Белорусский государственный университет информатики и радиоэлектроники

# Кафедра ЭВМ

### Лабораторная работа № 2

«Операции свертки и корреляции»

|  |  |
| --- | --- |
| **Выполнили:**  студенты группы 850502  Новиков И.А.  Карпович К.А. | **Проверил:**  Третьяков А. Г. |

#### Минск 2021

1. **Цель работы**

Целью работы является:

Изучение операций корреляции и свертки, их основных свойств, а также методики их получения с помощью быстрого преобразования Фурье (БПФ) на основе теорем о корреляции и свертке.

1. **Задание:**

|  |  |  |
| --- | --- | --- |
| **№ варианта** | **Заданная функция** | **N для БПФ** |
| 5 | y=cos(x)+sin(x) | 8 |

1. **Теоретические сведения, необходимые для выполнения лабораторной работы**

Свертка – это математический способ комбинирования двух сигналов для формирования третьего сигнала. Это один из самых важных методов ЦОС. Свертка связывает три сигнала: входной сигнал, выходной сигнал и импульсную характеристику.

Корреляция так же, как свертка, использует два сигнала для получения третьего. Этот третий сигнал называется корреляционным сигналом двух входных сигналов.

**Теорема свертки**

Если *{X(m)}* и *{Y(m)}* – последовательности действительных чисел, для которых *X(m)↔Cx(k)*, *Y(m)↔Cy(k)*, и свертка этих последовательностей определяется как

, , (2.1)

то

*Cz(k)=Cx(k)Cy(k)*.

Доказательство

Вычисляя *Z(m)*, получим

. (2.2)

Подставляя в (2.2) соотношение свертки (2.1), получим

.

Согласно теореме сдвига, имеем

.

Таким образом,

.

Эта теорема утверждает, что свертка временных последовательностей эквивалентна умножению их коэффициентов, полученных после дискретного преобразования Фурье.

**Теорема корреляции**

Если *X(m)↔Cx(k)* и *Y(m)↔Cy(k)*,а их функция корреляции определяется соотношением

, где , (2.3)

то ,

где  - комплексное сопряженное 

Доказательство

По определению имеем

. (2.4)

Подставляя (2.3) в (2.4) и меняя порядок суммирования, получаем



Применяя теорему сдвига, будем иметь

.

Так как , то .

Таким образом, .

Если последовательности *{X(m)}* и *{Y(m)}* идентичны друг другу, то

, .

Обратное ДПФ последовательности  есть .

Тогда

, т.е. справедлива теорема Парсеваля.

**Матричное представление корреляции и свертки**

Если *{X(m)}* и *{Y(m)}* –две *N*-периодические последовательности действительных чисел, то операции корреляции и свертки определяются соответственно как



В общем виде корреляцию двух последовательностей можно записать как

.

В свою очередь, соотношение свертки можно записать в общем виде как

.

Если последовательности *{X(m)}* и *{Y(m)}* аналогичны друг другу, то

, где .

Это соотношение определяет автокорреляцию последовательности *{X(m)}*.

С использованием БПФ схема вычислений корреляции будет иметь вид рис.2.1.



Рис. 1.1. Схема вычисления корреляции

В свою очередь, схему вычисления свертки можно представить как показано на рис.2.2.



Рис. 1.2. Схема вычисления свертки

1. **Результат работы**

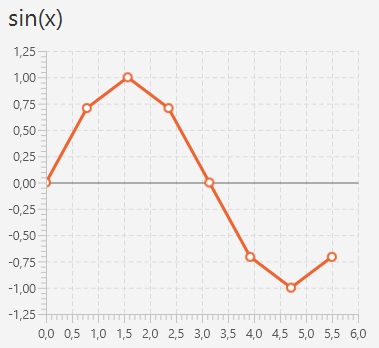


Рис. 1.3. График функции y=sin(x)

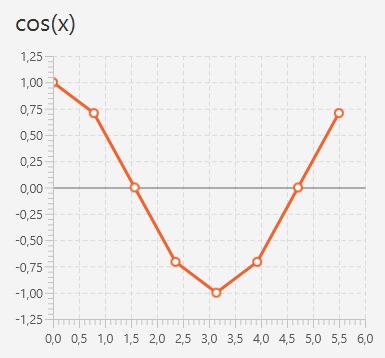


Рис. 1.4. График функции z=cos(x)

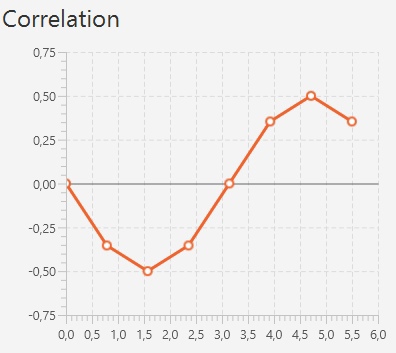


Рис. 1.5. Корреляция

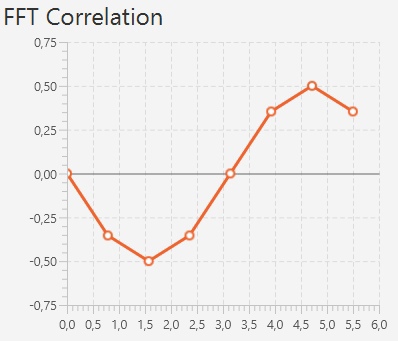


Рис. 1.6. Корреляция БПФ

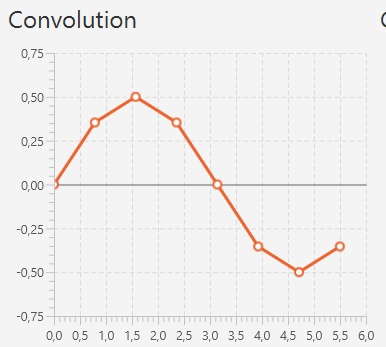


Рис. 1.7. Свертка

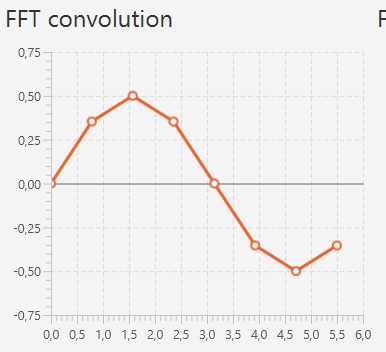


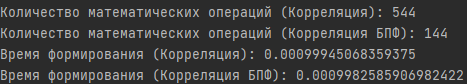
Рис. 1.8. Свертка БПФ

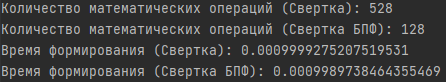
1. **Сравнительный анализ**

Таблица 1.1

Эффективность БПФ

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Операций (Корреляция)** | **Операций (Корреляция БПФ)** | **Эффективность БПФ** |
| 8 | 544 | 144 | 3.78 **:** 1 |
| **Скорость (Корреляция), с** | **Скорость (Корреляция БПФ), с** | **Эффективность БПФ** |
| 0.000999 | 0.000998 | 1 **:** 1 |
| **Операций (Свертка)** | **Операций (Свертка БПФ)** | **Эффективность БПФ** |
| 528 | 128 | 4.125 **:** 1 |
| **Скорость (Свертка), с** | **Скорость (Свертка БПФ), с** | **Эффективность БПФ** |
| 0.001 | 0.000999 | 1 **:** 1 |

****

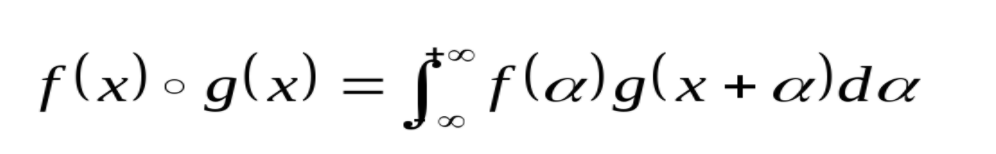


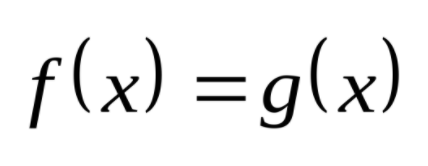
1. **Вывод**

В методике преобразования Фурье корреляция и свертка - очень важные операции. Свертка связывает три сигнала: входной сигнал, выходной сигнал и импульсную характеристику. Импульсная характеристика – сигнал, с помощью которого описываются системы (пользуясь стратегией импульсного разложения). Импульсное разложение представляет собой способ поточечного анализа сигнала.

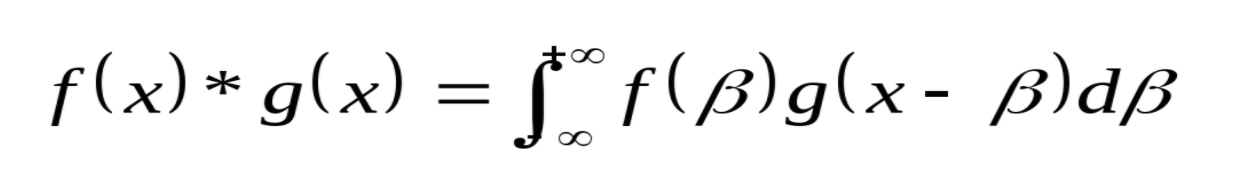
Корреляция так же, как свертка, использует два сигнала для получения третьего. Этот третий сигнал называется корреляционным сигналом двух входных сигналов.

Корреляция между двумя непрерывными функциями f(x) и g(x) определяется как:



где α- временная переменная для интегрирования. Корреляция называется автокорреляцией если и взаимной корреляцией в противном случае.

Свертка по определению есть:



Формы корреляции и свертки похожи, существует только одно различие между ними, показанное выше. При свертке g(x) сперва свертывается по вертикальной оси, а затем перемещается по х для получения g(x-). Затем эта функция умножается на f() и интегрируется.

Вычисление корреляционных функций при помощи БПФ является, особенно для длинных числовых рядов, в десятки и сотни раз более быстрым методом, чем последовательными сдвигами во временной области при больших интервалах корреляции.

# ПРИЛОЖЕНИЕ А

Листинг кода

**main.py**

import operations

import plot

def main():

number = 16

x\_list = operations.create\_x(number)

y\_list = operations.create\_y(x\_list)

z\_list = operations.create\_z(x\_list)

correlation\_list = operations.correlation(y\_list, z\_list, number)

correlation\_fft\_list = operations.correlation\_fft(y\_list, z\_list, number)

convolution\_list = operations.convolution(y\_list, z\_list, number)

convolution\_fft\_list = operations.convolution\_fft(y\_list, z\_list, number)

plot.create\_plot(x\_list, y\_list, 'Функция y = sin(2x)')

plot.create\_plot(x\_list, z\_list, 'Функция z = cos(7x)')

plot.create\_plot(x\_list, correlation\_list, 'Корреляция')

plot.create\_plot(x\_list, correlation\_fft\_list, 'Корреляция БПФ')

plot.create\_plot(x\_list, convolution\_list, 'Свертка')

plot.create\_plot(x\_list, convolution\_fft\_list, 'Свертка БПФ')

print('\nКоличество математических операций (Корреляция): ' + str(int(operations.counter\_correlation)))

print('Количество математических операций (Корреляция БПФ): ' + str(int(operations.counter\_correlation\_fft)))

print('Время формирования (Корреляция): ' + str(operations.time\_correlation))

print('Время формирования (Корреляция БПФ): ' + str(operations.time\_correlation\_fft))

print('\nКоличество математических операций (Свертка): ' + str(int(operations.counter\_convolution)))

print('Количество математических операций (Свертка БПФ): ' + str(int(operations.counter\_convolution\_fft)))

print('Время формирования (Свертка): ' + str(operations.time\_convolution))

print('Время формирования (Свертка БПФ): ' + str(operations.time\_convolution\_fft))

if \_\_name\_\_ == '\_\_main\_\_':

main()

**fourier.py**

import cmath

import time

import numpy

counter\_dft = 0

counter\_fft = 0

time\_dft = 0

time\_fft = 0

def create\_x(length):

points = []

for i in range(length):

# 2 \* pi / n - разбиваем на интервалы

points.append(i \* numpy.pi / 32)

return points

def create\_y(points):

function = []

for x in points:

function.append(complex(numpy.sin(2 \* x) + numpy.cos(7 \* x), x))

return function

def create\_frequency(length):

frequency = []

for i in range(-int(length / 2), int(length / 2), 1):

frequency.append(i)

return frequency

def create\_amplitude(points, length):

amplitude = []

for i in range(length):

amplitude.append(abs(points[i]))

return amplitude

def create\_phase(points, length):

phase = []

for i in range(length):

phase.append(cmath.phase(points[i]))

return phase

def create\_dft(points, length, direction):

global counter\_dft

global time\_dft

time\_dft = time.time()

n = length

dft = []

for m in range(n):

c = complex(0)

for k in range(n):

w = complex(numpy.cos(m \* k \* 2 \* numpy.pi / n), direction \* numpy.sin(m \* k \* 2 \* numpy.pi / n))

c += w \* points[k]

counter\_dft += 1

if direction == -1:

c /= n

dft.append(c)

time\_dft = time.time() - time\_dft

return dft

def create\_fft(points, direction, correction):

global time\_fft

time\_fft = time.time()

fft = []

for i in range(int(len(points))):

fft.append(complex(points[i]))

fft = create\_recursive\_fft(fft, direction, correction)

time\_fft = time.time() - time\_fft

return fft

def create\_recursive\_fft(points, direction, correction):

global counter\_fft

n = int(len(points))

fft\_first = []

fft\_second = []

fft = []

if n == 1:

return points

wn = complex(numpy.cos(2 \* numpy.pi / n), direction \* numpy.sin(2 \* numpy.pi / n))

w = complex(1, 0)

for i in range(int(n / 2)):

fft\_first.append(points[i] + points[i + int(n / 2)])

fft\_second.append((points[i] - points[i + int(n / 2)]) \* w)

w = w \* wn

counter\_fft += 1

even = create\_recursive\_fft(fft\_first, direction, correction)

uneven = create\_recursive\_fft(fft\_second, direction, correction)

for i in range(int(n / 2)):

fft.append(even[i])

fft.append(uneven[i])

if direction == -1:

for i in range(n):

fft[i] /= correction

return fft

**plot.py**

import matplotlib.pyplot

def create\_plot(x\_list, y\_list, title):

fig, (ax) = matplotlib.pyplot.subplots(1, 1)

ax.plot(x\_list, y\_list)

ax.set(title=title)

ax.grid()

matplotlib.pyplot.show()

**operations.py**

import time

import fourier

import numpy

counter\_correlation = 0

counter\_correlation\_fft = 0

counter\_convolution = 0

counter\_convolution\_fft = 0

time\_correlation = 0.0

time\_correlation\_fft = 0.0

time\_convolution = 0.0

time\_convolution\_fft = 0.0

def create\_x(length):

points = []

for i in range(length):

# 2 \* pi / length - разбиваем на интервалы

points.append(2 \* i \* numpy.pi / length)

return points

def create\_y(points):

function = []

for x in points:

function.append(numpy.sin(2 \* x))

return function

def create\_z(points):

function = []

for x in points:

function.append(numpy.cos(7 \* x))

return function

def get\_real(points):

result = []

for item in points:

result.append(item.real)

return result

def correlation(x\_list, y\_list, length):

global time\_correlation

global counter\_correlation

time\_correlation = time.time()

result = []

for m in range(length):

amount = 0

for h in range(length):

amount += x\_list[h] \* y\_list[m + h - length]

counter\_correlation += 2

amount /= length

result.append(amount / length)

counter\_correlation += 2

time\_correlation = time.time() - time\_correlation

return result

def correlation\_fft(x\_list, y\_list, length):

global time\_correlation\_fft

global counter\_correlation\_fft

time\_correlation\_fft = time.time()

fourier.counter\_fft = 0

cx = fourier.create\_fft(x\_list, 1, 1)

cy = fourier.create\_fft(y\_list, 1, 1)

result = []

for i in range(length):

cx[i] = numpy.conjugate(cx[i])

counter\_correlation\_fft += 1

for i in range(length):

result.append(cx[i] \* cy[i] / length)

counter\_correlation\_fft += 2

result = fourier.create\_fft(result, -1, 4)

counter\_correlation\_fft += fourier.counter\_fft

time\_correlation\_fft = time.time() - time\_correlation\_fft

return result

def convolution(x\_list, y\_list, length):

global time\_convolution

global counter\_convolution

time\_convolution = time.time()

result = []

for m in range(length):

amount = 0

for h in range(length):

amount += x\_list[h] \* y\_list[m - h]

counter\_convolution += 2

result.append(amount / length)

counter\_convolution += 1

time\_convolution = time.time() - time\_convolution

return result

def convolution\_fft(x\_list, y\_list, length):

global time\_convolution\_fft

global counter\_convolution\_fft

time\_convolution\_fft = time.time()

fourier.counter\_fft = 0

cx = fourier.create\_fft(x\_list, 1, 1)

cy = fourier.create\_fft(y\_list, 1, 1)

result = []

for i in range(length):

result.append((cx[i] \* cy[i]) / length)

counter\_convolution\_fft += 2

result = fourier.create\_fft(result, -1, 2)

counter\_convolution\_fft += fourier.counter\_fft

time\_convolution\_fft = time.time() - time\_convolution\_fft

return result