**Machine Learning for Signal Processing**

**[5LSL0]**

**Assignment 1: Optimum Linear Filters**

**REPORT**

**Group number: 2**

**Names including ID:  
 1:Martin van Leeuwen 0901497  
 2:Frouke Hekker 0897373**

**Date: 3-5-19**

## Known statistics

### Wiener filter







In the optimal solution the correlation between the error and the input is zero. This would mean that the model does not structural fault, the errors are as likely to occur on each place.



If statistical information is not available it might also be difficult to get it. Whit these approximations that isn’t necessary. However this approximation will change for every X and thus the loss landscape will seem to change too. This results in a sub-optimal path to the minimum.

### Steepest gradient descent



The gradient descent algorithm goes to a steady state if

2. GD filter update Python code (insert only the relevant lines)

for k in range(N):

w += [w[-1] + 2\*alpha\*(r\_yx-np.matmul(R\_x,w[-1]))]

|  |
| --- |
| *α is chosen to be 1/20. The blue line shows how w converges to the minimum. It does not take the shortest route, it takes a curved route. However it is stable and doesn’t overshoot.* |

### Newton algorithm



Newton converges for:

And α is not dependent on the filter weights so they can only converge at the same rate.



Because the w dimensions are whitened 0<α<1 is stable.

1. Newton filter update Python code (insert only the relevant line)

for k in range(N):

w += [ w[-1] + np.matmul( 2\*alpha\*Rinv,(r\_yx-np.matmul(R\_x,w[-1])))]

|  |
| --- |
| α *is chosen to be 0.5 . Now that Newton is used the weights converge in a straight line, which is the shortes route. The convergence is still smooth.* |

## Unknown statistics

### (N)LMS



for k in range(1,N-1):

inp = x[k-1:k+2]

y\_pred += [np.sum(inp \* w[-1])]

e += [y[k]-y\_pred[-1]]

w += [ w[-1] + 2 \* alpha \* np.array(inp) \* e[-1]]

|  |
| --- |
| *Interestingly, the LMS algorithm does not converge to the center. A possible explanation is that the given have not been predicted accurately for the given input and output data. Alternatively, if the system is dynamic, the optimum for the weights of the system may have been displaced overtime. Consequently, the point the system convergence towards varies.* |

Trade-off choosing :

A high causes the system to converge faster. However, if the is chosen too large the system might end up substantially overshooting the minimum. In the subsequent iteration the system attempts to move in the opposite direction but, overshoots the minimum again. In such a scenario the system will either keep oscillating around the local minimum or it might “jump” out of the valley completely and go in a random direction.

In conclusion the trade-off between the precision and convergence speed has to be made to choose .



for k in range(1,N-1):

inp = np.array(x[k-1:k+2])

y\_pred += [np.sum(inp \* w[-1])]

e += [y[k]-y\_pred[-1]]

sigma = np.matmul(inp.T,inp)/3 + eps

w += [ w[-1] + 2 \* alpha/sigma \* inp \* e[-1]]

|  |
| --- |
|  |

### RLS

RLS is related to the previous algorithms in the sense that it attempts to minimize a squared error and converge to the Wiener solution. However, the previously discussed methods contain various drawbacks that either hinders the convergence efficiency, may cause instability or require knowledge on the autocorrelation matrix .

The problem in LS is that is required in order to determine the optimal wiener solution. In order to obtain expensive computations are required and the result can be unstable. Gradient descent avoids this issue completely by estimating the optimum. However, gradient descent converges to the optimal solution in a curved path instead of a more efficient straight line. Furthermore, the algorithm needs in order to compute solutions. Like gradient descent, Newton uses knowledge of the matrix to update its weights. However, by using the input process can be whitened and thus Newton convergences in a straight line to the optimal solution. NLMS and LMS are different from the gradient descent and Newton methods since they estimate , and thus do not depend on knowledge of beforehand. However, these methods do not withen the input processes and thus often they do not converge in an efficient straight line to the optimal solution.

RLS iteratively estimates as well in order to include withening of the input process. Therefore, it has efficient convergence like the Newton method without depending on the knowledge of beforehand.

If is increased, less weights is given to older samples. They will be “forgotten”

If is decreased, older samples will be taken into account more for the new results.

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | Computational complexity | Convergence speed/stability/accuracy |
| LMS | 1 | 3 |
| NLMS | 2 | 2 |
| RLS | 3 | 1 |

The training chosen training method may have a significant impact for the computational complexity. Especially, the matrix multiplications within the training methods will require a large number of computations if a larger order is chosen for the filter.

NLMS is more computationally expensive because it adds an additional matrix multiplication to the LMS method in order to normalize the step size. However, RLS is the most expensive because it requires far more computation steps, most of them including matrix multiplications.

*In accordance with the results in this report LMS converges to the least accurate solutions. With the time-varying step-size of the NLMS a improvement in accuracy can be obtained since the step size can be adjusted to better fit the scenario. The RLS attempts to withen the input process which improves the efficiency of the convergence. Consequently, RLS leads to the most desirable convergence behavior.*