# Project II: Estimation Theory

## Maximum likelihood, CRLB & Least Squares Estimation

Simon Andreas Bjørn

February 28, 2023

# » Maximum Likelihood Estimator

We want to derive the maximum likelihod esimator for the time delay of a pulse in additive gaussian noise. The sent pulse $s(t)$ is received as $x(t)$ after the time delay $\tau_0 = 2r/c$ which is what we want to estimate.

Model for continues signal

$$x(t) = s(t - \tau_0) + W(t)$$

* $W(t)$ = AGN
* $s(t - \tau_0)$ = received signal.
* Assume $W$ is bandlimited by $B_w$.
* Sample $W(t)$ with $\Delta = \frac{1}{2} B_w$ s.t. $W[n] = W(n\Delta)$.
* Sampeling the entire noise bandwidth, we can assume the noise to be white.

Model for discrete signal

$$x[n] = s[n - n_0] + W[n]$$

* Discrete time-delay is given by $\tau_0 = n_0 \Delta$
* $s[n]$ is non-zero only during pulse length:

$$x[n] = \begin{cases} W[n] & , 0 \leq n \leq n_0 - 1 \\ s[n - n_0] + W[n] & , n_0 \leq n < n_0 + M \\ W[n] & , n_0 + M \leq n \leq N \end{cases}$$

# » Maximum Likelihood Estimator 2

White gaussian noise means $W[n]$ is independent, so the PDF of $s[n; n_0]$ can be factorized into the PDFs of each sub-interval. Split into 3:

1. $x[n] = W[n],\ 0 \leq n \leq n_0 - 1$:

$$\prod_{n=0}^{n_0-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2[n]}{2\sigma^2}\right)$$

2. $x[n] = S\left[n - n_0\right] + W[n],\ n_0 \leq n \leq n_0 + M - 1$:

$$\prod_{n=n_0}^{n_0+M-1} \frac{1}{\sqrt{(2\pi\sigma^2)}} \exp\left(-\frac{\left(x[n] - s[n - n_0]\right)^2}{2\sigma^2}\right)$$

3. $x[n] = W[n],\ n_0 + M \leq n \leq N_1$:

$$\prod_{n=n_0+M}^{N-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2[n]}{2\sigma^2}\right)$$

# » Maximum Likelihood Estimator 3

We can expand and factorize 1., 2. & 3.

$$P(x; n_0) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^{N-1} \exp\left(-\frac{\sum_{n=0}^{N-1} x^2[n]}{2\sigma^2}\right)$$
$$\cdot \exp\left(-\frac{1}{2\sigma^2} \underbrace{\sum_{n=n_0}^{n_0+M-1} (-2x[n]s[n-n_0] + s^2[n-n_0])}_{\text{We want to maximize this}}\right)$$

So we have

$$\underbrace{2}_{\text{const.}} \cdot \underbrace{\sum_{n=n_0}^{n_0+M-1} x[n]s[n-n_0]}_{\text{maximize}} + \underbrace{\sum_{n=n_0}^{n_0+M-1} s^2[n-n_0]}_{\text{not dependent of } n_0}$$

# » Maximum Likelihood Estimator 4

$s[n]$ is 0 outside of limits so we can expand the sum.

$$\underset{n_0}{\text{argmax}} \left\{ \sum_{n=n_0}^{n_0+M-1} x[n]s[n-n_0] \right\} = \underset{n_0}{\text{argmax}} \left\{ \underbrace{\sum_{n=0}^{N-1} x[n]s[n-n_0]}_{\text{This is just a cross correlation}} \right\}$$

We then end up with the Maximum Likelihood Estimator (MLE) to be the maximum value of the cross correlation of the signal

$$\widehat{n_0} = \underset{0 \leq m \leq N-M}{\text{argmax}} \{C_{xs}[m]\} \quad \text{with} \quad \hat{\tau} = \widehat{n_0}\Delta$$

# » Implementing the ML estimator

We now want to implement the ML esimator.

Note: Unnecessary code is left out. (Styling and so on). If curious, check repo at here

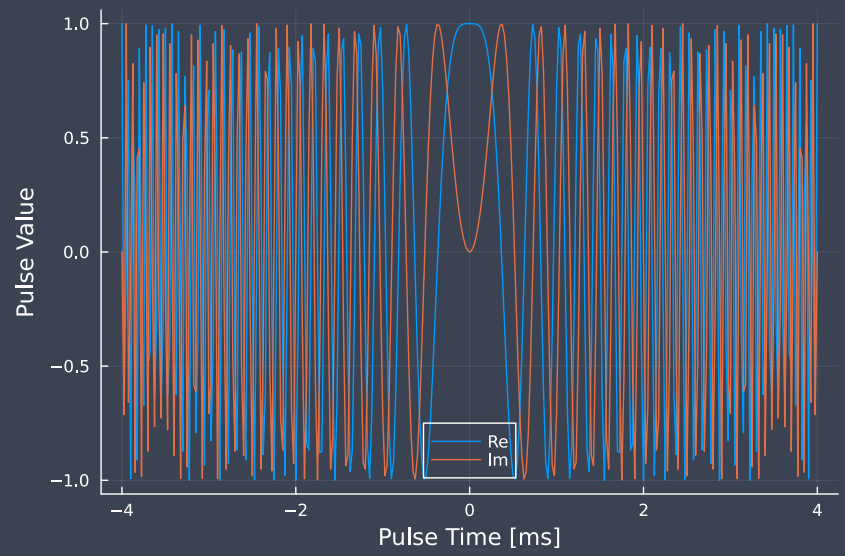
```
#  Start by getting some packages for this project
using DSP; using Plots; using Unitful; using MAT
import Unitful: Time, s, m, Hz
default(background_color=:transparent, foreground_color=:white)

# Import the MAT file
mat_file = matopen("sonardata2.mat")

B    = read(mat_file, "B")    * Hz
fs   = read(mat_file, "fs")   * Hz
c    = read(mat_file, "c")    * m/s
T_p  = read(mat_file, "T_p")  * s
fc   = read(mat_file, "fc")   * Hz
t_0  = read(mat_file, "t_0")  * s
data = read(mat_file, "data")
```

# » Implementing the ML estimator 2

Now we can define the signal and plot it over time

```
 1 # Signal model
 2 α = B / T_p
 3
 4 s_Tx(t::Time) = (-T_p/2<=t<=T_p/2)
 5     ? exp(2π*α*t^2/2*im) : 0.0im
 6 # Convert any input to Time
 7 s_Tx(t) = s_Tx(t*s)
 8
 9 # Plot simple signal
10 t = -T_p/2 : 1/fs : T_p/2
11 plot(t,real.(s_Tx.(t)))
12 plot!(t, imag.(s_Tx.(t)))
```

Plotted output



# » Implementing the ML estimator 3

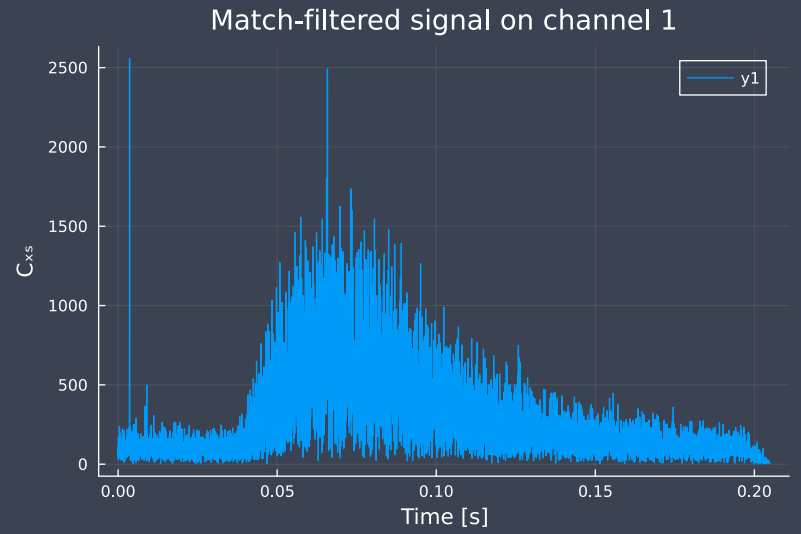
Now we implement the xcorr operation...

```
1  match_filter(ch,signal) = xcorr(ch, signal, padmode=:longest)[end÷2+1:end]
```

and we test it by applying it to 1 channel and the source signal

```
1  ml₁ = match_filter(data[:, 1], s_Tx.(t))
2  plot(
3      axes(ml₁,1)/upreferred(fs),
4      abs.(ml₁),
5  )
```

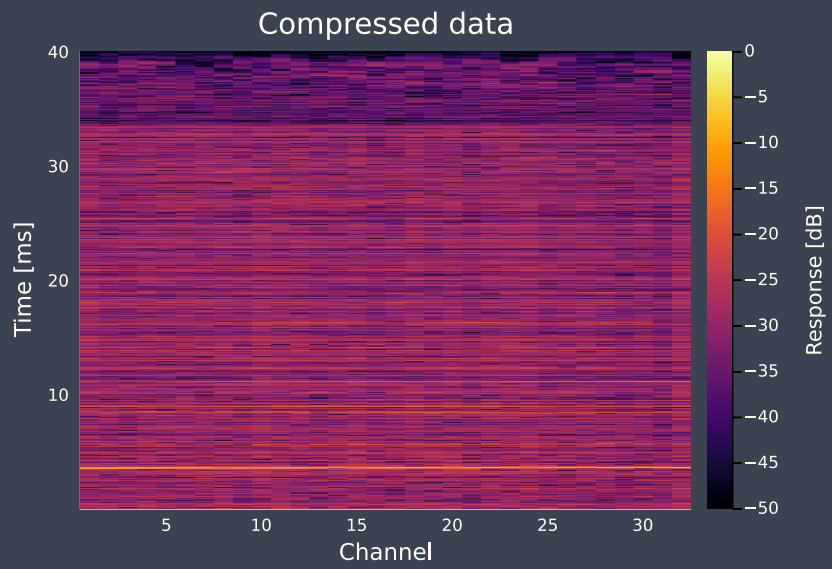
Plot generated from code

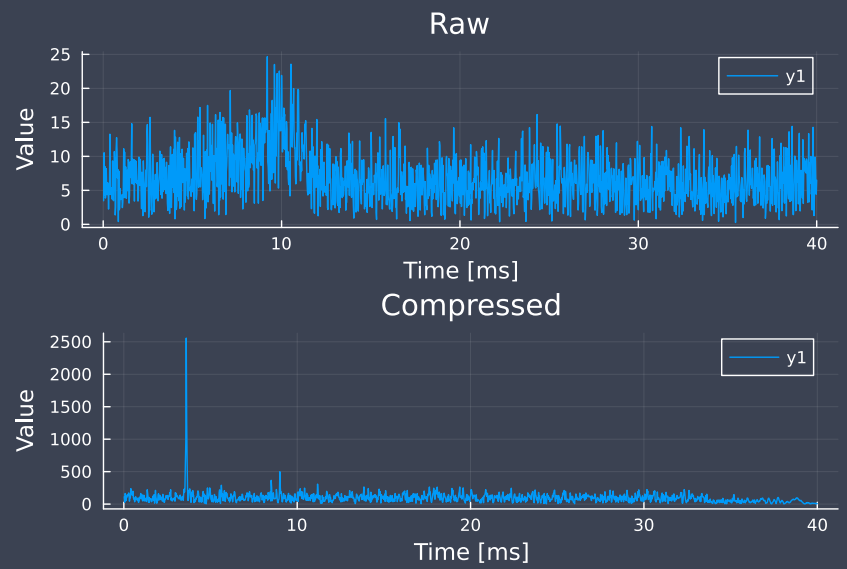


# » Applying the ML estimator

Now we map this as a function on each channel and plot the results.

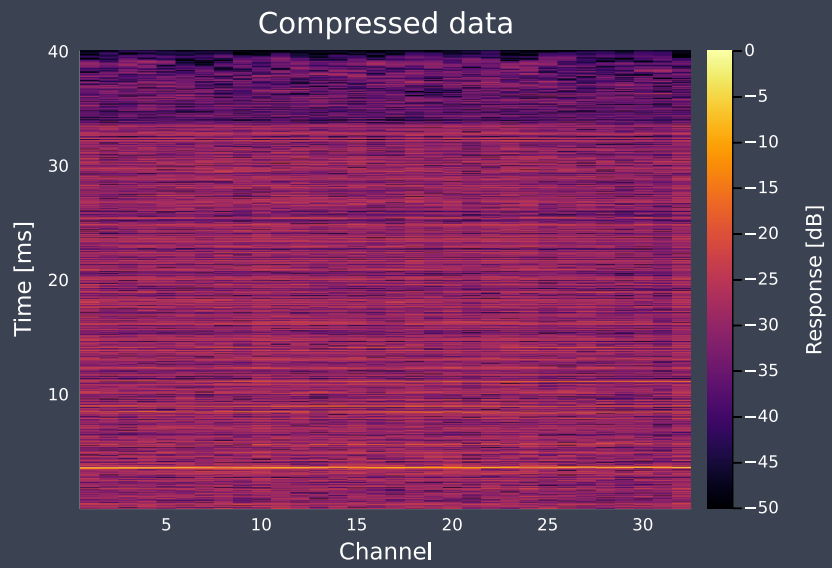
```
compressed = mapslices(
    ch -> match_filter(ch, s_Tx.(t)),
    data[1:1600,:], dims=(1,))

heatmap(axes(compressed,2), axes(compressed,1)/upreferred(fs),
    amp2db.(abs.(compressed)/maximum(abs.(compressed))))
plot([
    plot((1:1600)/upreferred(fs), abs.(d[1:1600,1]))
    for d in [data, compressed]
]..., layout=[1,1],)
```

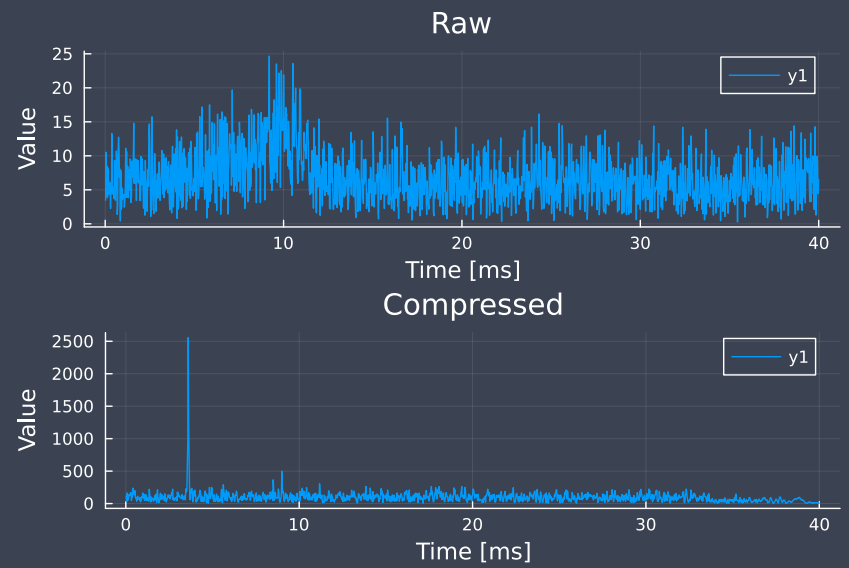


# » Applying the ML estimator 2

* Looking at the 1600 samples of the raw data, we see that it looksquite noisy, but there seems to be some kind of signal around 0.01s.
* When we perform the cross correlation, we see that a strong peak appear at about 5ms. By the ML estimator this is when it is most likely when we recieve the pulse.



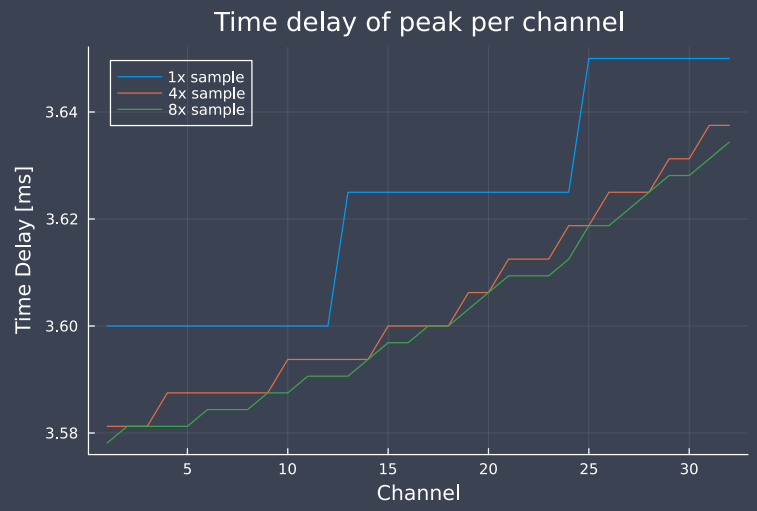



# » Estimate time delay using ML

We now want to estimate the time delay across all channels of the first 1600 samples using the ML estimator.

```
match_filter(ch,signal) = xcorr(ch, signal, padmode=:longest)[end÷2+1:end]
mle(ch, signal) = match_filter(ch, signal) .|> abs |> argmax
s0 = s_Tx.(t)
# Resample the signals
s4 = resample(s0, 4, dims=(1,))
s8 = resample(s0, 8, dims=(1,))

data_x0 = data[1:1600,:]
# Resample the data
data_x4 = resample(data[1:1600,:], 4, dims=(1,))
data_x8 = resample(data[1:1600,:], 8, dims=(1,))

# Perform mle on each channel
τx0 = mapslices(x -> mle(x, s0), data_x0, dims=(1,))
τx4 = mapslices(x -> mle(x, s4), data_x4, dims=(1,))
τx8 = mapslices(x -> mle(x, s8), data_x8, dims=(1,))
```

# » Estimate time delay usin ML 2




* We can see that upsampling allows the estimator to evaluate the time delay with more points. We can see this in how the 1x sample snapps to 0.01ms which corresponds with the sampling frequency, while the other allows the timedelay to be estimated at 4x and 8x the sampling frequency respectively.

# » Cramér-Rao Lower Bound of time delay

We now want to apply the CRLB on the time delay estimator. I choose to find the CRLB using the SNR of the match filtered data. We have that the CRLB is implemented by

$$\text{Var}\{\hat{\tau_0}\} = \frac{1}{\text{SNR} \cdot \beta^2_{\text{rms}}}$$

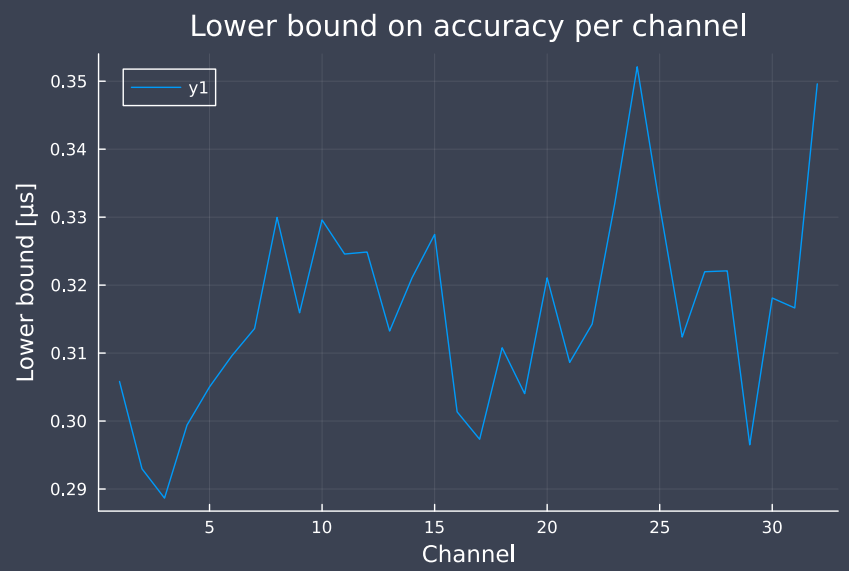
I start by finding $\beta^2_{\text{rms}}$ and the SNR.

```
# β²ᵣₘₛ is streight forward
β²ᵣₘₛ = (2π)^2*fc^2

# Map a function that gets the value of the peak, on each channel
Signal = map(ch -> abs.(compressed[τₓ₀[ch], ch]), 1:32)
# Map a function that gets the value of each sample except the peak,
# on each channel. Then, pass channel to mean.
Noise =  map(ch -> abs.(compressed[1:1600 .!= τₓ₀[ch], ch]), 1:32) .|> mean

SNR = Signal./Noise
```

# » Cramér-Rao Lower Bound of time delay 2

With each factor of the descriminator found, get the CRLB and plot it per channel

```
CRLB = 1 ./ (SNR.*β²ᵣₘₛ)
std = sqrt.(CRLB) .|> toUnit(u"μs")
plot(std)

waveperiod = 1/fc |> toUnit(u"μs")
res = [minimum(std)/waveperiod,
       maximum(std)/waveperiod]
@printf "Lower limit for accuracy:
        (%.3f - %.3f) wave periods"
        res...
# → Lower limit for accuracy: (0.029
        - 0.035) wave periods
```

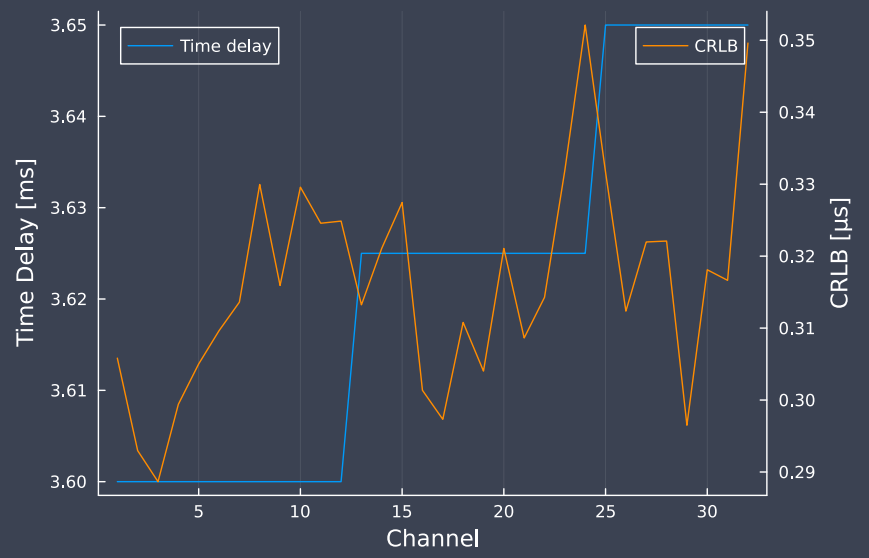



We see that not every channel has the same accuracy, and that is fluctuates between 0.029 and 0.035 wave periods.

# » Cramér-Rao Lower Bound of time delay 3

Overlaying the plot of $\tau_0$ per channel, we get the following plot.

We can see that the CRLB dips, for the channels at the center of time delay estimate "step".
I interpret this as the estimator to be more accurate at the channels in the center of each step while the transitions where the estimate changes, is more inaccurate.



# » Least Squares Estimation

Implementing the least squares estimator by using the algorithm from lecture 5.

* Upsample by 8 to get the more samples around the peak

```
upsample = 8
x = resample(compressed[:,1], upsample) .|> abs .|> x->x^2
N = size(x,1)
```

* Select the samples that are around the peak of the signal

```
# Index 1:N on boolian check
fwhm_filter = x .>= 0.5maximum(x)
fwhm_t = ((1:N)./(upsample*fs))[fwhm_filter] .|> toUnit(u"ms")
fwhm_x = x[fwhm_filter] .|> log
```

* Create observation matrix and perform the least squares fit

```
H = [r^c for r=fwhm_t, c=0:2]
θ = ustrip(H) \ fwhm_x ./ unit.(H[1,:])
# '\' doesn't work with units yet, so using 'ustrip' to remove units from H
        and 'unit' to reapply units to result
```

# » Least Squares Estimation 2

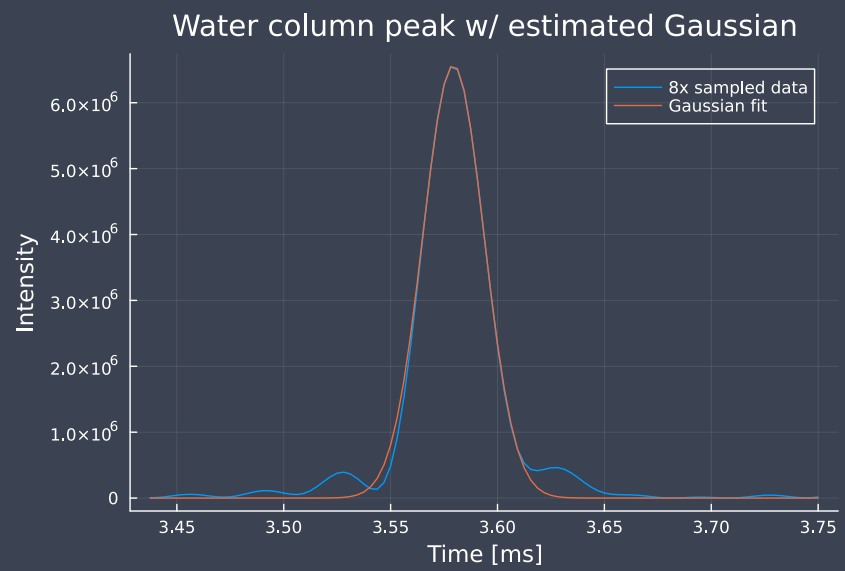
* Using the parameter $\hat{\theta}$, we calculate the time delay, pulse width and intensity

```
1 σ = sqrt(-1/(2θ[3]))          # → 0.0143ms
2 τ = -θ[2] / 2θ[3]             # → 3.5494ms
3 I = exp(θ[1] - θ[2]^2/4θ[3]) # → 6.5722e6
```

* Lastly, recostruct a gaussian with the specified parameters

```
1 g(t) = I*exp(-(t-τ)^2/(2σ^2))
```

Plotting the signal around the peak, we have the following plot



# » Least Squares Estimation 3

Wrapping it into a function of samples, which returns the parameters …

```
function lse(samples, upsample=8)
    x = resample(samples, upsample)
    N = size(x, 1)

    fwhm_filter = x .>= 0.5maximum(x)
    fwhm_t = ((1:N) ./ (upsample*fs))[fwhm_filter] .|> toUnit(u"ms")
    fwhm_x = x[fwhm_filter] .|> log

    H = [r^c for r=fwhm_t, c=0:2]
    θ = ustrip(H) \ fwhm_x ./ unit.(H[1,:])
    return sqrt(-1/(2θ[3])), -θ[2] / 2θ[3], exp(θ[1] - θ[2]^2/4θ[3])
end
```

allows us to map the `lse` function onto the compressed data as a function of the channels

```
τ_lse = mapslices(
    ch -> lse(ch)[2], # For each channel, get the 2nd parameter from lse
    compressed,
    dims=1
)
```

# » Least Squares Estimation 4

* We see that the time delay estimate is larger for later channels.
* Comparing the time delay from LSE to that of the MSE, we see that the MSE stagger between channels, while the LSE is a lot smoother.
* MSE estimates the time delay to be the most likely sample at which it can occur
* LSE interpolates the time delay to be a best possible fit to a quadratic polynomial.