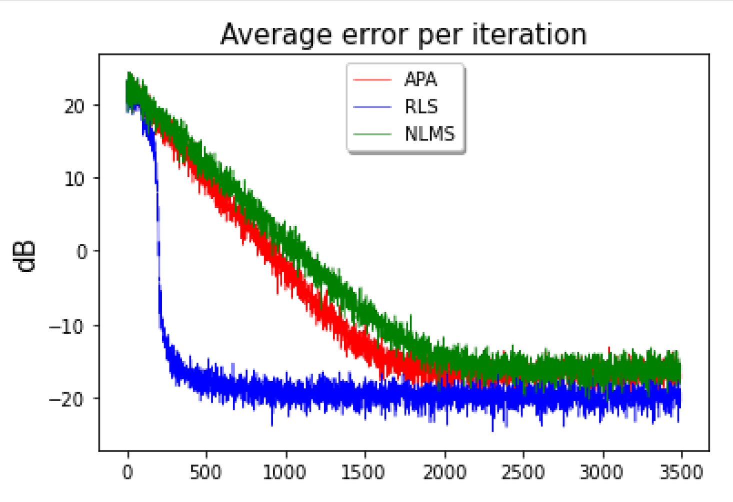
Question 4.2

In [1]:
```
import os
import sys
import numpy as np
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
from matplotlib import pyplot as plt
sys.path.append(os.getcwd())
sys.path.append('../')
```

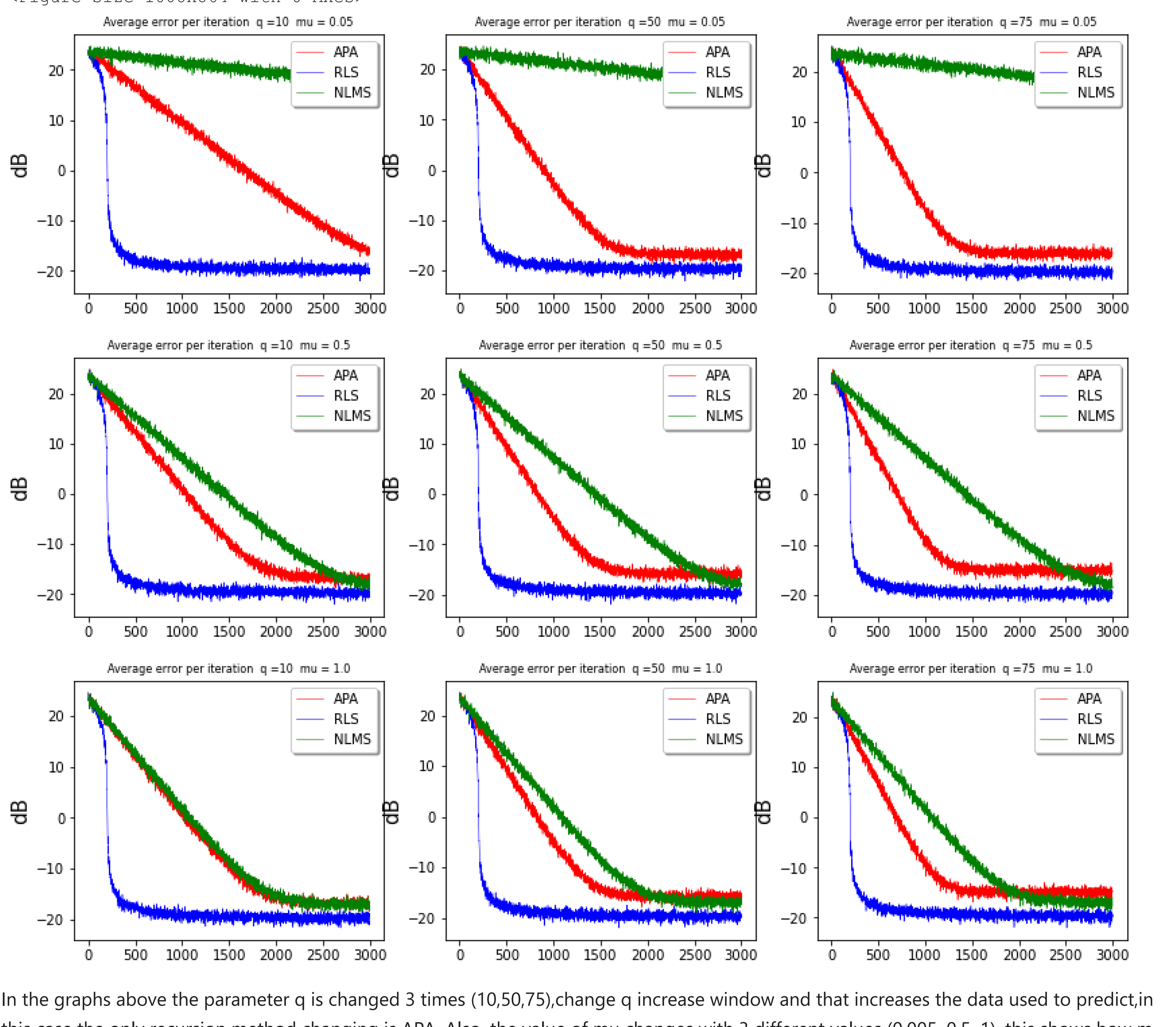
In [2]:
```
L = 200  # Dimension of the unknown vector
N = 3500  # Number of Data
theta = np.random.randn(L, 1)  # Unknown parameter

Iter_n = 30
MSE1 = np.zeros((N, Iter_n))
MSE2 = np.zeros((N, Iter_n))
MSE3 = np.zeros((N, Iter_n))

noisevar = 0.01
epsilon = np.sqrt(2) * noisevar


for t in range(0, Iter_n):

    X = np.random.randn(L, N)
    inputvec = lambda n: np.array([X[:, n].copy()]).conj().T


    noise = np.random.randn(N, 1) * np.sqrt(noisevar)

    y = np.zeros((N, 1))
    y[0:N] = np.dot(X[:, 0:N].conj().T, theta)
    y = y + noise
    w = np.zeros((L, 1))
    mu = 0.2
    delta = 0.001
    q = 30
    for i in range(0, N):
        if i > q:
            qq = range(i, i - q, -1)

            yvec = y[qq]
            Xq = inputvec(qq)
            Xq = np.reshape(Xq, newshape=(Xq.shape[0], Xq.shape[1]))
            e = yvec - np.dot(Xq, w)
            eins = y[i] - np.dot(w.conj().T, inputvec(i))
            w = w + mu * np.dot(np.dot(Xq.conj().T, np.linalg.inv(delta*np.eye(q)+np.dot(Xq, Xq.conj().T))), e)
            MSE1[i, t] = eins ** 2

    # RLS recursion
    w = np.zeros((L,1))
    delta = 0.001
    P = (1/delta) * np.eye(L)
    for i in range(0, N):

        gamma = 1/(1+np.dot(inputvec(i).conj().T, np.dot(P, inputvec(i))))
        gi = np.dot(P, inputvec(i)) * gamma
        e = y[i] - np.dot(w.conj().T, inputvec(i))
        w = w + gi * e
        P = P - np.dot(gi, gi.conj().T)/gamma
        MSE2[i, t] = e ** 2

    # NLMS Recursion
    w = np.zeros((L, 1))
    delta = 0.001
    mu = 1.2

    for i in range(0, N):

        e = y[i] - np.dot(w.conj().T, inputvec(i))
        mun = mu / (delta+np.dot(inputvec(i).conj().T, inputvec(i)))
        w = w + mun * e * inputvec(i)
        MSE3[i, t] = e ** 2

MSEav1 = sum(MSE1.conj().T) / Iter_n
MSEav2 = sum(MSE2.conj().T) / Iter_n
MSEav3 = sum(MSE3.conj().T) / Iter_n

plt.plot(10 * np.log10(MSEav1), 'r', lw=0.5)
plt.plot(10 * np.log10(MSEav2), 'b', lw=0.5)
plt.plot(10 * np.log10(MSEav3), 'g', lw=0.5)
plt.title("Average error per iteration", fontsize=15)
plt.ylabel('dB', fontsize=15)
plt.legend(('APA', 'RLS', 'NLMS'),
           loc='upper center', shadow=True)
plt.show()
```



From the graph above it is plausible to say that the fastest perfomnce comes from RLS

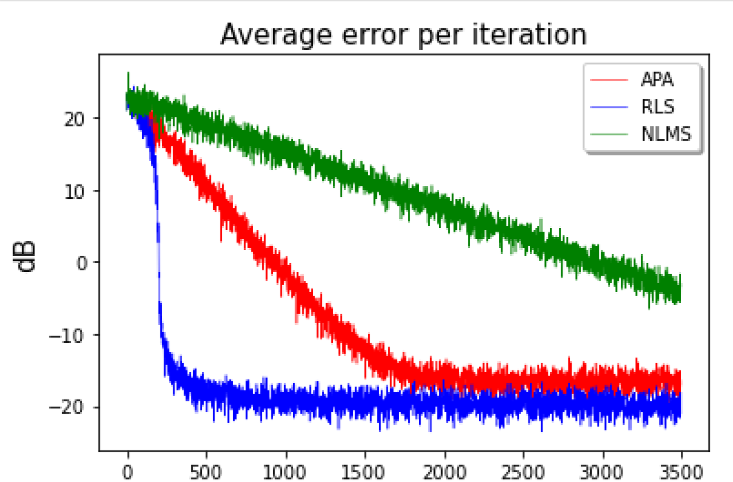
In [3]:
```
#changing mu & q

L = 200  # Dimension of the unknown vector
N = 3000  # Number of Data
theta = np.random.randn(L, 1)  # Unknown parameter

Iter_n = 100
MSE1 = np.zeros((N, Iter_n))
MSE2 = np.zeros((N, Iter_n))
MSE3 = np.zeros((N, Iter_n))

noisevar = 0.01
epsilon = np.sqrt(2) * noisevar

mu= np.array([0.05,0.5,1])
q = np.array([10,50,75])
#mu[t] = 0.2
delta = 0.001

g=1
fig = plt.figure(figsize=(14,12))
for k in range(len(mu)):
    fig = plt.figure(figsize=(14,12))
    for j in range(len(q)):

        for t in range(0, Iter_n):

            X = np.random.randn(L, N)
            inputvec = lambda n: np.array([X[:, n].copy()]).conj().T


            noise = np.random.randn(N, 1) * np.sqrt(noisevar)

            y = np.zeros((N, 1))
            y[0:N] = np.dot(X[:, 0:N].conj().T, theta)
            y = y + noise
            w = np.zeros((L, 1))

            for i in range(0, N):
                if i > q[j]:
                    qq = range(i, i - q[j], -1)

                    yvec = y[qq]
                    Xq = inputvec(qq)
                    Xq = np.reshape(Xq, newshape=(Xq.shape[0], Xq.shape[1]))
                    e = yvec - np.dot(Xq, w)
                    eins = y[i] - np.dot(w.conj().T, inputvec(i))
                    w = w + mu[k] * np.dot(np.dot(Xq.conj().T, np.linalg.inv(delta*np.eye(q[j])+np.dot(Xq, Xq.c
                    MSE1[i, t] = eins ** 2

            # RLS recursion
            w = np.zeros((L,1))
            P = (1/delta) * np.eye(L)
            for i in range(0, N):

                gamma = 1/(1+np.dot(inputvec(i).conj().T, np.dot(P, inputvec(i))))
                gi = np.dot(P, inputvec(i)) * gamma
                e = y[i] - np.dot(w.conj().T, inputvec(i))
                w = w + gi * e
                P = P - np.dot(gi, gi.conj().T)/gamma
                MSE2[i, t] = e ** 2

            # RLS recursion
            w = np.zeros((L, 1))

            for i in range(0, N):
                # mu[t]=1;%/(i^0.5);
                e = y[i] - np.dot(w.conj().T, inputvec(i))
                muu = mu[k] / (delta+np.dot(inputvec(i).conj().T, inputvec(i)))
                w = w + muu * e * inputvec(i)
                MSE3[i, t] = e ** 2

        MSEav1 = sum(MSE1.conj().T) / Iter_n
        MSEav2 = sum(MSE2.conj().T) / Iter_n
        MSEav3 = sum(MSE3.conj().T) / Iter_n

        plt.subplot(len(q),len(mu),g)
        g +=1
        plt.plot(10 * np.log10(MSEav1), 'r', lw=0.5)
        plt.plot(10 * np.log10(MSEav2), 'b', lw=0.5)
        plt.plot(10 * np.log10(MSEav3), 'g', lw=0.5)
        plt.title('Average error per iteration  q =' + str(q[j]) + '  mu = ' + str(mu[k]), fontsize=8)
        plt.ylabel('dB', fontsize=15)
        plt.legend(('APA', 'RLS', 'NLMS'), loc='upper right', shadow=True)
    plt.show()
```

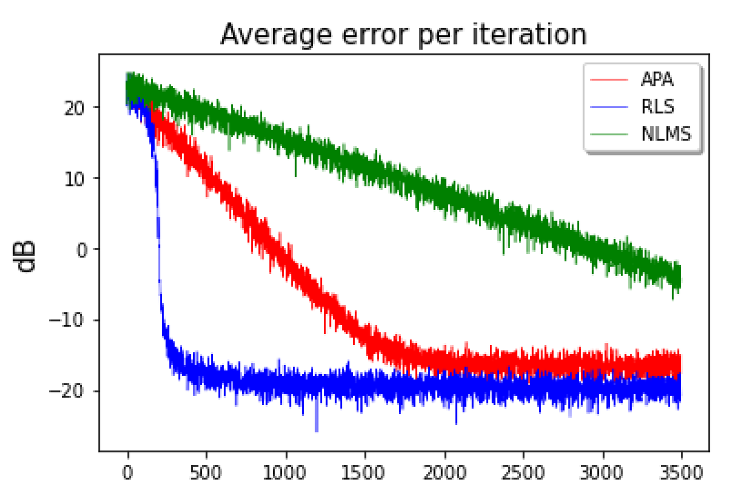
```
<Figure size 1008x864 with 0 Axes>
```

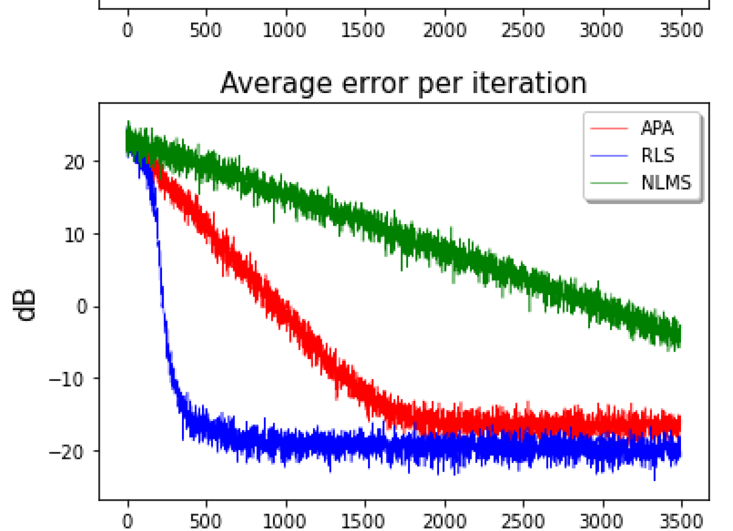


In the graphs above the parameter q is changed 3 times (10,50,75),change q increase window and that increases the data used to predict,in this case the only recursion method changing is APA. Also, the value of mu changes with 3 different values (0.005, 0.5, 1), this shows how m is related to learning rate, the bigger it is the fastets it update. It can be seen that the most effective recursion method is RLS and the slowest one is NLMS.

In [4]:
```
#changing delta

L = 200  # Dimension of the unknown vector
N = 3500  # Number of Data
theta = np.random.randn(L, 1)  # Unknown parameter

Iter_n = 30
MSE1 = np.zeros((N, Iter_n))
MSE2 = np.zeros((N, Iter_n))
MSE3 = np.zeros((N, Iter_n))

noisevar = 0.01
epsilon = np.sqrt(2) * noisevar

delta = np.array([0.001,0.1,1])
mu = 0.2
for j in range(len(delta)):
    for t in range(0, Iter_n):

        X = np.random.randn(L, N)
        inputvec = lambda n: np.array([X[:, n].copy()]).conj().T


        noise = np.random.randn(N, 1) * np.sqrt(noisevar)

        y = np.zeros((N, 1))
        y[0:N] = np.dot(X[:, 0:N].conj().T, theta)
        y = y + noise
        w = np.zeros((L, 1))
        q = 30
        for i in range(0, N):
            if i > q:
                qq = range(i, i - q, -1)

                yvec = y[qq]
                Xq = inputvec(qq)
                Xq = np.reshape(Xq, newshape=(Xq.shape[0], Xq.shape[1]))
                e = yvec - np.dot(Xq, w)
                eins = y[i] - np.dot(w.conj().T, inputvec(i))
                w = w + mu * np.dot(np.dot(Xq.conj().T, np.linalg.inv(delta[j]*np.eye(q)+np.dot(Xq, Xq.conj().T
                MSE1[i, t] = eins ** 2

        # RLS recursion
        w = np.zeros((L,1))
        P = (1/delta[j]) * np.eye(L)
        for i in range(0, N):

            gamma = 1/(1+np.dot(inputvec(i).conj().T, np.dot(P, inputvec(i))))
            gi = np.dot(P, inputvec(i)) * gamma
            e = y[i] - np.dot(w.conj().T, inputvec(i))
            w = w + gi * e
            P = P - np.dot(gi, gi.conj().T)/gamma
            MSE2[i, t] = e ** 2

        # NLMS Recursion
        w = np.zeros((L, 1))

        for i in range(0, N):

            e = y[i] - np.dot(w.conj().T, inputvec(i))
            muu = mu / (delta[j]+np.dot(inputvec(i).conj().T, inputvec(i)))
            w = w + muu * e * inputvec(i)
            MSE3[i, t] = e ** 2

    MSEav1 = sum(MSE1.conj().T) / Iter_n
    MSEav2 = sum(MSE2.conj().T) / Iter_n
    MSEav3 = sum(MSE3.conj().T) / Iter_n

    plt.plot(10 * np.log10(MSEav1), 'r', lw=0.5)
    plt.plot(10 * np.log10(MSEav2), 'b', lw=0.5)
    plt.plot(10 * np.log10(MSEav3), 'g', lw=0.5)
    plt.title("Average error per iteration", fontsize=15)
    plt.ylabel('dB', fontsize=15)
    plt.legend(('APA', 'RLS', 'NLMS'),
               loc='upper right', shadow=True)
    plt.show()
```







In the last graphs, the value of delta is changed 3 times(0.001,0.1,1), it is observable that there are not drastic changes in the different methods.