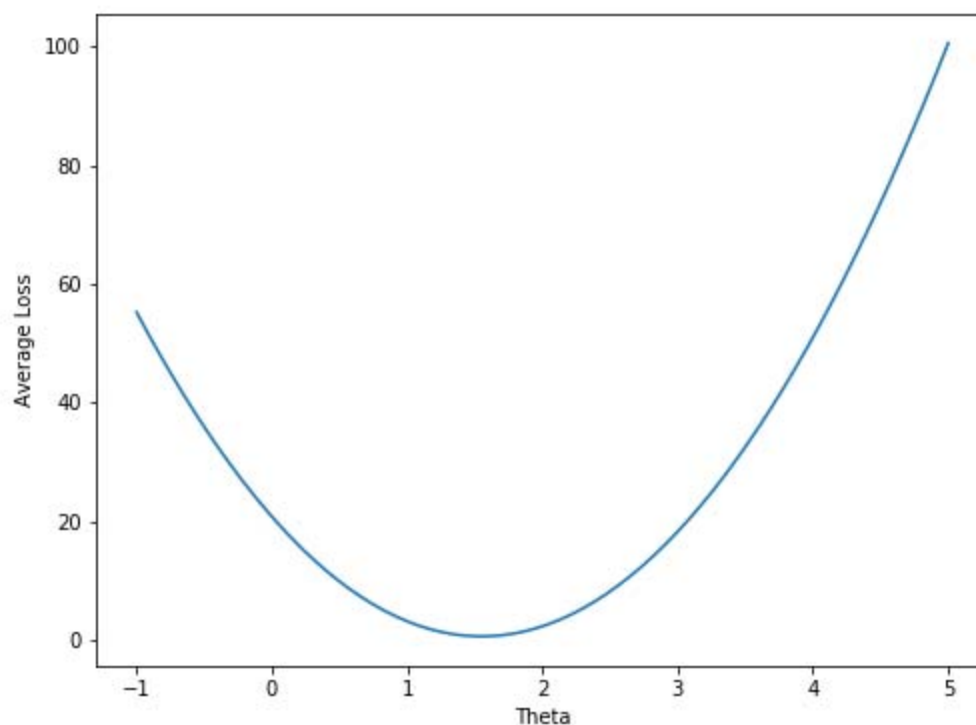
```
In [9]:  assert l2_loss(2, 1) == 1
         assert l2_loss(2, 0) == 4
         assert l2_loss(5, 1) == 16
         assert l2_loss(np.array([5, 6]), np.array([1, 1])) == 20.5
         assert l2_loss(np.array([1, 1, 1]), np.array([4, 1, 4])) == 6.0
```

Question 1e

First, visualize the L^2 loss as a function of θ, where several different values of θ are given. Be sure to label your axes properly. You plot should look something like this:

What looks like the optimal value, $\hat{\theta}$, based on the visualization? Set `theta_star_guess` to the value of θ that appears to minimize our loss.

In [10]:
```python
def visualize(x, y, thetas):
    """
    Plots the average l2 loss for given x, y as a function of theta.
    Use the functions you wrote for linear_model and l2_loss.

    Keyword arguments:
    x -- the vector of values x
    y -- the vector of values y
    thetas -- an array containing different estimates of the scalar theta
    """
    avg_loss = np.array([l2_loss(y, linear_model(x, theta)) for theta in thetas])

    plt.figure(figsize=(8,6))

    # Create your plot here
    sns.lineplot(x=thetas, y=avg_loss)
#     plt.legend()
    plt.xlabel(r"$\Theta$")
    plt.ylabel("Average Loss");

    res = {}
    res["avg_loss"] = avg_loss
    return res

    # YOUR CODE HERE
#     raise NotImplementedError()

thetas = np.linspace(-1, 5, 70)
r = visualize(x, y, thetas)
theta_star_guess = 1.5217
theta_star_guess = thetas[np.argmin(r["avg_loss"])]

# YOUR CODE HERE
# raise NotImplementedError()
```
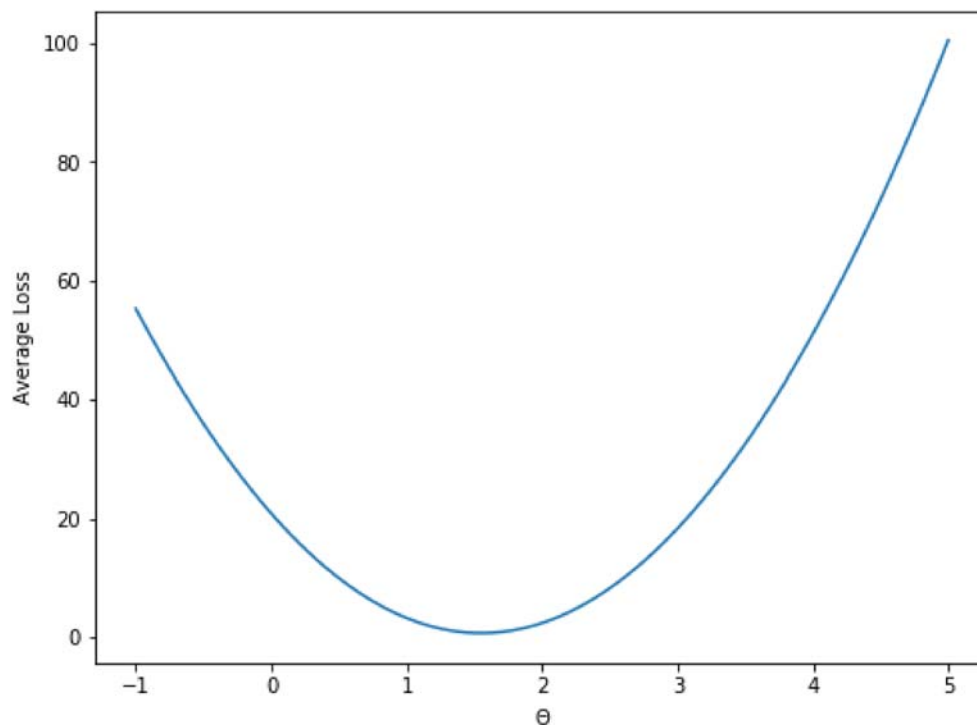
```
In [11]:  assert l2_loss(3, 2) == 1
          assert l2_loss(0, 10) == 100
          assert 1 <= theta_star_guess <= 2
```

2: Fitting our Simple Model

Now that we have defined a simple linear model and loss function, let's begin working on fitting our model to the data.

Question 2

Let's confirm our visual findings for optimal $\hat{\theta}$.

Question 2a

First, find the analytical solution for the optimal $\hat{\theta}$ for average L^2 loss. Write up your solution in the cell below using LaTex.

Hint: notice that we now have $\mathbf{x}$ and $\mathbf{y}$ instead of x and y. This means that when writing the loss function $L(\mathbf{x}, \mathbf{y}, \theta)$, you'll need to take the average of the squared losses for each $y_i, f_\theta(x_i)$ pair. For tips on getting started, see chapter [https://www.textbook.ds100.org/ch/10/modeling_loss_functions.html] (https://www.textbook.ds100.org/ch/10/modeling_loss_functions.html%5D)(chapter 10) of the textbook. Note that if you click "Open in DataHub", you can access the LaTeX source code of the book chapter, which you might find handy for typing up your work. Show your work, i.e. don't just write the answer.

Answer

In the matrix form, the simple linear regression is $\mathbf{x}\theta = \mathbf{y}$, where $\mathbf{x}, \mathbf{y}$ are column vector and θ is a scalar.

Define error vector $y^{er} = \mathbf{x}\theta - \mathbf{y}$, it is eivident that minimizing squared-loss is equivalent to minimizing the error vector, which is equivalent that dot product of $< y^{er}, \mathbf{x'} >= 0$ (where x' is the convention for x transpose)

$\mathbf{x'}(\mathbf{x}\theta - \mathbf{y}) = 0$
$\mathbf{x'}\mathbf{x}\theta = \mathbf{x'}\mathbf{y}$
$\theta = (\mathbf{x'}\mathbf{x})^{-1}\mathbf{x'}\mathbf{y}$
In other words,
$$\theta = \frac{< \mathbf{y}, \mathbf{x} >}{< \mathbf{x}, \mathbf{x} >}$$

Question 2b

Now that we have the analytic solution for $\hat{\theta}$, implement the function `find_theta` that calculates the numerical value of $\hat{\theta}$ based on our data $\mathbf{x}, \mathbf{y}$.

```
In [12]:  def find_theta(x, y):
              """
              Find optimal theta given x and y

              Keyword arguments:
              x -- the vector of values x
              y -- the vector of values y
              """
              theta_opt = np.dot(x, y) / np.dot(x, x)
              # YOUR CODE HERE
          #     raise NotImplementedError()
              return theta_opt
```
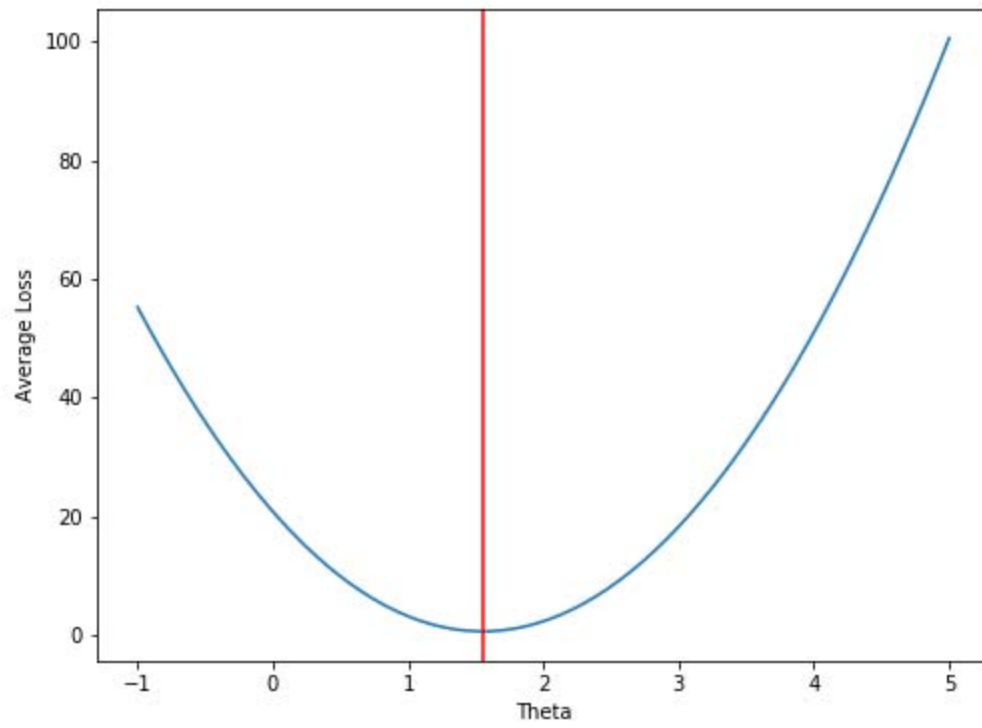
```
In [13]:  t_hat = find_theta(x, y)
          print(f'theta_opt = {t_hat}')

          assert 1.4 <= t_hat <= 1.6
```

```
          theta_opt = 1.550264808596222
```

Question 2c

Now, let's plot our loss function again using the `visualize` function. But this time, add a vertical line at the optimal value of theta (plot the line $x = \hat{\theta}$). Your plot should look something like this:

```
In [14]:  ### BEGIN Solution
          theta_opt = find_theta(x, y)
          thetas = np.linspace(-1, 5, 70)
          visualize(x, y, thetas)
          plt.axvline(x=theta_opt, color='r');
          ### END Solution
          # YOUR CODE HERE
          # raise NotImplementedError()
```
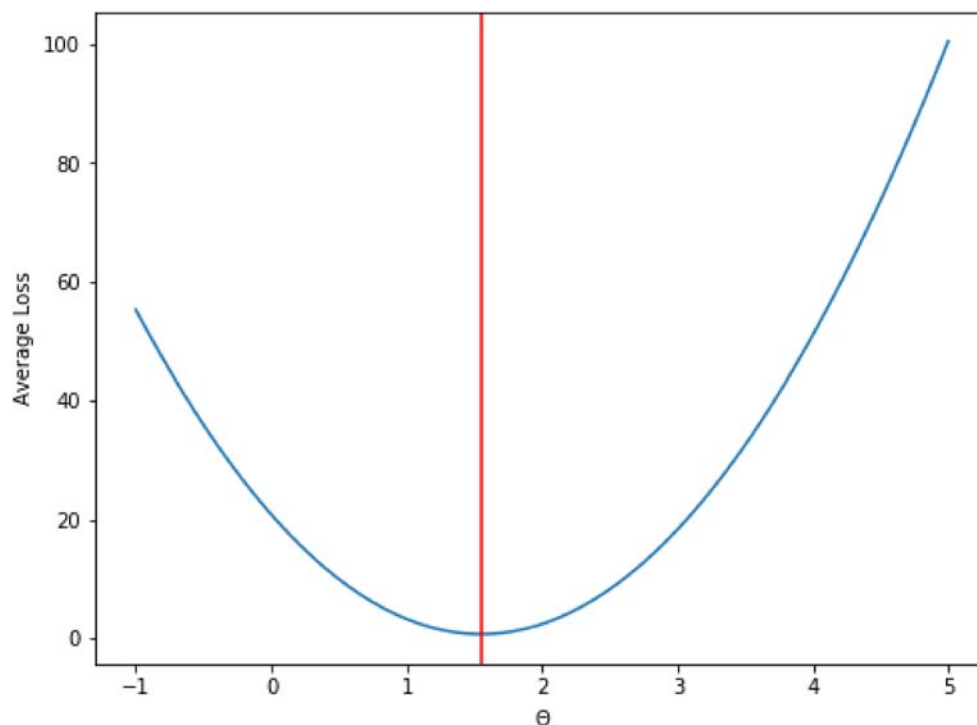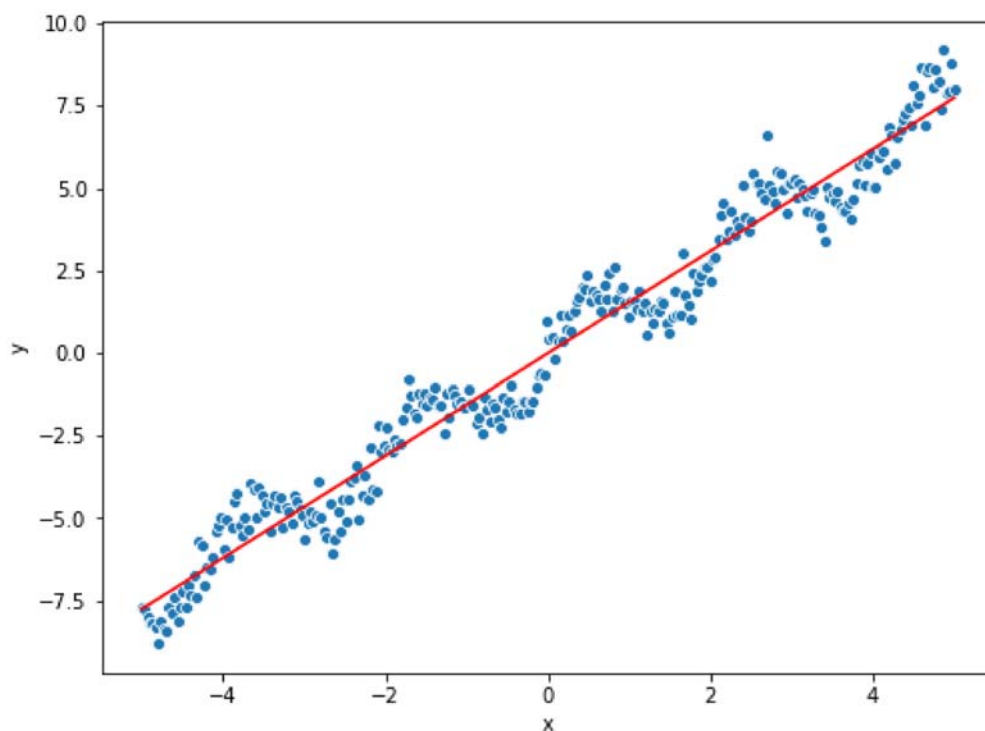

Question 2d

We now have an optimal value for θ that minimizes our loss. In the cell below, plot the scatter plot of the data from Question 1a (you can reuse the `scatter` function here). But this time, add the line $f_{\hat{\theta}}(x) = \hat{\theta} \cdot \mathbf{x}$ using the $\hat{\theta}$ you computed above. Your plot should look something like this:
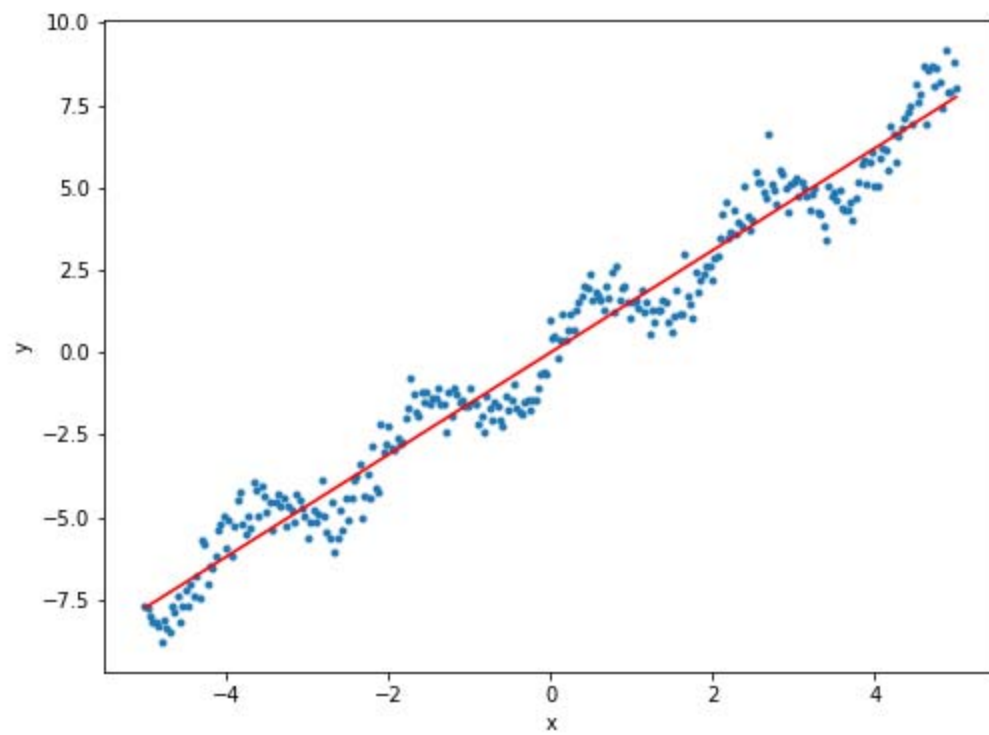
```
In [15]:  ### BEGIN Solution
          scatter(x,y)
          theta_opt = find_theta(x, y)
          y_hat = x * theta_opt
          sns.lineplot(x, y_hat, color='r');
          plt.ylabel("y");
          ### END Solution
          # YOUR CODE HERE
          # raise NotImplementedError()
```


Question 2e

Great! It looks like our estimator $f_{\hat{\theta}}(x)$ is able to capture a lot of the data with a single parameter θ. Now let's try to remove the linear portion of our model from the data to see if we missed anything.

The remaining data is known as the residual, $\mathbf{r} = \mathbf{y} - \hat{\theta} \cdot \mathbf{x}$. Below, write a function to find the residual and plot the residuals corresponding to x in a scatter plot. Plot a horizontal line at $y = 0$ to assist visualization.

```python
In [16]: def visualize_residual(x, y):
             """
             Plot a scatter plot of the residuals, the remaining
             values after removing the linear model from our data.

             Keyword arguments:
             x -- the vector of values x
             y -- the vector of values y
             """

             ...
             # YOUR CODE HERE
         #     raise NotImplementedError()
             ### BEGIN Solution
             sns.scatterplot(x, y-x*find_theta(x, y));
             plt.title("Residual plot")
             plt.ylabel("residual")
             ### END Solution

         visualize_residual(x, y)
```
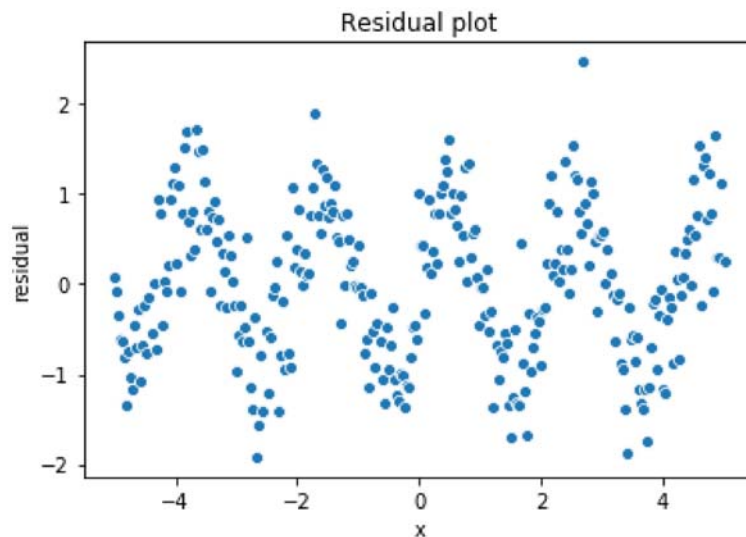

Residual plot

Question 2f

What does the residual look like? Do you notice a relationship between x and r?

Answer

1. the residual have mean zero
2. the plot has a period of around 2
3. x and r have sin/cos relationship

3: Increasing Model Complexity

It looks like the remaining data is sinusoidal, meaning our original data follows a linear function and a sinusoidal function. Let's define a new model to address this discovery and find optimal parameters to best fit the data:

$$f_{\boldsymbol{\theta}}(x) = \theta_1 x + sin(\theta_2 x)$$

Now, our model is parameterized by both θ_1 and θ_2, or composed together, $\boldsymbol{\theta}$.

Note that a generalized sine function $a \sin(bx + c)$ has three parameters: amplitude scaling parameter a, frequency parameter b and phase shifting parameter c. Looking at the residual plot above, it looks like the residual is zero at x = 0, and the residual swings between -1 and 1. Thus, it seems reasonable to effectively set the scaling and phase shifting parameter (a and c in this case) to 1 and 0 respectively. While we could try to fit a and c, we're unlikely to get much benefit. When you're done with the homework, you can try adding a and c to our model and fitting these values to see if you can get a better loss.

Question 3a

As in Question 1, fill in the `sin_model` function that predicts **y** (the y-values) using **x** (the x-values), but this time based on our new equation.

Hint: Try to do this without using for loops. The `np.sin` function may help you.

```
In [17]: def sin_model(x, theta_1, theta_2):
             """
             Predict the estimate of y given x, theta_1, theta_2

             Keyword arguments:
             x -- the vector of values x
             theta_1 -- the scalar value theta_1
             theta_2 -- the scalar value theta_2
             """
             ### BEGIN Solution
             y = x * theta_1 + np.sin(x * theta_2)
             ### END Solution
             # YOUR CODE HERE
         #     raise NotImplementedError()
             return y
```

```
In [18]: assert np.isclose(sin_model(1, 1, np.pi), 1.0000000000000002)
         # Check that we accept x as arrays
         assert len(sin_model(x, 2, 2)) > 1
```

Question 3b

Use the average L^2 loss to compute $\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}$.

First, we will use LaTex to write $L(\mathbf{x}, \mathbf{y}, \theta_1, \theta_2)$, $\frac{\partial L}{\partial \theta_1}$, and $\frac{\partial L}{\partial \theta_2}$ given $\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}$.

You don't need to write out the full derivation. Just the final expression is fine.

Answer

$$L(\mathbf{x}, \mathbf{y}, \theta_1, \theta_2) = \frac{1}{n} \sum_i (x_i\theta_1 + \sin(\theta_2 x_i) - y_i)^2$$

$$\frac{\partial L}{\partial \theta_1} = \frac{2}{n} \sum_i (x_i\theta_1 + \sin(\theta_2 x_i) - y_i)x_i$$

$$\frac{\partial L}{\partial \theta_2} = \frac{2}{n} \sum_i (x_i\theta_1 + \sin(\theta_2 x_i) - y_i)x_i \cos(\theta_2 x_i)$$

Question 3c

Now, implement the functions `dt1` and `dt2`, which should compute $\frac{\partial L}{\partial \theta_1}$ and $\frac{\partial L}{\partial \theta_2}$ respectively. Use the formulas you wrote for $\frac{\partial L}{\partial \theta_1}$ and $\frac{\partial L}{\partial \theta_2}$ in the previous exercise. In the functions below, the parameter `theta` is a vector that looks like (θ_1, θ_2).

Note: To keep your code a bit more concise, be aware that `np.mean` does the same thing as `np.sum` divided by the length of the numpy array.

```
In [19]: def dt1(x, y, theta):
             """
             Compute the numerical value of the partial of l2 loss with respect to theta_1

             Keyword arguments:
             x -- the vector of all x values
             y -- the vector of all y values
             theta -- the vector of values theta
             """
             ### BEGIN Solution
             return np.mean((sin_model(x, theta[0], theta[1])-y)*x) *2
             ### END Solution
             # YOUR CODE HERE
         #     raise NotImplementedError()
```

```
In [20]: def dt2(x, y, theta):
             """
             Compute the numerical value of the partial of l2 loss with respect to theta_2

             Keyword arguments:
             x -- the vector of all x values
             y -- the vector of all y values
             theta -- the vector of values theta
             """
             ### BEGIN Solution
             return np.mean((sin_model(x, theta[0], theta[1])-y)*np.cos(x*theta[1])*x) *2
             ### END Solution
             # YOUR CODE HERE
         #     raise NotImplementedError()
```

```
In [21]:  # This function calls dt1 and dt2 and returns the gradient dt. It is already imple
          def dt(x, y, theta):
              """
              Returns the gradient of l2 loss with respect to vector theta

              Keyword arguments:
              x -- the vector of values x
              y -- the vector of values y
              theta -- the vector of values theta
              """
              return np.array([dt1(x,y,theta), dt2(x,y,theta)])
```

```
In [ ]:
```

```
In [22]:  assert np.isclose(dt1(x, y, [0, np.pi]), -25.376660670924529)
          assert np.isclose(dt2(x, y, [0, np.pi]), 1.9427210155296564)
```

4: Gradient Descent

Now try to solve for the optimal $\hat{\theta}$ analytically...

Just kidding!

You can try but we don't recommend it. When finding an analytic solution becomes difficult or impossible, we resort to alternative optimization methods for finding an approximate solution.

Question 4

So let's try implementing a numerical optimization method: gradient descent!

Question 4a

Implement the `grad_desc` function that performs gradient descent for a finite number of iterations. This function takes in an array for $\mathbf{x}$ (x), an array for $\mathbf{y}$ (y), and an initial value for θ (theta). alpha will be the learning rate (or step size, whichever term you prefer). In this part, we'll use a static learning rate that is the same at every time step.

At each time step, use the gradient and alpha to update your current theta . Also at each time step, be sure to save the current theta in theta_history , along with the L^2 loss (computed with the current theta) in loss_history .

Hints:

- Write out the gradient update equation (1 step). What variables will you need for each gradient update? Of these variables, which ones do you already have, and which ones will you need to recompute at each time step?
- You may need a loop here to update theta several times

- Recall that the gradient descent update function follows the form:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha \left(\nabla_{\boldsymbol{\theta}} \mathbf{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}^{(t)}) \right)$$

```python
In [23]:  # Run me
          def init_t():
              """Creates an initial theta [0, 0] of shape (2,) as a starting point for gradi
              return np.zeros((2,))
```

```python
In [24]:  def grad_desc(x, y, theta, num_iter=20, alpha=0.1):
              """
              Run gradient descent update for a finite number of iterations and static learn

              Keyword arguments:
              x -- the vector of values x
              y -- the vector of values y
              theta -- the vector of values theta to use at first iteration
              num_iter -- the max number of iterations
              alpha -- the learning rate (also called the step size)

              Return:
              theta -- the optimal value of theta after num_iter of gradient descent
              theta_history -- the series of theta values over each iteration of gradient de
              loss_history -- the series of loss values over each iteration of gradient desc
              """
              theta_history = []
              loss_history = []

              ### BEGIN Solution
              for itr in range(num_iter):
                  theta = theta - alpha * dt(x, y, theta)
                  loss = np.mean((sin_model(x, theta[0], theta[1])-y)**2)
                  theta_history.append(theta);
                  loss_history.append(loss)
              ### END Solution

              # YOUR CODE HERE
          #     raise NotImplementedError()
              return theta, theta_history, loss_history
```

```python
In [25]:  t = init_t()
          t_est, ts, loss = grad_desc(x, y, t, num_iter=20, alpha=0.1)

          assert len(ts) == len(loss) == 20 # theta history and loss history are 20 items in
          assert ts[0].shape == (2,) # theta history contains theta values
          assert np.isscalar(loss[0]) # loss history is a list of scalar values, not vector

          assert loss[1] - loss[-1] > 0 # loss is decreasing

          assert np.allclose(np.sum(t_est), 4.5, atol=2e-1)  # theta_est should be close to
```

Question 4b

Now, let's try using a decaying learning rate. Implement `grad_desc_decay` below, which performs gradient descent with a learning rate that decreases slightly with each time step. You should be able to copy most of your work from the previous part, but you'll need to tweak how you update `theta` at each time step.

By decaying learning rate, we mean instead of just a number α, the learning should be now $\frac{\alpha}{i+1}$ where i is the current number of iteration. (Why do we need to add '+ 1' in the denominator?)

In [26]:
```python
def grad_desc_decay(x, y, theta, num_iter=20, alpha=0.1):
    """
    Run gradient descent update for a finite number of iterations and decaying lea

    Keyword arguments:
    x -- the vector of values x
    y -- the vector of values y
    theta -- the vector of values theta
    num_iter -- the max number of iterations
    alpha -- the learning rate

    Return:
    theta -- the optimal value of theta after num_iter of gradient descent
    theta_history -- the series of theta values over each iteration of gradient de
    loss_history -- the series of loss values over each iteration of gradient desc
    """
    theta_history = []
    loss_history = []

    ### BEGIN Solution
    for itr in range(num_iter):
        theta = theta - (alpha/(itr+1)) * dt(x, y, theta)
        loss = np.mean((sin_model(x, theta[0], theta[1])-y)**2)
        theta_history.append(theta);
        loss_history.append(loss)
    ### END Solution

    # YOUR CODE HERE
#     raise NotImplementedError()
    return theta, theta_history, loss_history
```

In [27]:
```python
t = init_t()
t_est_decay, ts_decay, loss_decay = grad_desc_decay(x, y, t, num_iter=20, alpha=0.

assert len(ts_decay) == len(loss_decay) == 20 # theta history and loss history are
assert ts_decay[0].shape == (2,) # theta history contains theta values
assert np.isscalar(loss[0]) # loss history should be a list of values, not vector

assert loss_decay[1] - loss_decay[-1] > 0 # loss is decreasing

assert np.allclose(np.sum(t_est_decay), 4.5, atol=2e-1)  # theta_est should be clo
```
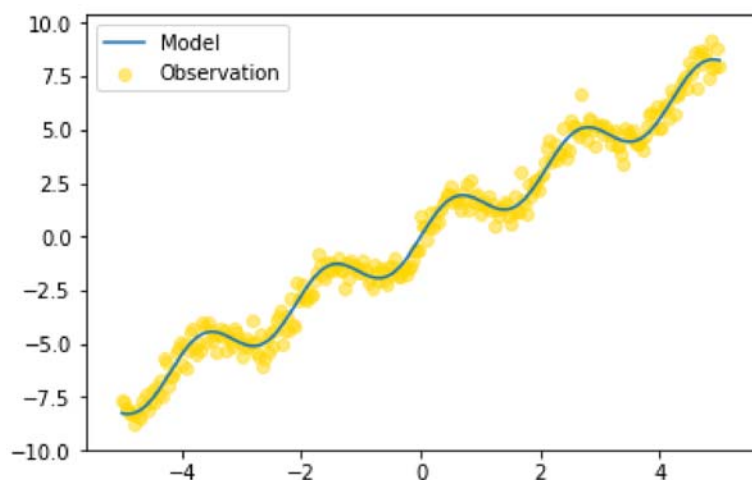
Question 4c

Let's visually inspect our results of running gradient descent to optimize $\boldsymbol{\theta}$. Plot our x-values with our model's predicted y-values over the original scatter plot. Did gradient descent successfully optimize $\boldsymbol{\theta}$?

In [28]:
```python
# Run me
t = init_t()
t_est, ts, loss = grad_desc(x, y, t)

t = init_t()
t_est_decay, ts_decay, loss_decay = grad_desc_decay(x, y, t)
```

In [29]:
```python
y_pred = sin_model(x, t_est[0], t_est[1])

plt.plot(x, y_pred, label='Model')
plt.scatter(x, y, alpha=0.5, label='Observation', color='gold')
plt.legend();
```
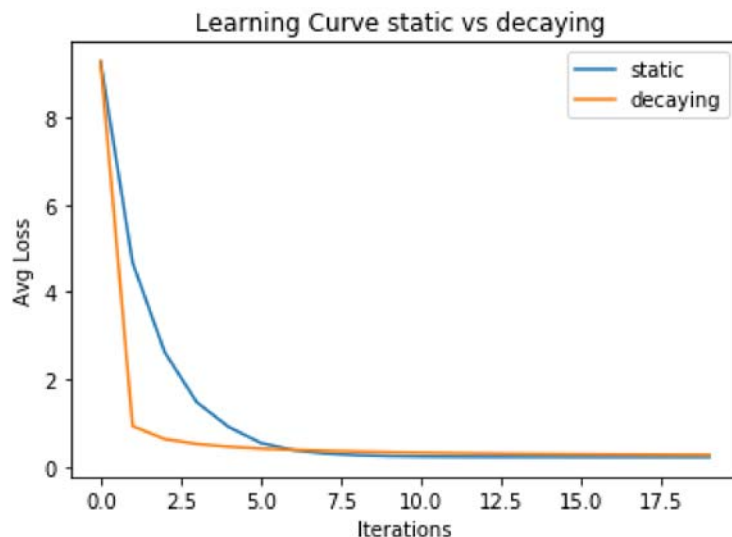

Answer

- sicne the blue curve fit the data point pretty well, the gradient descent method sucessfully optimize θ

Question 4d

Let's compare our two gradient descent methods and see how they differ. Plot the loss values over each iteration of gradient descent for both static learning rate and decaying learning rate.

```
In [30]:  # plt.plot(...) # Plot of loss history for static learning rate
          # plt.plot(...) # Plot of loss history for decaying learning rate
          ### BEGIN Solution
          sns.lineplot(x=np.arange(len(loss)), y=loss)
          sns.lineplot(x=np.arange(len(loss_decay)), y=loss_decay);
          plt.legend(["static", "decaying"])
          plt.xlabel("Iterations")
          plt.ylabel("Avg Loss")
          plt.title("Learning Curve static vs decaying ");
          ### END Solution
          # YOUR CODE HERE
          # raise NotImplementedError()
```


Question 4e

Compare and contrast the performance of the two gradient descent methods. Which method begins to converge more quickly?

Answer

- Decaying learning rate method converge faster.

5: Visualizing Loss

Question 5:

Let's visualize our loss functions and gain some insight as to how gradient descent and stochastic gradient descent are optimizing our model parameters.
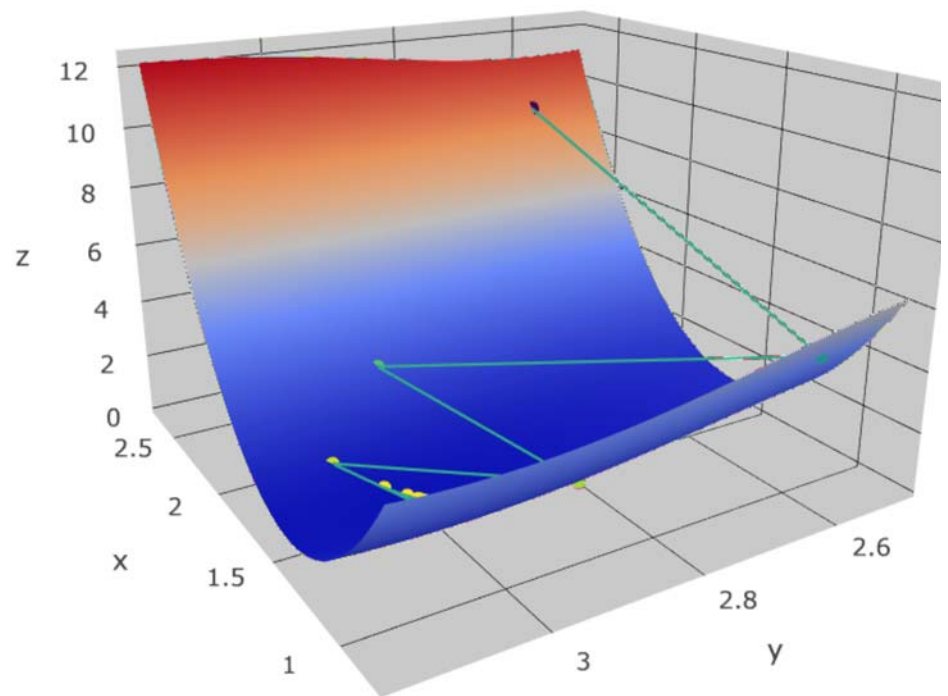
Question 5a:

In the previous plot is about the loss decrease over time, but what exactly is path the theta value? Run the following three cells.

In [31]:
```python
# Run me
ts = np.array(ts).squeeze()
ts_decay = np.array(ts_decay).squeeze()
loss = np.array(loss)
loss_decay = np.array(loss_decay)
```
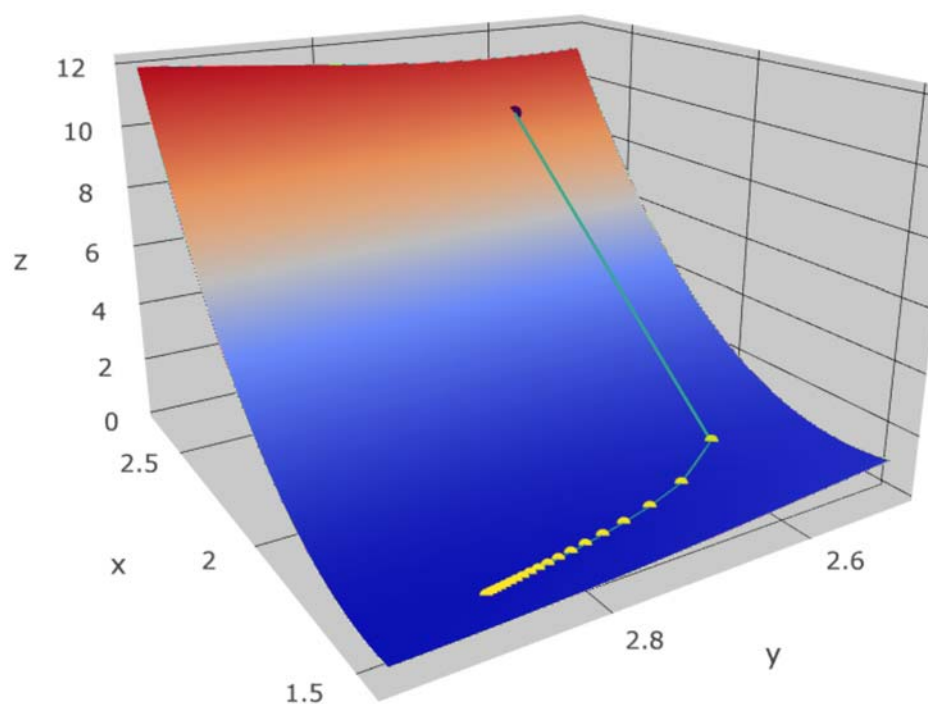
```
In [32]:   # Run me to see a 3D plot (gradient descent with static alpha)
           plot_3d(ts[:, 0], ts[:, 1], loss, l2_loss, sin_model, x, y)
```

Gradient Descent

In [33]:
```
# Run me to see another 3D plot (gradient descent with decaying alpha)
plot_3d(ts_decay[:, 0], ts_decay[:, 1], loss_decay, l2_loss, sin_model, x, y)
```

Gradient Descent

In the following cell, write 1-2 sentences about the differences between using a static learning rate and a learning rate with decay for gradient descent. Use the loss history plot as well as the two 3D visualization to support your answer.

Answer

- For the static learning rate, in the first 2 to 5 steps, the optimizer overshots a lot and hence it takes a while for it to find the minima.
- For the decaying learning rate, as time goes by and approaching the minima, the optimizer adapt smaller and smaller step sizes so that it can avoid overshoting around the minima. Around the minima region, it is a better staretgy to use small step size

Question 5b:

Another common way of visualizing 3D dynamics is with a *contour* plot.

Please refer to this notebook when you are working on the next question: Please refer to this notebook when you are working on the next question: http://www.ds100.org/fa18/assets/lectures/lec09/09-Models-and-Estimation-II.html (http://www.ds100.org/fa18/assets/lectures/lec09/09-Models-and-Estimation-II.html). Search the page for `go.Contour` .

In next question, fill in the necessary part to create a contour plot. Then run the following cells.

```
In [34]:   ## Run me
           import plotly
           import plotly.graph_objs as go
           plotly.offline.init_notebook_mode(connected=True)
```

```python
In [35]: def contour_plot(title, theta_history, loss_function, model, x, y):
             """
             The function takes the following as argument:
                 theta_history: a (N, 2) array of theta history
                 loss: a list or array of loss value
                 loss_function: for example, l2_loss
                 model: for example, sin_model
                 x: the original x input
                 y: the original y output
             """
             theta_1_series = theta_history[:,0] # a list or array of theta_1 value
             theta_2_series = theta_history[:,1] # a list or array of theta_2 value

             # Create trace of theta point
             # Uncomment the following lines and fill in the TODOS
             ### BEGIN Solution
             thata_points = go.Scatter(name="Theta Values",
                                       x= theta_1_series, #TODO
                                       y= theta_2_series, #TODO
                                       mode="lines+markers")
             ### END Solution

             ## In the following block of code, we generate the z value
             ## across a 2D grid
             t1_s = np.linspace(np.min(theta_1_series) - 0.1, np.max(theta_1_series) + 0.1)
             t2_s = np.linspace(np.min(theta_2_series) - 0.1, np.max(theta_2_series) + 0.1)

             x_s, y_s = np.meshgrid(t1_s, t2_s)
             data = np.stack([x_s.flatten(), y_s.flatten()]).T
             ls = []
             for t1, t2 in data:
                 l = loss_function(model(x, t1, t2), y)
                 ls.append(l)
             z = np.array(ls).reshape(50, 50)

             # Create the contour
         #     Uncomment the following lines and fill in the TODOS
             ### BEGIN Solution
             lr_loss_contours = go.Contour(x= t1_s, #TODO
                                           y= t2_s, #TODO
                                           z= z, #TODO
                                           colorscale='Viridis', reversescale=True)
             ### END Solution


             # YOUR CODE HERE
         #     raise NotImplementedError()

             plotly.offline.iplot(go.Figure(data=[lr_loss_contours, thata_points], layout={
```
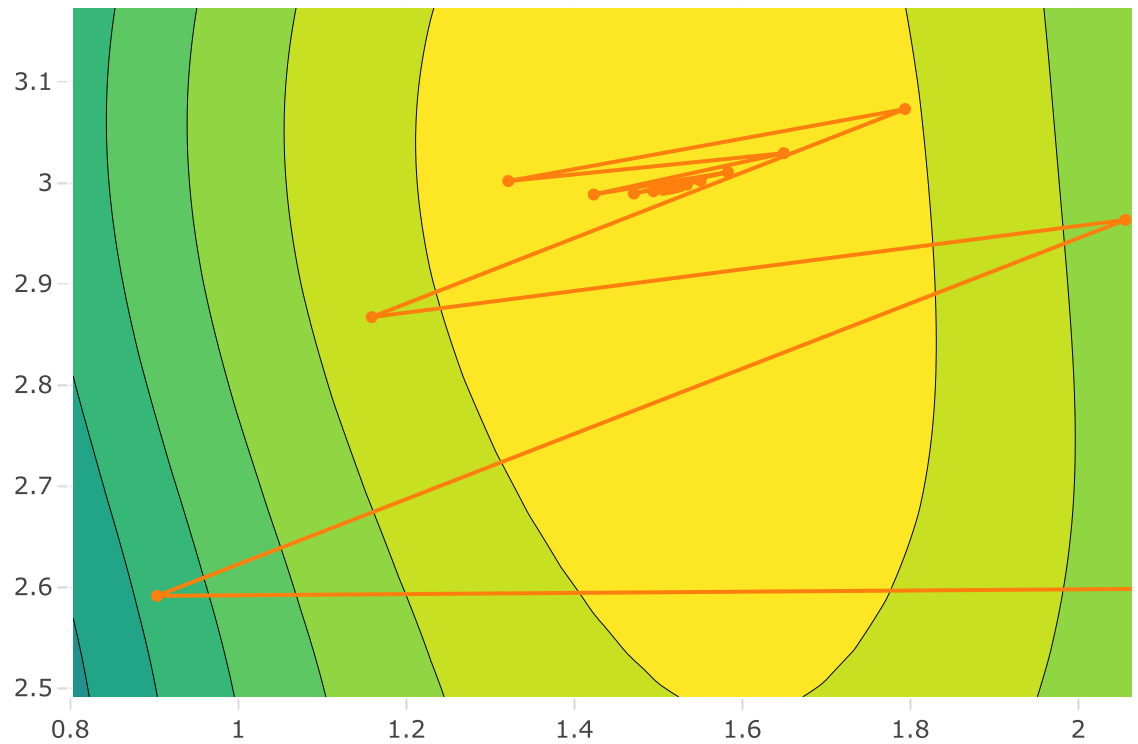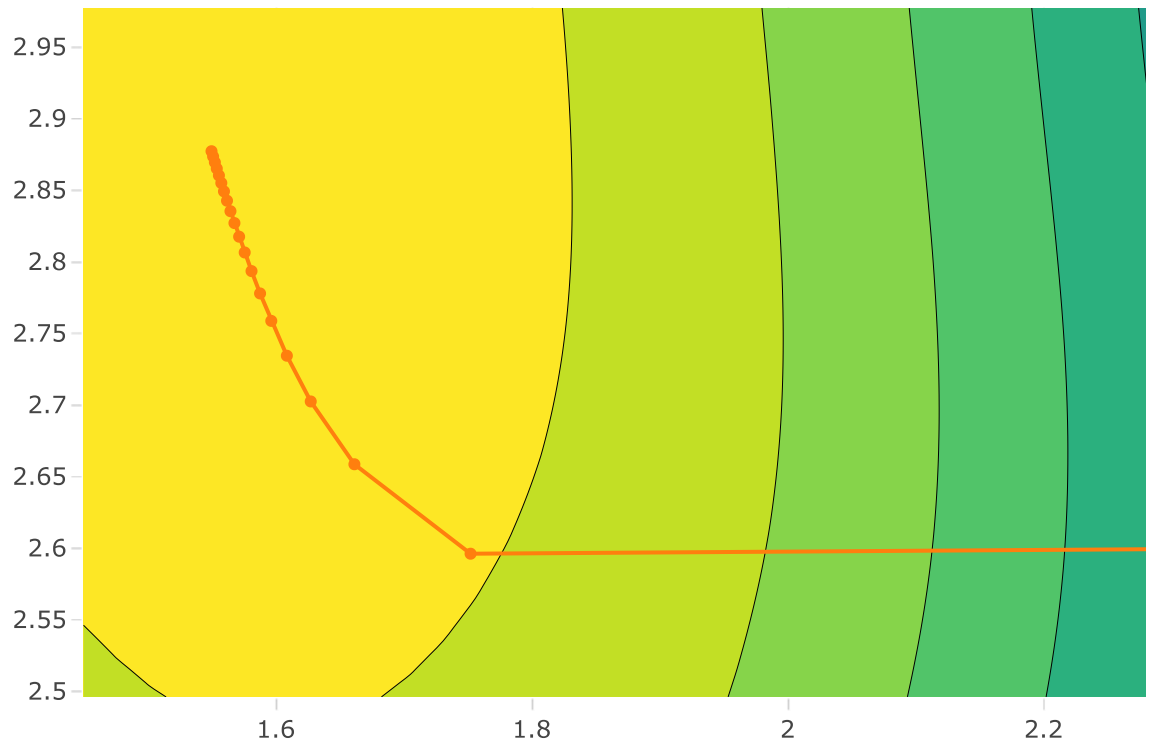
```
In [36]: # Run this
         contour_plot('Gradient Descent with Static Learning Rate', ts, l2_loss, sin_model,
```

Gradient Descent with Static Learnir

```
In [37]:   ## Run me
           contour_plot('Gradient Descent with Decay Learning Rate', ts_decay, l2_loss, sin_m
```

Gradient Descent with Decay Learnin

In the following cells, write down the answer to the following questions:

- How do you interpret the two contour plots?
- Compare contour plot and 3D plot, what are the pros and cons of each?

Answer

- The static learning rate optimizer fluctuates around the minima while the the decaying rate optimizer stably appoach the minima.
- The advantage of 3d plot is that the route of optimizer is clear but the disadvantage is that there are too much information in 3d plot which may confuse people
- The advantage of coutour is that it compress the 3-d information into 2-d, but in general, we cannot tell whether the optimizer have reached the minima or not.

How to Improve?

Question 5c (optional)

Try adding the two additional model parameters for phase and amplitude that we ignored (see 3a). What are the optimal phase and amplitude values for your four parameter model? Do you get a better loss?

YOUR ANSWER HERE

Question 5d (optional)

It looks like our basic two parameter model, a combination of a linear function and sinusoidal function, was able to almost perfectly fit our data. It turns out that many real world scenarios come from relatively simple models.

At the same time, the real world can be incredibly complex and a simple model wouldn't work so well. Consider the example below; it is neither linear, nor sinusoidal, nor quadratic.

Optional: Suggest how we could iteratively create a model to fit this data and how we might improve our results.

Extra optional: Try and build a model that fits this data.

```
In [38]: x = []
         y = []
         for t in np.linspace(0,10*np.pi, 200):
             r = ((t)**2)
             x.append(r*np.cos(t))
             y.append(r*np.sin(t))

         plt.scatter(x,y)
         plt.show()
```
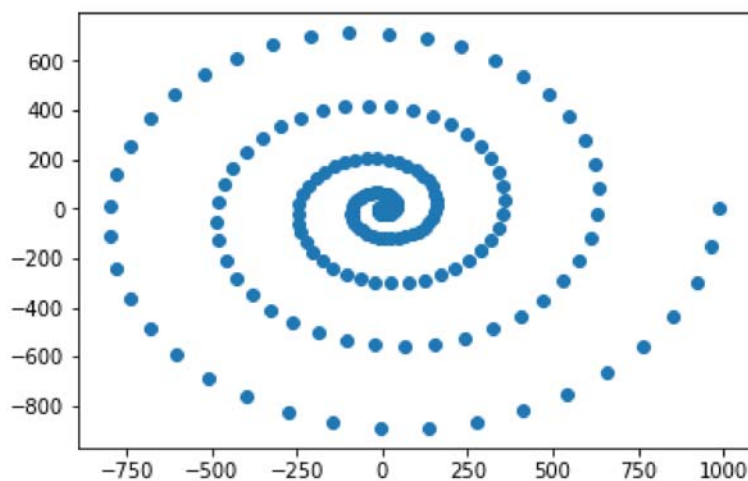

- Can use model $x = r cos(t), y = r sin(t)$ to fit the data

6: Short Analytic Problems

Let's work through some problems to solidify the foundations of gradient descent. If these questions are hard, consider reviewing lecture and supplementary materials.

Question 6

Complete the problems below. **Show your work and solution in LaTeX**. Here are some useful examples of LaTex syntax:

Summation: $\sum_{i=1}^{n} a_i$

Exponent: a^2

Fraction: $\frac{a}{b}$

Multiplication: $a \cdot b$

Derivative: $\frac{\partial}{\partial a}$

Symbols: α, β, θ

Convexity

Question 6a

In [lecture 8 (http://www.ds100.org/fa18/syllabus#lecture-week-5)](http://www.ds100.org/fa18/syllabus#lecture-week-5), we introduced the idea of a convex function. Let $h(x) = f(x) + g(x)$ where f, g are convex functions. Prove that h is convex.

Answer

For any x, y in domain,
$$\alpha h(x) + (1 - \alpha)h(y) = \alpha f(x) + (1 - \alpha)f(y) + \alpha g(x) + (1 - \alpha)g(y)$$
Since f, g are convex,
$$\alpha f(x) + (1 - \alpha)f(y) \geq f(\alpha x + (1 - \alpha)y)$$
$$\alpha g(x) + (1 - \alpha)g(y) \geq g(\alpha x + (1 - \alpha)y)$$
Hence, h is convex.
$$\alpha h(x) + (1 - \alpha)h(y) \geq h(\alpha x + (1 - \alpha)y)$$

Mutlivariable/vector calculus mechanical problems

Question 6b

Show that the sum of the squared error
$$L(\mathbf{w}) = ||\mathbf{Xw} - \mathbf{y}||_2^2$$

can be expanded into

$$L(\mathbf{w}) = \mathbf{w}^T \mathbf{X}^T \mathbf{X}\, \mathbf{w} - 2\mathbf{y}^T \mathbf{X}\, \mathbf{w} + \mathbf{y}^T \mathbf{y}$$

using vector/matrix notation.

Answer

$$||Xw - y||_2^2 = (Xw - y)'(Xw - y)$$

where A' means A transpose

$$(Xw - y)'(Xw - y) = (Xw)'Xw - (Xw)'y - y'(Xw) + y'y = w'X'Xw - 2y'Xw + y'y$$

In other words,

$$L(\mathbf{w}) = \mathbf{w}^T \mathbf{X}^T \mathbf{X}\, \mathbf{w} - 2\mathbf{y}^T \mathbf{X}\, \mathbf{w} + \mathbf{y}^T \mathbf{y}$$

Question 6c

Solve for the optimal $\mathbf{w}$, assuming $\mathbf{X}$ is full rank. Use the Matrix Derivative rules from lecture 11 (http://www.ds100.org/fa18/syllabus#lecture-week-6).

Answer

Let $\frac{\partial}{\partial w} L(w) = 0$

$$2X'Xw - 2X'y = 0$$

Hence, w is solved by (X is full rank)

$$w = (X'X)^{-1}X'y$$

Question 6d

Repeat the steps above for ridge regression as described in lecture 12 (http://www.ds100.org/fa18/syllabus#lecture-week-6). Recall that ridge regression uses the following l2 regularized sum of squared error.

$$L(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_2^2$$

Answer

$$L(w) = (Xw)'Xw - (Xw)'y - y'(Xw) + y'y = w'X'Xw - 2y'Xw + y'y + \lambda w'w$$

Let $\frac{\partial}{\partial w} L(w) = 0$

$$2X'Xw - 2X'y + 2\lambda w = 0$$

$$w = (X'X + \lambda I)^{-1}X'y$$

Question 6e

Compare the analytic solutions of least squares and ridge regression. Why does ridge regression guarantee that we can find a unique solution? What are some of the tradeoffs (pros/cons) of using ridge regression?

Answer

- Since the matrix $X'X + \lambda I$ is always invertible (eigenvalues of new matrix is positive), we can always find a unique solution.
- Pro: The usage of ridge regression helps to mitigate the overfitting issues, since the params w are regularized to be small.
- Con: Although using ridge regression, the model will still be sensitive to outliers.

Expectation and Variance

Question 6f

In [lecture 10 (http://www.ds100.org/fa18/syllabus#lecture-week-6)](http://www.ds100.org/fa18/syllabus#lecture-week-6), we completed half of the proof for the linearity of expectation. Your task in this question is to complete the second half.

For reference, in lecture we showed that:

$$\mathbb{E}[aX + bY + c] = a\mathbb{E}[X] + \sum_{x \in \mathbb{X}}\sum_{y \in \mathbb{Y}}P(x, y)by + c$$

To complete this proof, prove that:

$$b\mathbb{E}[Y] = \sum_{x \in \mathbb{X}}\sum_{y \in \mathbb{Y}}P(x, y)by$$

Note: You cannot simply start with the given equation and use linearity of expectation. Start with the summation on the right side and manipulate it to get the left side.

Hint: What can we do with the order of the summations?

Answer

$$ \sum_{x \in \mathbb{X}}\sum_{y \in \mathbb{Y}}P(x, y)by = b\sum_{y \in \mathbb{Y}}y\sum_{x \in \mathbb{X}}P(x, y) = b\sum_{y \in \mathbb{Y}}yP(y) = b\mathbb{E}[Y] $$

Question 6g

Prove that if two random variables X and Y are independent, then $Var(X - Y) = Var(X) + Var(Y)$.

Answer

If X and Y are independent, cov(X, Y) = 0

$$ Var(X-Y) = Var(X) + Var(-Y) + 2Cov(X, -Y) = Var(X) + Var(Y) $$

7: Quick Regex Problems

Here are some quick problems to review your knowledge of regular expressions.

Question 7a

Write a regular expression to match the following strings without using the | operator.

1. **Match:** `abcdefg`
2. **Match:** `abcde`
3. **Match:** `abc`
4. **Skip:** `c abc`

```
In [39]: ### BEGIN Solution
         # regxa = r"^(?:(?!c ).)+" # fill in your pattern
         regxa = r"^abc.*"
         ### END Solution
         # YOUR CODE HERE
         # raise NotImplementedError()
```

```
In [40]: assert ("|" not in regxa)
         assert (re.search(regxa, "abc").group() == "abc")
         assert (re.search(regxa, "abcde").group() == "abcde")
         assert (re.search(regxa, "abcdefg").group() == "abcdefg")
         assert (re.search(regxa, "c abc") is None)
```

Question 7b

Write a regular expression to match the following strings without using the | operator.

1. **Match:** `can`
2. **Match:** `man`
3. **Match:** `fan`
4. **Skip:** `dan`
5. **Skip:** `ran`
6. **Skip:** `pan`

```
In [41]: ### BEGIN Solution
         regxb = r"[cmf]{1}an" # fill in your pattern
         ### END Solution

         # YOUR CODE HERE
         # raise NotImplementedError()
```

```
In [42]:   assert ("|" not in regxb)
           assert (re.match(regxb, 'can').group() == "can")
           assert (re.match(regxb, 'fan').group() == "fan")
           assert (re.match(regxb, 'man').group() == "man")
           assert (re.match(regxb, 'dan') is None)
           assert (re.match(regxb, 'ran') is None)
           assert (re.match(regxb, 'pan') is None)
```

Question 7c:

Write a regular expression to extract and print the quantity and type of objects in a string. You may assume that a space separates quantity and type, ie. "{quantity} {type}" . See the example string below for more detail.

1. **Hint:** use `re.findall`
2. **Hint:** use `\d` for digits and one of either `*` or `+` .

```
In [43]:   text_qc = "I've got 10 eggs that I stole from 20 gooses belonging to 30 giants."

           res_qc = re.findall(r"\d+ [a-zA-Z]+", text_qc)
           # YOUR CODE HERE
           # raise NotImplementedError()

           res_qc
```

```
Out[43]:   ['10 eggs', '20 gooses', '30 giants']
```

```
In [44]:   assert res_qc == ['10 eggs', '20 gooses', '30 giants']
```

Question 7d:

Write a regular expression to replace at most 2 occurrences of space, comma, or dot with a colon.

Hint: use `re.sub(regex, "newtext", string, number_of_occurences)`

```
In [45]:   text_qd = 'Python Exercises, PHP exercises.'
           res_qd = re.sub(r"[ |,|.]", ":", text_qd, 2)
           # YOUR CODE HERE
           # raise NotImplementedError()

           res_qd
```

```
Out[45]:   'Python:Exercises: PHP exercises.'
```

```
In [46]:   assert res_qd == 'Python:Exercises: PHP exercises.'
```

Question 7e (optional):

Write a regular expression to replace all words that are not `"mushroom"` with `"badger"`.

```
In [47]:  text_qe = 'this is a word mushroom mushroom'
          res_qe = re.sub(r"\b(?!mushroom|badger)\b\S+", "badger", text_qe) # Hint: https://
          # YOUR CODE HERE
          # raise NotImplementedError()
          res_qe
          # re.findall("((?!\bmushroom\b)\S+)+", text_qe)
```

```
Out[47]:  'badger badger badger badger mushroom mushroom'
```

Submission - IMPORTANT, PLEASE READ

For this assignment and future assignments (homework and projects) you will also submit your free response and plotting questions to gradescope. To do this, you can download as PDF (`File->Download As->PDF via Latex (.pdf)`). You are responsible for submitting and tagging your answers in gradescope. For each free response and plotting question, please include:

1. Relevant code used to generate the plot or inform your insights
2. The written free response or plot

We are doing this to make it easier on our graders and for you, in the case you need to submit a regrade request. Gradescope (as of now) is still better for manual grading.

Submission

You're done!

Before submitting this assignment, ensure to:

1. Restart the Kernel (in the menubar, select Kernel->Restart & Run All)
2. Validate the notebook by clicking the "Validate" button

Finally, make sure to **submit** the assignment via the Assignments tab in Datahub