The least-mean-squares (LMS) adaptive filter

February 5, 2020

[2]:
```
import numpy as np
from IPython.display import Image
import matplotlib.pylab as plt
import padasip as pa
```

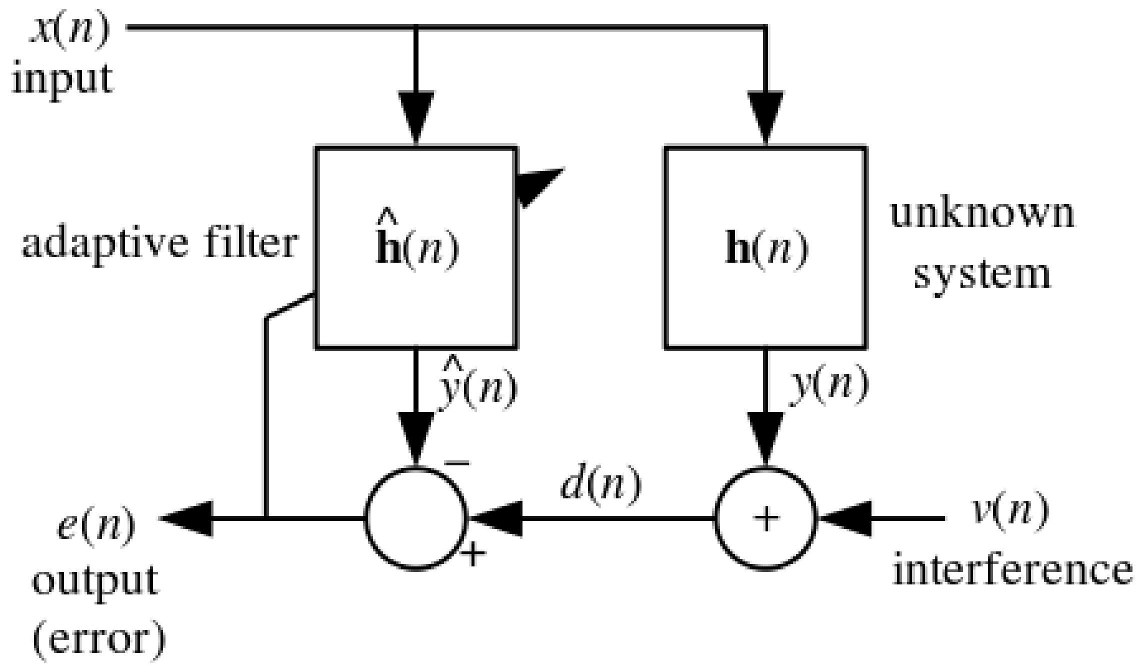
1 Least Mean Square Adaptative Filter

An unknown system h(n) is to be identified and the adaptive filter attempts to adapt the filter h ˆ (n) to make it as close as possible to h (n), while using only observable signals x(n), d(n) and e(n) but y(n), v(n) and h(n) are not directly observable.[1]

The basic idea behind LMS filter is to approach the optimum filter weights by updating the filter weights in a manner to converge to the optimum filter weight. This is based on the gradient descent algorithm.[1]

[2]:
```
PATH = "/home/mitsy/mysandbox/AF/"
Image(filename = PATH + "444px-Lms_filter.svg.png", width=500, height=500)
```

[2]:



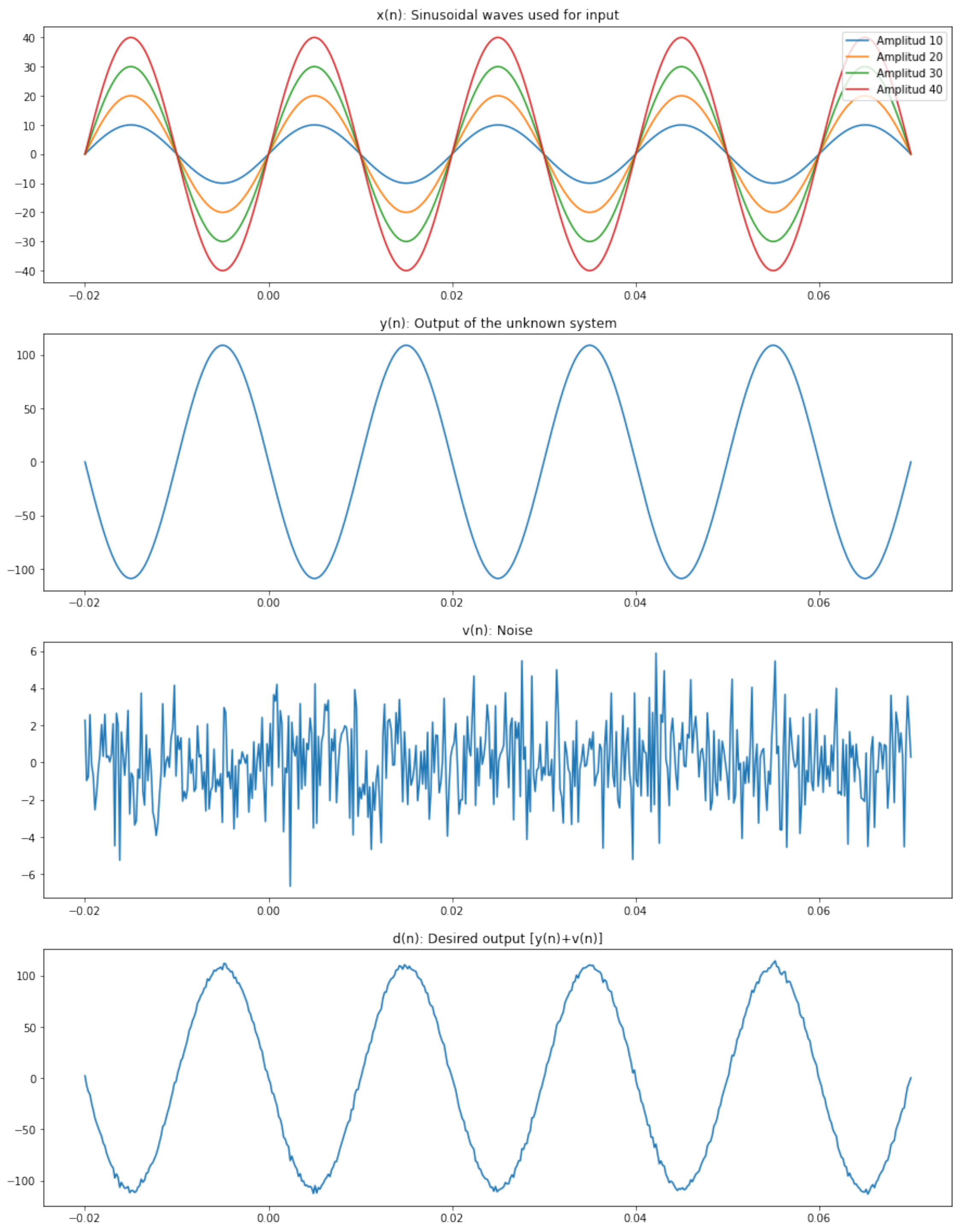
As the LMS algorithm does not use the exact values of the expectations, the weights would never reach the optimal weights in the absolute sense, but a convergence is possible in mean. That is, even though the weights may change by small amounts, it changes about the optimal weights. However, if the variance with which the weights change, is large, convergence in mean would be misleading. This problem may occur, if the value of step-size (mu) is not chosen properly.

Creation of the data

In the folowing cell I create the typical data an LMS algorithm could work with. The input x(n) consists of an 2d array of 4 sinusodal waves with different amplitudes but same frecuencies. The output of the unknown system y(n) is made performing operations over x(n), and the desired output d(n) is y(n)+noise

[51]:
```
# creation of data
N = 500 #number of data points

#t = np.linspace(-0.02, 0.05, N)#time domain goes from -0.02 to 0.05
t = np.linspace(-0.02, 0.07, N)

#Here I build the input, which is an array of N data points of 4 sinusodal␣
↪waves (N by 4)
x=np.zeros((4,N))
for i in range(4):
    x[i]= ((i+1)*10) * np.sin(2*np.pi*50*t)#each wave has an amplitud of␣
↪10,20,30 and 40 respectively
x=x.transpose()

v = np.random.normal(0, 2, N) # noise

y = 3*x[:,0] + 0.2*x[:,1] - 5.3*x[:,2] + 0.4*x[:,3]#output of the unknown␣
↪system h(n)
d = 3*x[:,0] + 0.2*x[:,1] - 5.3*x[:,2] + 0.4*x[:,3] + v # target
#each zero, first, second and third data point of each wave is multiplied by 3,␣
↪0.2, 5.3 and 0.4 respectively, sum together and added the noise v
plt.figure(figsize=(15,20))
plt.subplot(411)
plt.plot(t,x)
plt.legend(('Amplitud 10', 'Amplitud 20', 'Amplitud 30', 'Amplitud 40'),
           loc='upper right')
plt.title('x(n): Sinusoidal waves used for input')
plt.subplot(412)
plt.plot(t,y)
plt.title('y(n): Output of the unknown system')
plt.subplot(413)
plt.plot(t,v)
plt.title('v(n): Noise')
plt.subplot(414)
plt.plot(t,d)
```

```python
plt.title('d(n): Desired output [y(n)+v(n)]')
plt.show()
```



LMS adaptive filtering

Using LMS adaptive filtering requires a series of matrix operations (multiplications and sums) that can be consulted in the book Adaptative Filter Theory by Simon Haykin, chapter 6, [2] or other reliable sources. These operations can be done in python. In this case I decided to not re-invent the wheel and use an openly available library (padasip[3]) that offers a class with functions that perfom the basic operations for LMS, together with useful examples.

The key argument should be set to really small number in most of the cases[3]. The optimum value of is between 0 and 1/lambda(max), where lambda(max) is the maximum eigenvalue of the correlation matrix of the tap imputs.

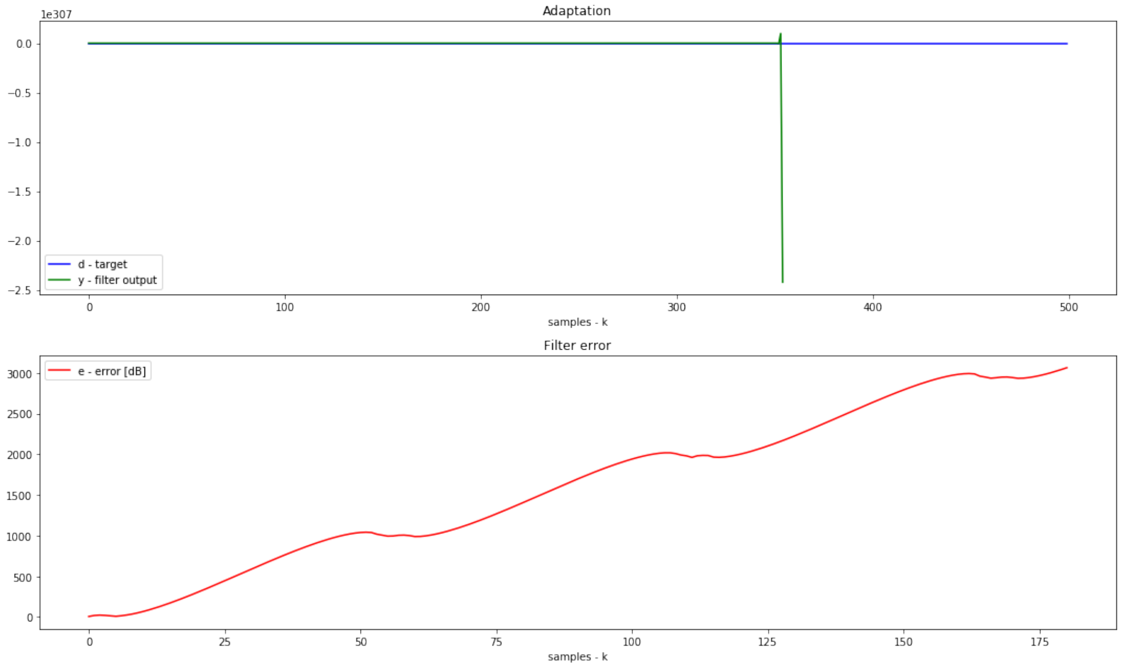
If is chosen to be large, the amount with which the weights change depends heavily on the gradient estimate, and so the weights may change by a large value so that gradient which was negative at the first instant may now become positive. And at the second instant, the weight may change in the opposite direction by a large amount because of the negative gradient and would thus keep oscillating with a large variance about the optimal weights. On the other hand, if is chosen to be too small, time to converge to the optimal weights will be too large.[1]

Here I variate the value of from 0.01 to 0.000001, and compare the plots of y and d, and observe the error plot. LMS adaptative filter should make 'y' as close to 'd' as possible, and stabilize the error plot as close to zero as possible. For this reason I also print the maximum absolute error and the average of the last 50 points of the error. The best is the othe with smaller error and more similarity of 'y' and 'd'.

[62]:
```
# identification
f = pa.filters.FilterLMS(n=4, mu=0.01, w="random")
"""
This class (FilterLMS) represents an adaptive LMS filter.

Arguments:

- `n` : length of filter (integer) - how many input is input array
  (row of input matrix), in this case is 4 because x shape is (500,4)

- `mu` : learning rate (float). Also known as step size. If it is too slow,
  the filter may have bad performance. If it is too high,
  the filter will be unstable. The default value can be unstable
  for ill-conditioned input data.

- `w` : initial weights of filter. "random" : creates random weights

"""
y, e, w = f.run(d, x)
"""
This function filters multiple samples in a row.

Arguments:

- `d` : desired value (1 dimensional array of lengh N)

- `x` : input matrix (2-dimensional array of shape (N,4)). Rows are samples,
  columns are input arrays.
```

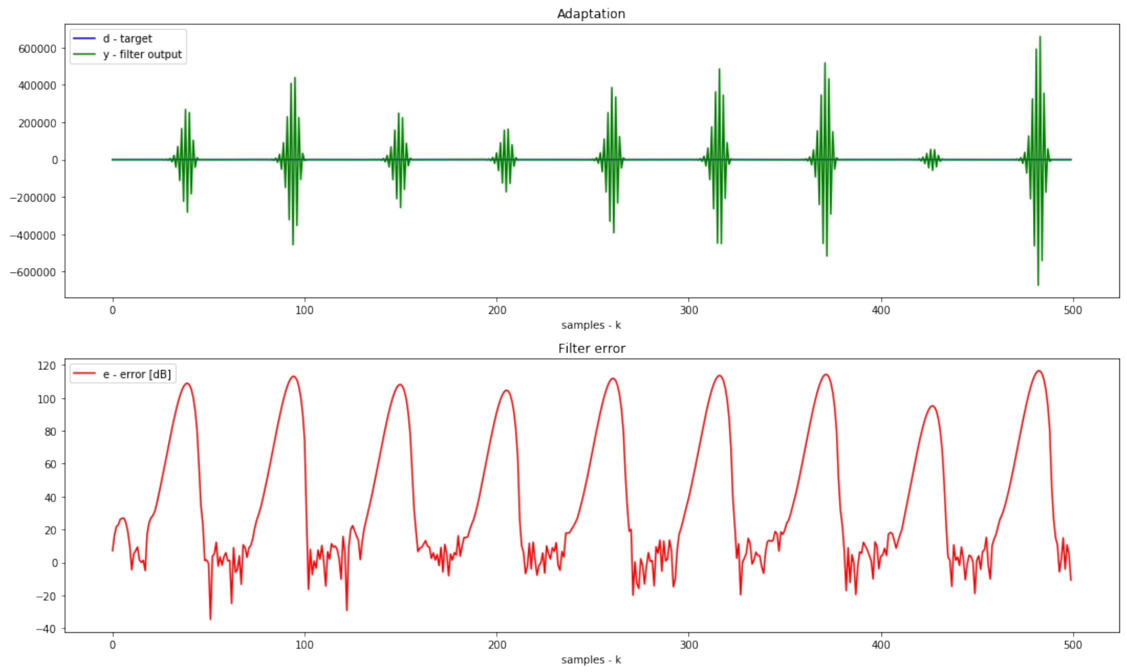
```
Returns:

- `y` : output value (1 dimensional array).
  The size corresponds with the desired value.

- `e` : filter error for every sample (1 dimensional array).
  The size corresponds with the desired value.

- `w` : history of all weights (2 dimensional array).
  Every row is set of the weights for given sample.
"""
error=10*np.log10(e**2)
# show results
plt.figure(figsize=(15,9))
plt.subplot(211);plt.title("Adaptation");plt.xlabel("samples - k")
plt.plot(d,"b", label="d - target")#desired output
plt.plot(y,"g", label="y - filter output");plt.legend()#filter output
plt.subplot(212);plt.title("Filter error");plt.xlabel("samples - k")#error
plt.plot(error,"r", label="e - error [dB]");plt.legend()
plt.tight_layout()
plt.show()
print('maximum absolute error=', max(abs(error)))
print('average of the last 50 points of the error=', np.mean(error[N-50:N]))
```

```
/home/mitsy/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:41:
RuntimeWarning: overflow encountered in square
```



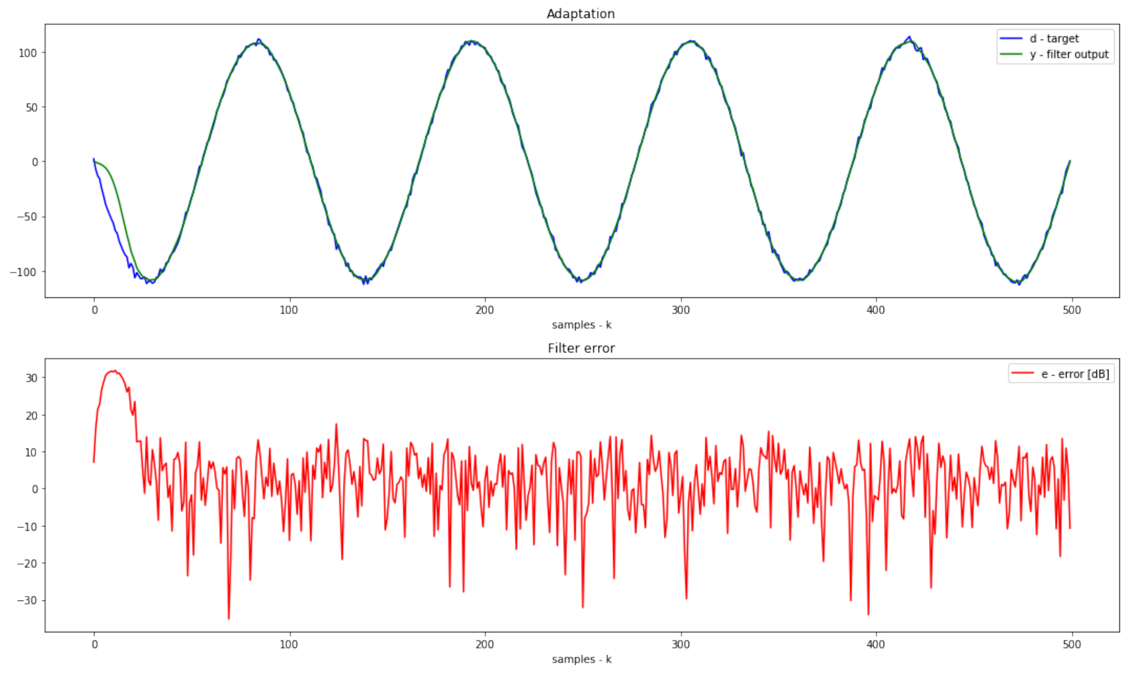
```
maximum absolute error= inf
average of the last 50 points of the error= nan
```

[63]:
```
# identification
f = pa.filters.FilterLMS(n=4, mu=0.001, w="random")
y, e, w = f.run(d, x)
error=10*np.log10(e**2)
# show results
plt.figure(figsize=(15,9))
plt.subplot(211);plt.title("Adaptation");plt.xlabel("samples - k")
plt.plot(d,"b", label="d - target")#desired output
plt.plot(y,"g", label="y - filter output");plt.legend()#filter output
plt.subplot(212);plt.title("Filter error");plt.xlabel("samples - k")#error
plt.plot(error,"r", label="e - error [dB]");plt.legend()
plt.tight_layout()
plt.show()
print('maximum absolute error=', max(abs(error)))
print('average of the last 50 points of the error=', np.mean(error[N-50:N]))
```



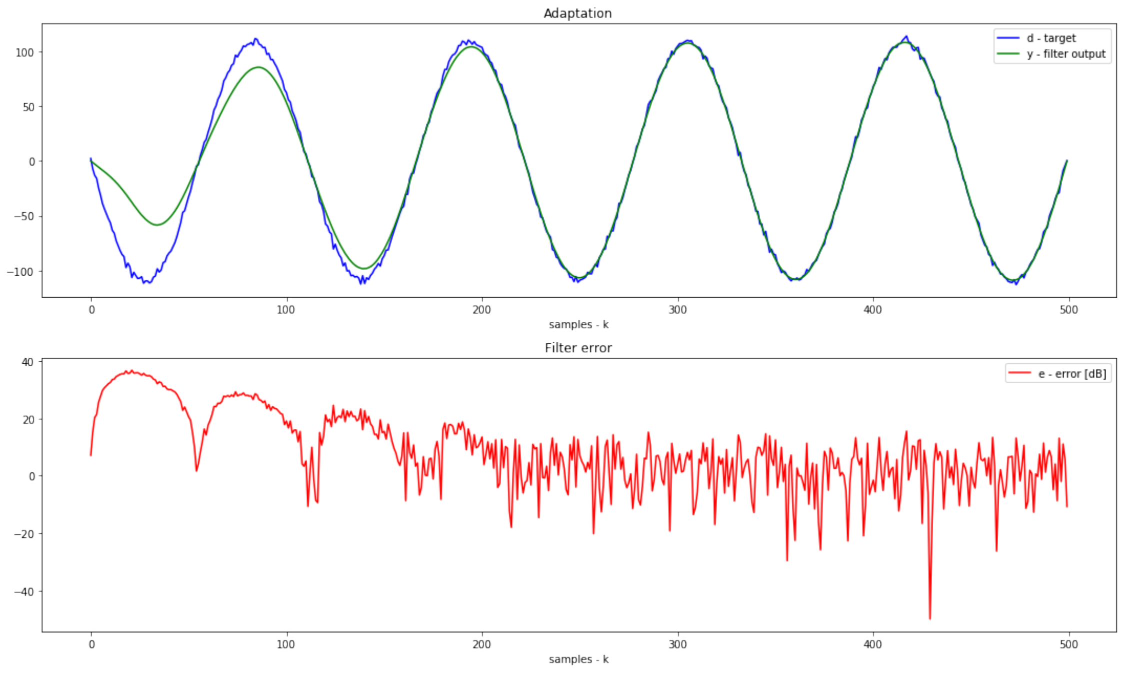
```
maximum absolute error= 116.55462574819936
average of the last 50 points of the error= 47.11490145947806
```

[71]:
```
# identification
f = pa.filters.FilterLMS(n=4, mu=0.0001, w="random")
y, e, w = f.run(d, x)
error=10*np.log10(e**2)
# show results
plt.figure(figsize=(15,9))
plt.subplot(211);plt.title("Adaptation");plt.xlabel("samples - k")
plt.plot(d,"b", label="d - target")#desired output
plt.plot(y,"g", label="y - filter output");plt.legend()#filter output
plt.subplot(212);plt.title("Filter error");plt.xlabel("samples - k")#error
plt.plot(error,"r", label="e - error [dB]");plt.legend()
plt.tight_layout()
plt.show()
print('maximum absolute error=', max(abs(error)))
print('average of the last 50 points of the error=', np.mean(error[N-50:N]))
```



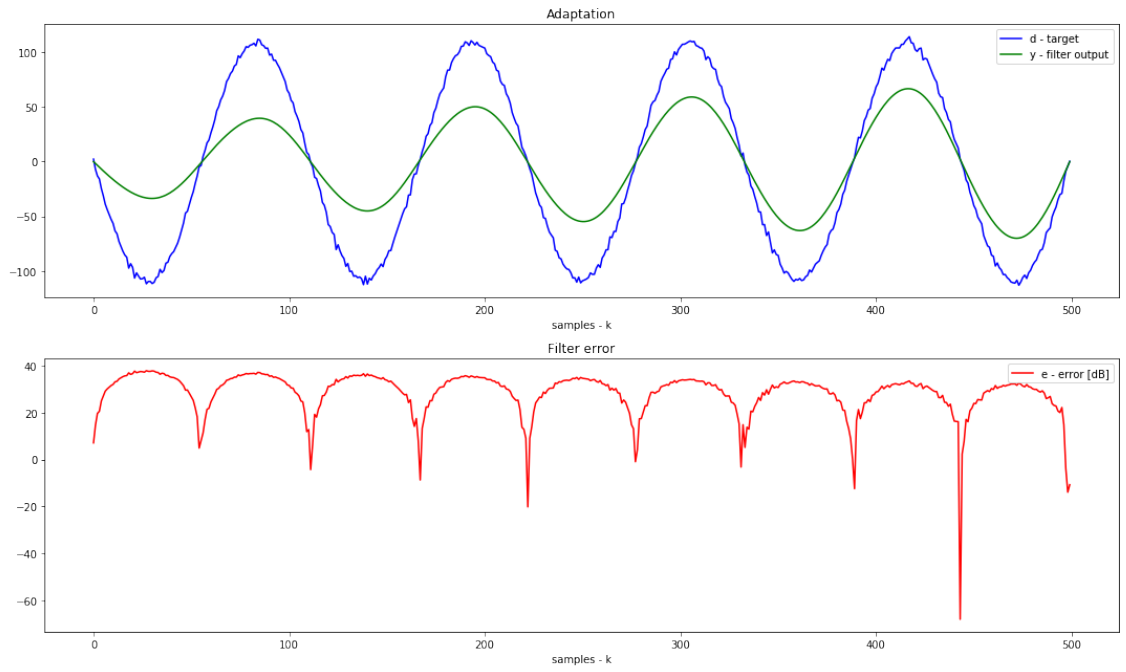
```
maximum absolute error= 35.17567500193132
average of the last 50 points of the error= 2.072438729735343
```

[72]:
```
# identification
f = pa.filters.FilterLMS(n=4, mu=0.00001, w="random")
y, e, w = f.run(d, x)
error=10*np.log10(e**2)
# show results
plt.figure(figsize=(15,9))
plt.subplot(211);plt.title("Adaptation");plt.xlabel("samples - k")
plt.plot(d,"b", label="d - target")#desired output
plt.plot(y,"g", label="y - filter output");plt.legend()#filter output
plt.subplot(212);plt.title("Filter error");plt.xlabel("samples - k")#error
plt.plot(error,"r", label="e - error [dB]");plt.legend()
plt.tight_layout()
plt.show()
print('maximum absolute error=', max(abs(error)))
print('average of the last 50 points of the error=', np.mean(error[N-50:N]))
```



```
maximum absolute error= 49.81592577307768
average of the last 50 points of the error= 1.4860175695831006
```

[66]:
```
# identification
f = pa.filters.FilterLMS(n=4, mu=0.000001, w="random")
y, e, w = f.run(d, x)
error=10*np.log10(e**2)
# show results
plt.figure(figsize=(15,9))
plt.subplot(211);plt.title("Adaptation");plt.xlabel("samples - k")
plt.plot(d,"b", label="d - target")#desired output
plt.plot(y,"g", label="y - filter output");plt.legend()#filter output
plt.subplot(212);plt.title("Filter error");plt.xlabel("samples - k")#error
plt.plot(error,"r", label="e - error [dB]");plt.legend()
plt.tight_layout()
plt.show()
print('maximum absolute error=', max(abs(error)))
print('average of the last 50 points of the error=', np.mean(error[N-50:N]))
```



```
maximum absolute error= 68.02397428698984
average of the last 50 points of the error= 26.089501136429607
```

Based on the previous explanation about , the best results were obtaine for a =0.0001.

References

[1]Monson H. Hayes: Statistical Digital Signal Processing and Modeling, Wiley, 1996, ISBN 0-471-59431-8

[2]S. Haykin: Adaptative Filter Theory, Hamilton, 2014, ISBN 978-0-132-67145-3

[3]https://matousc89.github.io/padasip/_modules/padasip/filters/lms.html#FilterLMS.run

[ ]:

```
```