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

Значення автокореляційної функції фізично представляє зв'язок між значенням однієї і тієї ж величини, тобто для конкретних моментів t_k, τ_s, значення $R_{xx}(t,\tau)$ оцінюється друге змішаним центральним моментом 2-х перетинів випадкових процесів $x(t_k), x(t_k+\tau_s)$

$$R_{xx}(t,\tau_s)=\lim_{N\to\infty}\frac{1}{N-1}\sum_{i=1}^{N}(\overbrace{x_i(t_k)-M_x(t_k)}^{\overset{0}{x}(t_k)})\cdot(\overbrace{x_i(t_k+\tau_s)-M_x(t_k+\tau_s)}^{\overset{0}{x}(t_k+\tau_s)})$$

для кожного конкретного інтервалу потрібно проходити по всім t_k (перетинах).

Центральні значення можна замінити:

$$\overset{0}{x}(t_k),\ \overset{0}{x}(t_k,\tau_s),\ \text{тобто їх}\ M_x=0$$

$$\left[\begin{array}{l} R_{xx}(t,\tau)=\lim\limits_{N\to\infty}\dfrac{1}{N-1}\sum\limits_{i=1}^{N}\overset{0}{x}_i(t)\cdot\overset{0}{x}_i(t+\tau) \\ R_{xx}(t,\tau)=\lim\limits_{N\to\infty}\dfrac{1}{N-1}\sum\limits_{i=1}^{N}\overset{0}{x}_i(t)\cdot\overset{0}{x}_i(t+\tau) \end{array}\right]$$

Обчислення кореляційної функції $R_{xx}(t,\tau)$ є відносно складним, оскільки необхідно попереднє обчислення математичного очікування M_x для виконання кількісної оцінки, іноді виповнюється ковариационной функцією:

$$C_{xx}(t,\tau)=\lim_{N\to\infty}\frac{1}{N-1}\sum_{i=1}^{N}x_i(t)\cdot x_i(t+\tau)$$

У завданнях управління частіше використовується нормована кореляційна функція:

$$S_{xx}(t,\tau)=\frac{R_{xx}(t,\tau)}{D_x(t)}<1$$

Дослідження нестандартних випадкових сигналів вимагає значних обсягів пам'яті, тому в більшості наукових досліджень приймається гіпотеза про стаціонарності випадкового сигналу на інтервалі (t_0 … t_1).

Кореляційна функція для стаціонарного сигналу:

$$R_x(\tau_s) = \lim_{N\to 0} \cdot \frac{1}{N-1} \cdot \sum_{i=1}^{N} (\underbrace{x_i(t_k) - M_x}_{X(t_k)}) \cdot (\underbrace{x_i(t_k + \tau_s) - M_x}_{x(t_s)}) =$$

$$= \lim_{n\to 0} \cdot \frac{1}{n-1} \cdot (x_i(t_k) - M_x) \cdot (x_i(t_k + \tau_s) - M_x)$$

$x(t)$ в межах однієї реалізації показує наскільки швидко змінюється сигнал.

Коваріаційна функція для стаціонарного сигналу:

$$C_{xx}(\tau) = \lim_{N\to 0} \cdot \frac{1}{n-1} \cdot \sum_{k=1}^{n} Lx(t_k) \cdot x(t_k + \tau)$$

показує ступінь зв'язності між значеннями одного і того ж сигналу.

Таким чином для стаціонарних і ергодичні процесів обчислення параметрів сигналів реалізуються шляхом усереднення за часом у межах однієї реалізації.

Статистичне вимірювання зв'язків між двома стаціонарними випадковими процесами

Дуже важливим виявляється не тільки обчислення автокореляційної функції $R_{xx}(\tau)$, але і обчислення взаємної кореляційної функції $R_{xy}(\tau)$ для двох випадкових процесів $x(y)$, $y(t)$, для якої не можна на основі зовнішнього спостереження сказати, чи є залежність між ними. Для розрахунку взаємної кореляційної функції:

$$R_{xy}(\tau) = \lim_{n\to 0} \cdot \frac{1}{n-1} \cdot \sum_{i=1}^{n} (\underbrace{x_i(t_k) - M_x}_{X(t_k)}) \cdot (\underbrace{y(t_k + \tau) - M_y}_{y(t_k - \tau)}) =$$

τ - випробувальний інтервал, на конкретному значенні якого досліджується взаємний вплив.

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 C++.