Gradient descent in depth

Fraida Fund

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
import seaborn as sns
sns.set()
colors = sns.color_palette("hls", len(w_true))

# for 3d interactive plots
from ipywidgets import interact, fixed, widgets
from mpl_toolkits import mplot3d

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-7291d55ec314> in <module>
      5 import seaborn as sns
      6 sns.set()
----> 7 colors = sns.color_palette("hls", len(w_true))
      8
      9 # for 3d interactive plots

NameError: name 'w_true' is not defined
```

Gradient descent for simple linear regression

Generate data

```
def generate_linear_regression_data(n=100, d=1, coef=[5], intercept=1, sigma=0):
  x = np.random.randn(n,d)
  y = (np.dot(x, coef) + intercept).squeeze() + sigma * np.random.randn(n)
  return x, y
```

```
# y = 2 + 3x
w_true = np.array([2, 3])
n_samples = 100
```

```
x, y = generate_linear_regression_data(n=n_samples, d=1, coef=w_true[1:], intercept=
    w_true[0])
```

```
# create the "design matrix" with a ones column at beginning
X = np.hstack((np.ones((n_samples, 1)), x))
X.shape
```

```
(100, 2)
```

Define a descent step

In each gradient descent step, we will compute

$$w^{t+1} = w^t - \alpha^t \nabla L(w^t)$$

With a mean squared error loss function

$$\begin{aligned} L(w) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \langle w, x_i \rangle)^2 \\ &= \frac{1}{n}\|y - Xw\|^2 \end{aligned}$$

we will compute the weights at each step as

$$\begin{aligned} w^{t+1} &= w^t + \frac{\alpha^t}{n}\sum_{i=1}^{n}(y_i - \langle w^t, x_i \rangle)x_i \\ &= w^t + \frac{\alpha^t}{n}X^T(y - Xw^t) \end{aligned}$$

```
def gd_step(w, X, y, lr):
  # use current parameters to get y_hat
  y_hat = np.dot(X,w)
  error = y_hat-y
  # compute gradient for this y_hat
  grad = np.matmul(X.T, error)
  # update weights
  w_new = w - (lr/X.shape[0])*grad

  # we don't have to actually compute MSE
  # but I want to, for visualization
  mse = np.mean(error**2, axis=0)

  return (w_new, mse, grad)
```

Perform gradient descent

```
# gradient descent settings: number of iterations, learning rate, starting point
itr = 50
lr = 0.1
w_init = np.random.randn(len(w_true))
print(w_init)
```

```
[-0.27657397  0.92304132]
```

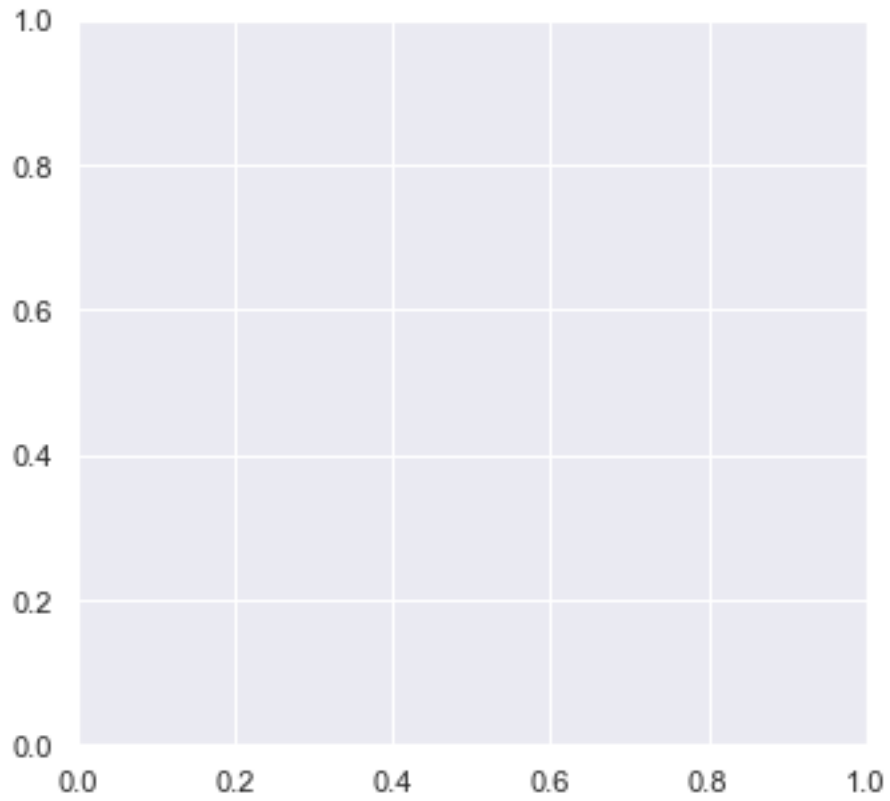
```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)
grad_steps = np.zeros((itr, len(w_init)))
```

```
w_star = w_init
for i in range(itr):
  w_star, mse, gradient = gd_step(w_star, X, y, lr)
  w_steps[i] = w_star
  mse_steps[i] = mse
  grad_steps[i] = gradient
```

Visualize

```
plt.figure(figsize=(18,5))
plt.title("After %d iterations with rate %f: %s" % (itr, lr, w_star))
plt.subplot(1,3,1);

for n in range(len(w_true)):
  plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);

plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");

plt.subplot(1,3, 2);
sns.lineplot(x=np.arange(itr), y=mse_steps);
#plt.yscale("log")
plt.xlabel("Iteration");
plt.ylabel("Mean Squared Error");


plt.subplot(1, 3, 3);
for n in range(len(coef)+1):
  sns.lineplot(x=np.arange(itr), y=grad_steps[:,n], color=colors[n]);
plt.xlabel("Iteration");
plt.ylabel("Gradient");

plt.suptitle("Estimate after %d iterations with rate %s: %s" % (itr, "{0:0.4f}".format(lr),
    ["{0:0.4f}".format(w) for w in w_star]));
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-75f07a4a60b9> in <module>
      4
      5 for n in range(len(w_true)):
----> 6   plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
      7   sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);
      8

NameError: name 'colors' is not defined
```



Other things to try

- What happens if we increase the learning rate?
- What happens if we decrease the learning rate?

Descent path on MSE contour

MSE contour

```
coefs = np.arange(2, 8, 0.05)

coef_grid = np.array(np.meshgrid(coefs, coefs)).reshape(1, 2, coefs.shape[0], coefs.shape[0])
y_hat_c = (intercept + np.sum(coef_grid * x.reshape(x.shape[0], 2, 1, 1), axis=1) )
mses_coefs = np.mean((y.reshape(-1, 1, 1)- y_hat_c)**2,axis=0)

plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs)
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-af86d78a9cbc> in <module>
      2
```

```
      3 coef_grid = np.array(np.meshgrid(coefs, coefs)).reshape(1, 2, coefs.shape[0],
          coefs.shape[0])
----> 4 y_hat_c = (intercept + np.sum(coef_grid * x.reshape(x.shape[0], 2, 1, 1), axis=1) )
      5 mses_coefs = np.mean((y.reshape(-1, 1, 1)- y_hat_c)**2,axis=0)
      6

NameError: name 'intercept' is not defined
```

```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', sort=False, alpha=0.5);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
plt.title("Estimate after %d iterations with rate %s: %s" % (itr, "{0:0.4f}".format(lr),
    ["{0:0.4f}".format(w) for w in w_star]));
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-c9d13ff246c6> in <module>
      1 plt.figure(figsize=(5,5));
      2 X1, X2 = np.meshgrid(coefs, coefs);
----> 3 p = plt.contour(X1, X2, mses_coefs, levels=5);
      4 plt.clabel(p, inline=1, fontsize=10);
      5 plt.xlabel('w1');

NameError: name 'mses_coefs' is not defined
```

```
<Figure size 360x360 with 0 Axes>
```

Stochastic gradient descent

For stochastic gradient descent, we will compute the gradient and update the weights using one sample (or a mini-batch of samples) in each step.

A note on sampling: In practice, the samples are often sampled without replacement, but the statistical guarantee of convergence is for sampling with replacement. In this example, we sample with replacement. You can read more about different varieties of gradient descent and stochastic gradient descent in How is stochastic gradient descent implemented in the context of machine learning and deep learning.

Define a descent step

```
def sgd_step(w, X, y, lr, n):

  idx_sample = np.random.choice(X.shape[0], n, replace=True)

  X_sample = X[idx_sample, :]
  y_sample = y[idx_sample]

  # use current parameters to get y_hat
```

```
  y_hat = np.dot(X_sample,w)
  error = y_hat-y_sample
  # compute gradient for this y_hat
  grad = np.matmul(X_sample.T, error)
  # update weights
  w_new = w - (lr/n)*grad

  # we don't have to actually compute MSE
  # but I want to, for visualization
  # note: MSE is computed on entire data, not sample
  mse = np.mean((y-np.dot(X, w))**2, axis=0)

  return (w_new, mse, grad)
```

Perform gradient descent

```
itr = 50
lr = 0.1
n = 1
w_init = [intercept, 2, 8]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-fb812a62142d> in <module>
      2 lr = 0.1
      3 n = 1
----> 4 w_init = [intercept, 2, 8]

NameError: name 'intercept' is not defined
```

```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)

w_star = w_init
for i in range(itr):
  w_star, mse, grad = sgd_step(w_star, X, y, lr, n)
  w_steps[i] = w_star
  mse_steps[i] = mse
```

```
w_star
```

```
array([1.99439742, 2.99921534])
```

Visualize

```
colors = sns.color_palette("hls", len(coef) + 1)

plt.axhline(y=intercept, linestyle='--', color=colors[0]);
sns.lineplot(x=np.arange(itr), y=w_steps[:,0], color=colors[0]);

for n in range(len(coef)):
```

```
  plt.axhline(y=coef[n], linestyle='--', color=colors[n+1]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n+1], color=colors[n+1]);

plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-f12978862262> in <module>
----> 1 colors = sns.color_palette("hls", len(coef) + 1)
      2
      3 plt.axhline(y=intercept, linestyle='--', color=colors[0]);
      4 sns.lineplot(x=np.arange(itr), y=w_steps[:,0], color=colors[0]);
      5

NameError: name 'coef' is not defined
```

```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', sort=False, alpha=0.5);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-b527b063ebd3> in <module>
      1 plt.figure(figsize=(5,5));
      2 X1, X2 = np.meshgrid(coefs, coefs);
----> 3 p = plt.contour(X1, X2, mses_coefs, levels=5);
      4 plt.clabel(p, inline=1, fontsize=10);
      5 plt.xlabel('w1');

NameError: name 'mses_coefs' is not defined
```

```
<Figure size 360x360 with 0 Axes>
```

Other things to try

- Increase learning rate?
- Decrease learning rate?
- Use decaying learning rate $\alpha^t = \frac{C}{t}$?
- Increase number of samples used in each iteration?

Gradient descent with noise

Generate data

This time, we will use the `sigma` argument in our `generate_linear_regression_data` function to generate data that does not perfectly fit a linear model.

```
w_true = [2, 6, 5]
intercept = w_true[0]
coef = w_true[1:]
print(intercept, coef)
```

```
2 [6, 5]
```

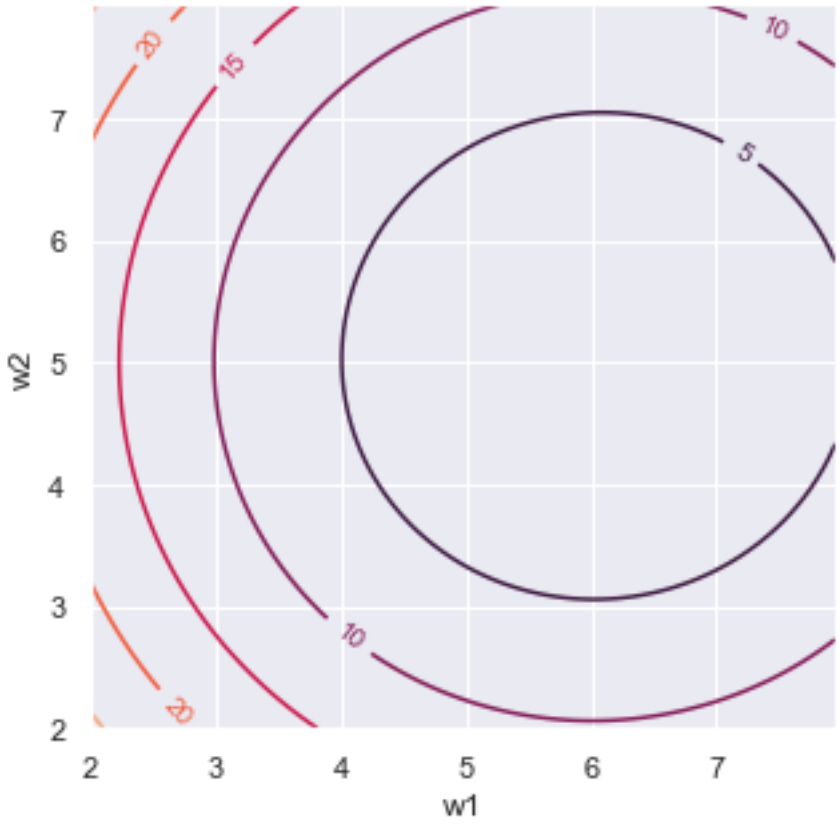
```
n_samples = 1000
```

```
x, y = generate_linear_regression_data(n=n_samples, d=2, coef=coef, intercept=intercept,
    sigma=1)
```

MSE contour

```
coefs = np.arange(2, 8, 0.05)

coef_grid = np.array(np.meshgrid(coefs, coefs)).reshape(1, 2, coefs.shape[0], coefs.shape[0])
y_hat_c = (intercept + np.sum(coef_grid * x.reshape(x.shape[0], 2, 1, 1), axis=1) )
mses_coefs = np.mean((y.reshape(-1, 1, 1)- y_hat_c)**2,axis=0)
```

```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs)
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
```



Perform gradient descent

This time, the gradient descent may not necessarily arrive at the "true" coefficient values. That's not because it does not find the coefficients with minimum MSE; it's because the coefficients with minimum MSE on the noisy training data are not necessarily the "true" coefficients.

```
X = np.column_stack((np.ones((n_samples, 1)), x))
X.shape
```

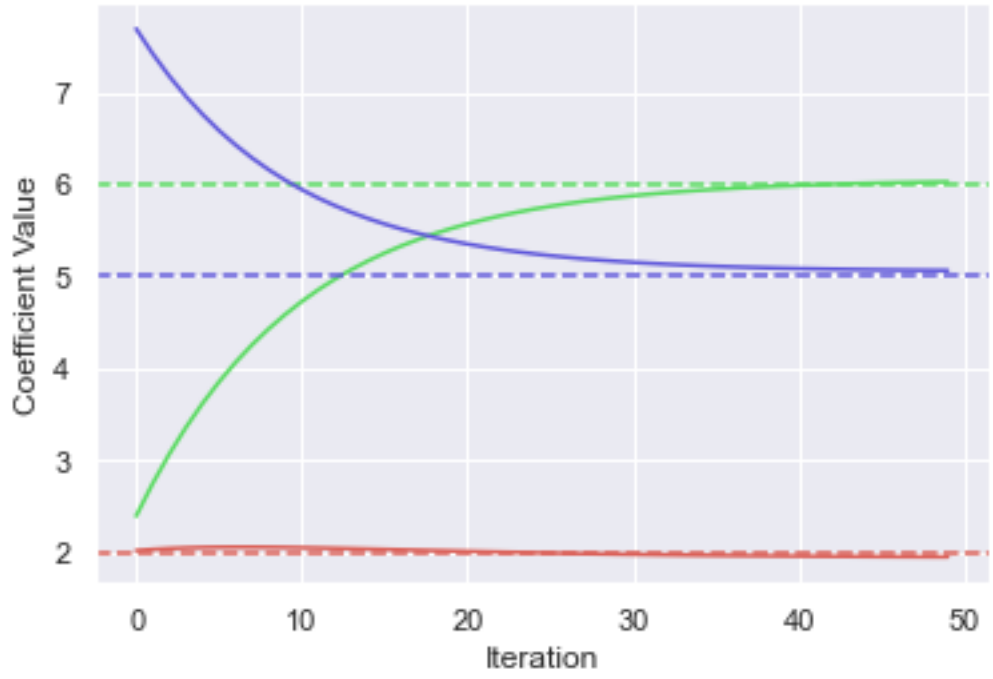
```
(1000, 3)
```

```
itr = 50
lr = 0.1
w_init = [intercept, 2, 8]
```

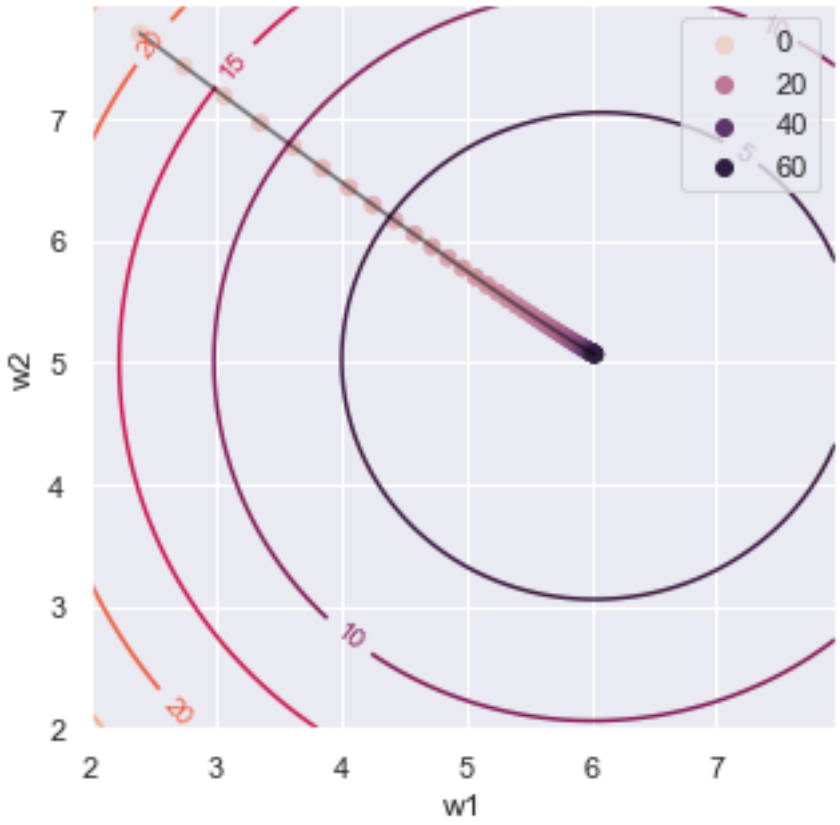
```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)

w_star = w_init
for i in range(itr):
  w_star, mse, gradient = gd_step(w_star, X, y, lr)
  w_steps[i] = w_star
  mse_steps[i] = mse
```

Visualize gradient descent

```
colors = sns.color_palette("hls", len(w_true))

for n in range(len(w_true)):
  plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);

plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");
```




```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', sort=False, alpha=0.5);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
```




```
w_star
```

```
array([1.94835177, 6.0304585 , 5.06558084])
```

```
def plot_3D(elev=20, azim=-20, X1=X1, X2=X2, mses_coefs=mses_coefs,
            w_steps=w_steps, mse_steps=mse_steps):

    plt.figure(figsize=(10,10))
    ax = plt.subplot(projection='3d')


    # Plot the surface.
    ax.plot_surface(X1, X2, mses_coefs, alpha=0.5, cmap=cm.coolwarm,
                          linewidth=0, antialiased=False)
    ax.scatter3D(w_steps[:, 1], w_steps[:, 2], mse_steps, s=5, color='black')
    ax.plot(w_steps[:, 1], w_steps[:, 2], mse_steps, color='gray')


    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('w1')
    ax.set_ylabel('w2')
    ax.set_zlabel('MSE')

interact(plot_3D, elev=widgets.IntSlider(min=-90, max=90, step=10, value=20),
          azim=widgets.IntSlider(min=-90, max=90, step=10, value=20),
         X1=fixed(X1), X2=fixed(X2), mses_coefs=fixed(mses_coefs),
```

```
         w_steps=fixed(w_steps), mse_steps=fixed(mse_steps));
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-f98d9610c682> in <module>
     18     ax.set_zlabel('MSE')
     19
---> 20 interact(plot_3D, elev=widgets.IntSlider(min=-90, max=90, step=10, value=20),
     21           azim=widgets.IntSlider(min=-90, max=90, step=10, value=20),
     22          X1=fixed(X1), X2=fixed(X2), mses_coefs=fixed(mses_coefs),

NameError: name 'interact' is not defined
```

Perform stochastic gradient descent

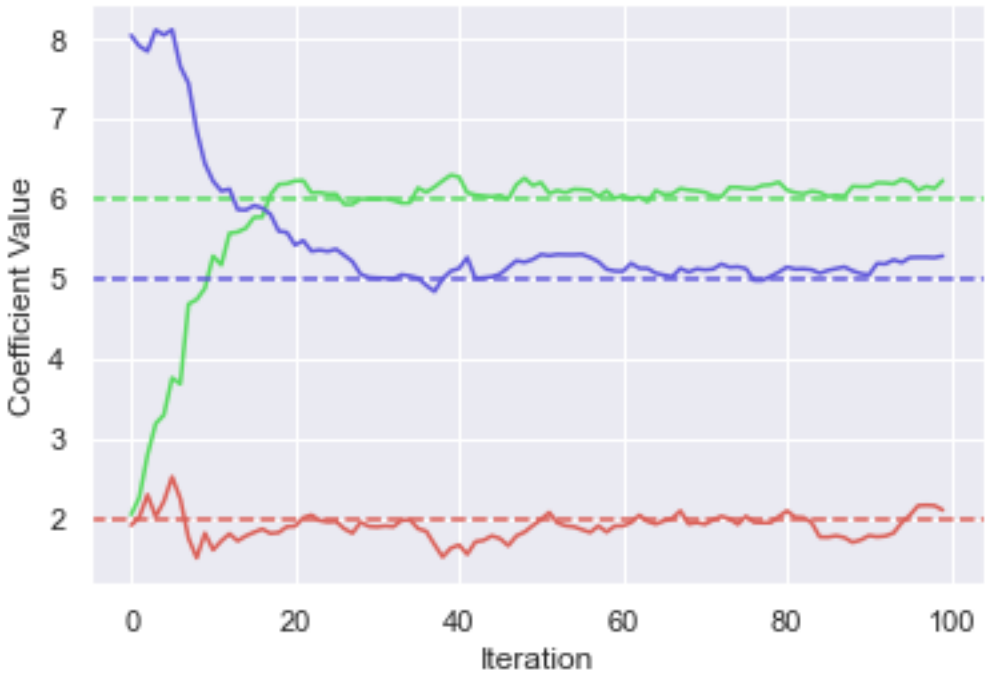
With data that does not perfectly fit the linear model, the stochastic gradient descent converges to a "noise ball" around the optimal solution.

```
itr = 100
lr = 0.1
w_init = [intercept, 2, 8]
```

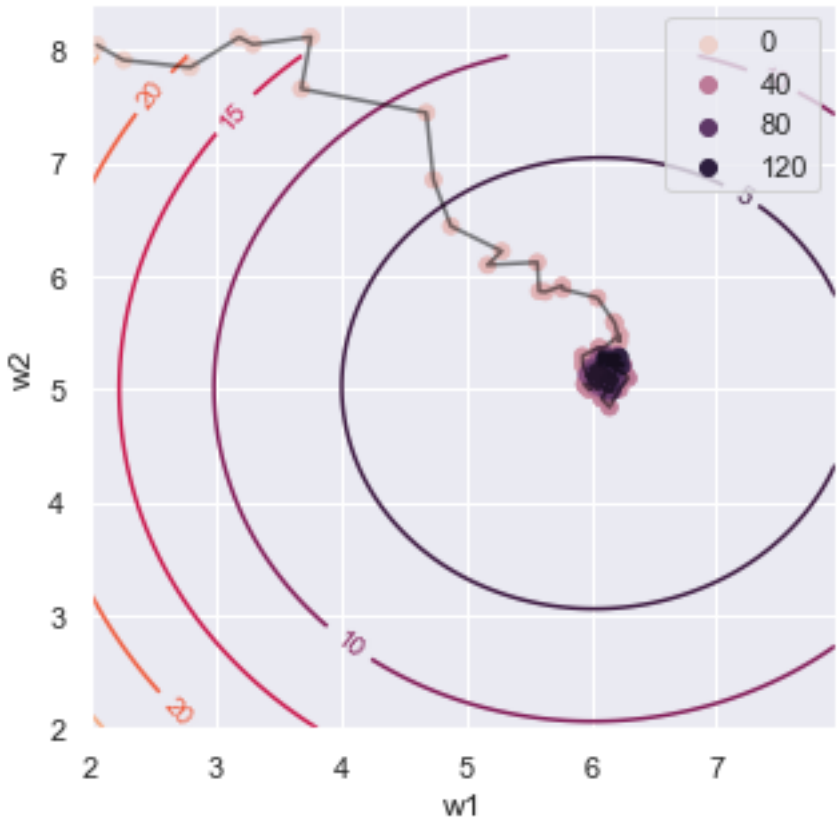
```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)

w_star = w_init
for i in range(itr):
  w_star, mse, grad = sgd_step(w_star, X, y, lr, n)
  w_steps[i] = w_star
  mse_steps[i] = mse
```

Visualize stochastic gradient descent

```
colors = sns.color_palette("hls", len(w_true))

for n in range(len(w_true)):
  plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);

plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");
```




```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', sort=False, alpha=0.5);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
```




```
w_star
```

```
array([2.10595877, 6.22858996, 5.285646  ])
```

```
def plot_3D(elev=20, azim=-20, X1=X1, X2=X2, mses_coefs=mses_coefs,
            w_steps=w_steps, mse_steps=mse_steps):

    plt.figure(figsize=(10,10))
    ax = plt.subplot(projection='3d')


    # Plot the surface.
    ax.plot_surface(X1, X2, mses_coefs, alpha=0.5, cmap=cm.coolwarm,
                          linewidth=0, antialiased=False)
    ax.scatter3D(w_steps[:, 1], w_steps[:, 2], mse_steps, s=5, color='black')
    ax.plot(w_steps[:, 1], w_steps[:, 2], mse_steps, color='gray')


    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('w1')
    ax.set_ylabel('w2')
    ax.set_zlabel('MSE')

interact(plot_3D, elev=widgets.IntSlider(min=-90, max=90, step=10, value=20),
          azim=widgets.IntSlider(min=-90, max=90, step=10, value=20),
         X1=fixed(X1), X2=fixed(X2), mses_coefs=fixed(mses_coefs),
```

```
         w_steps=fixed(w_steps), mse_steps=fixed(mse_steps));
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-f98d9610c682> in <module>
     18     ax.set_zlabel('MSE')
     19
---> 20 interact(plot_3D, elev=widgets.IntSlider(min=-90, max=90, step=10, value=20),
     21           azim=widgets.IntSlider(min=-90, max=90, step=10, value=20),
     22          X1=fixed(X1), X2=fixed(X2), mses_coefs=fixed(mses_coefs),

NameError: name 'interact' is not defined
```

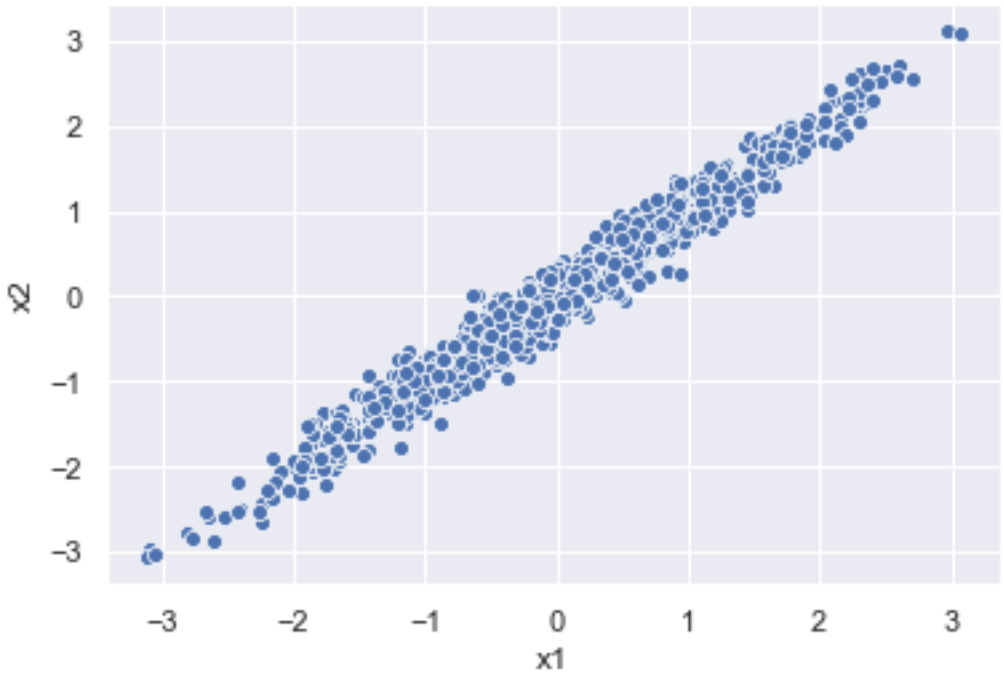
A less friendly loss surface

```
w_true = [2, 5, 4]
intercept = w_true[0]
coef = w_true[1:]
print(intercept, coef)
```

```
2 [5, 4]
```

```
n_samples = 1000
d = 1
sigma = 1

x1 = np.random.randn(n_samples,d)
x2 = x1 + (sigma/5)*np.random.randn(n_samples,1)
x = np.column_stack([x1, x2])
y = (np.dot(x, coef) + intercept).squeeze() + sigma * np.random.randn(n_samples)


X = np.column_stack((np.ones((n_samples, 1)), x))
```

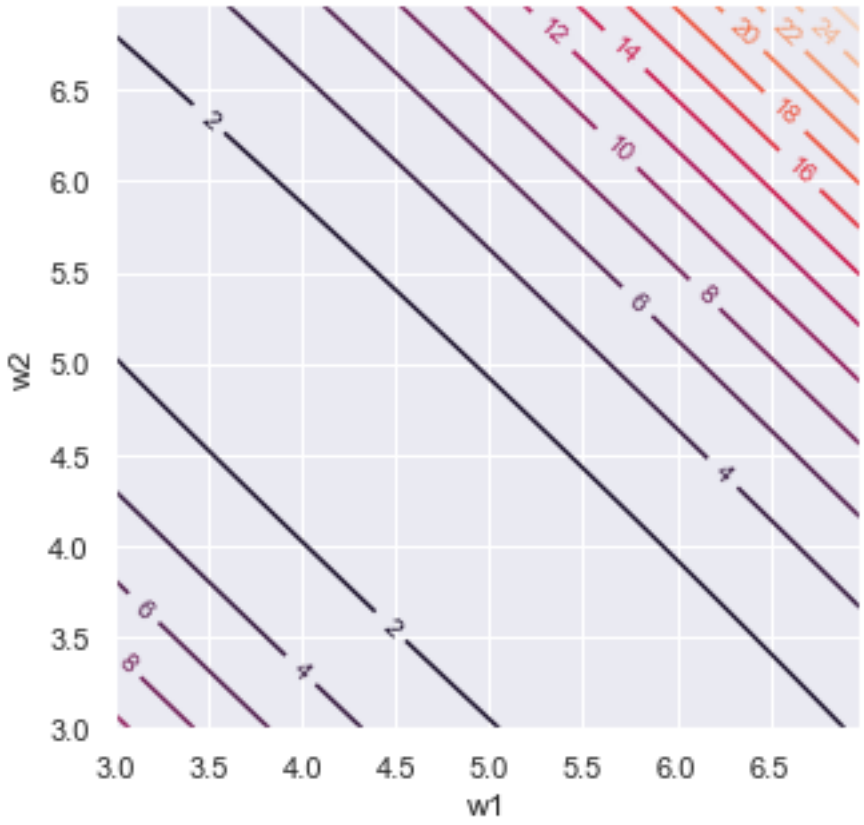
```
sns.scatterplot(x=x1.squeeze(), y=x2.squeeze());
plt.xlabel('x1');
plt.ylabel('x2');
```



MSE contour

```
coefs = np.arange(3, 7, 0.02)

coef_grid = np.array(np.meshgrid(coefs, coefs)).reshape(1, 2, coefs.shape[0], coefs.shape[0])
y_hat_c = (intercept + np.sum(coef_grid * x.reshape(x.shape[0], 2, 1, 1), axis=1) )
mses_coefs = np.mean((y.reshape(-1, 1, 1)- y_hat_c)**2,axis=0)
```

```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs)
p = plt.contour(X1, X2, mses_coefs, levels=15);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
```



Perform gradient descent

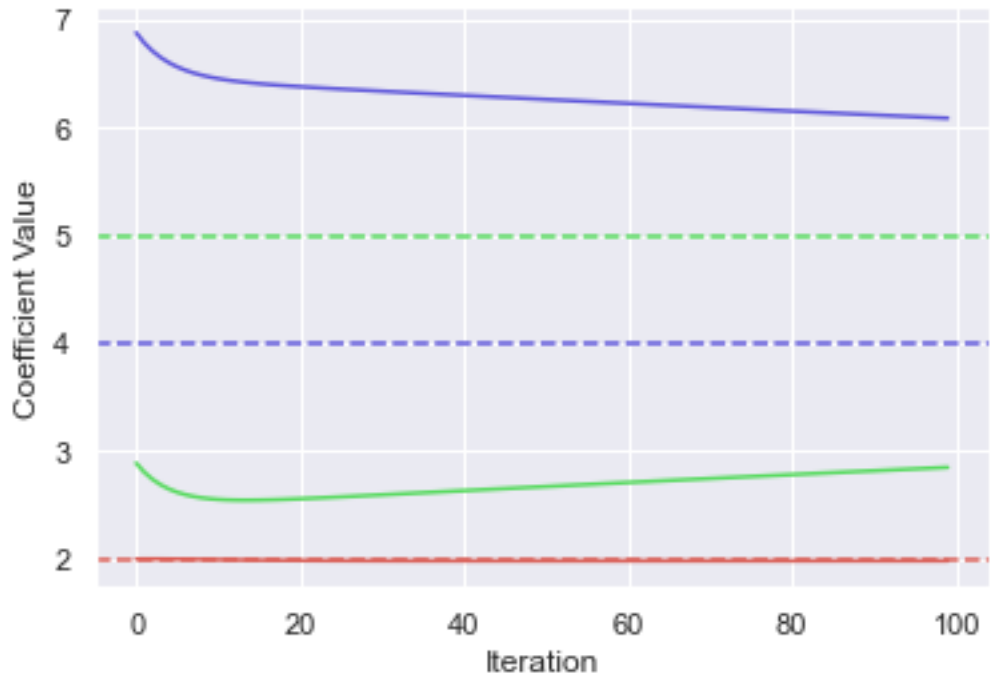
```
itr = 100
lr = 0.1
w_init = [intercept, 3, 7]
```

```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)
grad_steps = np.zeros((itr, len(w_init)))

w_star = w_init
for i in range(itr):
  w_star, mse, gradient = gd_step(w_star, X, y, lr)
  w_steps[i] = w_star
  mse_steps[i] = mse
  grad_steps[i] = gradient
```

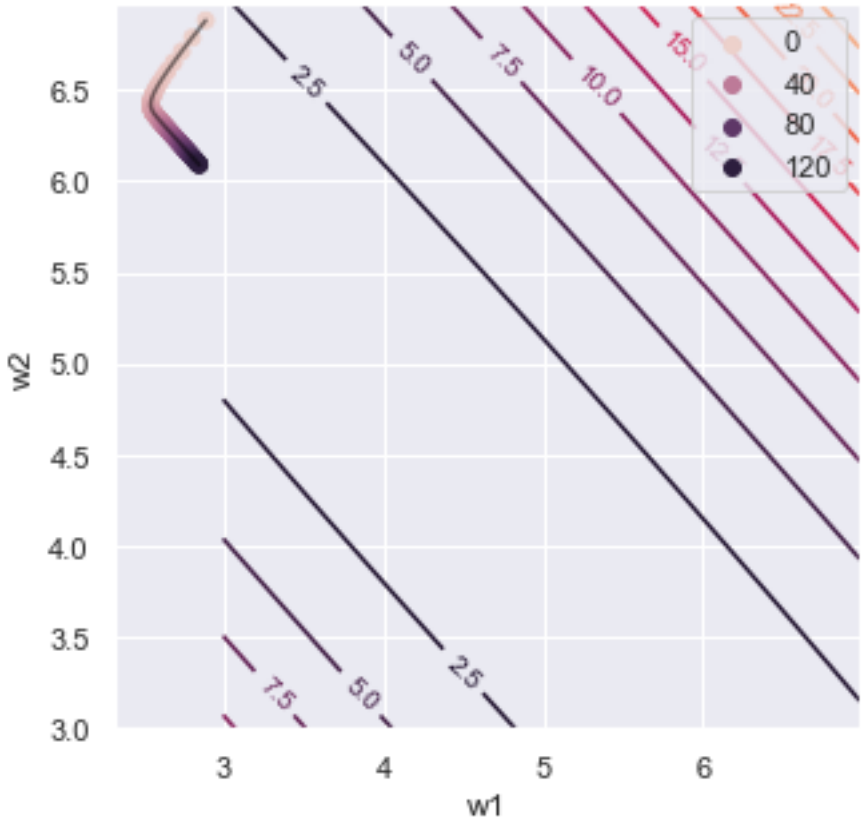
Visualize gradient descent

```
colors = sns.color_palette("hls", len(w_true))

for n in range(len(w_true)):
  plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);
```

```
plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");
```




```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=10);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', alpha=0.5, sort=False);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
```



Momentum

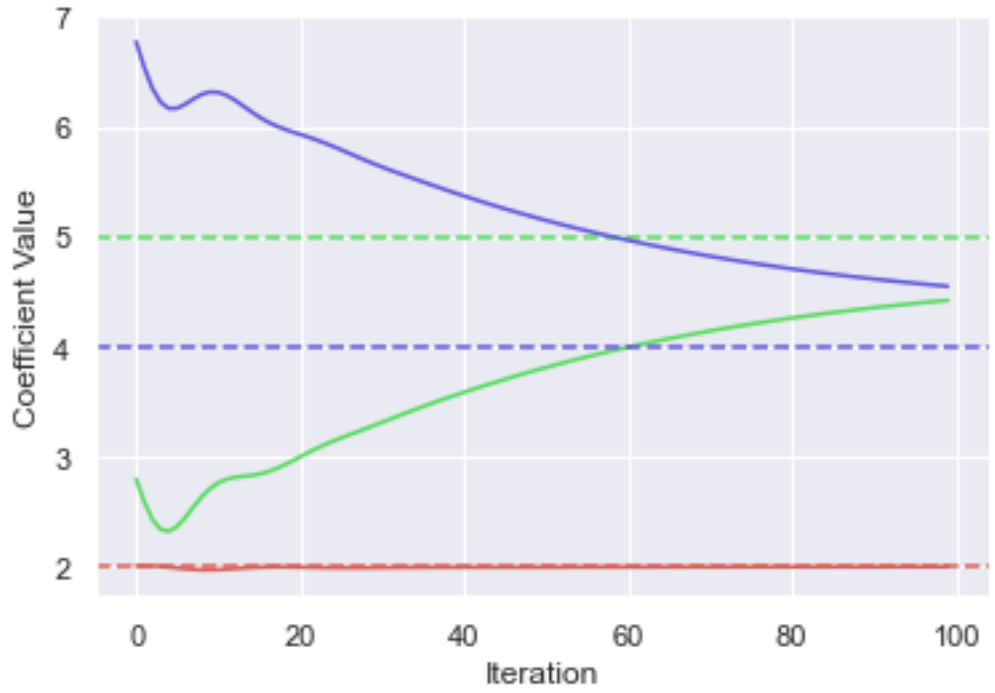
```
def gd_step_momentum(w, X, y, lr, eta, v):
  # use current parameters to get y_hat, error
  y_hat = np.dot(X,w)
  error = y_hat-y
  # compute gradient and velocity
  grad = np.matmul(X.T, error)
  v_new = eta*v - (lr/X.shape[0])*grad
  # update weights
  w_new = w - (lr/X.shape[0])*grad + eta*v_new

  # we don't have to actually compute MSE
  # but I want to, for visualization
  mse = np.mean(error**2, axis=0)

  return (w_new, mse, grad, v_new)
```

```
itr = 100
lr = 0.1
eta = 0.9
w_init = [intercept, 3, 7]
```

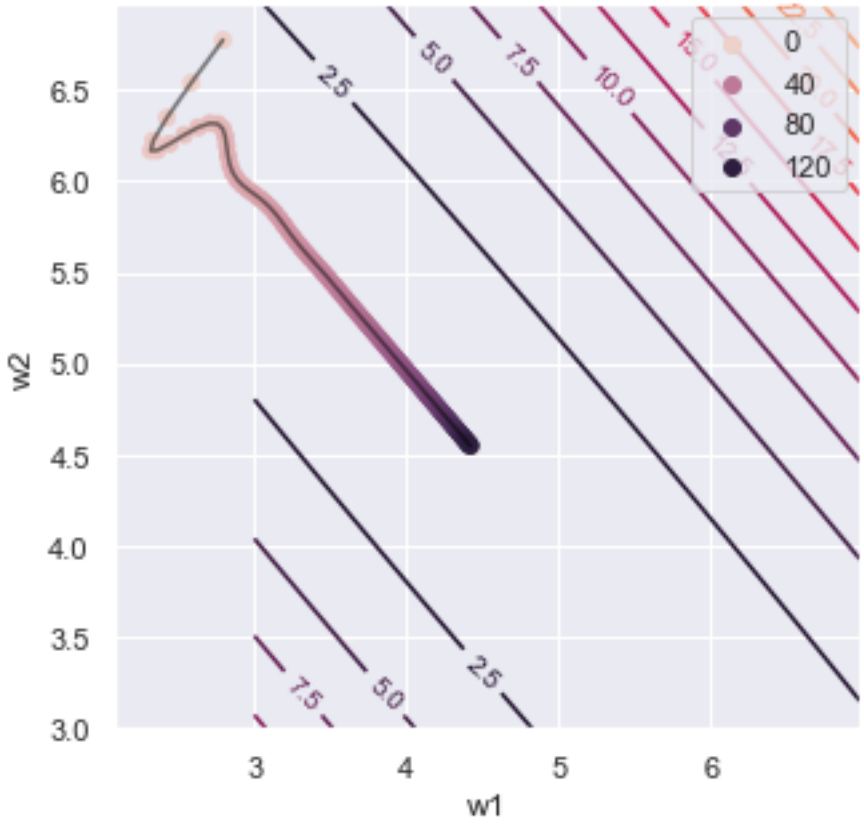
```
w_steps = np.zeros((itr, len(w_init)))
mse_steps = np.zeros(itr)
grad_steps = np.zeros((itr, len(w_init)))
```

```
v_steps = np.zeros((itr, len(w_init)))

w_star = w_init
velocity = np.zeros(len(w_init))
for i in range(itr):
  w_star, mse, gradient, velocity = gd_step_momentum(w_star, X, y, lr, eta, velocity)
  w_steps[i] = w_star
  mse_steps[i] = mse
  grad_steps[i] = gradient
  v_steps[i] = velocity
```

```
colors = sns.color_palette("hls", len(w_true))

for n in range(len(w_true)):
  plt.axhline(y=w_true[n], linestyle='--', color=colors[n]);
  sns.lineplot(x=np.arange(itr), y=w_steps[:,n], color=colors[n]);

plt.xlabel("Iteration");
plt.ylabel("Coefficient Value");
```




```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs);
p = plt.contour(X1, X2, mses_coefs, levels=10);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w1');
plt.ylabel('w2');
sns.lineplot(x=w_steps[:,1], y=w_steps[:,2], color='black', alpha=0.5, sort=False);
sns.scatterplot(x=w_steps[:,1], y=w_steps[:,2], hue=np.arange(itr), edgecolor=None);
```


0
40
80
120
2.5
5.0
7.5
10.0
w1
w2
3
4
5
6
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5