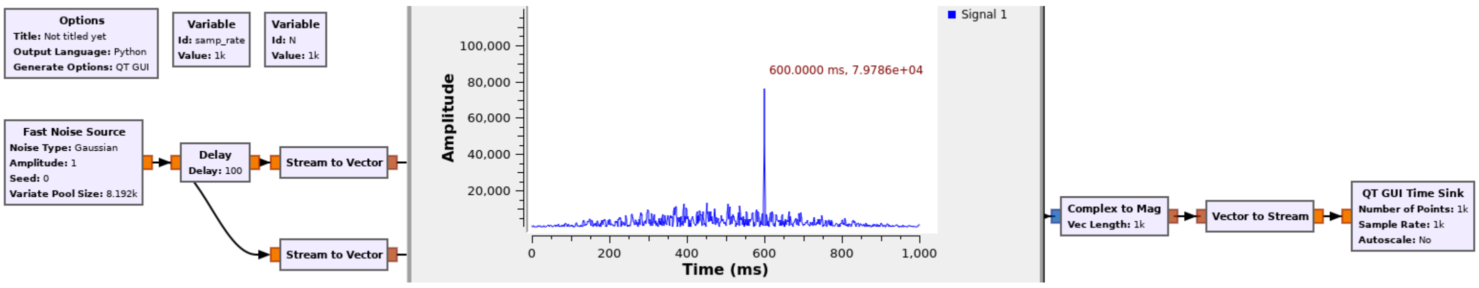
We have extensively discussed cross correlation as a core part of modern digital communication for detecting the preamble of resource sharing communication protocols, whether CDMA or FDMA. The GNU/Octave cross correlation implementation is provided at `https://sourceforge.net/p/octave/signal/ci/default/tree/inst/xcorr.m` Analyzing this code should allow for answering:

1. What is the definition of the cross-correlations between time dependent signals $x(t)$ and $y(t)$?

2. How is the cross-correlation implemented in GNU/Octave, according to the link provided in the introduction ? What fundamental theorem does this implementation rely on ?

3. What is the reason for the difference between answers to questions 1 and 2 ? Justify quantitatively on a 16384-sample long cross-correlation.

4. Based on 2, provide a flowchart returning the time delay τ between a random source and the time delayed copy of this same source using the processing blocks available in GNU Radio Companion. Ideally, provide the GNU Radio Companion flowchart and a screenshot demonstrating the result with a delay of 100 samples (see provided sample for source and sink ... fill the blocks hidden under the output image). How is the output image provided below analyzed in this context ?


Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 1k

Variable
Id: N
Value: 1k

Fast Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 0
Variate Pool Size: 8.192k

Delay
Delay: 100

Stream to Vector

Stream to Vector

Complex to Mag
Vec Length: 1k

Vector to Stream

QT GUI Time Sink
Number of Points: 1k
Sample Rate: 1k
Autoscale: No

5. We have introduced two variables in these questions: the time delay τ between datasets and the dataset length t. How does each quantity impact on the cross-correlation ? Analyze the `xcorr` function in case $\tau \ll t$. How can computation time be improved under such an assumption ?

Answers

1. $xcorr(x(t),y(t))(\tau)=\int_{-\infty}^{+\infty}x(t)\times y(t+\tau)dt$ in continuous time, discretized as

$$xcorr(x,y)(m)=\sum_{k=1}^{N}x_n\times y_{n+m}$$

with N the length of datasets x and y, for $m\in[-N:N]\in\mathbb{Z}$.

2. Thanks to the convolution theorem relating the convolution $conv(x(t),y(t))(\tau)=\int_{-\infty}^{+\infty}x(t)\times y(\tau-t)dt$ to its Fourier transform FT relation $FT(conv(x,y))=FT(x)\cdot FT(y)$, and since time of the second argument is reversed as achieved with the complex conjugate of the complex argument, then $FT(xcorr(x,y))=FT(x)\cdot FT^*(y)$ with * the complex conjugate. As the Fourier transform is efficiently implemented as a fast Fourier transform with $N\log_2(N)$ complexity for datasets of length N, the gain becomes significant over the complexity N^2 (N values of τ each requiring N multiplications in the discretized integral) for large N.

   On the provided url including Octave's implementation of `xcorr`, the core function is

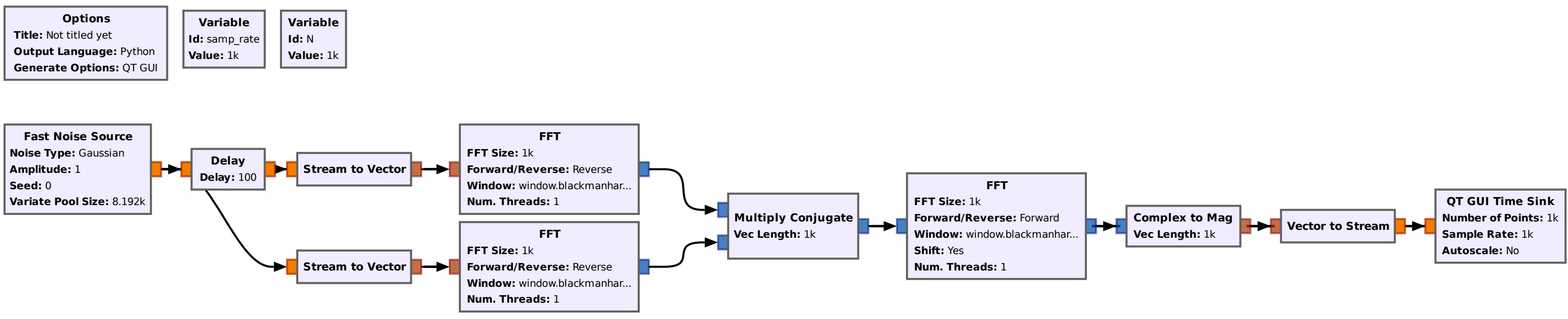
```
function [R, lags] = xcorr (X, Y, maxlag, scale)

  N = max(length(X),length(Y));
  maxlag=N-1;
  M = 2^nextpow2(N + maxlag);
  pre  = fft( postpad( prepad( X(:), length(X)+maxlag ), M) );
  post = fft( postpad( Y(:), M ) );
  cor = ifft( pre .* conj(post) );
  R = cor(1:2*maxlag+1);
```

   so indeed the cross-correlation is computed as the inverse Fourier transform of the product of the Fourier transform of one dataset, times the complex conjugate of the Fourier transform of the second dataset.

3. computing in the time domain the N values of τ, each requiring N multiplications, requires a total of 268 million multiplications if $N=16384$, while the Fast Fourier transform with its $N\times\log_2(N)$ complexity, only requires 230 thousand multiplications. The gain of 230000 will have a significant impact on the computation time needed on a digital computer processing discretized samples.

4. Based on 2, provide a flowchart returning the time delay τ between a random source and the time delayed copy of this same source using the processing blocks available in GNU Radio Companion. Ideally, provide the GNU Radio Companion flowchart and a screenshot demonstrating the result with a delay of 100 samples (see provided sample for source and sink ... fill the blocks hidden under the output image). How is the output image provided below analyzed in this context ?


Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 1k

Variable
Id: N
Value: 1k

Fast Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 0
Variate Pool Size: 8.192k

Delay
Delay: 100

Stream to Vector

FFT
FFT Size: 1k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Num. Threads: 1

Stream to Vector

FFT
FFT Size: 1k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Num. Threads: 1

Multiply Conjugate
Vec Length: 1k

FFT
FFT Size: 1k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Complex to Mag
Vec Length: 1k

Vector to Stream

QT GUI Time Sink
Number of Points: 1k
Sample Rate: 1k
Autoscale: No

5. the prototype of the `xcorr` function introduces the `maxlag` variable which is the expected range of τ.

```
function [R, lags] = xcorr (X, Y, maxlag, scale)

  N = max(length(X),length(Y));
  maxlag=N-1;
  M = 2^nextpow2(N + maxlag);
```

Indeed the sample length `N` and delay `maxlag` have different meanings:

- the sample length can be considered as the averaging duration over which noise will be reduced or energy accumulate coherently when the signal is detected. In the pulse compression definition of the processing gain time$\times$bandwidth, the sample length is the time parameter and the processing gain will be increased by increasing the sample length
- the time delay `maxlag` is related to the physical process delaying the reference signal in the measurement signal, and might be much smaller than the integration time.

Let us consider the case of RADAR signals: the time delay is given by the furthest target the receiver can detect, and might only be a few discretized samples in the future. On the other hand, the sample length will provide noise rejection capability and long samples are required for far target detection. As a numerical example, sampling at 100 MHz for 1.5 m range resolution, a target located at 300 m will appear as a correlation peak at sample 2 μs or index 200 (since the sampling period is 10 ns). Hence, `maxlag` can be limited to a few hundred samples. On the other hand, typical integration duration involved several tens of thousand samples for efficient pulse compression, or a ratio of about 100 between `N` and `maxlag`. We see in the listing sample above that the datasets are pre- and post- padded with a number of empty samples to adapt the vector length to the output: the input datasets with length `N` must be 0-padded to reach a length of `2N+1` ranging from `-N` to `+N` the length of the output correlation vector. If an assumption is given on `maxlag` $\ll$ `N`, then the pre- and post-padding can be much shorter than N, saving computation time in the FFT computation.