НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1.2

з дисципліни

«Інтелектуальні вбудовані системи»

на тему

«Дослідження і розробка моделей випадкових сигналів.

Аналіз їх характеристик»

Виконав:

студент групи ІП-83

ЗК ІП-8311

Карпюк Ігор

Київ 2021

**Варіант № / Без варіанту**

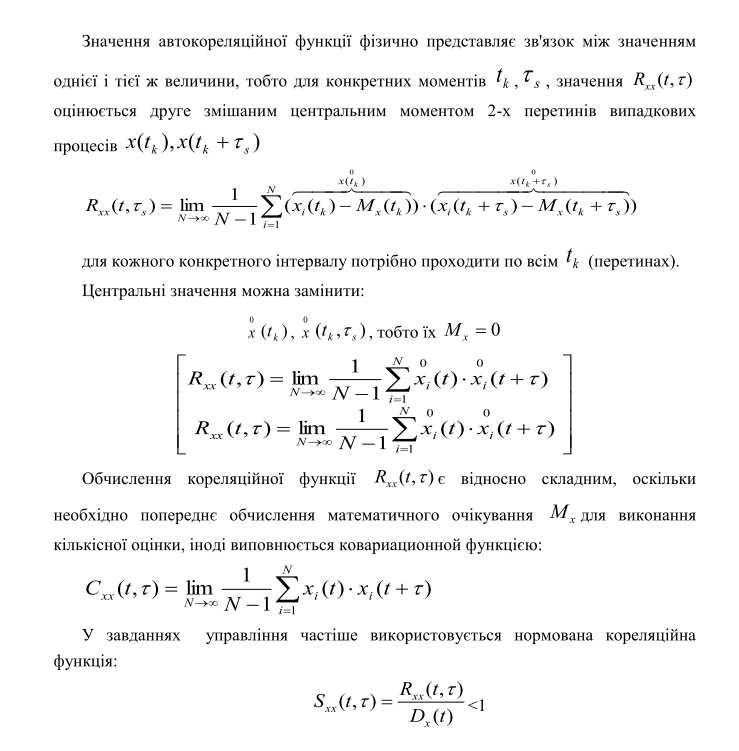
Варіант: 11

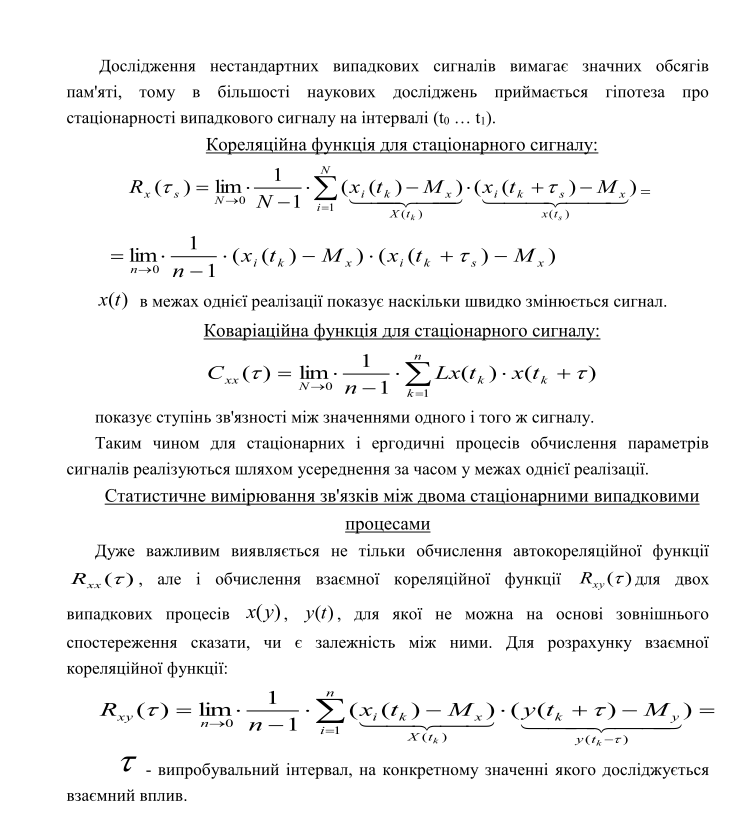
Число гармонік в сигналі n: 10

Гранична частота, ω гр: 1500

Кількість дискретних відліків, N: 256

**Основні теоретичні відомості**





**Завдання**

Для згенерованого випадкового сигналу з Лабораторної роботи N 1 відповідно до заданого варіантом (Додаток 1) розрахувати його автокореляційної функцію. Згенерувати копію даного сигналу і розрахувати взаімнокорреляціонную функцію для 2-х сигналів. Розробити відповідну програму і вивести отримані значення і графіки відповідних параметрів.

**Лістинг коду**

**libLab.h**

|  |
| --- |
| #include <iostream> |
|  | #include <fstream> |
|  | #include <omp.h> |
|  | #include <random> |
|  | #include <math.h> |
|  |  |
|  | int convertToInt(char \*a, int size) |
|  | { |
|  | int i; |
|  | std::string s = ""; |
|  | for (i = 0; i < size; i++) |
|  | { |
|  | s = s + a[i]; |
|  | } |
|  | return (int)std::stoi(s); |
|  | } |
|  |  |
|  | struct Point |
|  | { |
|  | double x = 0.0; |
|  | double y = 0.0; |
|  |  |
|  | Point(double xCoor = 0.0, double yCoor = 0.0) |
|  | { |
|  | x = xCoor; |
|  | y = yCoor; |
|  | } |
|  | }; |
|  |  |
|  | Point \*makeFunction(int n, int W, int N) |
|  | { |
|  | Point \*xt = reinterpret\_cast<Point \*>(malloc(sizeof(Point) \* N + 1)); |
|  |  |
|  | // Init randomizer |
|  | std::default\_random\_engine generator; |
|  | std::uniform\_real\_distribution<double> amp\_and\_fi(0, 1); |
|  |  |
|  | // Iterate through the several harmonics |
|  | for (int w = W / n; w <= W; w += W / n) |
|  | { |
|  | double A = amp\_and\_fi(generator); |
|  | double fi = amp\_and\_fi(generator); |
|  |  |
|  | // Iterate through one harmonica |
|  | for (int x = 0; x < N; x += 1) |
|  | { |
|  | xt[x].x = (double)x; |
|  | double y = A \* sin(w \* x + fi); |
|  | xt[x].y += y; |
|  | } |
|  | } |
|  | return xt; |
|  | } |
|  |  |
|  | double calcMathExp(Point \*xt, int N) |
|  | { |
|  | double sumOfPoints = 0.0; |
|  |  |
|  | for (int i = 0; i < N; i++) |
|  | { |
|  | sumOfPoints += xt[i].y; |
|  | } |
|  | return sumOfPoints / N; |
|  | } |
|  |  |
|  | double calcDispersion(Point \*xt, int N, double mathematicalExpectation) |
|  | { |
|  | double sumOfDiffs = 0.0; |
|  | for (int i = 0; i < N; i++) |
|  | { |
|  | sumOfDiffs += (xt[i].y - mathematicalExpectation) \* (xt[i].y - mathematicalExpectation); |
|  | } |
|  | return sumOfDiffs / N; |
|  | } |
|  |  |
|  | void writeCalcsToFile(Point \*xt, int N, std::string fileName) |
|  | { |
|  | std::ofstream dataSheet; |
|  | dataSheet.open("lab1.2/" + fileName + ".xlsx"); |
|  | for (int i = 0; i < N; i++) |
|  | { |
|  | dataSheet << xt[i].x << "\t" << xt[i].y << '\n'; |
|  | } |
|  | } |
|  |  |
|  | Point \*mutualCorrelation(Point \*firstFunc, Point \*secondFunc, int N) |
|  | { |
|  | Point \*mutCorFunVals = reinterpret\_cast<Point \*>(malloc(sizeof(Point) \* N + 1)); |
|  | double mathExpFirstFun = calcMathExp(firstFunc, N); |
|  | double mathExpSecondFun = calcMathExp(secondFunc, N); |
|  | double dispFirstFun = calcDispersion(firstFunc, N, mathExpFirstFun); |
|  | double dispSecondFun = calcDispersion(secondFunc, N, mathExpSecondFun); |
|  |  |
|  | // Loop through function values |
|  | for (int x = 0; x < N; x++) |
|  | { |
|  | mutCorFunVals[x] = Point(x, ((firstFunc[x].y - mathExpFirstFun) \* (secondFunc[x].y - mathExpSecondFun)) / ((N - 1) \* sqrt(dispFirstFun) \* sqrt(dispSecondFun))); |
|  | } |
|  | return mutCorFunVals; |
|  | } |
|  |  |
|  | Point \*offsetFun(Point \*xt, double Tau, int N) |
|  | { |
|  | Point \*offsetFunVals = reinterpret\_cast<Point \*>(malloc(sizeof(Point) \* N + 1)); |
|  |  |
|  | for (int i = 0; i < N; i++) |
|  | { |
|  | offsetFunVals[i].x = xt[i].x + Tau; |
|  | offsetFunVals[i].y = xt[i].y; |
|  | } |
|  |  |
|  | return offsetFunVals; |
|  | } |

**Code.cpp**

|  |
| --- |
| #include "labLib.h" |
|  |  |
|  | int main(int argc, char \*\*argv) |
|  | { |
|  | // init variables |
|  | int n = 10; // Harmonica |
|  | int W = 1500; // Critical frequency |
|  | int N = 256; // Discrete vidclick |
|  | double tau = 5.0; // The offset |
|  | // The check is conducted to assert having all the three needed arguments |
|  | // if the program is going to be used with a different data set |
|  | if (argc == 4) |
|  | { |
|  | n = convertToInt(argv[1], sizeof(argv[1])); |
|  | W = convertToInt(argv[2], sizeof(argv[2])); |
|  | N = convertToInt(argv[3], sizeof(argv[3])); |
|  | } |
|  |  |
|  | // Init array of Point's |
|  | Point \*xt = makeFunction(n, W, N); |
|  |  |
|  | // Calculating correlation functions |
|  | Point \*xtOfsset = offsetFun(xt, tau, N); |
|  | Point \*autoCor = mutualCorrelation(xt, xtOfsset, N); |
|  |  |
|  | Point \*xtWithDifferentHarmonics = makeFunction(100, W, N); |
|  | Point \*mutCor = mutualCorrelation(xt, xtWithDifferentHarmonics, N); |
|  |  |
|  | // Write calculations to file |
|  | writeCalcsToFile(autoCor, N, "Auto\_correlation\_function"); |
|  | writeCalcsToFile(mutCor, N, "Mutual\_correlation\_function"); |
|  | return 0; |
|  | } |

**Висновок**

Я навчився будувати автокроеляційну функцію, розібрався з реалізацією потрібних функцій на С++.