```
In [1]: import csv
        import numpy as np
        from sklearn.metrics import accuracy_score
        import random
        import matplotlib.pyplot as plt
```

Perceptron Learning algorithm

[2 points] Implement the Perceptron Learning algorithm. Run it on the data file "classification.txt" ignoring the 5th column. That is, consider only the first 4 columns in each row. The first 3 columns are the coordinates of a point; and the 4th column is its classification label +1 or -1. Report your results (weights and accuracy after the final iteration).

```
In [2]: data = []
        with open('classification.txt') as f:
            file = csv.reader(f)
            for line in file:
                data.append(line)
        data = np.array(data).astype('float64')
```

```
In [3]: def perceptron_learning_ALG(x, y_true, alpha):
            w = np.zeros(len(x[0]))
            iteration = 0
            yhat = -1*(x.dot(w) < 0) + 1*(x.dot(w) >= 0)
            while np.any(yhat != y_true):
                for idx in range(len(x)):
                    z = x[idx].dot(w)
                    if y_true[idx]==1 and z<0:
                        w = w + alpha*x[idx]
                        break
                    if y_true[idx]==-1 and z>=0:
                        w = w - alpha*x[idx]
                        break
                yhat = -1*(x.dot(w) < 0) + 1*(x.dot(w) >= 0)
                iteration+=1
            acc = accuracy_score(y_true, yhat)
            return iteration, w, acc
```

```
In [4]: data_points = data[:,:3]
        yi = data[:,3]
        xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
        iteration, w, acc = perceptron_learning_ALG(xi, yi, 0.1)
        iteration, w, acc

Out[4]: (23884, array([ 0.        , 11.33717993, -9.11213384, -6.78511535]), 1.0)
```

Pocket algorithm

[1 point] Implement the Pocket algorithm and run it on the data file "classification.txt" ignoring the 4th column. That is, consider only the first 3 columns and the 5th column in each row. The first 3 columns are the coordinates of a point; and the 5th column is its classification label +1 or -1. Plot the number of misclassified points against the number of iterations of the algorithm. Run up to 7000 iterations. Also, report your results (weights and accuracy after the final iteration).

```
In [178]: data = []
          with open('classification.txt') as f:
              file = csv.reader(f)
              for line in file:
                  data.append(line)
          data = np.array(data).astype('float64')
```

In [179]:
```python
def sign(z):
    yhat = -1*(z < 0) + 1*(z >= 0)
    return yhat
```

In [180]:
```python
def find_all_diff(w, x, y):
    zi = x.dot(w)
    yhat = sign(zi)
    all_diff_idx = np.where(yhat!=y)[0]
    return all_diff_idx
```

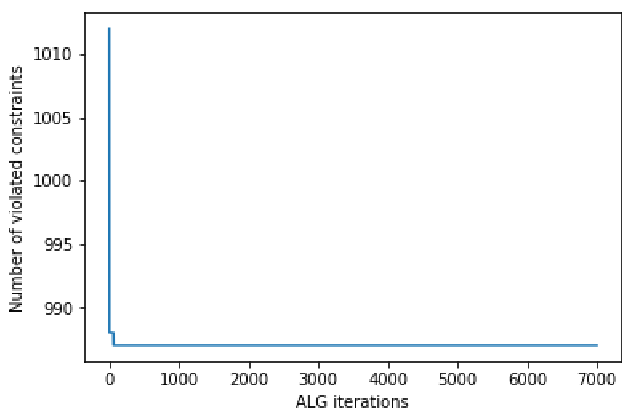
ALG1: Only update w when w' makes fewer mistakes than w

In [249]:
```python
def pocket_ALG(x, y_true, alpha, iteration_num):
    w = np.zeros(len(x[0]))
    w_diff_idx = find_all_diff(w, x, y_true)
    violated_amount = []
    for i in range(iteration_num):
        violated_amount.append(len(w_diff_idx))
        diff_idx = random.choice(w_diff_idx)
        w2 = w + alpha*y_true[diff_idx]*x[diff_idx]
        w2_diff_idx = find_all_diff(w2, x, y_true)
        if len(w2_diff_idx) < len(w_diff_idx):
            w = w2.copy()
            w_diff_idx = w2_diff_idx.copy()
    zi = x.dot(w)
    yhat = sign(zi)
    acc = accuracy_score(y_true, yhat)
    return w, w_diff_idx, acc, violated_amount, yhat
```

In [286]:
```python
data_points = data[:,:3]
yi = data[:,4]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w, w_diff_idx, acc, violated_amount, yhat = pocket_ALG(xi, yi, 0.1, 7000)
w, acc
```

Out[286]:
```
(array([ 0.        , -0.02579255,  0.00195799,  0.05170381]), 0.5065)
```

In [287]:
```python
x = [i for i in range(1,7001)]
y = violated_amount
plt.plot(x,y)
plt.xlabel('ALG iterations')
plt.ylabel('Number of violated constraints')
plt.show()
```



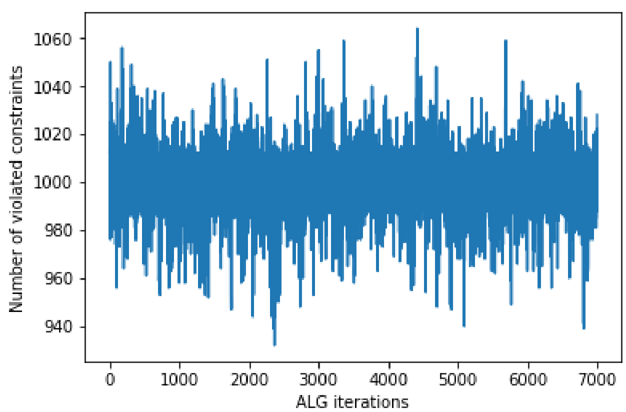
ALG2: Randomly pick one that is wrong and update w every time, but remember the w that have least wrong prediction

In [298]:
```
def pocket_ALG(x, y_true, alpha, iteration_num):
    w = np.zeros(len(x[0]))
    w_diff_idx = find_all_diff(w, x, y_true)
    violated_amount = []
    w_diff_least = len(y)
    for i in range(iteration_num):
        violated_amount.append(len(w_diff_idx))
        diff_idx = random.choice(w_diff_idx)
        w = w + alpha*y_true[diff_idx]*x[diff_idx]
        w_diff_idx = find_all_diff(w, x, y_true)
        if len(w_diff_idx) < w_diff_least:
            w_diff_least = len(w_diff_idx)
            w_best = w
    zi = x.dot(w)
    yhat = sign(zi)
    acc = accuracy_score(y_true, yhat)
    return w_best, w_diff_idx, acc, violated_amount, yhat
```

In [299]:
```
data_points = data[:,:3]
yi = data[:,4]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w_best, w_diff_idx, acc, violated_amount, yhat = pocket_ALG(xi, yi, 0.1, 7000)
w_best, acc
```

Out[299]:
```
(array([ 0.        , -0.15457602,  0.12474349,  0.02682324]), 0.506)
```

In [300]:
```
x = [i for i in range(1,7001)]
y = violated_amount
plt.plot(x,y)
plt.xlabel('ALG iterations')
plt.ylabel('Number of violated constraints')
plt.show()
```



Logistic Regression

[3 points] Implement Logistic Regression and run it on the points in the data file "classification.txt" ignoring the 4th column. That is, consider only the first 3 columns and the 5th column in each row. The first 3 columns are the coordinates of a point; and the 5th column is its classification label +1 or -1. Use the sigmoid function Θ(s) = es/(1+es). Run up to 7000 iterations. Report your results (weights and accuracy after the final iteration).

In [162]:
```
data = []
with open('classification.txt') as f:
    file = csv.reader(f)
    for line in file:
        data.append(line)
data = np.array(data).astype('float64')
```

In [164]:
```
def sigmoid(s):
    z = np.exp(s)/(1+np.exp(s))
    return z
```

In [165]:
```
def gradient_Ein(w, xi, yi):
    sum2=0
    for x,y in zip(xi,yi):
        sum2 += y*x*(1/(1+np.exp(y*x.dot(w))))
    return sum2*(-1/len(xi))
```

In [168]:
```
def Logistic_Reg(xi, yi, alpha, iteration_num):
    w = np.zeros(len(xi[0]))
    for i in range(iteration_num):
        w = w - alpha*gradient_Ein(w, xi, yi)
    s = xi.dot(w)
    theta = sigmoid(s)
    yhat = -1*(theta < 0.5) + 1*(theta >= 0.5)
    acc = accuracy_score(yi, yhat)
    return w, acc, yhat, theta
```

In [169]:
```
data_points = data[:,:3]
yi = data[:,4]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w, acc, yhat, theta = Logistic_Reg(xi, yi, 0.1, 7000)
w, acc
```

Out[169]:
```
(array([-0.03149498, -0.17769975,  0.11444872,  0.07669738]), 0.5295)
```

Linear Regression

[1 point] Implement Linear Regression and run it on the points in the data file "linear-regression.txt". The first 2 columns in each row represent the independent X and Y variables; and the 3rd column represents the dependent Z variable. Report your results (weights after the final iteration).

In [170]:
```
data = []
with open('linear-regression.txt') as f:
    file = csv.reader(f)
    for line in file:
        data.append(line)
data = np.array(data).astype('float64')
```

In [173]:
```
def Linear_Reg(xi, yi):
    DDT = xi.transpose().dot(xi)
    DDT_inv = np.linalg.inv(DDT)
    w = DDT_inv.dot(xi.transpose()).dot(yi)
    return w
```

In [174]:
```
data_points = data[:,:2]
yi = data[:,2]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w = Linear_Reg(xi, yi)
w
```

Out[174]:
```
array([0.01523535, 1.08546357, 3.99068855])
```