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 = "Weijie Yuan"
COLLABORATORS = "N/A"
```

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/)](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	Points
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 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

In [2]:
```python
# 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)
```

In [3]:
```python
# 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]:
```python
# Run this cell to load our sample data
data = pd.read_csv("https://github.com/DS-100/fa18/raw/gh-pages/assets/datasets/hw3_data.csv", index_col=0)
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

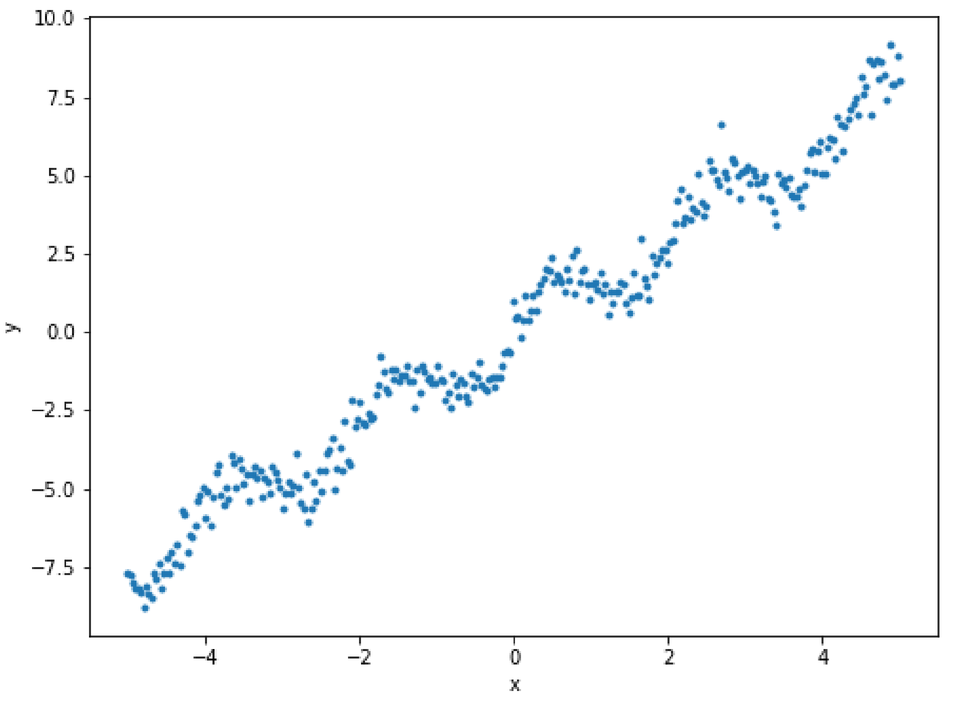
1: A Simple Model

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

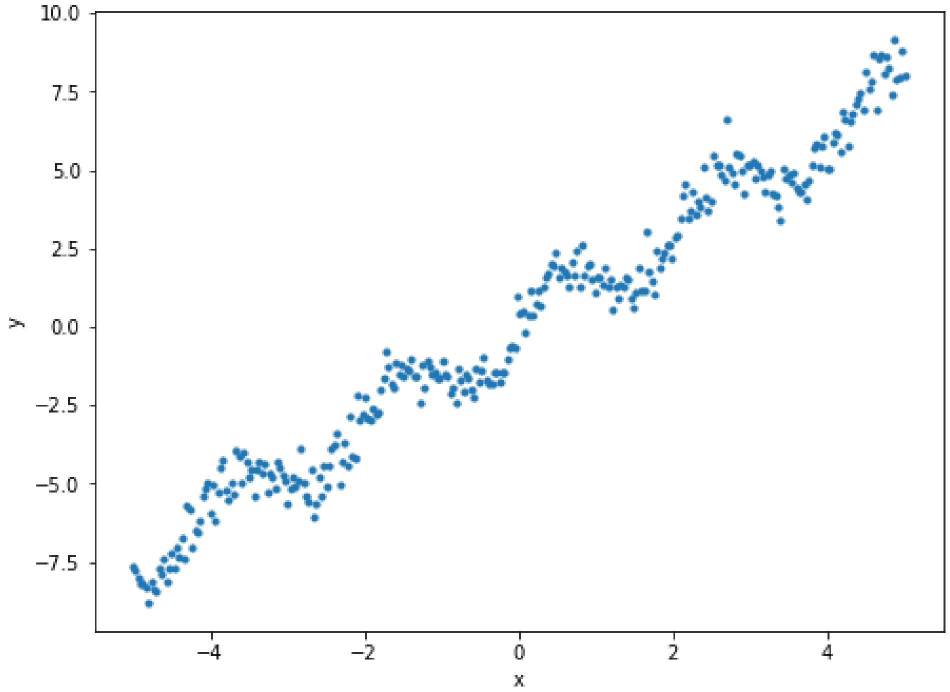
Question 1

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]:
```python
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))
    plt.scatter(x,y,s=10)
    plt.xlabel('x')
    plt.ylabel('y')
    # YOUR CODE HERE

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?

y has posive correlation with x and the slope of this positive correlation shows nearly periodic change. It looks roughly linear, with some extra noise terms.

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 $\mathbf{y}$ (the y-values) given $\mathbf{x}$ (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
    """
    y = x*theta
    # YOUR CODE HERE
    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

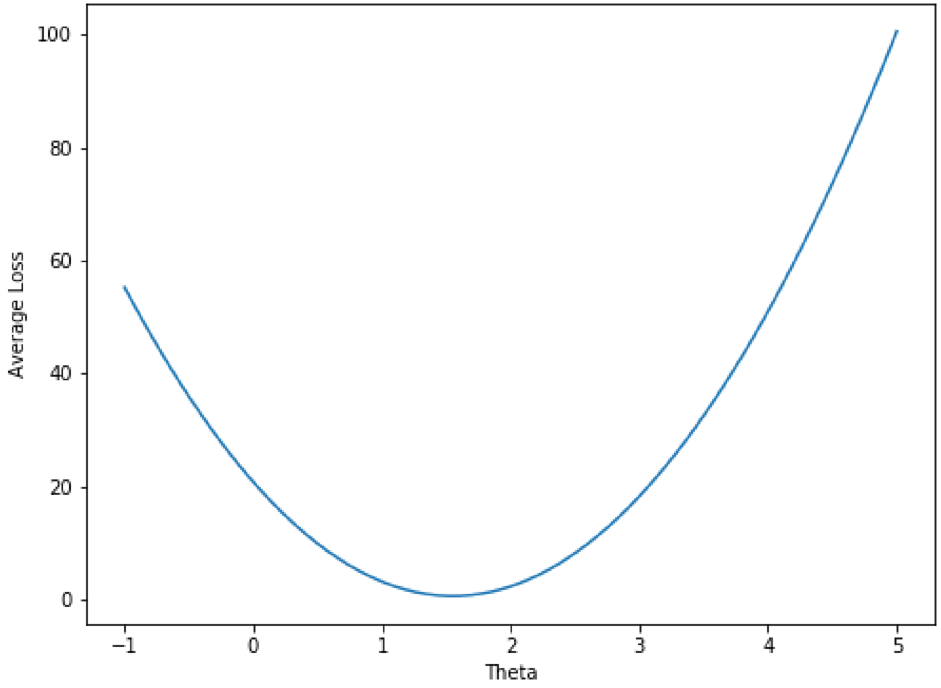
In class, we learned that the L^2 (or squared) loss function is smooth and continuous. Let's use L^2 loss to evaluate our estimate θ, which we will use later to identify an optimal θ, represented as $\hat{\theta}$. Define the L^2 loss function `l2_loss` below.

In [8]:
```python
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
    """
    return np.mean(np.power((y-y_hat),2))
    # YOUR CODE HERE
```

In [9]:
```python
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

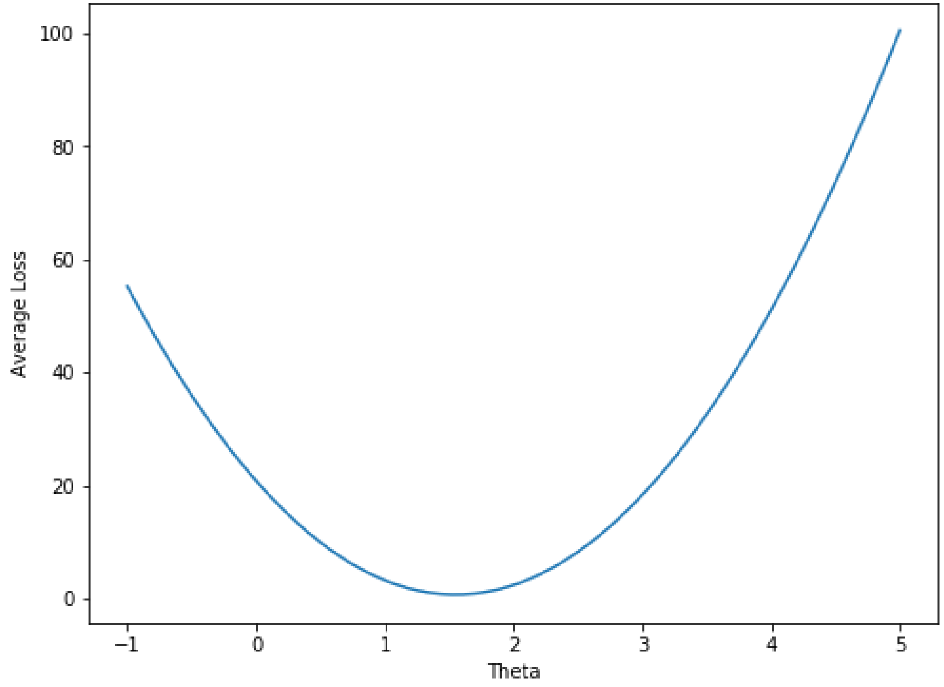
First, visualize the L^2 loss as a function of θ, where several different values of θ are given. Be sure to label your axes properly. You plot should look something like this:



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 = pd.Series([l2_loss(y, x*theta) for theta in thetas]) # Calculate the loss here for each value of theta

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

    plt.plot(thetas,avg_loss)
    plt.xlabel('Theta')
    plt.ylabel('Average Loss')

    # YOUR CODE HERE

thetas = np.linspace(-1, 5, 70)
visualize(x, y, thetas)
theta_star_guess = 1.5
# YOUR CODE HERE
```



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.

$$
\begin{aligned}
L(\mathbf{x}, \mathbf{y}, \theta) &= \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \\
\frac{\partial}{\partial \theta} L(\mathbf{x}, \mathbf{y}, \theta) &= \frac{1}{n} \sum_{i=1}^{n} -2(y_i - x_i\theta) * x_i \\
&= -\frac{2}{n} \sum_{i=1}^{n} (y_i - x_i\theta) * x_i
\end{aligned}
$$

If we want to find the analytical solution for the optimal θ, we can set the derivative of loss function with respect to θ to zero.

$$
\begin{aligned}
-\frac{2}{n} \sum_{i=1}^{n} (y_i - x_i\theta) * x_i &= 0 \\
\sum_{i=1}^{n} (y_i - x_i\theta) * x_i &= 0 \\
\sum_{i=1}^{n} y_i * x_i - \sum_{i=1}^{n} \theta x_i^2 &= 0 \\
\theta \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} y_i x_i \\
\theta &= \frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2} \\
\hat{\theta} = \theta &= \frac{\mathbf{y}^T \mathbf{x}}{\mathbf{x}^T \mathbf{x}}
\end{aligned}
$$

Validation:

$$
\frac{\partial^2}{\partial \theta^2} L(\mathbf{x}, \mathbf{y}, \theta) = \frac{2}{n} \sum_{i=1}^{n} x_i^2 \geq 0
$$

So, $\hat{\theta} = \theta = \frac{\mathbf{y}^T \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$ minimizes the loss function.

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 $\mathbf{x}$, $\mathbf{y}$.

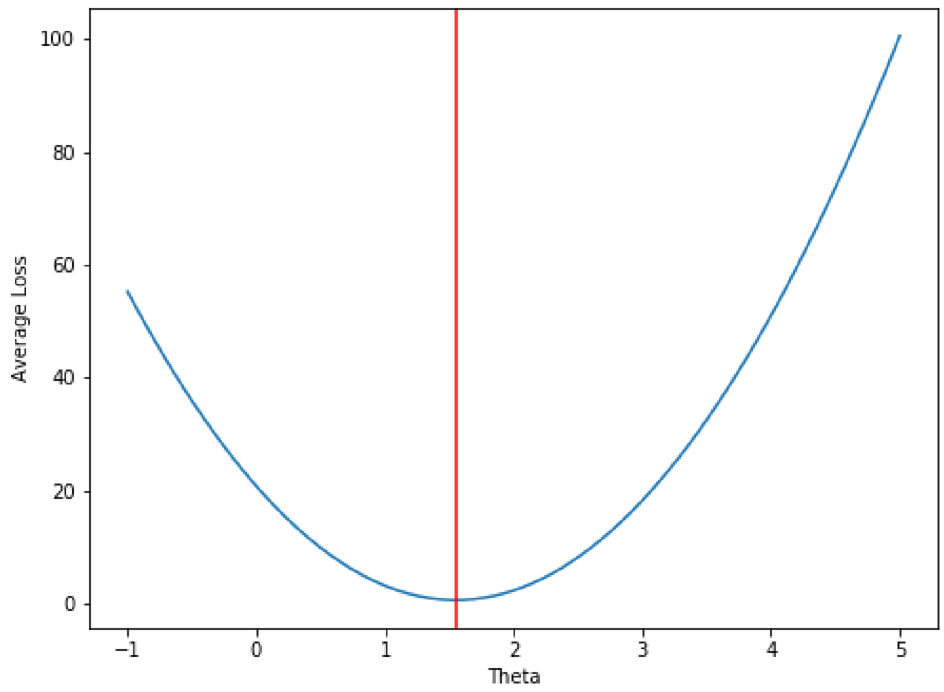
In [12]:
```
def find_theta(x, y):
    """
    Find optimal theta given x and y

    Keyword arguments:
    x -- the vector of values x
    y -- the vector of values y
    """
    theta_opt = sum(y*x)/sum(x*x)
    # YOUR CODE HERE
    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 = 1.5502648085962216
```

Question 2c

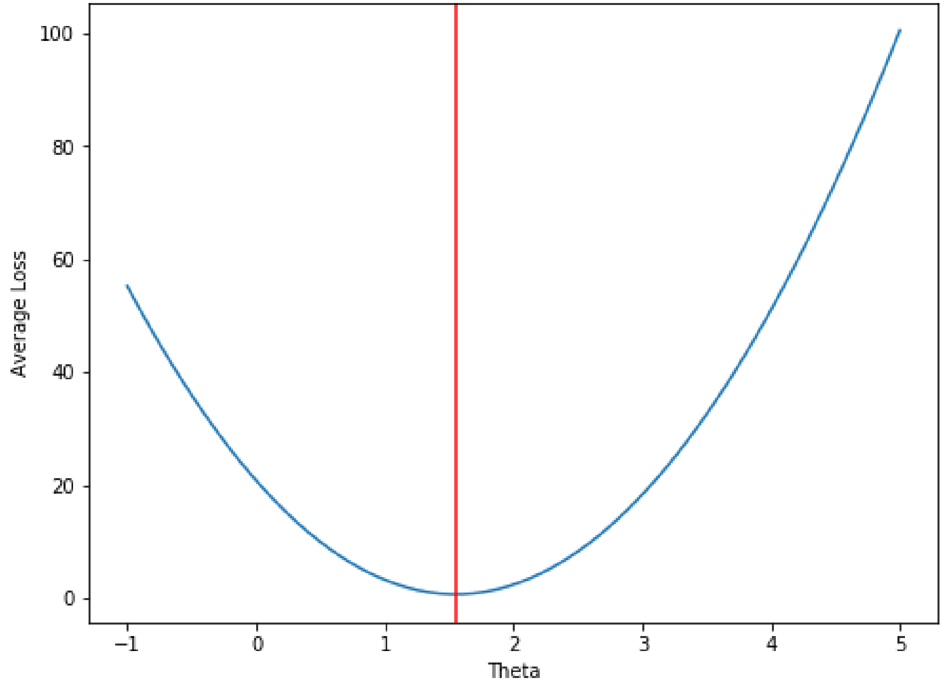
Now, let's plot our loss function again using the `visualize` function. But this time, add a vertical line at the optimal value of theta (plot the line $x = \hat{\theta}$). Your plot should look something like this:



100
80
60
40
20
0
Average Loss
−1
0
1
2
3
4
5
Theta

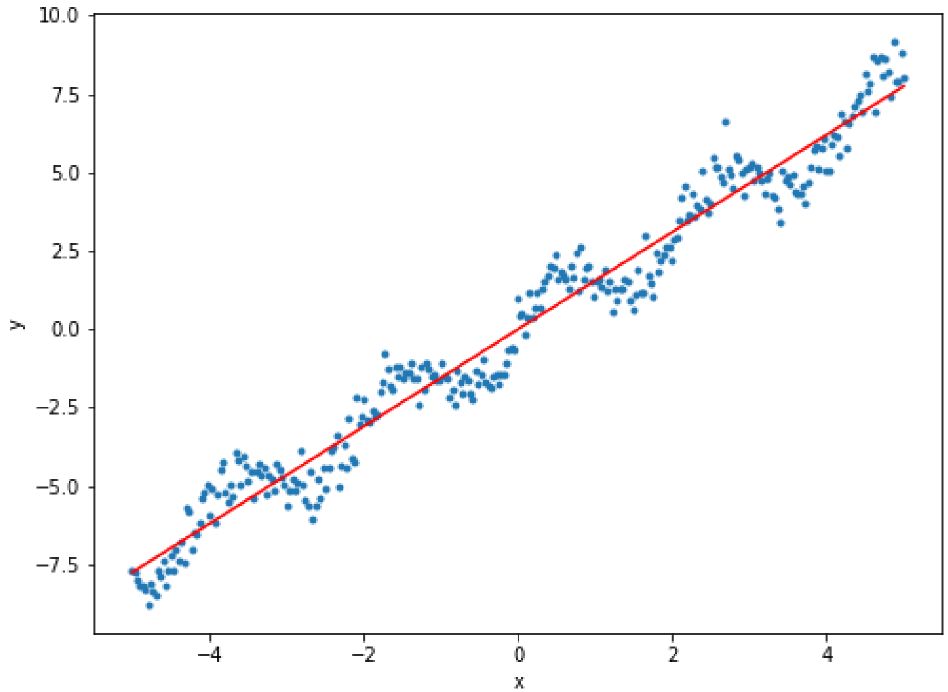
In [14]:

```
theta_opt = t_hat
visualize(x, y, thetas)
plt.axvline(t_hat,color='red')
plt.show()
# YOUR CODE HERE
```

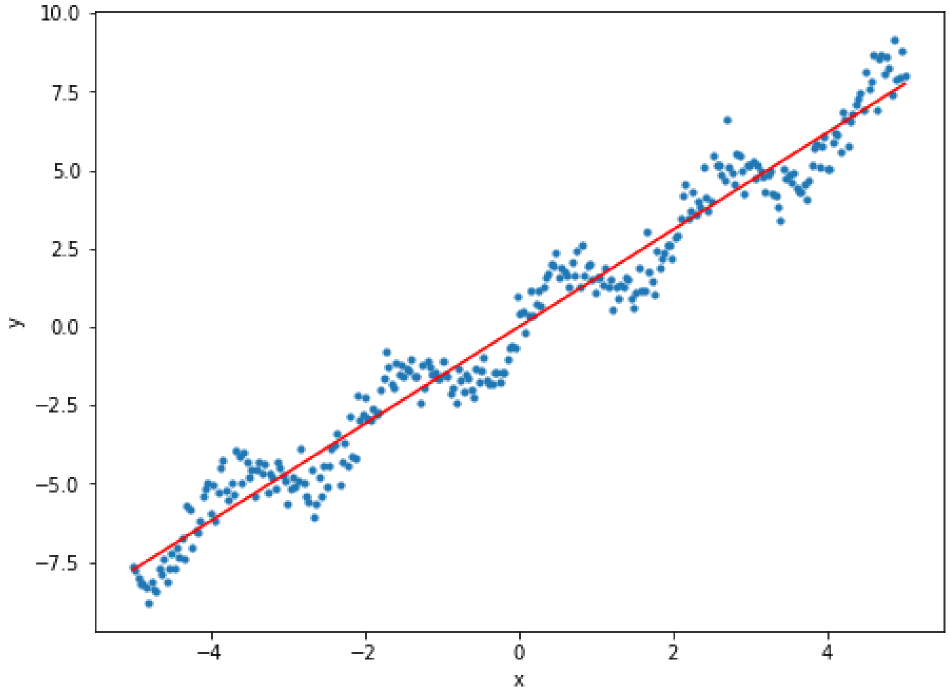


Question 2d

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:


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
y
−4
−2
0
2
4
x

In [15]:
```
theta_opt = t_hat
scatter(x, y)
x_line = np.linspace(x.min(),x.max(),50)
plt.plot(x_line,x_line*t_hat,color='red')
plt.show()
# YOUR CODE HERE
```

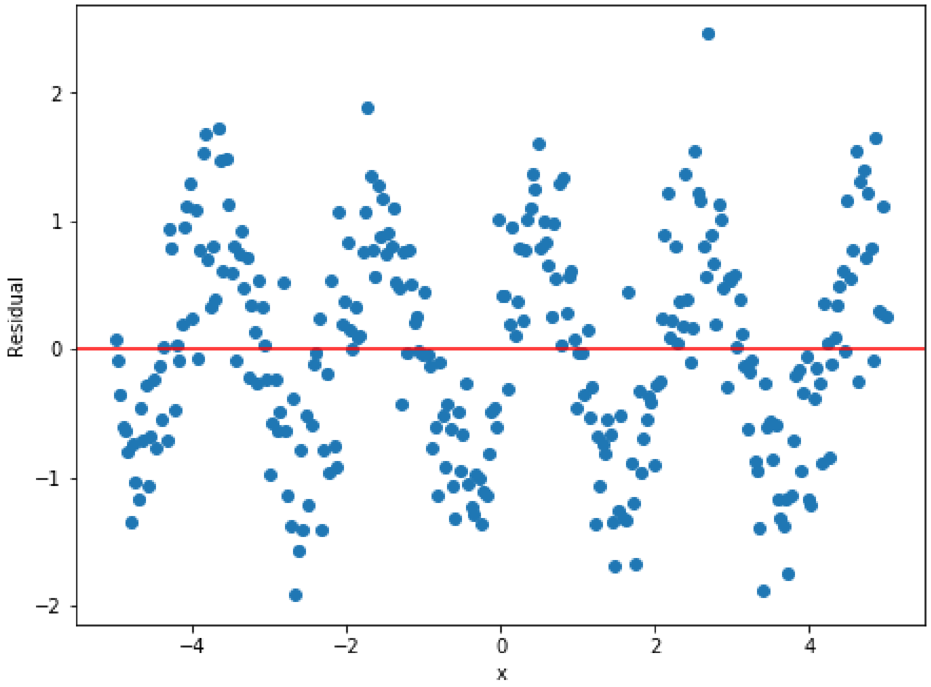


Question 2e

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.

In [16]:
```python
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
    """

    r = y-t_hat*x
    plt.figure(figsize=(8, 6))
    plt.scatter(x,r)
    plt.xlabel('x')
    plt.ylabel('Residual')
    plt.axhline(0,color='red')
    # YOUR CODE HERE
visualize_residual(x, y)
```



Question 2f

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

The residual looks like a sin/cos function graphic fluctuating up and down around y=0. Residual is kind of periodic with respect to x with some extra noise terms.

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 $\mathbf{y}$ (the y-values) using $\mathbf{x}$ (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
    """
    y = theta_1*x+np.sin(theta_2*x)
    # YOUR CODE HERE
    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.

$$L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

$$L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_1 x_i - sin(\theta_2 x_i))^2$$

$$\frac{\partial}{\partial \theta_1}L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{-2}{n}\sum_{i=1}^{n}[(y_i - \theta_1 x_i - sin(\theta_2 x_i))x_i]$$

$$\frac{\partial}{\partial \theta_2}L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{-2}{n}\sum_{i=1}^{n}[(y_i - \theta_1 x_i - sin(\theta_2 x_i))cos(\theta_2 x_i)x_i]$$

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
    """
    return np.mean((-2)*(y-theta[0]*x-np.sin(theta[1]*x))*x)
    # YOUR CODE HERE
```

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
    """
    return np.mean(-2*(y-theta[0]*x-np.sin(theta[1]*x))*np.cos(theta[1]*x)*x)
    # YOUR CODE HERE
```

In [21]:
```
# This function calls dt1 and dt2 and returns the gradient dt. It is already implemented for you.
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 [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:
$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha \left(\nabla_{\boldsymbol{\theta}} \mathbf{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}^{(t)})\right)$$

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

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

    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 descent
    loss_history -- the series of loss values over each iteration of gradient descent
    """
    theta_history = []
    loss_history = []
    for i in range(num_iter):
        theta = theta - alpha * dt(x, y, theta)
        theta_history.append(theta)
        loss = np.mean(np.power((y-theta[0]*x-np.sin(theta[1]*x)),2))
        loss_history.append(loss)

    # YOUR CODE HERE
    return theta, theta_history, loss_history
```

In [25]:
```python
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 them
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 our value
```

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 learning rate

    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 descent
    loss_history -- the series of loss values over each iteration of gradient descent
    """
    theta_history = []
    loss_history = []
    for i in range(num_iter):
        alpha_update = alpha/(i+1)
        theta = theta - alpha_update * dt(x, y, theta)
        theta_history.append(theta)
        loss = np.mean(np.power((y-theta[0]*x-np.sin(theta[1]*x)),2))
        loss_history.append(loss)

    # YOUR CODE HERE
    return theta, theta_history, loss_history
```

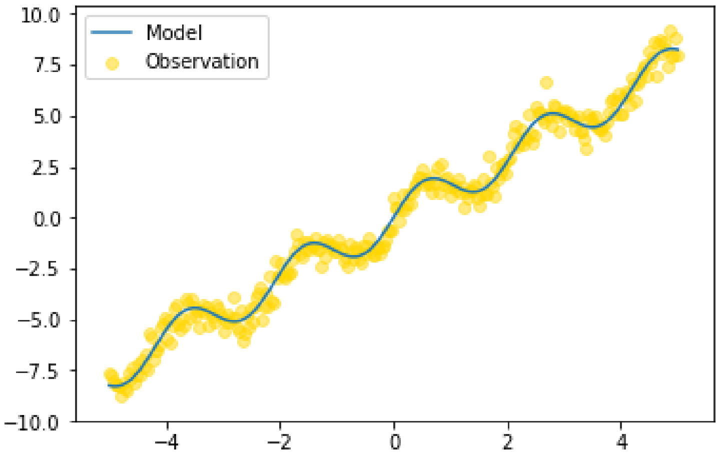
In [27]:
```python
t = init_t()
t_est_decay, ts_decay, loss_decay = grad_desc_decay(x, y, t, num_iter=20, alpha=0.1)

assert len(ts_decay) == len(loss_decay) == 20 # theta history and loss history are 20 items in them
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 close to our value
```

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]:
```
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();
```

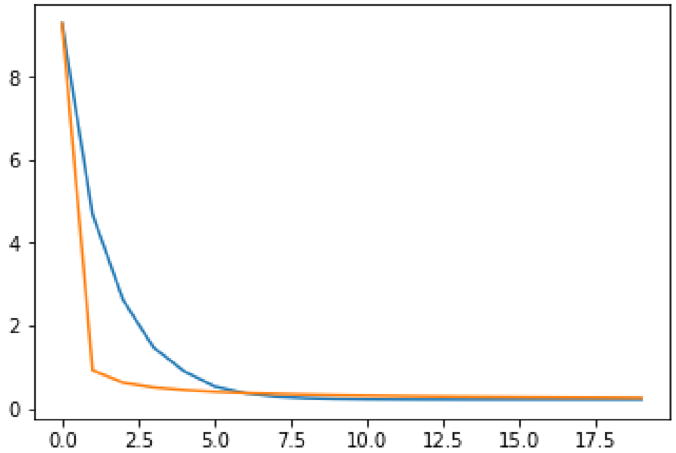


I think gradient descent successfully optimize θ because the model fits the observation quite well including the extra noise. Still, some outliers can not be fitted well.

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
plt.plot(loss)
plt.plot(loss_decay)
plt.show()
# YOUR CODE HERE
```



Question 4e

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

These two gradient descent methods converge to loss levels which are almost same. And the gradient descent method with decaying learning rate converges more quickly to steady state.

5: Visualizing Loss

Question 5:

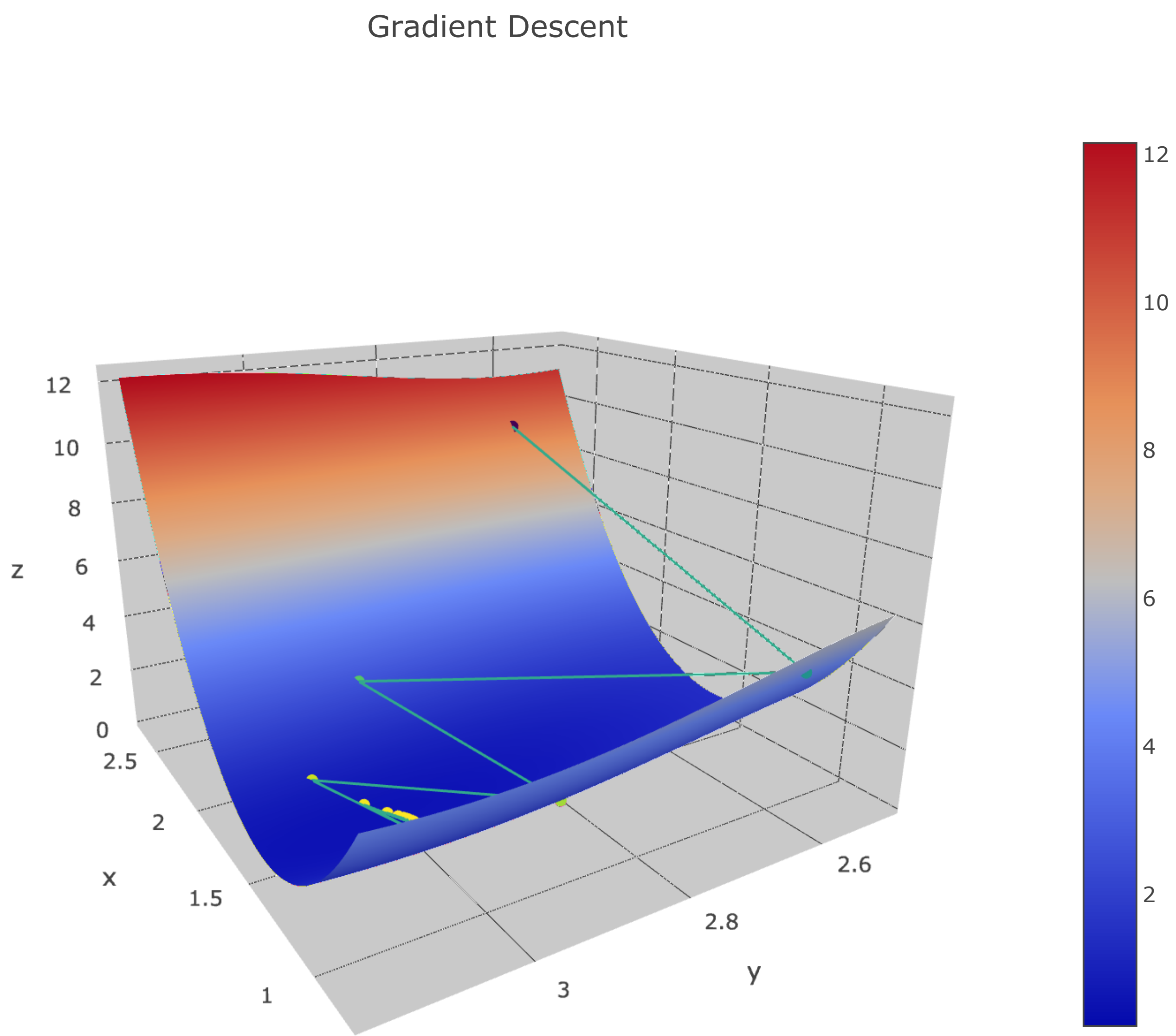
Let's visualize our loss functions and gain some insight as to how gradient descent and stochastic gradient descent are optimizing our model parameters.

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.

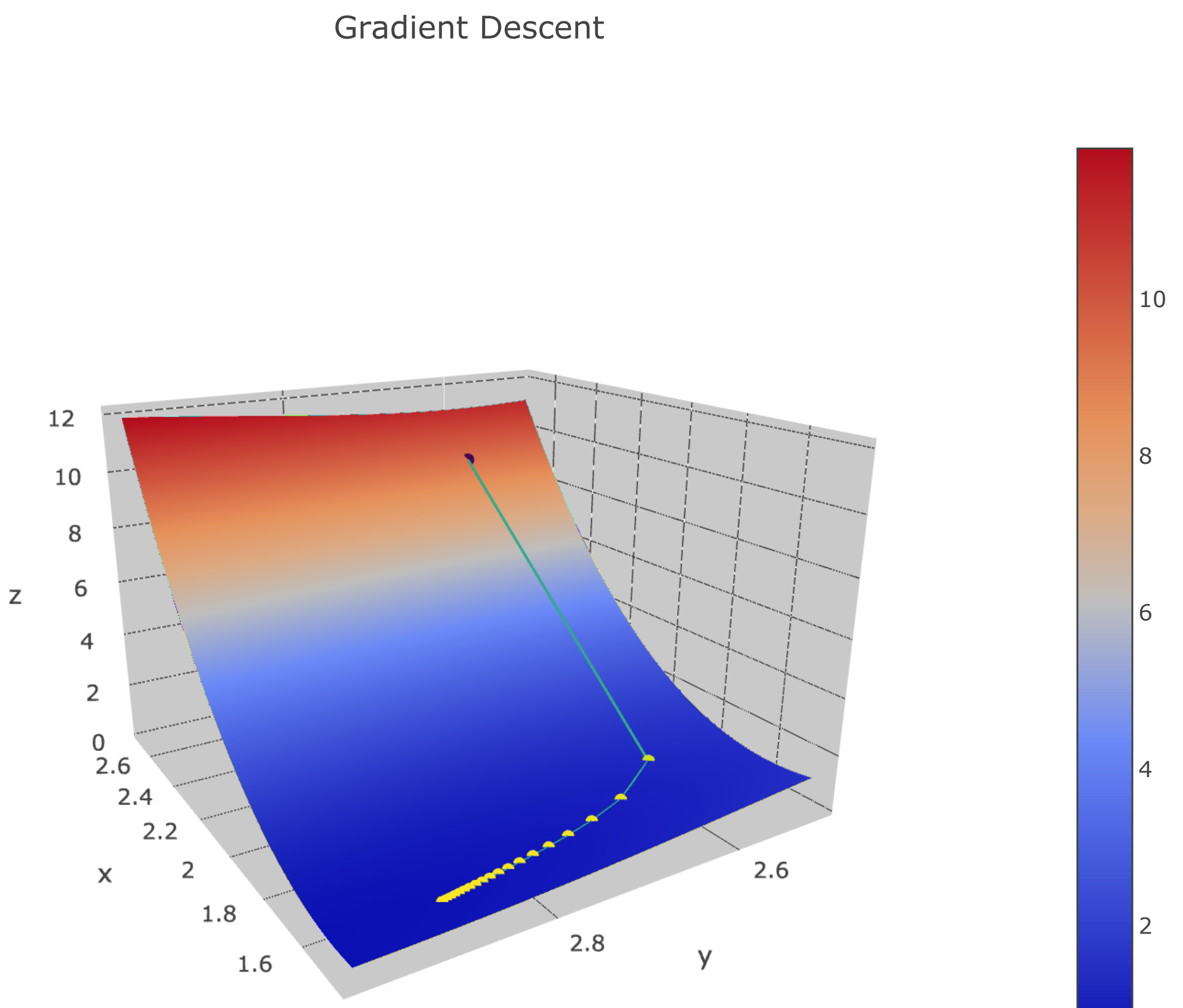
In [31]:
```
# Run me
ts = np.array(ts).squeeze()
ts_decay = np.array(ts_decay).squeeze()
loss = np.array(loss)
loss_decay = np.array(loss_decay)
```

In [32]:
```python
# 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)
```



Export to plot.ly »

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)
```



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.

The speed of convergence of the gradient descent method with a decaying learning rate is much higher than that of method with a static learning rate. The loss history plot shows directly that the gradient descent method with decaying learning rate converges more quickly to steady state. The two 3D visualization show that gradient descent method with a decaying learning rate converge in a zigzag path and gradient descent method with a static alpha converge in a nearly linear path after the first step. These lead to the inevitable fact that gradient descent method with a decaying learning rate converges at a faster rate.

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` .

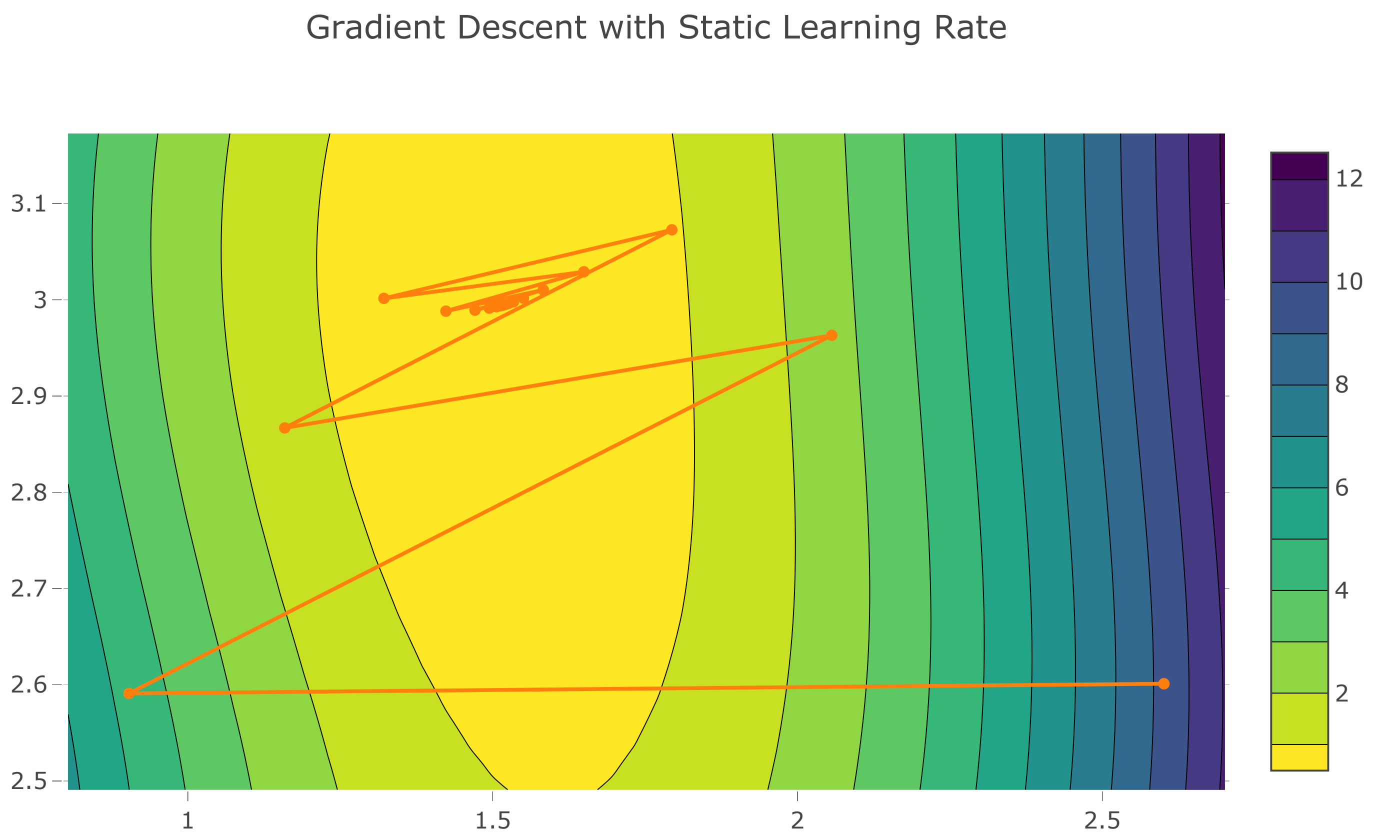
In next question, fill in the necessary part to create a contour plot. Then run the following cells.

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

In [35]:

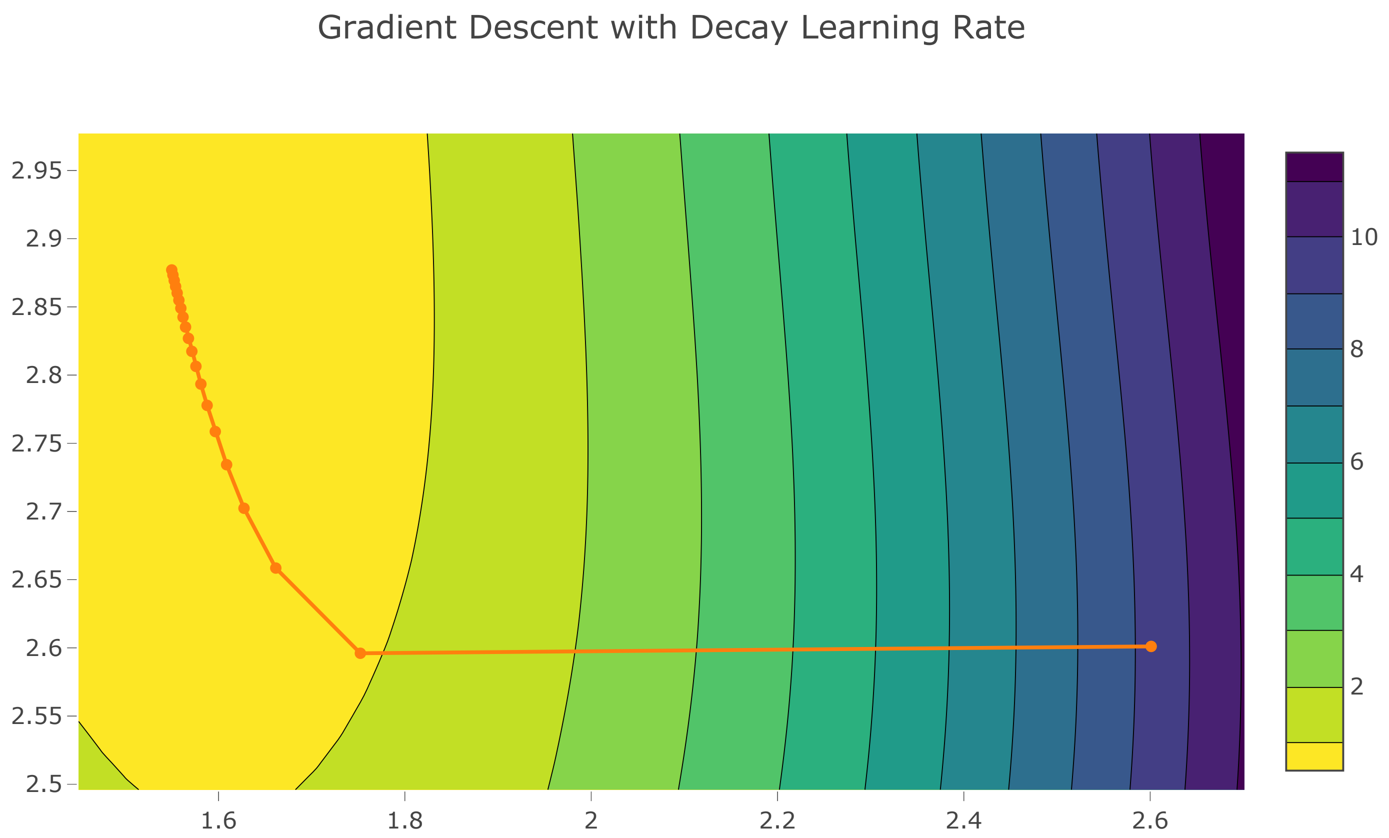
```python
def contour_plot(title, theta_history, loss_function, model, x, y):
    """
    The function takes the following as argument:
        theta_history: a (N, 2) array of theta history
        loss: a list or array of loss value
        loss_function: for example, l2_loss
        model: for example, 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
    thata_points = go.Scatter(name="Theta Values",
                              x=theta_1_series, #TODO
                              y=theta_2_series, #TODO
                              mode="lines+markers")

    ## 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
    lr_loss_contours = go.Contour(x=t1_s, #TODO
                                  y=t2_s, #TODO
                                  z=z, #TODO
                                  colorscale='Viridis', reversescale=True)

    # YOUR CODE HERE
    plotly.offline.iplot(go.Figure(data=[lr_loss_contours, thata_points], layout={'title': title}))
```

In [36]:
```
# Run this
contour_plot('Gradient Descent with Static Learning Rate', ts, l2_loss, sin_model, x, y)
```



Export to plot.ly »

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



Export to plot.ly »

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?

Interpretation: The color of background denotes the values of loss. The lighter color denotes the smaller loss corresponding a specific $\boldsymbol{\theta}$. And the orange line denotes the path of changing of loss along with the improvement on $\boldsymbol{\theta}$. Besides, each marker point on the orange line denotes each pair of $\boldsymbol{\theta}$ with x-coordinate corresponding value of θ_1 and y-coordinate corresponding value of θ_2.

Contour plot

Pros:

- Easy to read the values of θ_1 and θ_2 by reading the x-coordinate and y-coordinate of each marker point.
- The path of changing of loss is easy to interpret and imagine.
- Color denoting the values of loss is easy to distinguish for human's eye and clear to understand.

Cons:

- It is hard to read values of loss for each step.
- The last few optimization steps is hard to distinguish because of overplotting resulting from concentrated marker points and lines.

3D plot

Pros:

- Easy to examine the path of changing of loss in a three-dimensional way because almost none overlapping exists.
- Easy to imagine the decreasing of the loss because it is denoted by height of marker points in 3D plot.
- Able to change perspective to get more information of the changing path of loss.

Cons:

- Hard to read the values of θ_1, θ_2 and loss directly from the plot.

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?

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

It should have a better loss because the model mentioned in 3a is a special situation of the model above.

In [38]:
```
def dt1_new(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
    """
    return np.mean((-2)*(y-theta[0]*x-theta[2]*np.sin(theta[1]*x+theta[3]))*x)
    # YOUR CODE HERE

def dt2_new(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
    """
    return np.mean(-2*(y-theta[0]*x-theta[2]*np.sin(theta[1]*x+theta[3]))*theta[2]*np.cos(theta[1]*x+theta[3])*x)
    # YOUR CODE HERE

def dt_A(x, y, theta):
    """
    Compute the numerical value of the partial of l2 loss with respect to A

    Keyword arguments:
    x -- the vector of all x values
    y -- the vector of all y values
    theta -- the vector of values theta
    """
    return np.mean(-2*(y-theta[0]*x-theta[2]*np.sin(theta[1]*x+theta[3]))*np.sin(theta[1]*x+theta[3]))
    # YOUR CODE HERE

def dt_phi(x, y, theta):
    """
    Compute the numerical value of the partial of l2 loss with respect to phi

    Keyword arguments:
    x -- the vector of all x values
    y -- the vector of all y values
```

```python
    theta -- the vector of values theta
    """
    return np.mean(-2*(y-theta[0]*x-theta[2]*np.sin(theta[1]*x+theta[3]))*theta[2]*np.cos(theta[1]*x+theta[3]))
    # YOUR CODE HERE

def dt_up(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_new(x,y,theta), dt2_new(x,y,theta), dt_A(x, y, theta), dt_phi(x, y, theta)])

def grad_desc_decay_new(x, y, theta, num_iter=20, alpha=0.1):
    """
    Run gradient descent update for a finite number of iterations and decaying learning rate

    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 descent
    loss_history -- the series of loss values over each iteration of gradient descent
    """
    theta_history = []
    loss_history = []
    for i in range(num_iter):
        alpha_update = alpha/(i+1)
        theta = theta - alpha_update * dt_up(x, y, theta)
        theta_history.append(theta)
        loss = np.mean(np.power((y-theta[0]*x-theta[2]*np.sin(theta[1]*x+theta[3])),2))
        loss_history.append(loss)

    # YOUR CODE HERE
    return theta, theta_history, loss_history
```

```
t = np.array([0,0,1,0])
t_est_decay_new, ts_decay_new, loss_decay_new = grad_desc_decay_new(x, y, t, num_iter=20, alpha=0.1)
```

In [39]:
```
loss_decay_new[19]-loss_decay[19]
```

Out[39]:
```
-0.0006461720791709479
```

So the result above shows that the loss of new model is a little better than the two-parameter model. But it does not look like a obvious improvement, so the two-parameter model is quite enough to fit the original data.

Question 5d (optional)

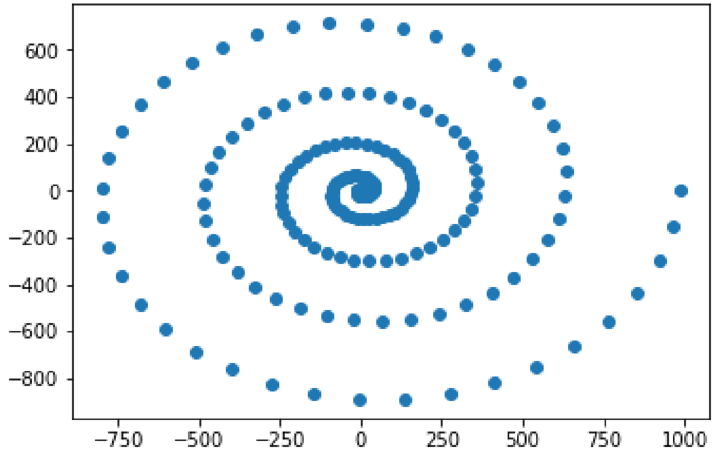
It looks like our basic two parameter model, a combination of a linear function and sinusoidal function, was able to almost perfectly fit our data. It turns out that many real world scenarios come from relatively simple models.

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 [40]:
```python
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()
```



I use polar transformation to model this data. We have

$r = t^2$

$x = rcos(t)$

$y = rsin(t)$

So, we can get the polar coordinates expression: $r = \theta^2$

where

$r = \sqrt{x^2 + y^2}$

$\theta = arctan(\frac{y}{x})$

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.

Proof:

Because f,g are convex function, we have:

$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$
$tg(a) + (1-t)g(b) \geq g(ta + (1-t)b)$
$\forall a, \forall b, t \in [0, 1]$

Then, because h(x)=f(x)+g(x), we have:

$$
\begin{aligned}
th(a) + (1-t)h(b) &= t(f(a) + g(a)) + (1-t)(f(b) + g(b)) \\
&= [tf(a) + (1-t)f(a)] + [tg(a) + (1-t)g(a)] \\
&\geq f(ta + (1-t)b) + g(ta + (1-t)b) \\
&= h(ta + (1-t)b)
\end{aligned}
$$

So, via the definition of convexity, the h is convex.

Mutlivariable/vector calculus mechanical problems

Question 6b

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

Proof:

$$
\begin{aligned}
L(\mathbf{w}) &= ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \\
&= (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= ((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\ \mathbf{w} - \mathbf{y}^T\mathbf{X}\ \mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}
\end{aligned}
$$

Because $\mathbf{y}^T\mathbf{X}\ \mathbf{w}$ and $\mathbf{w}^T\mathbf{X}^T\mathbf{y}$ are both numbers and as we all know, a number has the same value as its transpose.

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

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)](http://www.ds100.org/fa18/syllabus#lecture-week-6).

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

We need the solution for the optimal $\mathbf{w}$, so we let:
$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = 0$$

Then, we have:
$$\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} = 0$$
$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This does not mean that $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ minimizes the $L(\mathbf{w})$. We are supposed to prove it.

Let $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

for $\forall\mathbf{w}$, we have
$$\begin{aligned} L(\mathbf{w}) &= ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \\ &= ||\mathbf{y} - \mathbf{X}\hat{\mathbf{w}} + \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w})||_2^2 \\ &= ||\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}||_2^2 + (\hat{\mathbf{w}} - \mathbf{w})^T\mathbf{X}^T\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}) + 2(\hat{\mathbf{w}} - \mathbf{w})^T\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) \end{aligned}$$

Because $\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}^T\mathbf{y} = 0$, so we have $\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) = 0$

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

And $\mathbf{X}^T\mathbf{X}$ is a positive smemi-definite matrix, so:
$$(\hat{\mathbf{w}} - \mathbf{w})^T\mathbf{X}^T\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}) \geq 0$$

So we have:

$$||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \geq ||\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}||_2^2$$

Finally, we reach the conclusion that: $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ minimizes the $L(\mathbf{w})$

Question 6d

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

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

$$
\begin{aligned}
L(\mathbf{w}) &= ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_2^2 \\
&= (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^T\mathbf{w} \\
&= ((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^T\mathbf{w} \\
&= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\ \mathbf{w} - 2\mathbf{y}^T\mathbf{X}\ \mathbf{w} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

The derivative of loss function is:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{I}\mathbf{w}$$

Then, we have:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{I}\mathbf{w} = 0$$
$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y}$$
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

We can use similar method as mentioned in 6c to prove that:

$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$ minimizes the $L(\mathbf{w})$ of ridge regression.

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?

If $\mathbf{X}$ is not full rank, $\mathbf{X}^T\mathbf{X}$ is not invertible and there is no unique solution for $\mathbf{w}$.□ This problem does not occur with ridge regression, however. Because for any design matrix $\mathbf{X}$, the quantity $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ is always invertible; thus, there is always a unique solution for ridge regression.

Ridge regressions benefits from bias-variance trade-off. i.e. As λ increases, the flexibility of ridge regression coefficients decreases which means variance decreases but bias increases.

Pros

- Ridge regression can reduce the variance with an increasing bias and works best in situations where the OLS estimates have high variance.
- Can improve predictive performance.
- Works in situations where p<n.
- Mathematically simple computations.

Cons

- Ridge regression is not able to shrink coefficients to exactly zero.

- As a result, it can not perform variable selection.

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?

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

Question 6g

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

Proof:

$$\begin{aligned} Var(X-Y) &= \mathbb{E}[(X-Y)^2] - [\mathbb{E}(X-Y)]^2 \\ &= \mathbb{E}[(X^2 + Y^2 - 2XY)] - [\mathbb{E}(X) - \mathbb{E}(Y)]^2 \\ &= \mathbb{E}(X^2) + \mathbb{E}(Y^2) - 2\mathbb{E}(X)\mathbb{E}(Y) - [\mathbb{E}(X)]^2 - [\mathbb{E}(Y))]^2 + 2\mathbb{E}(X)\mathbb{E}(Y) \\ &= Var(X) + Var(Y) \end{aligned}$$

To get the conclusion, we only need to prove that $\mathbb{E}(XY) = \mathbb{E}(X)\mathbb{E}(Y)$ when random variables X and Y are independent.

$$\begin{aligned} \mathbb{E}(XY) &= \sum_{x\in\mathbb{X}}\sum_{y\in\mathbb{Y}} P(x,y)xy \\ Independence \quad &= \sum_{x\in\mathbb{X}}\sum_{y\in\mathbb{Y}} P(X=x)P(Y=y)xy \\ &= \sum_{x\in\mathbb{X}} xP(X=x)\sum_{y\in\mathbb{Y}} P(Y=y)y \\ &= \mathbb{E}(X)\mathbb{E}(Y) \end{aligned}$$

Finally, we reach the conclusion that $Var(X - Y) = Var(X) + Var(Y)$ when random variables X and Y are independent.

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 [41]:

```
regxa = r"^(abc).*" # fill in your pattern
# YOUR CODE HERE
```

In [42]:
```
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 [43]:
```
regxb = r"[cmf]{1}(an)" # fill in your pattern
# YOUR CODE HERE
```

In [44]:
```
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 [45]:
```
text_qc = "I've got 10 eggs that I stole from 20 gooses belonging to 30 giants."

res_qc = re.findall(pattern=r"\d+\s[a-z]*",string=text_qc)
# YOUR CODE HERE

res_qc
```

Out[45]:
```
['10 eggs', '20 gooses', '30 giants']
```

In [46]:
```
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 [47]:
```
text_qd = 'Python Exercises, PHP exercises.'
res_qd = re.sub(pattern='[\s,.]',string=text_qd,repl=':',count=2) # Hint: use re.sub()
# YOUR CODE HERE

res_qd
```

Out[47]:
```
'Python:Exercises: PHP exercises.'
```

In [48]:
```
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 [49]:
```
text_qe = 'this is a word mushroom mushroom'
res_qe = re.sub(pattern=r"\b(?!mushroom)\b\S+",string=text_qe,repl='badger') # Hint: https://www.regextester.com/94017
# YOUR CODE HERE
res_qe
```

Out[49]:
```
'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