20260122

Normalized Crosscorrelation

Project 1
Normalized Crosscorrelation

Homework
Normalized Crosscorrelation on a Spreadsheet

Normalized Crosscorrelation

References

[Proakis] (book) John Proakis and Dimitris Manolakis: *Digital Signal Processing*, 4th ed, 2007.

[Walpole] (book) Walpole, et al: *Probability and Statistics for Engineers and Scientists*, Pearson, 9th ed, 2012. *(we need the material on correlation)*

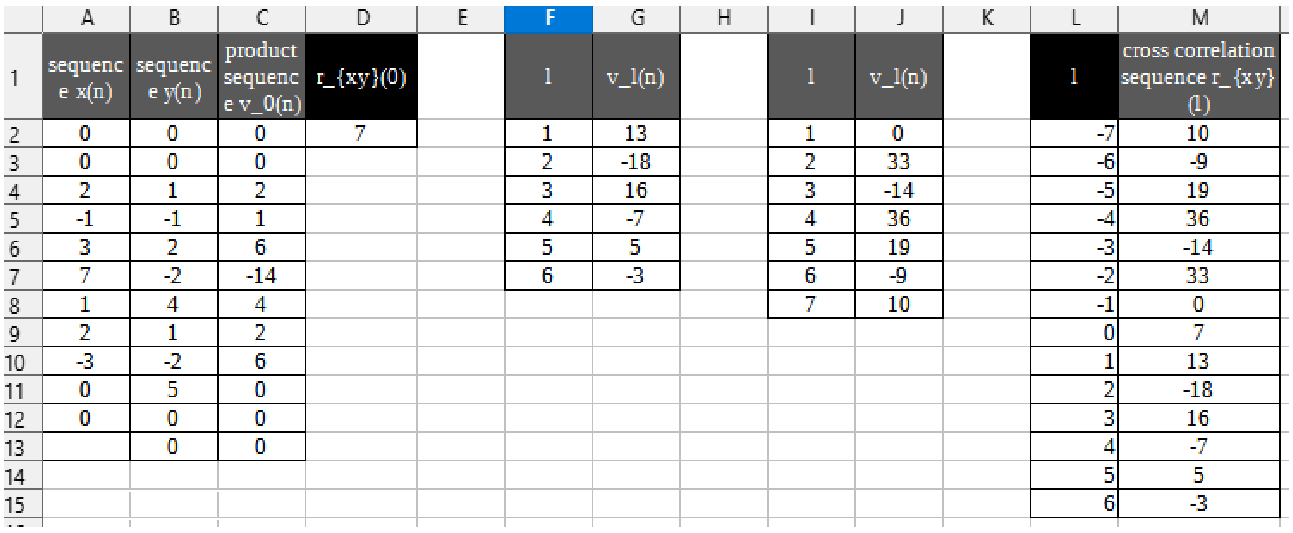
Crosscorrelation

* How similar are two signals?

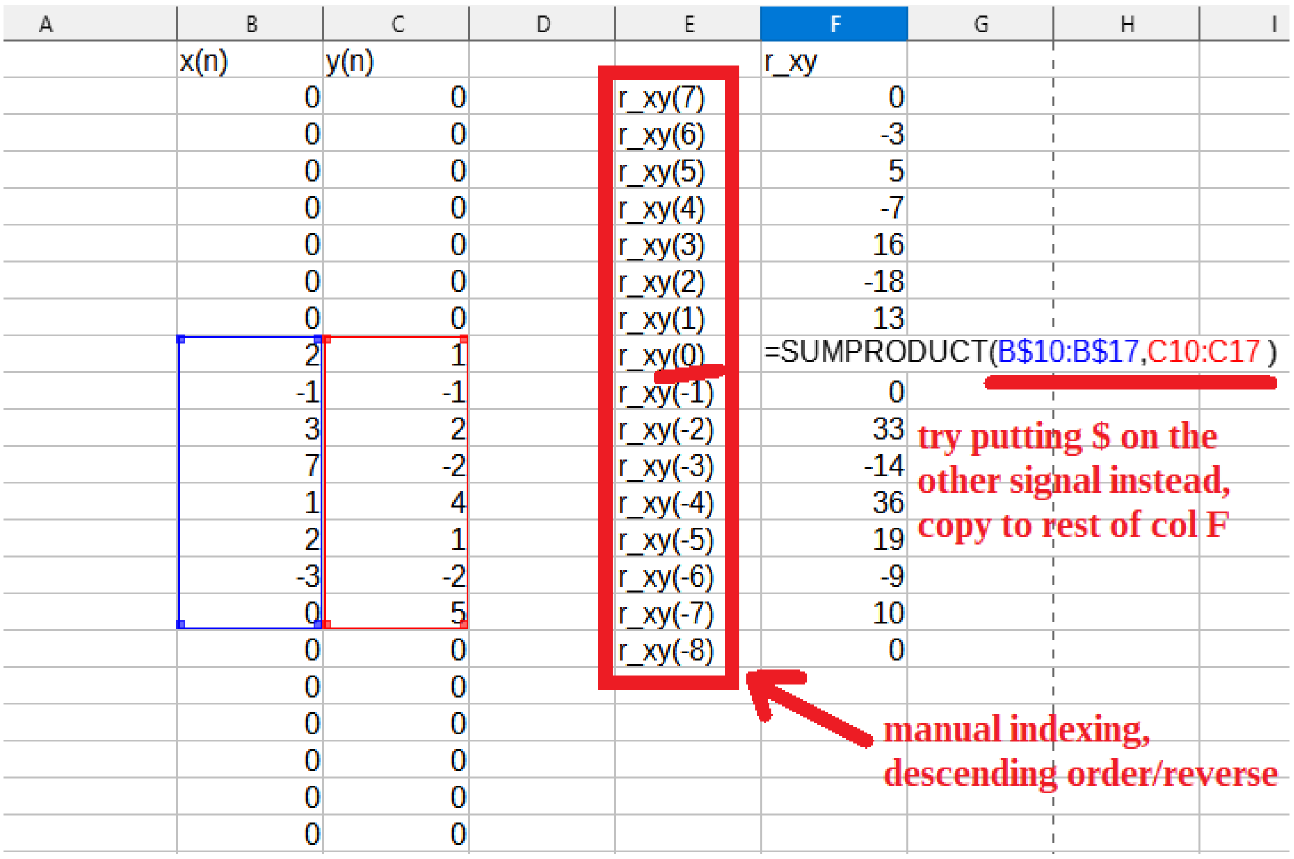
* Can we move one signal left or right for a better alignment?
  (so they would look more similar)

Study Example 2.6.1 on computing the crosscorrelation. Use the `sumproduct()` function (in LibreOffice Calc) in your spreadsheet to obtain the same results as in the example. Arrange the signal values in columns.

* best if crosscorrelation is produced directly in a single column
(from the two columns containing the signals)



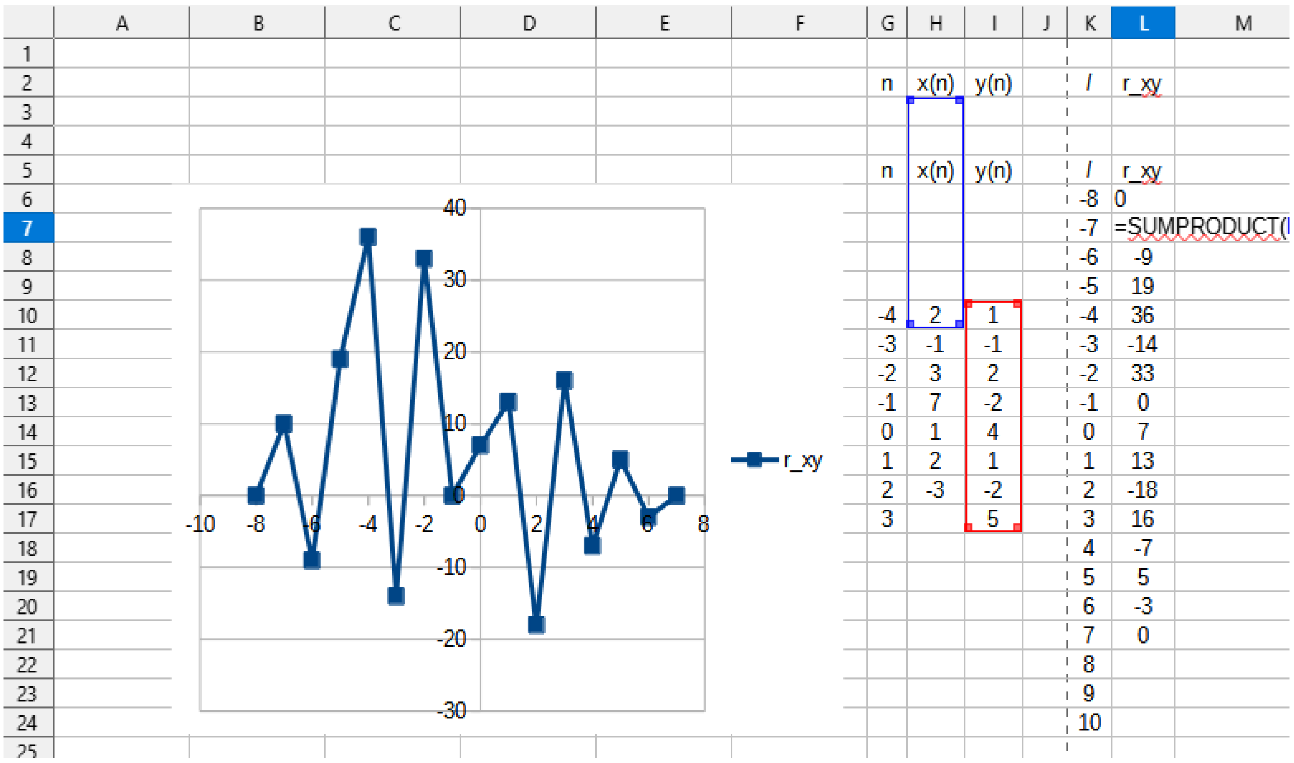
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	sequence x(n)	sequence y(n)	product sequence v_0(n)	r_{xy}(0)		l	v_l(n)		l	v_l(n)		l	cross correlation sequence r_{xy}(l)
2	0	0	0	7		1	13		1	0		-7	10
3	0	0	0			2	-18		2	33		-6	-9
4	2	1	2			3	16		3	-14		-5	19
5	-1	-1	1			4	-7		4	36		-4	36
6	3	2	6			5	5		5	19		-3	-14
7	7	-2	-14			6	-3		6	-9		-2	33
8	1	4	4						7	10		-1	0
9	2	1	2									0	7
10	-3	-2	6									1	13
11	0	5	0									2	-18
12	0	0	0									3	16
13		0	0									4	-7
14												5	5
15												6	-3



A	B	C	D	E	F	G	H	I
	x(n)	y(n)			r_xy			
	0	0		r_xy(7)	0			
	0	0		r_xy(6)	-3			
	0	0		r_xy(5)	5			
	0	0		r_xy(4)	-7			
	0	0		r_xy(3)	16			
	0	0		r_xy(2)	-18			
	0	0		r_xy(1)	13			
	2	1		r_xy(0)	=SUMPRODUCT(B$10:B$17,C10:C17)			
	-1	-1		r_xy(-1)	0			
	3	2		r_xy(-2)	33			
	7	-2		r_xy(-3)	-14			
	1	4		r_xy(-4)	36			
	2	1		r_xy(-5)	19			
	-3	-2		r_xy(-6)	-9			
	0	5		r_xy(-7)	10			
	0	0		r_xy(-8)	0			
	0	0						
	0	0						
	0	0						
	0	0						
	0	0						

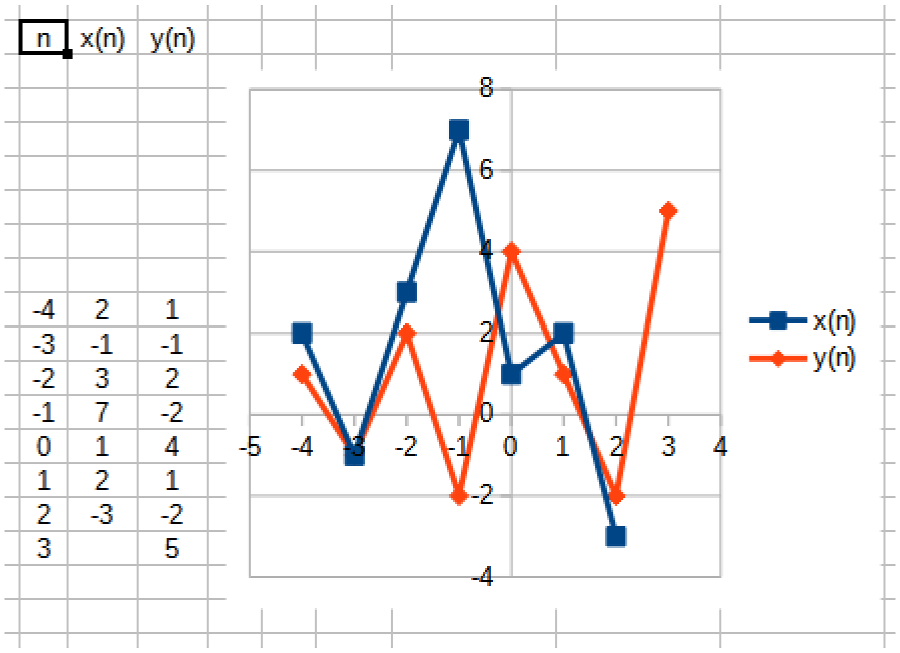
try putting $ on the other signal instead, copy to rest of col F

manual indexing, descending order/reverse



	G	H	I	J	K	L	M
1							
2	n	x(n)	y(n)		*l*	r_xy	
3							
4							
5	n	x(n)	y(n)		*l*	r_xy	
6					-8	0	
7					-7	=SUMPRODUCT(	
8					-6	-9	
9					-5	19	
10	-4	2	1		-4	36	
11	-3	-1	-1		-3	-14	
12	-2	3	2		-2	33	
13	-1	7	-2		-1	0	
14	0	1	4		0	7	
15	1	2	1		1	13	
16	2	-3	-2		2	-18	
17	3		5		3	16	
18					4	-7	
19					5	5	
20					6	-3	
21					7	0	
22					8		
23					9		
24					10		
25							

* if formula at *l=0* is correct, just copy it to other *l* values



n	x(n)	y(n)
-4	2	1
-3	-1	-1
-2	3	2
-1	7	-2
0	1	4
1	2	1
2	-3	-2
3		5

2.6.1 Crosscorrelation and Autocorrelation Sequences

Suppose that we have two real signal sequences $x(n)$ and $y(n)$ each of which has finite energy. The *crosscorrelation* of $x(n)$ and $y(n)$ is a sequence $r_{xy}(l)$, which is defined as

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l), \qquad l = 0, \pm 1, \pm 2, \ldots \tag{2.6.3}$$

or, equivalently, as

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n+l)y(n), \qquad l = 0, \pm 1, \pm 2, \ldots \tag{2.6.4}$$

* peak at l=2 means $x(n)$ aligns best with $y(n - 2)$

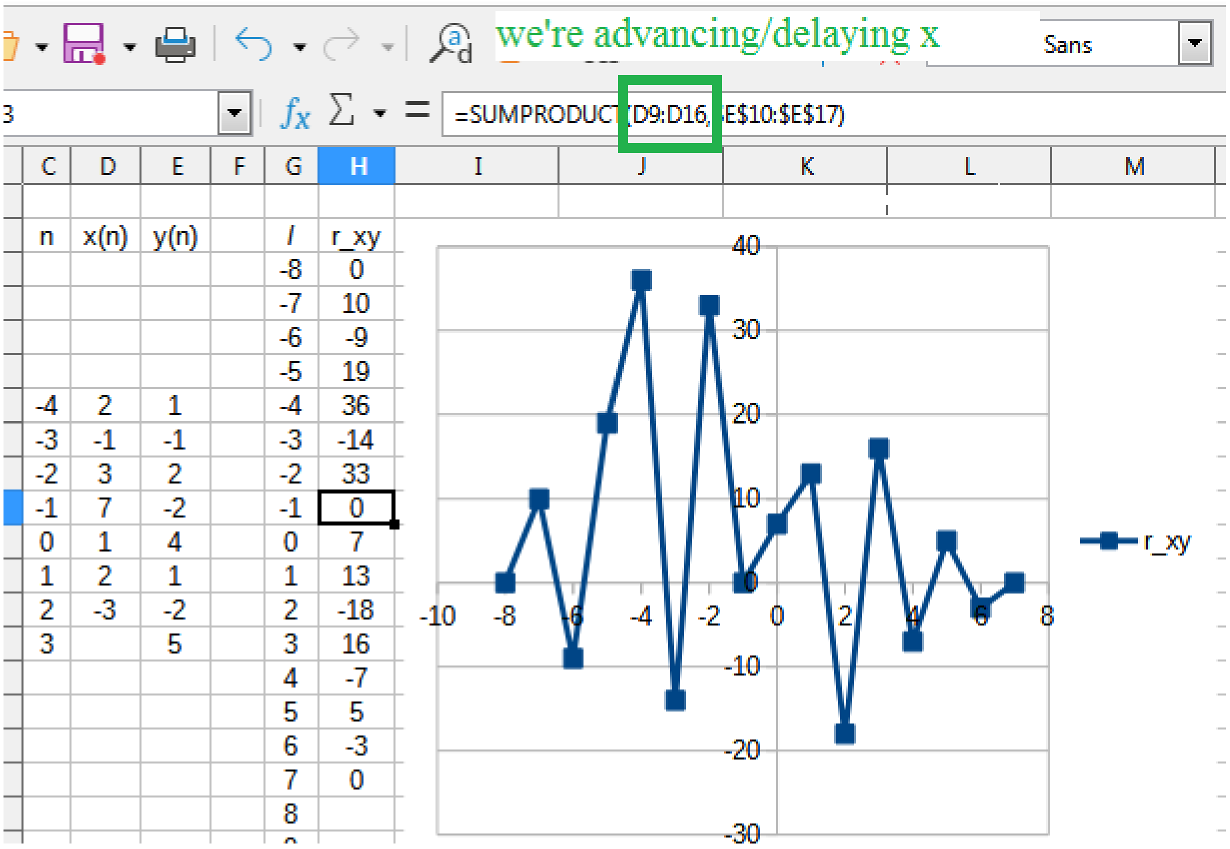
// delay y by 2

// equivalent to advancing x by 2

* peak at l=-3 means $x(n)$ aligns best with $y(n- (-3)) = y(n+3)$

// advance y by 3

// equivalent to delaying x by 3



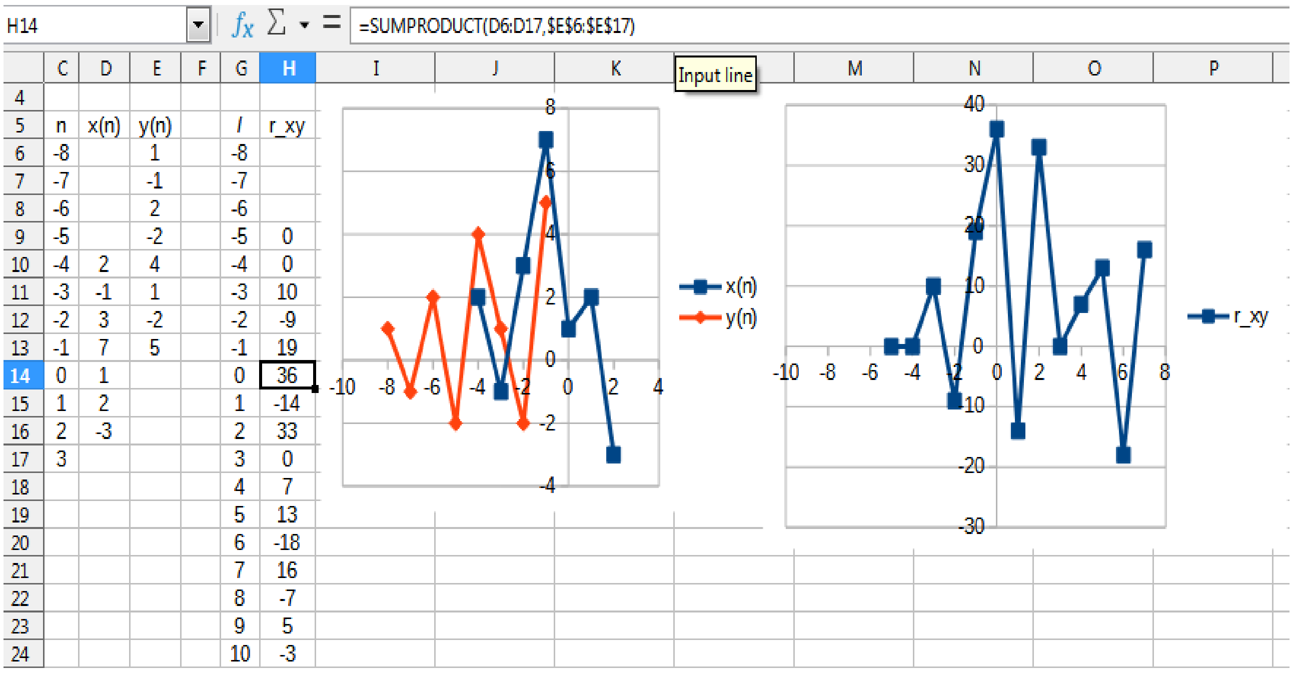
=SUMPRODUCT(D9:D16,$E$10:$E$17)

n	x(n)	y(n)		*l*	r_xy
				-8	0
				-7	10
				-6	-9
				-5	19
-4	2	1		-4	36
-3	-1	-1		-3	-14
-2	3	2		-2	33
-1	7	-2		-1	0
0	1	4		0	7
1	2	1		1	13
2	-3	-2		2	-18
3		5		3	16
				4	-7
				5	5
				6	-3
				7	0
				8	

* max. value at *l* = -4

* close alternative: *l* = -2

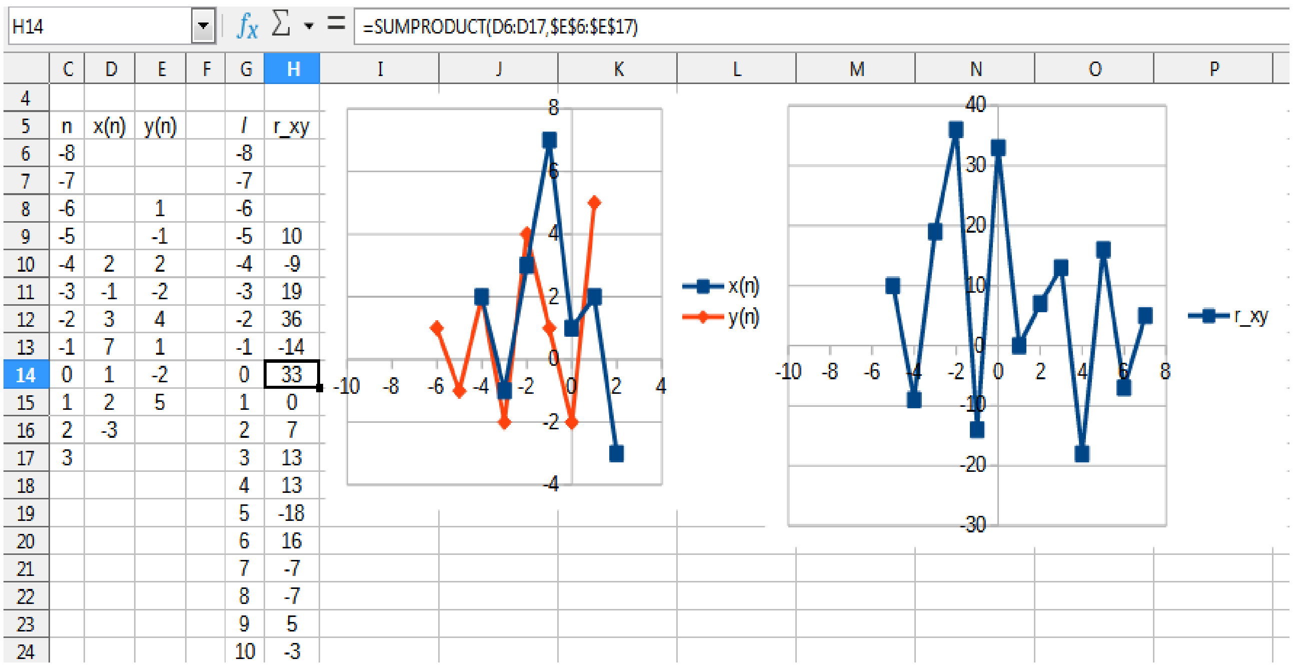
* best alignment: advance *y* by 4 (alternative: advance by 2)



H14 =SUMPRODUCT(D6:D17,$E$6:$E$17)

	n	x(n)	y(n)		*l*	r_xy
4						
5	n	x(n)	y(n)		*l*	r_xy
6	-8		1		-8	
7	-7		-1		-7	
8	-6		2		-6	
9	-5		-2		-5	0
10	-4	2	4		-4	0
11	-3	-1	1		-3	10
12	-2	3	-2		-2	-9
13	-1	7	5		-1	19
14	0	1			0	36
15	1	2			1	-14
16	2	-3			2	33
17	3				3	0
18					4	7
19					5	13
20					6	-18
21					7	16
22					8	-7
23					9	5
24					10	-3

alternative alignment



H14 =SUMPRODUCT(D6:D17,$E$6:$E$17)

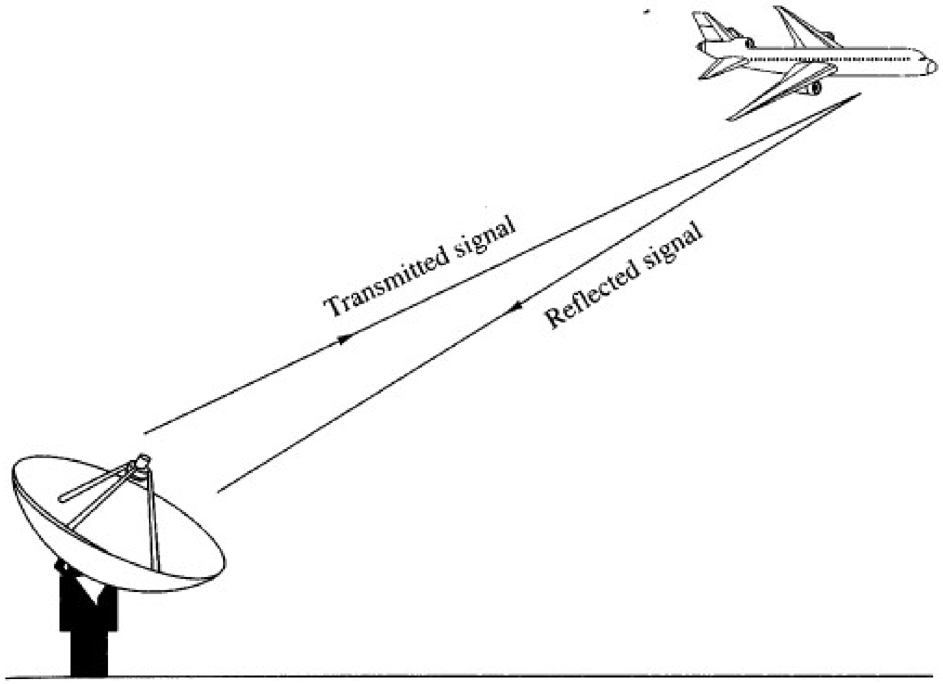
	C	D	E	F	G	H
4						
5	n	x(n)	y(n)		*l*	r_xy
6	-8				-8	
7	-7				-7	
8	-6		1		-6	
9	-5		-1		-5	10
10	-4	2	2		-4	-9
11	-3	-1	-2		-3	19
12	-2	3	4		-2	36
13	-1	7	1		-1	-14
14	0	1	-2		0	33
15	1	2	5		1	0
16	2	-3			2	7
17	3				3	13
18					4	13
19					5	-18
20					6	16
21					7	-7
22					8	-7
23					9	5
24					10	-3

Applications

our objective in computing the correlation between the two signals is to measure the degree to which the two signals are similar and thus to extract some information that depends to a large extent on the application. Correlation of signals is often encountered in radar, sonar, digital communications, geology, and other areas in science and engineering.

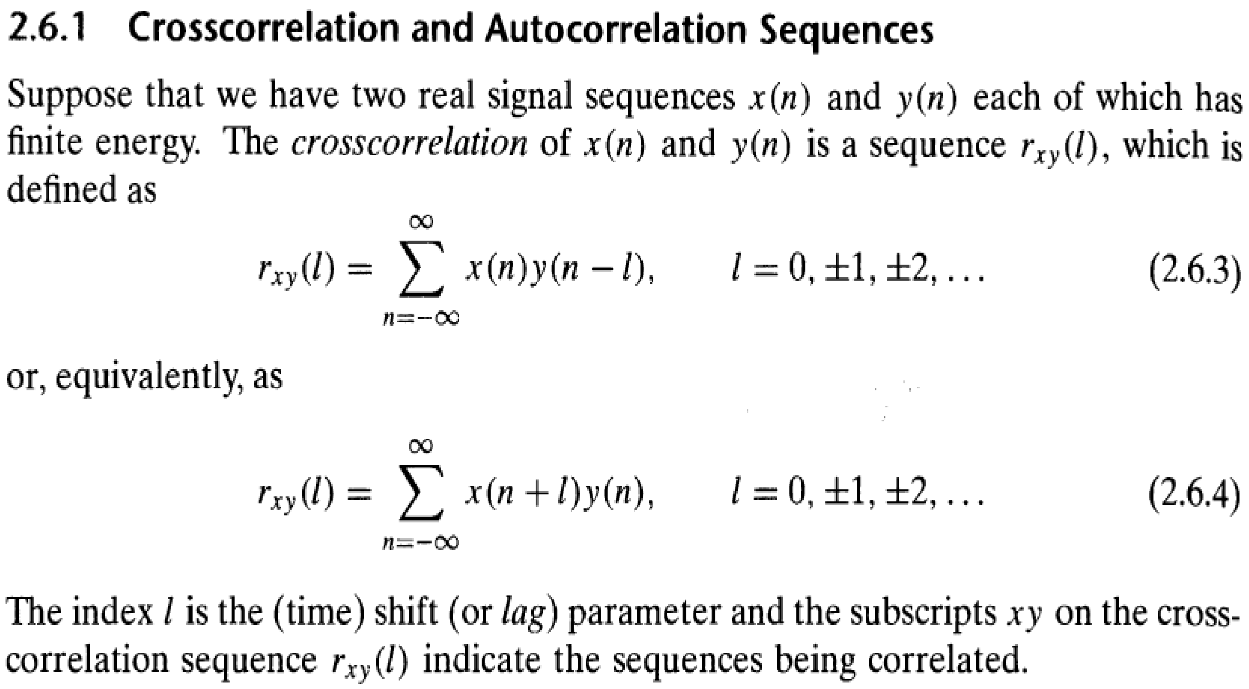
* correlation in machine learning?

Application: Radar



Having the two signal sequences, $x(n)$, which is called the reference signal or transmitted signal, and $y(n)$, the received signal, the problem in radar and sonar detection is to compare $y(n)$ and $x(n)$ to determine if a target is present and, if so, to determine the time delay D and compute the distance to the target. In practice, the signal $x(n-D)$ is heavily corrupted by the additive noise to the point where a visual inspection of $y(n)$ does not reveal the presence or absence of the desired signal reflected from the target. Correlation provides us with a means for extracting this important information from $y(n)$.

Crosscorrelation



2.6.1 Crosscorrelation and Autocorrelation Sequences

Suppose that we have two real signal sequences $x(n)$ and $y(n)$ each of which has finite energy. The *crosscorrelation* of $x(n)$ and $y(n)$ is a sequence $r_{xy}(l)$, which is defined as

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l), \qquad l = 0, \pm 1, \pm 2, \ldots \tag{2.6.3}$$

or, equivalently, as

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n+l)y(n), \qquad l = 0, \pm 1, \pm 2, \ldots \tag{2.6.4}$$

The index l is the (time) shift (or *lag*) parameter and the subscripts xy on the cross-correlation sequence $r_{xy}(l)$ indicate the sequences being correlated.

Note: infinite bounds

* in practice, finite-duration signals will be correlated
= infinite-duration signals where the values are 0 outside the finite duration

* formulas are neater for infinite-duration signals

* some terminology for finite-duration signals

duration = the number of samples

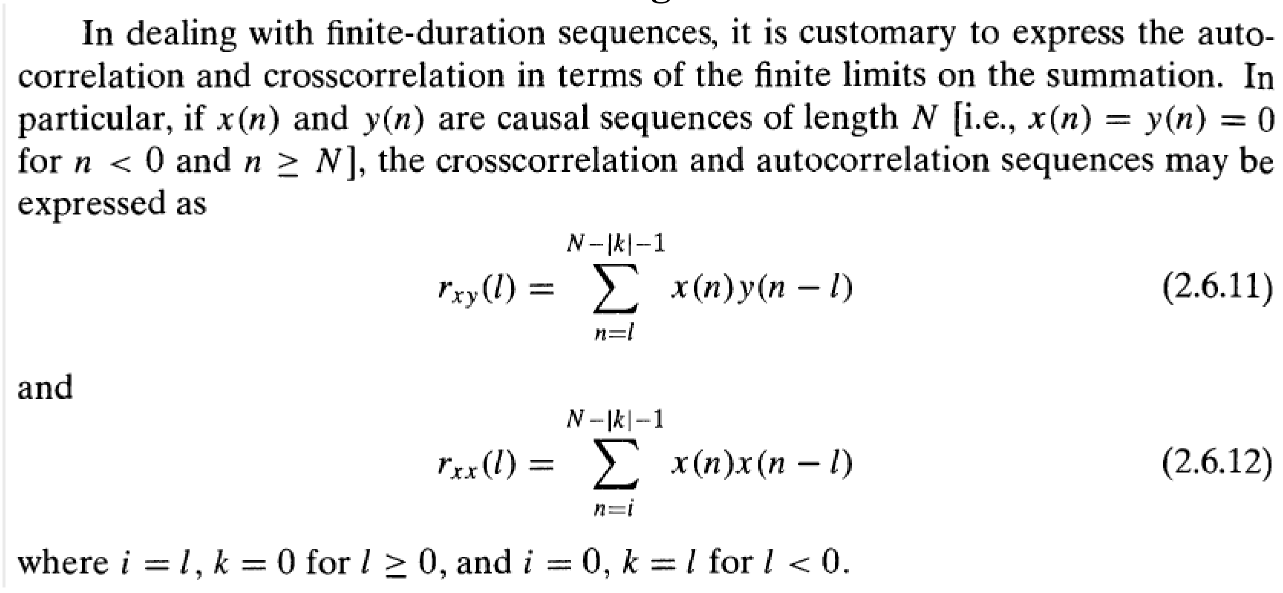
A signal *starts* at $n = a$ if all signal values in the corresponding infinite-duration signal are 0 for $n < a$.

A signal *ends* at $n = b$ if all signal values in the corresponding infinite-duration signal are 0 for $n > b$. The finite-duration signal *lasts* from a to b.

$$duration = b - a + 1$$

A signal lasting from 0 to 99 starts at 0, ends at 99 and has a duration of 100
(100 = 99 - 0 + 1).

Crosscorrelation: Finite-duration signals



In dealing with finite-duration sequences, it is customary to express the autocorrelation and crosscorrelation in terms of the finite limits on the summation. In particular, if $x(n)$ and $y(n)$ are causal sequences of length N [i.e., $x(n) = y(n) = 0$ for $n < 0$ and $n \geq N$], the crosscorrelation and autocorrelation sequences may be expressed as

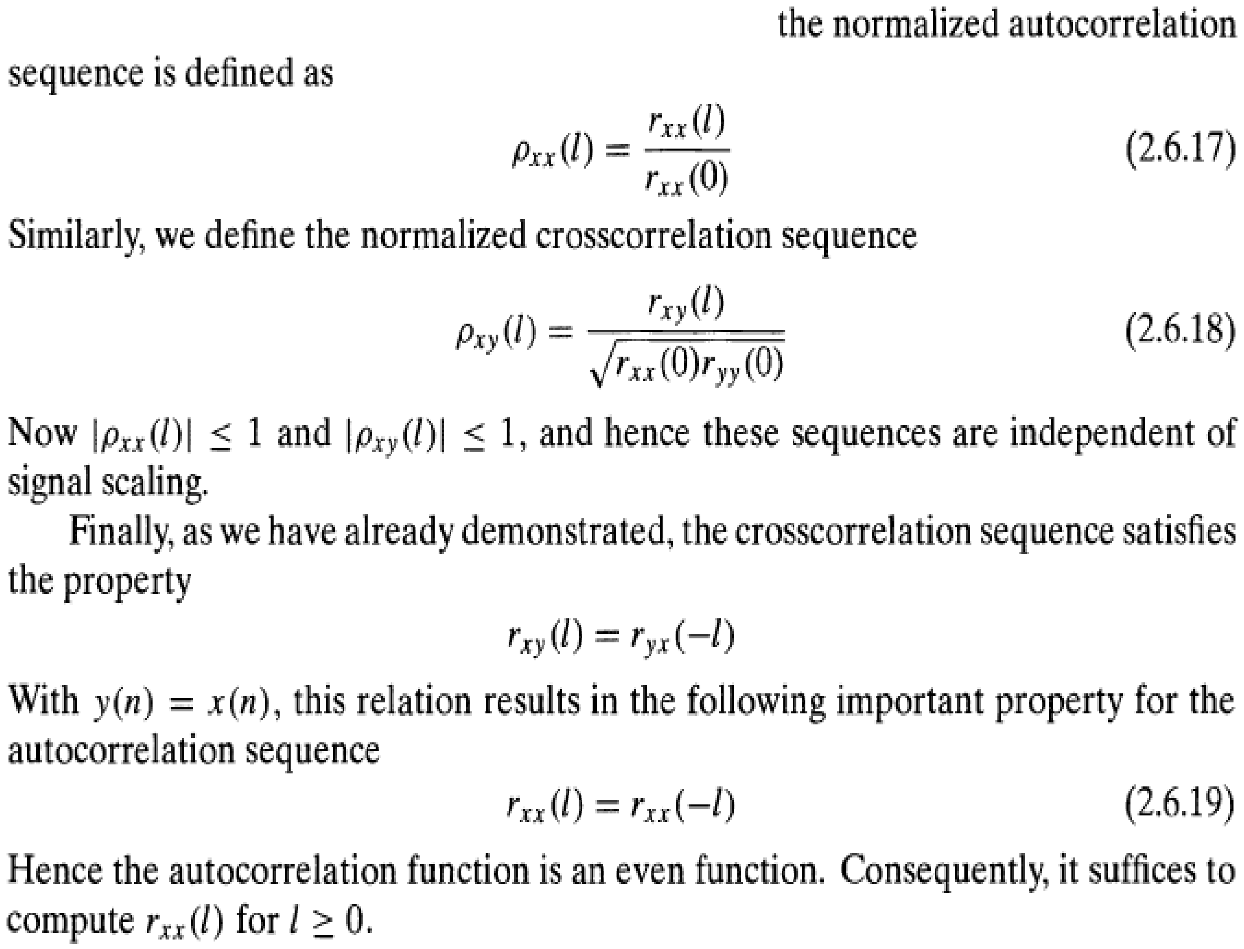
$$r_{xy}(l) = \sum_{n=l}^{N-|k|-1} x(n)y(n-l) \qquad (2.6.11)$$

and

$$r_{xx}(l) = \sum_{n=i}^{N-|k|-1} x(n)x(n-l) \qquad (2.6.12)$$

where $i = l$, $k = 0$ for $l \geq 0$, and $i = 0$, $k = l$ for $l < 0$.

Normalization



the normalized autocorrelation sequence is defined as

$$\rho_{xx}(l) = \frac{r_{xx}(l)}{r_{xx}(0)} \qquad (2.6.17)$$

Similarly, we define the normalized crosscorrelation sequence

$$\rho_{xy}(l) = \frac{r_{xy}(l)}{\sqrt{r_{xx}(0)r_{yy}(0)}} \qquad (2.6.18)$$

Now $|\rho_{xx}(l)| \le 1$ and $|\rho_{xy}(l)| \le 1$, and hence these sequences are independent of signal scaling.

Finally, as we have already demonstrated, the crosscorrelation sequence satisfies the property

$$r_{xy}(l) = r_{yx}(-l)$$

With $y(n) = x(n)$, this relation results in the following important property for the autocorrelation sequence

$$r_{xx}(l) = r_{xx}(-l) \qquad (2.6.19)$$

Hence the autocorrelation function is an even function. Consequently, it suffices to compute $r_{xx}(l)$ for $l \ge 0$.

* always yields values between -1 and 1 (independence of scale)

* values near 1: strongly correlated
(both signals are high/low, increase/decrease at the same time)

* values near -1: strongly anti-correlated
(one signal high when other is low,
increasing when other is decreasing)

* values near 0: no strong correlation

* remember the dot product?

Why normalize?

* independence of scale

Correlation in Statistics

Correlation Coefficient The measure ρ of linear association between two variables X and Y is estimated by the **sample correlation coefficient** r, where

$$r = b_1\sqrt{\frac{S_{xx}}{S_{yy}}} = \frac{S_{xy}}{\sqrt{S_{xx}S_{yy}}}.$$

$$r^2 = \frac{S_{xy}^2}{S_{xx}S_{yy}} = \frac{SSR}{S_{yy}},$$

sample coefficient of determination

where

$$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2, \quad S_{yy} = \sum_{i=1}^{n}(y_i - \bar{y})^2, \quad S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}).$$

* important idea: center signals on horizontal axis

* no shifting (not so realistic, remember radar?)

* finite-duration sequences (more realistic)

*   with the same duration (not so realistic)

Shifting

* some applications depend on shifting

time-domain reflectometry

bathymetry

RADAR

SONAR

* another example: upstream and downstream river water levels

Use crosscorrelation to relate surge upstream to surge downstream.

How much time do people downstream have to react to surge upstream?

* shifting is expensive

no shifting: O (n), where n is the duration (of both signals, for simplicity)

with shifting: O (n^2)
O (n) computation repeated around $2n$ times

find alternatives where posible (Discrete Fourier Transform)

Removal of DC-content

* detect negative crosscorrelation between positive-valued signals

* DC content carries no information

* required for normalization

* avoid where possible

    audio samples that are known to center around 0

General Application

* both shifting and removal of DC content needed

* maximize applications/flexibility

* tradeoff: computational efficiency

.

Correlation in Statistics

* two finite sequences of the same duration
* removes average (centers signals on horizontal axis)
* computes just one coefficient (no shifting)

Correlation in Signal Processing

* no removal of average (ok if signals always center on 0)
* neat formula for infinite-duration signals

Reality

* centering a signal is an important generalization
  (applications outside signal processing,
  non-zero average is misleading)
* may need to correlate signals with different finite durations
  (must shift)

Project 1
Normalized Crosscorrelation

Due: start of class time, February 19, 2026

Progress Report 1	Jan. 29
Progress Report 2	Feb. 5
Progress Report 3	Feb. 12
Early Work Submission	Feb. 16

Digital Signal Processing Project

Normalized Crosscorrelation

Luisito L. Agustin
2025 01 29

General Specifications

Develop an application in C++ for computing the normalized crosscorrelation of two finite-duration signals.

Detailed Specifications

Develop an application in C++ for computing the normalized crosscorrelation of two finite-duration signals.

The application should work when invoked from the command line in the following manner:

```
> xcorr [xdata] [ydata] [output file]
```

where
`xcorr` may be the name of the executable,
`[xdata]` is the filename of the signal file containing the *x* values,
`[ydata]` is the filename of the signal file containing the *y* values, and
`[output file]` is the filename of the output signal file for the crosscorrelation
(">" is the command prompt).

Providing an interactive mode for cases where the user does not provide appropriate command line arguments is optional.

Obtain raw signals $x_{raw}(n)$ and $y_{raw}(n)$ from the text files specified by the user. These text files must comply with the *Signal File Format*. Do not impose any limitations on the sizes of signal files.

From the raw signals, obtain $x(n)$ and $y(n)$ by first obtaining averages x_{ave} and y_{ave}, of the values in $x_{raw}(n)$ and $y_{raw}(n)$, respectively, and defining

$$x(n) = x_{raw}(n) - x_{ave} \text{ for all applicable } n\text{, and}$$
$$y(n) = y_{raw}(n) - y_{ave} \text{ for all applicable } n.$$

The crosscorrelation is computed as

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-l)$$

or

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n+l)\, y(n)$$

for all applicable l depending on the actual extent of the raw signals.

The normalized crosscorrelation is computed from the definition

$$\rho_{xy}(l)=\frac{r_{xy}(l)}{\sqrt{r_{xx}(0)r_{yy}(0)}}$$

where $r_{xx}(0)$ and $r_{yy}(0)$ are the autocorrelations (crosscorrelations of signals with themselves) of *x* and *y*, respectively, evaluated at *l*=0.

The normalized crosscorrelation is itself a signal and should be exported to a signal file of the same format as the input files, with the filename as specified by the user.

Implement all *Recommendations for Handling Signal Files* in the *Signal File Format* specification.

The application provides feedback containing details of files read and written. The feedback must mention the filename, indicate whether a signal was successfully read or written, and specify the duration and start index of the signal if such information is available.
e.g.

“Signal with start index -4, duration 7, imported from x.text”
“Unable to import a valid signal from y.csv”
“Crosscorrelation signal with start index -7, duration 14, exported to xcorr.csv”
“Unable to export signal with start index -7, duration 14, to xcorr.txt”

Implement a correlation function that accepts the raw signals as parameters and computes the normalized crosscorrelation. The normalized crosscorrelation is written to parameters that the function can modify. This function does not call on other functions except possibly those available thru the C++ standard library. All data must be in arrays of `doubles` at this point and passed to the correlation function, directly or indirectly, thru pointers to `double`. Vectors are not acceptable for use with this function.

Possible prototypes (do not deviate significantly from these):

```
void computeNormalizedCrosscorrelation(
  double * xData, int xDuration, int xStart,
  double * yData, int yDuration, int yStart,
  double ** xcorrData, int & xcorrDuration, int & xcorrStart);

void computeNormalizedCrosscorrelation(
   engg151Signal x,  engg151Signal y,  engg151Signal & xcorr);

engg151Signal normalizedXCorr ( engg151Signal x, engg151Signal y);
```

The task of the function is to compute the normalized crosscorrelation. The function may call on C++ standard library functions to do portions of the task. Neither the crosscorrelation function nor any C++ standard library function called by it should get any input from a file or from the console or send any output to a file or the console.

If the duration of the normalized crosscorrelation signal is less than 20, the contents of the signal file representing the normalized crosscorrelation are also shown on the console in addition to being exported to a file.

Do not modify any filename provided by the user. Do not assume, nor require, nor add a ".txt" extension. Use whatever filename is provided.

References

[Proakis] (book) John Proakis and Dimitris Manolakis: Digital Signal Processing, 4th ed, 2007 (Sec. 2.6. Correlation of Discrete-Time Signals).

[Walpole] (book) Walpole, et al: Probability and Statistics for Engineers and Scientists, Pearson, 9th ed, 2012 (Ch. 11. Simple Linear Regression and Correlation).

Revision History - Normalized Crosscorrelation

	in use since 2020-1
2022 02 28	1st formalization as a project specification document
2023 01 25	revised detailed specifications
2024 08 21	required self-contained crosscorrelation function
2025 01 29	improved clarity of some instructions

Variable Specifications

Specifications and instructions from this point onwards may vary from one use of the document to the next. These are not tracked as revisions to the project specifications.

Group Work

Work on the C++ code may be done by groups. Documentation and testing must be done individually.

It is expected that C++ code submitted by groupmates may be similar or exactly the same. Points earned for the entire project will be subject to penalties for unauthorized collaboration if documentation submitted by at least two students contain at least one distinctive phrase, image, or item in common.

Notes

In the definition of the normalized crosscorrelation,

$$\rho_{xy}(l)=\frac{r_{xy}(l)}{\sqrt{r_{xx}(0)r_{yy}(0)}},$$

note that the normalization coefficient $\sqrt{r_{xx}(0)r_{yy}(0)}$ is a constant and needs to be computed only once.

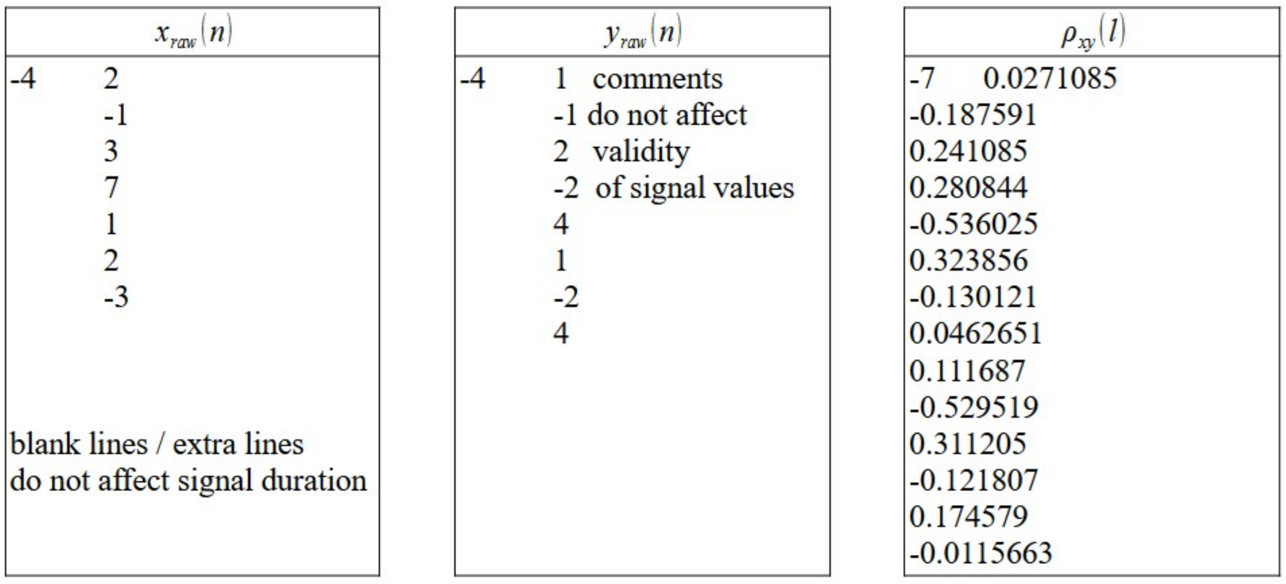
The recommended way of parsing for integers or floating point numbers is to use facililties such as the extraction operator (>>) in the standard library of C++. A properly working parser will most likely take more time to write from scratch than the time allocated for this project. It will most likely work only in very special cases.

Basic Test

$x_{raw}(n)$
-4 2
-1
3
7
1
2
-3
blank lines / extra lines
do not affect signal duration

$y_{raw}(n)$
-4 1 comments
-1 do not affect
2 validity
-2 of signal values
4
1
-2
4

$\rho_{xy}(l)$
-7 0.0271085
-0.187591
0.241085
0.280844
-0.536025
0.323856
-0.130121
0.0462651
0.111687
-0.529519
0.311205
-0.121807
0.174579
-0.0115663



$x_{raw}(n)$
-4 2
-1
3
7
1
2
-3
blank lines / extra lines
do not affect signal duration

$y_{raw}(n)$
-4 1 comments
-1 do not affect
2 validity
-2 of signal values
4
1
-2
4

$\rho_{xy}(l)$
-7 0.0271085
-0.187591
0.241085
0.280844
-0.536025
0.323856
-0.130121
0.0462651
0.111687
-0.529519
0.311205
-0.121807
0.174579
-0.0115663

The application must be able to obtain $\rho_{xy}(l)$ from $x_{raw}(n)$ and $y_{raw}(n)$ as shown.

Submission

Do the basic test with exactly the same data provided. Obtain a screenshot of the application with the results of the basic test on the console. Since the duration of the normalized crosscorrelation in the basic test is less than 20, the contents of the resulting signal file should also be shown onscreen. Take a screenshot of the console at this point with the results onscreen.

Implementation and Testing Environment Report

Provide details of at least one platform used in implementing and testing the project.

* name and version of compiler
* name and version of integrated development environment (IDE)
* operating system name/distribution and version

screenshot of compilation

The screenshot must show the files involved and show feedback from the IDE reporting successful compilation and linking; if the project is compiled on the command line, the screenshot must show the commands issued; if a makefile was used, include a copy of the makefile in addition to the screenshot.

screenshot of program run on the console

Copy the executable to a location whose path contains a recognizable version of your name. If working with a group, make sure that the path does not contain any version of the names of any groupmates. Create a new folder named after yourself as needed. Run the executable by typing its name at the command prompt, and take a screenshot as the executable starts. Do not run the application from within an IDE.

Code locator

Fill in the table locating the specified code items. Fill in this table after the code has been finalized. Inaccurate details will be penalized.

code items	file name	line numbers
correlation function		
declaration of double * that are passed to the correlation function		
separate processing of first line of signal file		
input validation of integers in processing signal files		
input validation of floating point numbers in processing signal files		

Do a self-evaluation using the project evaluation table and evaluation procedures provided. Fill in your scores and add your points correctly.

Place all screenshots, the code locator table, and the self-evaluation table together in a single document and export this document to pdf. Label all screenshots accurately.

Submit a single zip file containing
* C++ implementation files and header files for the project
* required documentation

Do NOT include executables in your submission.

Project Evaluation - Normalized Crosscorrelation		
Item	Points	Rubrics
early work	8/8	(all or nothing) 8: at most 10% of the code in the final implementation differs from that in early work submission
implementation and testing environment report	4/4	4 - all instructions followed correctly
basic test	50/50	50 - correctly computes the correct normalized crosscorrelation from basic test data
	6/6	10 - screenshot submitted showing the correctly computed normalized crosscorrelation as it appears on the console output
generality of correlation	6/6	6 - the normalized crosscorrelation is computed correctly and efficiently at all times
correlation function	6/6	6 - the normalized crosscorrelation is computed correctly and efficiently at all times, by a function as specified
processing of signal files	4/4	4 - The file is opened only once and the first line of the signal file is processed correctly and separately from code processing the rest of the signal file
input validation	4/4	4 - integers and floating point numbers are fully validated when signal files are processed
command line	4/4	4 - application processes command line arguments correctly at all times
application feedback	4/4	4 - application provides appropriate feedback at all times, as specified
self-evaluation	4/4	4 - self-evaluation accurate (or evaluating this item leads to an error)
total	100/100	

Evaluation Procedures

Special Cases

Score for the entire project is 0 if
* no source files are submitted
* the source files, as submitted, will not compile as a C++ project
* the implementation uses resources that are not allowed

early work

Early work credits are based on the final code submitted, assuming the final and early work code use the same style. The total number of lines in the final code that are not in the early work version, or that were modified from the early work version, other than formatting changes that do not modify compiled code, must not be more than 10% of the number of lines in the final code, excluding blank lines.

basic test

The correct result must have been computed from the provided test data, not hardcoded in any manner. Score 0 if there is any anomaly in the basic test.

screenshot

must show the command line and the complete results produced on the console; values must be the same as expected. Score 0 if the complete results are not all visible, or values are not the same as expected, or screenshot shows the contents of the output file instead of the console output. Score 0 if there is any anomaly in the basic test.

generality of correlation

Score 0 if basic test fails.
Score 0 if limits are placed on the amount of data.
Stress test application.
Change the starting indices of the test data. Check that the starting index of the result is always correctly determined from the starting indices and duration of the test data. The sequence of values must remain the same.
Change the test data and check results.
Do more tests if in doubt.
Check for suspicious code. Verify that suspicious code leads to errors.
Score 0 if any error manifests.

correlation function

Score 0 if generality test fails. Score 0 if code not put into a function.
Check that location of function is reported correctly. Check that parameters are correct.
Scale points by half if related code elements are not located accurately.

processing of signal files

Score 0 if signal files are opened more than once.

Score 0 if the first line is not processed correctly or not processed separately from the rest of the signal file. Check that a “1” as the first token on the first line is interpreted differently from a “1.” .

Check that data extracted are put into dynamically allocated arrays of doubles. Check that contents of vectors used while extracting data from signal files are copied into arrays. Do more tests if in doubt. Insert comments. Insert blank lines. Test with invalid files. Score 0 if any deficiency manifests. Scale points by half if related code elements are not located accurately.

input validation

Score 0 if not done.

Make a signal value invalid by adding an invalid character to a signal value (e.g. -2a). Check that this is treated as a comment and the signal extracted shorter in duration than if there were no invalid character. Do more tests if in doubt. Score 0 if any deficiency manifests. Scale points by half if related code elements are not located accurately.

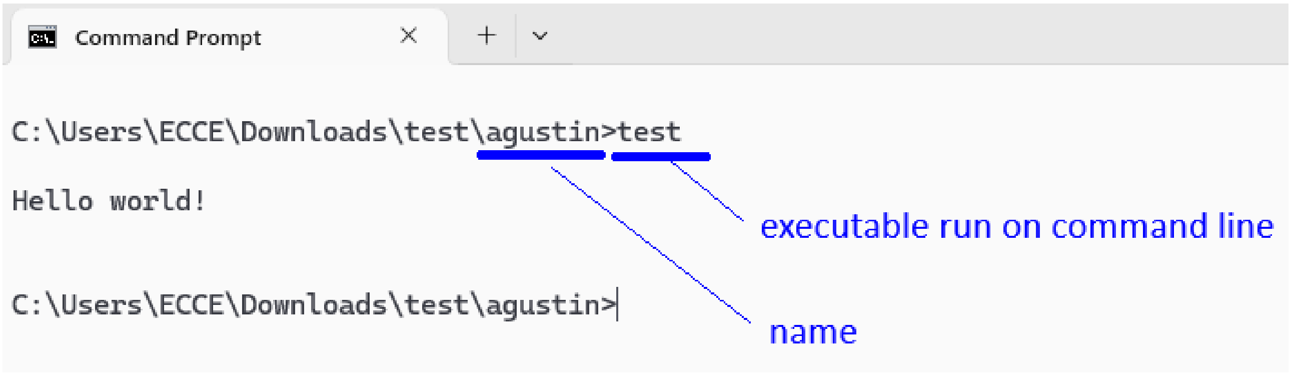
command line

Check that application accepts filenames specified on the command line. Change filename extensions and check that the application accepts them (e.g. remove the .txt, change .txt to .exe or .jpg, etc). Score 0 if there are deficiencies.

application feedback

Score 0 if application does not provide feedback in some instances, or feedback from application misses some details. Score 0 if application ever crashes, goes into an infinite loop, or behaves erratically.

Implementation and Testing Environment (4 pts)
1 - all details of implementation platform are provided
1 - screenshot of compilation: all instructions are followed correctly
1 - screenshot of program run on the console: all instructions are followed correctly
1 - all filenames appearing in the screenshots are consistent with files in the project submission

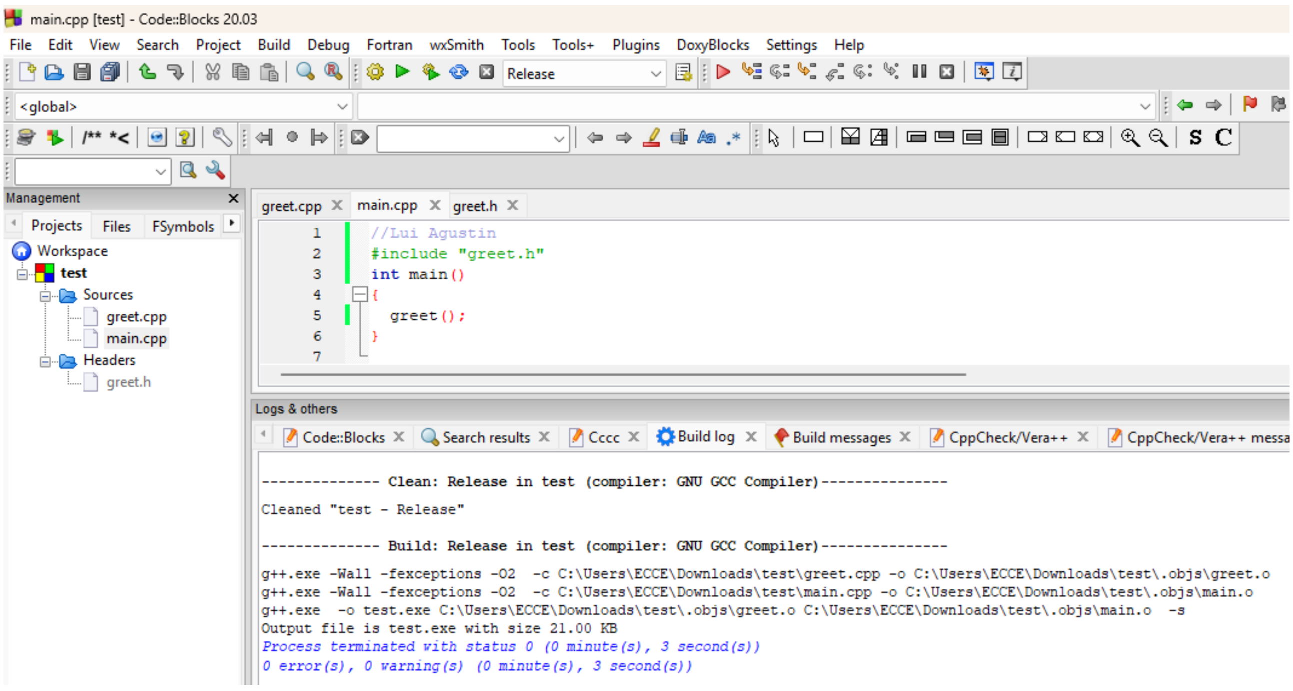


```
C:\Users\ECCE\Downloads\test\agustin>test

Hello world!


C:\Users\ECCE\Downloads\test\agustin>
```

sample screenshot - executable test.exe run by typing its name at the prompt from a path with name of author


main.cpp [test] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Release
<global>
Management
Projects Files FSymbols
Workspace
test
Sources
greet.cpp
main.cpp
Headers
greet.h
greet.cpp main.cpp greet.h
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messa

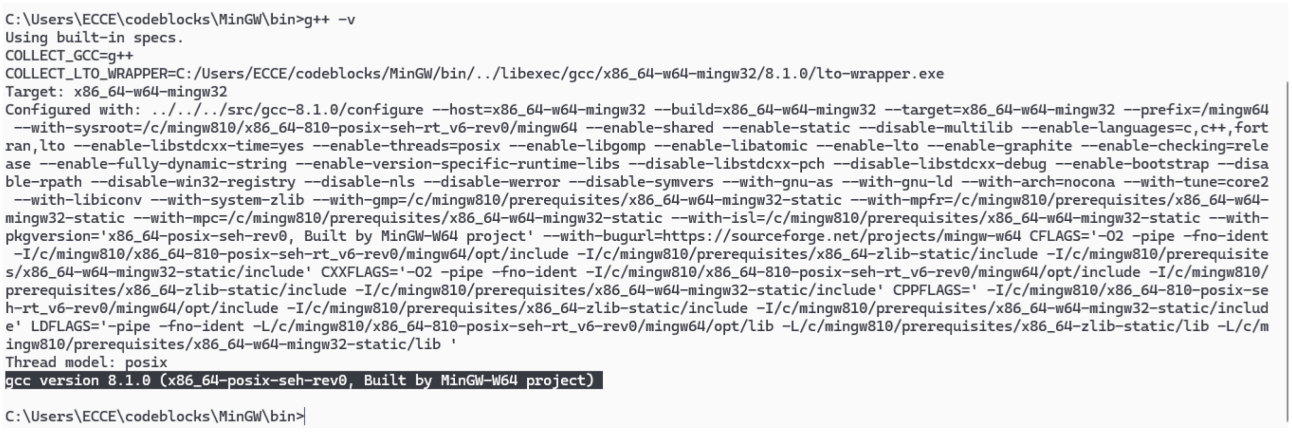
```
//Lui Agustin
#include "greet.h"
int main()
{
    greet();
}

```

```
-------------- Clean: Release in test (compiler: GNU GCC Compiler)---------------
Cleaned "test - Release"

-------------- Build: Release in test (compiler: GNU GCC Compiler)---------------
g++.exe -Wall -fexceptions -O2  -c C:\Users\ECCE\Downloads\test\greet.cpp -o C:\Users\ECCE\Downloads\test\.objs\greet.o
g++.exe -Wall -fexceptions -O2  -c C:\Users\ECCE\Downloads\test\main.cpp -o C:\Users\ECCE\Downloads\test\.objs\main.o
g++.exe  -o test.exe C:\Users\ECCE\Downloads\test\.objs\greet.o C:\Users\ECCE\Downloads\test\.objs\main.o  -s
Output file is test.exe with size 21.00 KB
Process terminated with status 0 (0 minute(s), 3 second(s))
0 error(s), 0 warning(s) (0 minute(s), 3 second(s))
```

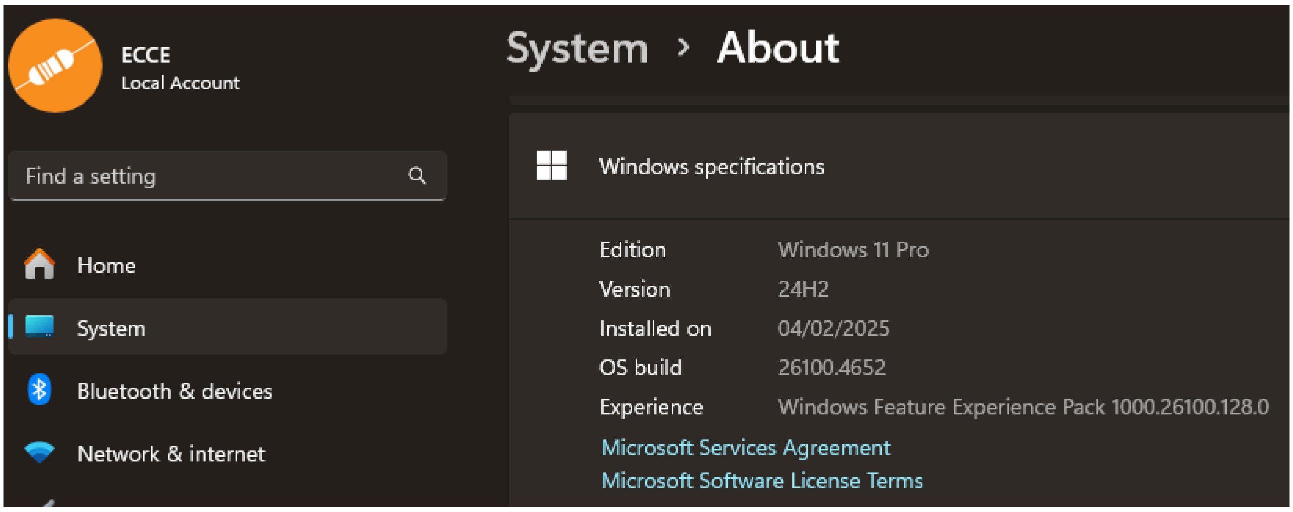
sample screenshot - compilation on Code::Blocks



```
C:\Users\ECCE\codeblocks\MinGW\bin>g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=C:/Users/ECCE/codeblocks/MinGW/bin/../libexec/gcc/x86_64-w64-mingw32/8.1.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../../../src/gcc-8.1.0/configure --host=x86_64-w64-mingw32 --build=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --prefix=/mingw64
 --with-sysroot=/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64 --enable-shared --enable-static --disable-multilib --enable-languages=c,c++,fort
ran,lto --enable-libstdcxx-time=yes --enable-threads=posix --enable-libgomp --enable-libatomic --enable-lto --enable-graphite --enable-checking=rele
ase --enable-fully-dynamic-string --enable-version-specific-runtime-libs --disable-libstdcxx-pch --disable-libstdcxx-debug --enable-bootstrap --disa
ble-rpath --disable-win32-registry --disable-nls --disable-werror --disable-symvers --with-gnu-as --with-gnu-ld --with-arch=nocona --with-tune=core2
 --with-libiconv --with-system-zlib --with-gmp=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpfr=/c/mingw810/prerequisites/x86_64-w64-
mingw32-static --with-mpc=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-isl=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-
pkgversion='x86_64-posix-seh-rev0, Built by MinGW-W64 project' --with-bugurl=https://sourceforge.net/projects/mingw-w64 CFLAGS='-O2 -pipe -fno-ident
 -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisite
s/x86_64-w64-mingw32-static/include' CXXFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/
prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CPPFLAGS=' -I/c/mingw810/x86_64-810-posix-se
h-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/includ
e' LDFLAGS='-pipe -fno-ident -L/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/lib -L/c/mingw810/prerequisites/x86_64-zlib-static/lib -L/c/m
ingw810/prerequisites/x86_64-w64-mingw32-static/lib '
Thread model: posix
gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)

C:\Users\ECCE\codeblocks\MinGW\bin>
```

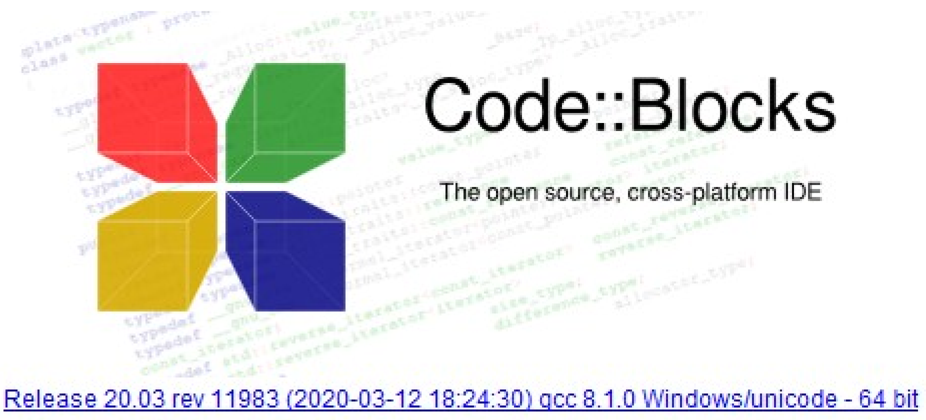
compiler - gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)



Edition	Windows 11 Pro
Version	24H2
Installed on	04/02/2025
OS build	26100.4652
Experience	Windows Feature Experience Pack 1000.26100.128.0

Microsoft Services Agreement
Microsoft Software License Terms

operating system - Windows 11 Pro, 24H2, build 26100.4652


Code::Blocks
The open source, cross-platform IDE
Release 20.03 rev 11983 (2020-03-12 18:24:30) gcc 8.1.0 Windows/unicode - 64 bit

IDE -Code::Blocks 20.03
rev 11983 (2020-03-12 18:24:30) gcc 8.1.0 Windows/unicode - 64 bit

self-evaluation
0 - not done
2 - self-evaluation score differs from project score by more than 10 points or scores not tallied properly
4 - self-evaluation accurate (or evaluating this item leads to an error)

More Tests

Changing the start indices by the same amount will give the same results.

$x_{raw}(n)$	
-50	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
-50	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$	
-7	0.0271085
	-0.187591
	0.241085
	0.280844
	-0.536025
	0.323856
	-0.130121
	0.0462651
	0.111687
	-0.529519
	0.311205
	-0.121807
	0.174579
	-0.0115663

$x_{raw}(n)$	
34	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
34	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$	
-7	0.0271085
	-0.187591
	0.241085
	0.280844
	-0.536025
	0.323856
	-0.130121
	0.0462651
	0.111687
	-0.529519
	0.311205
	-0.121807
	0.174579
	-0.0115663

Change start index of y.

$x_{raw}(n)$	
-4	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
-11	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$	
0	0.0271085
-0.187591	
0.241085	
0.280844	
-0.536025	
0.323856	
-0.130121	
0.0462651	
0.111687	
-0.529519	
0.311205	
-0.121807	
0.174579	
-0.0115663	

$x_{raw}(n)$	
-4	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
0	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$	
-11	0.0271085
-0.187591	
0.241085	
0.280844	
-0.536025	
0.323856	
-0.130121	
0.0462651	
0.111687	
-0.529519	
0.311205	
-0.121807	
0.174579	
-0.0115663	

Change start index of x

$x_{raw}(n)$	
3	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
-4	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$
0 0.0271085
-0.187591
0.241085
0.280844
-0.536025
0.323856
-0.130121
0.0462651
0.111687
-0.529519
0.311205
-0.121807
0.174579
-0.0115663

$x_{raw}(n)$	
0	2
	-1
	3
	7
	1
	2
	-3

$y_{raw}(n)$	
-4	1
	-1
	2
	-2
	4
	1
	-2
	4

$\rho_{xy}(l)$
-3 0.0271085
-0.187591
0.241085
0.280844
-0.536025
0.323856
-0.130121
0.0462651
0.111687
-0.529519
0.311205
-0.121807
0.174579
-0.0115663

Invalid value - signal is valid up to the previous value

$x_{raw}(n)$	$y_{raw}(n)$	$\rho_{xy}(l)$
-4 2	-4 1	-5 0.00193441
-1	-1	0.0251473
3	2	-0.246959
7	-2	0.357866
1	4	0.250828
2	1	-0.288872
-3	-2a	0.148949
	4	-0.706059
	duration of y is 6	0.417187
		-0.168294
		0.228905
		-0.0206337

$x_{raw}(n)$	$y_{raw}(n)$	$\rho_{xy}(l)$
-4 2	-4. 1	-11 0.0265203
-1	-1	-0.175034
3	2	0.189178
7	-2	0.277579
1	4	-0.394268
2	1	0.311171
-3	-2	-0.106081
	4	0.00235736
	The -4. is not an integer.	0.298795
	Treat as a signal value.	-0.527459
	Implied start index = 0	-0.186821
		-0.100777
		0.064238
		0.3206

Appendix: Signal File Format

See next page.

Signal File Format

Luisito L. Agustin

2023 01 18

Introduction

This document specifies a text file format for representing finite-duration signals. In this file format, signal values are expected to be floating point values in general.

All signal values of a finite-duration signal, $x(n)$, are zero except for values of the integer index n in a finite range from $n = start$ to $n = end$ where $start < end$. The signal is then said to have a *duration* given by

$$duration = end - start + 1 \; .$$

A signal file is a text file regardless of whatever extension may be used in the filename.

Format

The format for a signal file is as follows:

```
[optional starting index] signal value  optional comments
signal value  optional comments
.
.
.
signal value  optional comments
optional comments
```

The finite set of signal values must be the first or leftmost tokens in consecutive lines of the text file starting with the first line of the file, except that the first signal value (on the first line) may be preceded by an integer indicating the index $n = start$ of the signal value that follows on the line. Each signal value may be preceded by any amount of white space on each line. Only the starting index needs to be specified. It is understood that succeeding signal values correspond to successive increments of the index.

Any text that follows a signal value is considered a comment and may be ignored. A line which does not start with a valid floating point value is a comment. A signal file does not encode any valid signal if the first line does not start with a valid integer or floating point number.

The duration of the signal is determined from the number of consecutive lines from which a valid signal value can be extracted. The signal values either terminate when the end of a file is reached or when the next line extracted from the file does not contain a valid signal value as its first token.

If the optional starting index is absent, the implied starting index is 0. That is, if the first item on the first line is not a valid integer but is a valid floating point number, then the start index of the signal is zero and the signal value at $n = 0$ is the floating point number extracted from the first line.

If the first line contains a single numerical value, that numerical value must be a signal value and the implied start index is 0.

Examples

```
-4 2 This is a comment to be ignored
-1 Comment: the -4 on the first line means the signal starts with x(-4) = 2
3
7
1
2
-3
This line does not start with a floating point value.
The parser will conclude that -3 is the end of the finite duration signal.
Everything that comes after may be ignored.
This signal is the sequence x(n) from Example 2.6.1 in
John Proakis and Dimitris Manolakis: Digital Signal Processing, 4th ed, 2007.
x(-4) = 2
x(-3) = -1
x(-2) = 3
x(-1) = 7
x(0) = 1
x(1) = 2
x(2) =-3

start = -4
end = 2
duration = 7  ( = 2 - (-4) + 1 )
```

Without comments, the signal file would simply contain:

```
-4 2
-1
3
7
1
2
-3
```

Comments do not change the signal being represented.

Notes

Some confusion may be possible in the first line if there is no starting index (implied start = 0) but the floating point signal value appears as an integer. The signal value might be incorrectly interpreted as the start index.

Recommendations for Handling Signal Files

The extraction operator (>>) in C++ skips white space and terminates processing on any white space character. It would skip whatever leading white space there might be before the signal values.

The first line of a signal file needs to be processed differently from the rest of a signal file. It should not be processed as part of a loop.

There's no limit to how large a signal file could be. Applications should not impose any limit. Nothing in the file format specifies the duration of the signal. A C++ vector should be used to store the signal values obtained until the last signal value is extracted from the file, at which point the duration would be known. Once the duration is known, an array must be dynamically allocated and the contents of the vector copied to the dynamically allocated array. Any processing to be done must be done on the dynamically allocated array. Processing arrays is faster than processing vectors.

Whether a signal file is valid depends on its contents, never on the extension used for the filename. While a .txt extension may be common for text files, applications that process signal files should neither expect nor require a .txt extension. They should work with whatever filename is provided, as is, without any modification.

A signal file should be opened only once, and its lines processed one at a time from start to finish without any need to process any line twice.

Suggested C++ Signal Class API

```
class engg151Signal
{
  public:
    engg151Signal ();
    // default constructor
    // suggested default: one-element signal with value 0.0, start index 0

    engg151Signal ( double * x, int start, int duration );

    bool importSignalFromFile ( string filename);
    // returns true if a valid signal was actuallly obtained from filename
    // returns false otherwise

    bool exportSignalToFile ( string filename );
    // returns true if the signal was successfully exported to a file
    // returns false otherwise

    int start();
    int end();
    int duration();
    // as implied by the names,
    // return the start index, end index, and duration of the signal
    // respectively

    double * data();
    // returns a pointer to an array of double containing the signal values
}

engg151Signal normalizedXCorr ( engg151Signal x, engg151Signal y);
// computes the normalized crosscorrelation of x crosscorrelated with y
// and returns the normalized crosscorrelation as a signal object.
// Note that the crosscorrelation is not commutative.
// This does not need to be a class member.
```

Revision History - Signal File Format

2021 02 17	first version
2023 01 18	revisions for clarity, recommendations

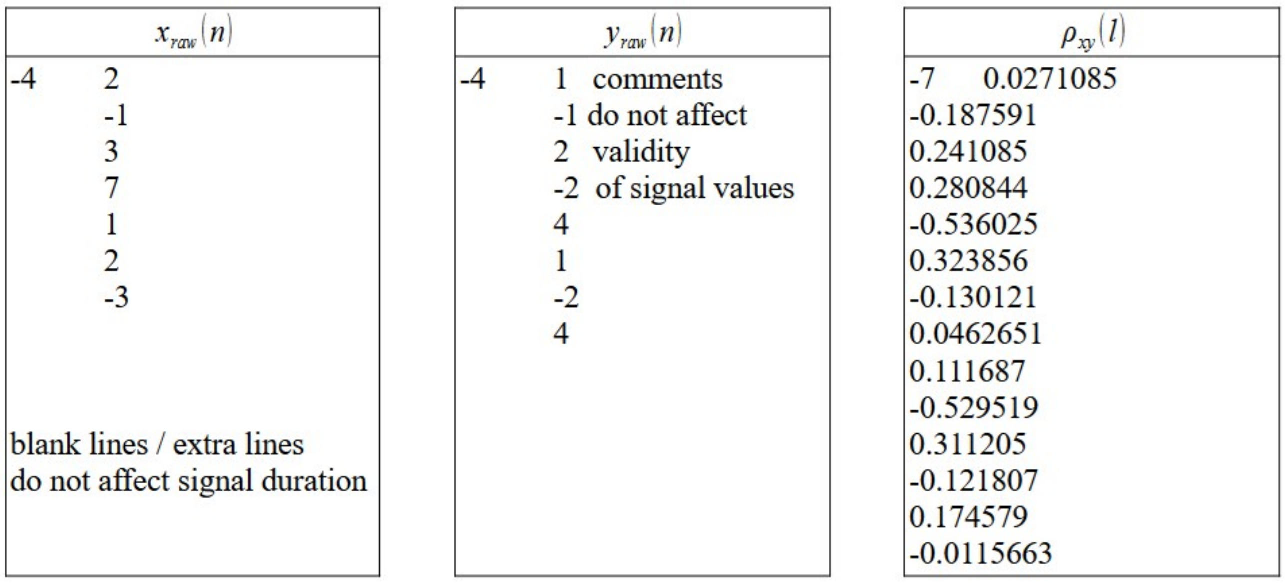
Homework
Normalized Crosscorrelation on a Spreadsheet

30 minor points
asynchronous task for Jan. 26
due: Jan. 29

Refer to the specifications of Project 1 Normalized Crosscorrelation. Use a spreadsheet to compute the normalized crosscorrelation of the data provided for the "Basic Test".

Basic Test

$x_{raw}(n)$	$y_{raw}(n)$	$\rho_{xy}(l)$
-4 2	-4 1 comments	-7 0.0271085
-1	-1 do not affect	-0.187591
3	2 validity	0.241085
7	-2 of signal values	0.280844
1	4	-0.536025
2	1	0.323856
-3	-2	-0.130121
	4	0.0462651
		0.111687
		-0.529519
blank lines / extra lines		0.311205
do not affect signal duration		-0.121807
		0.174579
		-0.0115663



$x_{raw}(n)$	$y_{raw}(n)$	$\rho_{xy}(l)$
-4 2	-4 1 comments	-7 0.0271085
-1	-1 do not affect	-0.187591
3	2 validity	0.241085
7	-2 of signal values	0.280844
1	4	-0.536025
2	1	0.323856
-3	-2	-0.130121
	4	0.0462651
		0.111687
		-0.529519
blank lines / extra lines		0.311205
do not affect signal duration		-0.121807
		0.174579
		-0.0115663

Obtain $\rho_{xy}(l)$ from $x_{raw}(n)$ and $y_{raw}(n)$ as shown.

Obtain *rho_xy (l)* from *x_raw(n)* and *y_raw(n)* as shown.

.

Generate the correct results by applying spreadsheet operations and formulas on the given data. Do not just copy and paste values into spreadsheet cells. Each result must be obtained using formulas whose correctness could be verified on the spreadsheet.

Submit the spreadsheet.