sheet04-schrott

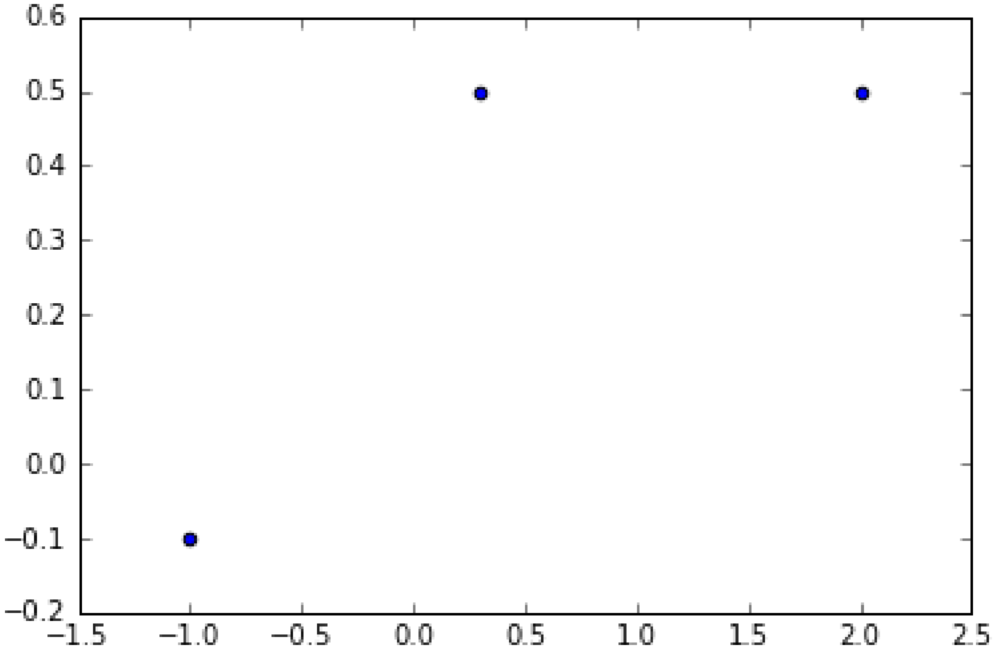
November 17, 2016

1 H4.2 Comparison of gradient descent methods

```
In [1]: import numpy as np

        # plotting stuff
        %matplotlib inline
        import matplotlib.pyplot as plt
```

Initialize and plot the training data.

```
In [2]: x = np.array([-1.,0.3,2.0])[np.newaxis,:]
        t = np.array([-0.1,0.5,0.5])[:,np.newaxis]

        X = np.append(np.ones(x.shape), x, axis=0)

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

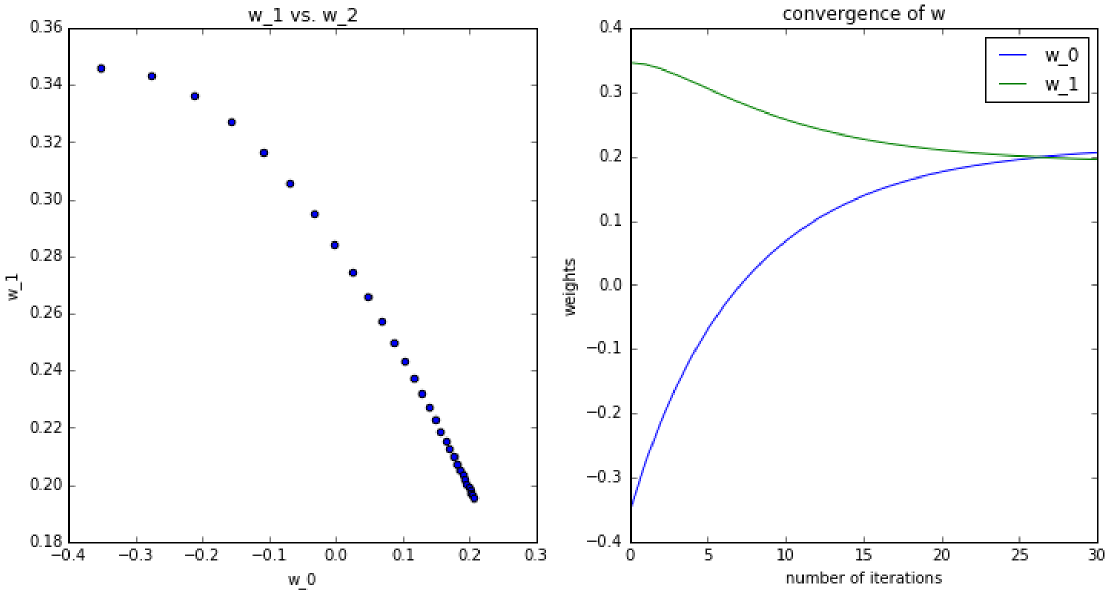


```
In [3]: # initialization
        H = X.dot(X.T)
        b = -1 * X.dot(t)
        w_initial = np.random.uniform(-0.5,0.5,(2))[:,np.newaxis]

        n_max_iter = 30
```

```
In [4]: def plot_w_1_vs_w_2(W):
            plt.scatter(W[0,:], W[1,:])
            plt.xlabel('w_0')
            plt.ylabel('w_1')
            plt.title('w_1 vs. w_2')


        def plot_w_vs_iterations(W):
            plt.plot(W[0,:], label='w_0')
            plt.plot(W[1,:], label='w_1')
            plt.xlabel('number of iterations')
            plt.ylabel('weights')
            plt.title('convergence of w')
            plt.legend()
```

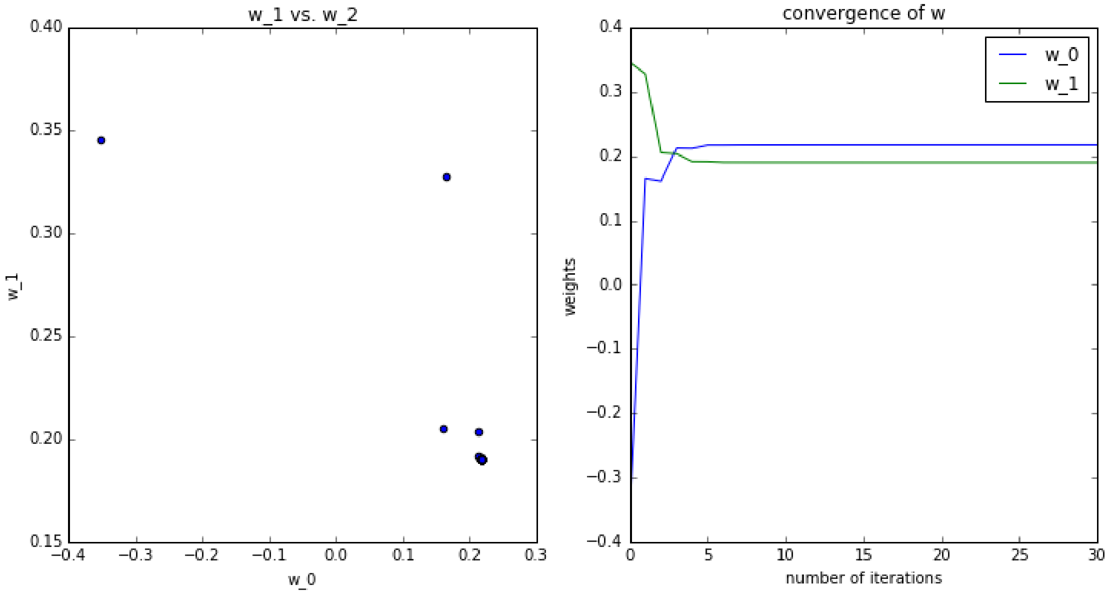
1.1 H4.2 a) Gradient Descent

```
In [5]: learning_rate = 0.05

        W = np.zeros((2, n_max_iter+1))
        W[:,0] = w_initial.reshape(2)

        for i in range(0, n_max_iter):
            # compute the gradient
            g_t = H.dot(W[:,i][:,np.newaxis]) + b
            # update the weights
            W[:,i+1] = (W[:,i][:,np.newaxis] - learning_rate * g_t)[:,0]

        # plot
        fig = plt.figure(figsize=(12,6))
        # w_0 vs. w_1
        plt.subplot(1, 2, 1)
        plot_w_1_vs_w_2(W)

        # w over the iterations
        plt.subplot(1, 2, 2)
        plot_w_vs_iterations(W)

        plt.show()
```



1.2 H4.2 b) Line Search

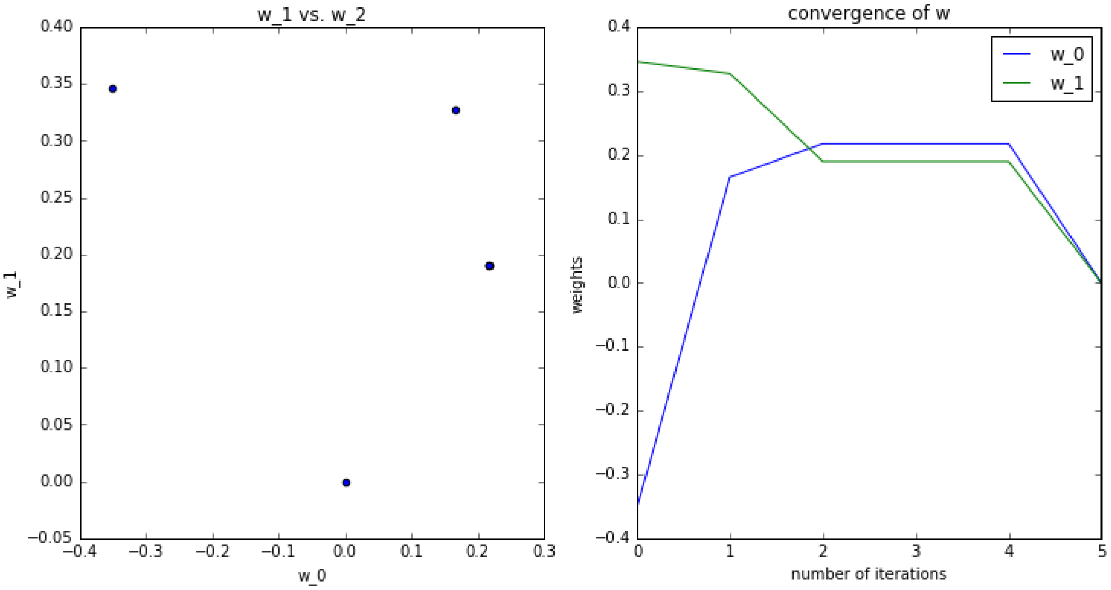
```
In [6]: learning_rate = 0.05

        W = np.zeros((2, n_max_iter+1))
        W[:,0] = w_initial.reshape(2)

        for i in range(0, n_max_iter):
            # current weight
            w = W[:,i][:,np.newaxis]

            # compute the gradient
            g = H.dot(w) + b

            # compute the new learning rate
            # regularize to ensure there is not div by 0
            var_e = g.T.dot(H.dot(g))
            if (var_e != 0): var_e = var_e + 0.00001
            learning_rate = g.T.dot(g) / var_e

            # update the weichts
            W[:,i+1] = (w - learning_rate * g)[:,0]


        # plot
        fig = plt.figure(figsize=(12,6))
        # w_0 vs. w_1
        plt.subplot(1,2,1)
        plot_w_1_vs_w_2(W)

        # w over the iterations
        plt.subplot(1, 2, 2)
        plot_w_vs_iterations(W)

        plt.show()
```



1.3 H4.2 c) Conjugate Gradient

```
In [7]: n_max_iter = 5

        learning_rate = 0.05

        W = np.zeros((2, n_max_iter+1))
        W[:,0] = w_initial.reshape(2)

        g = H.dot(w_initial)+b
        d = -g

        for i in range(0, n_max_iter):
            w = W[:,i][:,np.newaxis]
            var_e = d.T.dot(H.dot(d))
            # to make it converge save
            if (var_e != 0):
                learning_rate = -d.T.dot(g)/var_e
```

```
    else:
        break
    w = w + learning_rate * d
    g_next = H.dot(w) + b
    beta = - g_next.T.dot(g_next)/g.T.dot(g)
    d = g_next + beta * d
    g = g_next
    W[:,i+1] = w.reshape(2)


# plot
fig = plt.figure(figsize=(12,6))
# w_0 vs. w_1
plt.subplot(1,2,1)
plot_w_1_vs_w_2(W)

# w over the iterations
plt.subplot(1, 2, 2)
plot_w_vs_iterations(W)

plt.show()
```


w_1 vs. w_2
w_1
w_0
convergence of w
w_0
w_1
weights
number of iterations