In [107]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import numpy as np;
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error as MSE
#getting boston house price data from sklearn
boston = load_boston()
```

In [2]:

```
#finding no of datapoints in boston data

X = pd.DataFrame(data=boston['data'], columns=boston['feature_names']);
print("Shape of feature matrix: ", X.shape)
print(X.head(5))

print('*'*100)

y = pd.DataFrame(np.array(boston['target']))
print("Shape of output matrix", y.shape)
```

```
Shape of feature matrix:  (506, 13)
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
****************************************************************************************************

Shape of output matrix (506, 1)
```

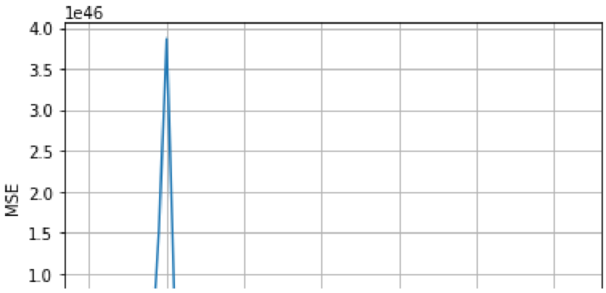
In [183]:

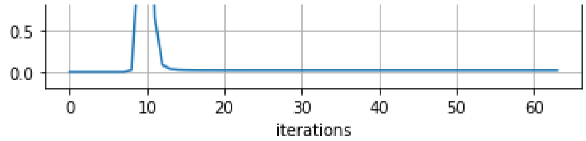
```
class LinearRegression():
    def __init__(self, X, y, plot_error_per_iteration = False):
        self.X = X
        self.y = y #dimension: n*1
        self.data = pd.concat([X, y], axis=1);
        self.lr = 0.001 #defining learning rate
        self.b = np.random.normal(0,1,1);
        self.w = np.zeros((self.X.shape[1], 1)) #generating intial weight vector of dimension 1*d,
where d is no of columns/features
        self.MSE_list = [];
        self.plot_error_per_iteration = plot_error_per_iteration;

    def computeDerivative(self, X, y):
        """
        function to obtain derivative value which is -2*x * (y - W_T * X)
        """
        wT_x = np.dot(X, self.w)#calculating x * w => multiplying X(n*d - dim) with weight(d*1), ou
tput will be of n*1 - dim
        y_wT_x = y - wT_x #calculating y - (w_t * x) => output will be of dimension n*1
        x_prod = -2 * X;
        der = np.dot(y_wT_x.T, x_prod) #der will be of dimension 1 * d
        return self.w - self.lr * der.T;

    def computeIntercept(self, X, y):
        """
        function to compute new intercept
```

```
        function to compute new intercept
        """
        wT_x = np.dot(X, self.w)#calculating x * w => multiplying X(n*d - dim) with weight(d*1), ou
tput will be of n*1 - dim
        y_wT_x = y - wT_x #calculating y - (w_t * x) => output will be of dimension n*1
        y_prod = -2 * y_wT_x;
        intercept = y_prod.sum(axis=0)
        intercept = intercept[0]
        return self.b - (self.lr * intercept)

    def predict(self, X):
        """
        function to predict output given query points
        """
        wT_x = np.dot(self.w.T, X.T)
        y = wT_x + self.b;
        return y.T

    def addMSE(self, ):
        """
        function to add mean squared error for each W, obtain in each iteration
        """
        y_pred = self.predict(self.X);
        self.MSE_list.append(MSE(y, y_pred));

    def plotError(self, ):
        """
        function to plot error
        """
        plt.plot(np.arange(0, len(self.MSE_list)), self.MSE_list);
        plt.ylabel('MSE');
        plt.xlabel('iterations')
        plt.grid();
        plt.show();

    def fit(self, ):
        itr = 1;
        while True:
            sample_data = self.data.sample(n=5)
            X = sample_data.loc[:, sample_data.columns != 0];
            y = sample_data.loc[:, [0]]
            new_w = self.computeDerivative(X, y);
            new_b = self.computeIntercept(X, y);
            if (np.all(new_w - self.w) < 0.01 and np.all(new_b - self.b) < 0.01):
                self.w = new_w
                self.b = new_b
                if self.plot_error_per_iteration:
                    self.addMSE();
                    self.plotError();
                break
            else:
                self.lr /= 2
                self.w = new_w
                self.b = new_b
                if self.plot_error_per_iteration:
                    self.addMSE();
```

In [186]:

```
LR = LinearRegression(X, y, plot_error_per_iteration = True);
LR.fit()
y_pred = LR.predict(X);
self_model_error = MSE(y, y_pred)
```





In [177]:

```
from sklearn import linear_model
model = linear_model.SGDRegressor(penalty='none');
model.fit(X, y);
y_pred = model.predict(X);
sgd_model_error = MSE(y, y_pred)
```

In [178]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Mean squared error"]
x.add_row(["SGDRegressor", sgd_model_error])
x.add_row(["our created model",self_model_error ])
print(x)
```

```
+-------------------+------------------------+
|       Model       |   Mean squared error   |
+-------------------+------------------------+
|    SGDRegressor   | 1.216532166852742e+29  |
| our created model | 1.3757560711443702e+43 |
+-------------------+------------------------+
```