**Machine Learning for Signal Processing**

**[5LSL0]**

**Assignment 1: Optimum Linear Filters**

**REPORT**

**Group number: -**

**Names including ID:  
 1:  
 2:Frouke Hekker 0897373**

**Date:**

## Known statistics

### Wiener filter








### 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, 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]]

|  |
| --- |
|  |

Trade-off choosing :

A high makes the convergence go faster, however if is too large it might step over the minimum. When it “bounces” back it might also not hit the minimum. In the worst case the w might “jump” out of the valley completely and go in a random direction.

A low makes convergence slow.



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

The problem in LS is that the optimal wiener solution uses . This is expensive to compute and can be unstable. Gradient descent avoids this completely by estimating the optimum. N)LMS improves on that by directly estimation the gradient. This way there is no need to know . RLS iteratively estimates so it is closer to the optimal solution while avoid the problematic computation.

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.

|  |
| --- |
| *Insert plot here* |

|  |  |  |
| --- | --- | --- |
|  | 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 number of taps is chosen for the filter.*

*NLMS is more computational expensive because it adds an additional matrix multiplication to LMS method in order to normalize the step size. RLS is the most expensive because it requires many computation most including matrix multiplications.*

*In accordance with the results in this report LMS has the worst convergence. Normalizing the dimensions gives a straighter path, thus NLMS is better. RLS goes back to the optimal solution and approximates that, therefore it is the best solution.*