# РАБОТА №4. АВТОКОРРЕЛЯЦИОННАЯ И ВЗАИМНОКОРРЕЛЯЦИОННАЯ ФУНКЦИЯ ДИСКРЕТНЫХ И НЕПРЕРЫВНЫХ СИГНАЛОВ

Цель работы: изучение АКФ и ВКФ дискретных и непрерывных сигналов.

Планируемая продолжительность: 2 академических часа.

Тип работы:  с использованием компьютерных средств.

В данной работе в качестве основного инструмента используется язык программирования Python. Общие принципы решения задач описаны только для Python вне зависимости от какого-либо окружения, поэтому вы можете использовать любой редактор кода, например JupyterLab, Jupyter Notebook, VSCode и т.д.

### Теоретические основы

В общем случае взаимная корреляция сигналов $f(t)$ и $g(t)$ описывается выражением:

$$(f \cdot g)(\tau) = \int_{-\infty}^{\infty} \bar{f}(t) g(t-\tau)\, dt,$$

а в некоторых источниках встречается другое обозначение, отличное знаком перед *t*:

$$(f \cdot g)(\tau) = \int_{-\infty}^{\infty} \bar{f}(t) g(t+\tau)\, dt,$$

где $\bar{f}(t)$ обозначает комплексное сопряжение и в случае используемых нами действительных сигналов может быть опущен.

Эта функция характеризует степень «похожести» $f(t)$ на $g(t)$ при текущем сдвиге $\tau$.

То есть, по функции ВКФ двух сигналов можно построить график со значениями сдвига $\tau$ по оси абсцисс. На данном графике видно, где сигналы более похожи – при смещении второго на положительный сдвиг (часть графика при $\tau > 0$) или же при его смещении на отрицательный сдвиг (часть графика при $\tau < 0$). Пример графика ВКФ

приведен на рисунке ниже. Здесь мы видим, что сигналы максимально похожи при сдвиге второго из них на $\tau = 2$.

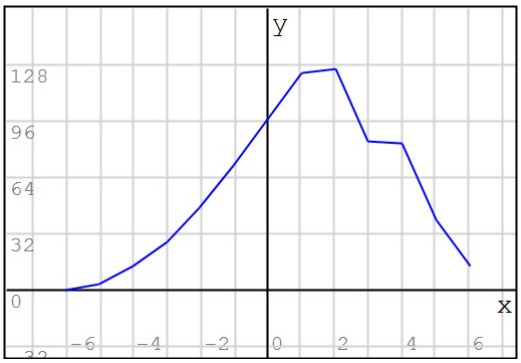


*Рис. 4.1*. Пример ВКФ

Автокорреляционная функция (АКФ) — это частный случай ВКФ, в котором в качестве второго сигнала используется первый (то есть ВКФ сигнала с самим собой):

$$(f \cdot f)(\tau) = \int_{-\infty}^{\infty} f(t)^* f(t-\tau)\, dt.$$

Функция АКФ симметрична относительно $\tau = 0$ и обязательно имеет максимум в $\tau = 0$, поскольку сигнал не может быть более похож сам на себя, чем при нулевом сдвиге. В случае с АКФ можно строить только половину графика (обычно ту, что соответствует $\tau \geq 0$) из-за его симметричности. Пример АКФ представлен на рисунке ниже.

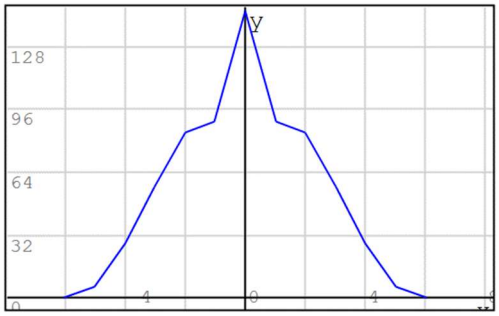


*Рис. 4.2*. Пример АКФ

АКФ используют для выявления зависимостей в сигналах, не поддающихся визуальному анализу. К примеру, АКФ случайного сигнала имеет максимум при $\tau = 0$ и не имеет никаких выраженных зависимостей на графике при $\tau \neq 0$. Если же, как на рисунке ниже, визуально случайный сигнал является результатом сложения

периодического сигнала (в данном случае синусоиды) и шума, периодическая составляющая будет четко видна на графике АКФ.

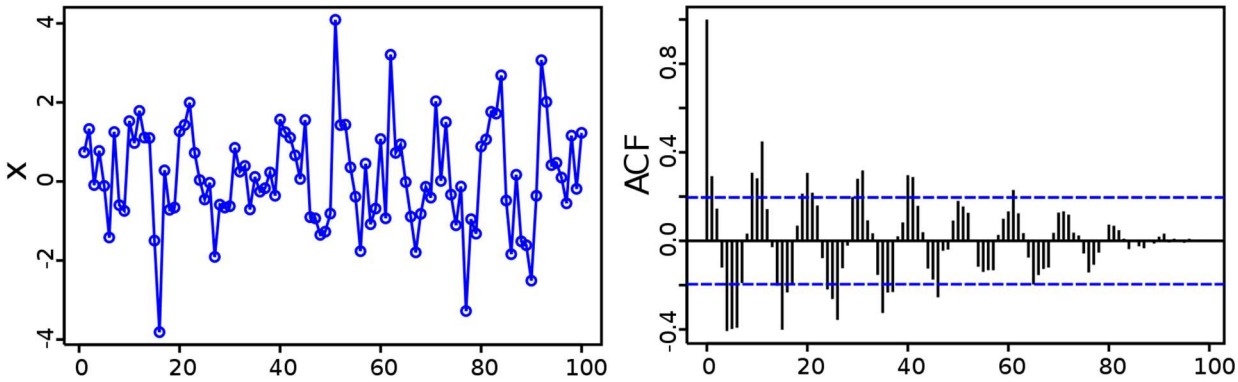


*Рис. 4.3.* Пример выявления полезной составляющей по АКФ

**ВКФ и АКФ дискретного сигнала**

Подготовим окружение. Через pip установим все необходимые пакеты.

```
1  ! pip install -U pip
2  ! pip install -U matplotlib
3  ! pip install -U numpy
4  ! pip install -U scipy
```

*Рис. 4.4.* Установка пакетов

И импортируем следующее:

```
1  import numpy as np
2  import math
3  from matplotlib import pyplot as plt
4  from scipy.integrate import quad
5  from typing import Callable
```

*Рис. 4.5*. Импортирование модулей

Напишем заранее метод для отображения графика для данных вида $y = f(x)$.

```
1  def plot(x, y):
2      plt.plot(x, y)
3      plt.grid(True)
4      plt.ylabel('y')
5      plt.xlabel('x')
6      plt.show()
```

*Рис. 4.6*. Функция построения графика

Зададим два дискретных сигнала, в виде массивов чисел с одинаковым размером:

```
1  y_1 = [1, 2, 3, 4, 5, 6, 7]
2  y_2 = [2, 4, 7, 2, 8, 3, 0]
```

*Рис. 4.7*. Задание массивов

Для осуществления различных преобразований сигналов удобно задавать эти преобразования в общем виде функцией и вызывать для обработки конкретных массивов. Зададим функцию для вычисления ВКФ при текущем сдвиге, ориентируясь на следующее выражение:

$$\sum_{k=0}^{K-n} s_k \, s_{k-n},$$

где n – сдвиг между массивами (а также номер элемента в ВКФ),

k – номер элемента в первом массиве.

То есть, ВКФ двух массивов также будет массивом, причем каждый из его элементов будет рассчитываться как сумма произведений всех перекрывающихся элементов. Чтобы задать функцию вычисления текущего (k-го) элемента ВКФ, можно реализовать следующий метод.

```python
def cross_corelation_element(
        array_1: np.ndarray,
        array_2: np.ndarray,
        lag: int) -> float:
    """
    При отрицательном lag
    становится меньше размера одного из исходных массивов на
    значение lag, что не позволяет выйти за пределы второго массива.
    """
    last_index = 0
    if lag < 0:
        last_index = len(array_1) + lag
    else:
        last_index = len(array_1)

    """
    При положительном сдвиге данное условие
    заставляет оператор суммирования начать не с первого элемента
    массивов, а с 1 + lag, поскольку иначе будут перемножаться не
    перекрывающиеся элементы. Для первого массива всегда будут взяты
    элементы, начиная с первого.
    """
    begin_index = 1
    if lag > 0:
        begin_index = lag + 1

    result = []
    for i in range(begin_index, last_index):
        result.append(array_1[i] * array_2[i - lag])
    return sum(result)
```

*Рис. 4.8*. Алгоритм вычисления
текущего (k-го) элемента ВКФ

Пояснения в шести кавычках писать не нужно, это всего лишь комментарий к тому, как работает алгоритм. Также необязательно писать типизацию к входным и выходным параметрам, это, например np.ndarray или `-> float`. Такая типизация может помочь интегрированным средам разработки, понять, что за объект будет на входе и выходе, и вызвать подсказки к методам в соответствии с типом, если среда опять же эти подсказки поддерживает.

Теперь нужно задать сдвиг. Как было сказано в рамках прошлого занятия, для дискретных отсчетов функция корреляции может быть найдена в диапазоне сдвига от $-(N-1)$ до $(N-1)$, где N – число

отсчетов в сигнале. Напишем для этого функцию, приведённую ниже.

```
1  def lags(array):
2      return np.arange(-(len(array) - 1), len(array), 1, dtype=int)
```

*Рис. 4.9*. Функция создания смещений
на основе существующего массива

Чтобы вручную не задавать пределы массива смещений на вход будем передавать некоторый массив и на его основе получаем N встроенной функцией len. При помощи функции np.arange создаём массив смещений от –(len(array) - 1) до –(len(array) - 1) (второй аргумент в np.arrange берёт предел не включительно).

Теперь осталось скомбинировать наши функции, при каждом сдвиге нужно записать результат вычисления текущего (k-го) элемента ВКФ в массив.

```
1  def cross_corelation(
2          array_1: np.ndarray,
3          array_2: np.ndarray) -> list:
4      result = []
5      for lag in lags(array_1):
6          result.append(cross_corelation_element(array_1, array_2, lag))
7      return result
```

*Рис. 4.10*. Функция создания смещений
на основе существующих массивов

Вызовем эти функции и нарисуем полученные графики при помощи реализованного в самом начале метода plot.

```
1  cc = cross_corelation(y_1, y_2)
2  plot(np.arange(0, len(y_1)), y_1)
3  plot(np.arange(0, len(y_1)), y_2)
4  plot(lags(y_1), cc)
```

*Рис. 4.11*. Вызов функций

Таким образом получим следующие графики.

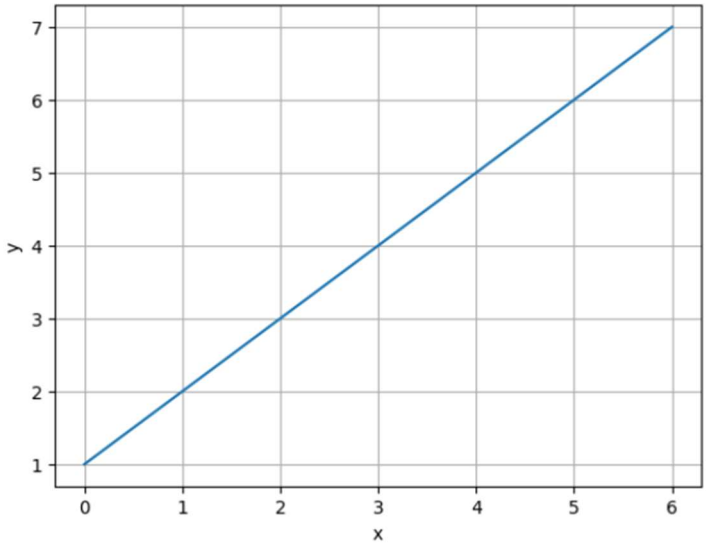


*Рис. 4.12*. График сигнала y_1

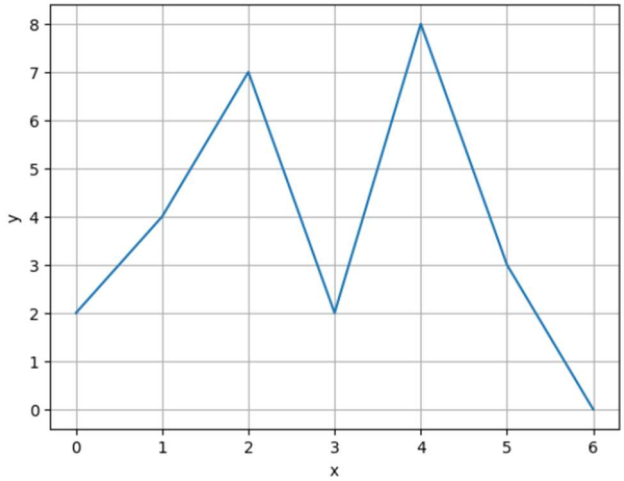


*Рис. 4.13*. График сигнала y_2

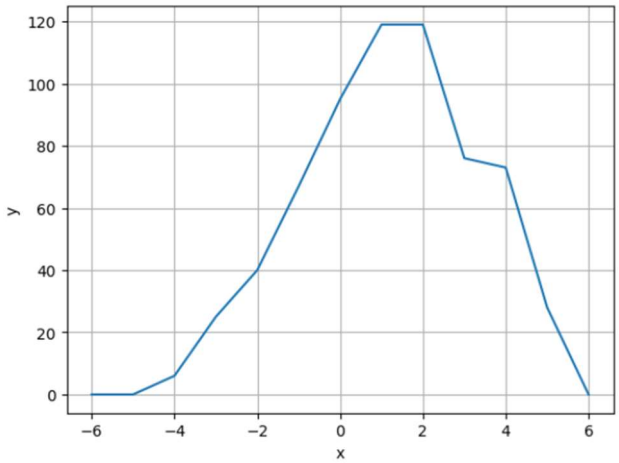


*Рис. 4.14.* График ВКФ y_1 и y_2

Как видно из последнего графика наиболее похожей функция y_1 похожа на y_2 при смещении равном 2.

Теперь найдем АКФ для сигнала, заданного некоторой функцией. Для этого сначала реализуем функцию $y = \sin(\frac{x \cdot \pi}{12})$ по которой будем создавать массив данных.

```
1  def func_1(x):
2      return math.sin((x * math.pi) / 12)
```

*Рис.* 4.15 Функция для генерации данных

Также зная то, что автокорреляционная функция (АКФ) – это частный случай ВКФ, в котором в качестве второго сигнала используется первый (то есть ВКФ сигнала с самим собой) напишем функцию АКФ.

```
1  def auto_corelation(
2      array: np.ndarray
3  ) -> list:
4      return cross_corelation(array, array)
```

*Рис. 4.16* Функция АКФ

Теперь создадим массив данных по func_1 при помощи функции np.linespace, которая создаст массив от 1 до 100 на 100 элементов с равными промежутками между элементами и выведем её.

```
1  x = np.linspace(1, 100, 100)
2  y_1 = [func_1(value) for value in x]
3  plot(np.linspace(1, 100, 100), y_1)
```

*Рис. 4.17.* Алгоритм создания данных

Получим график нашей функции на рисунке 4.18.

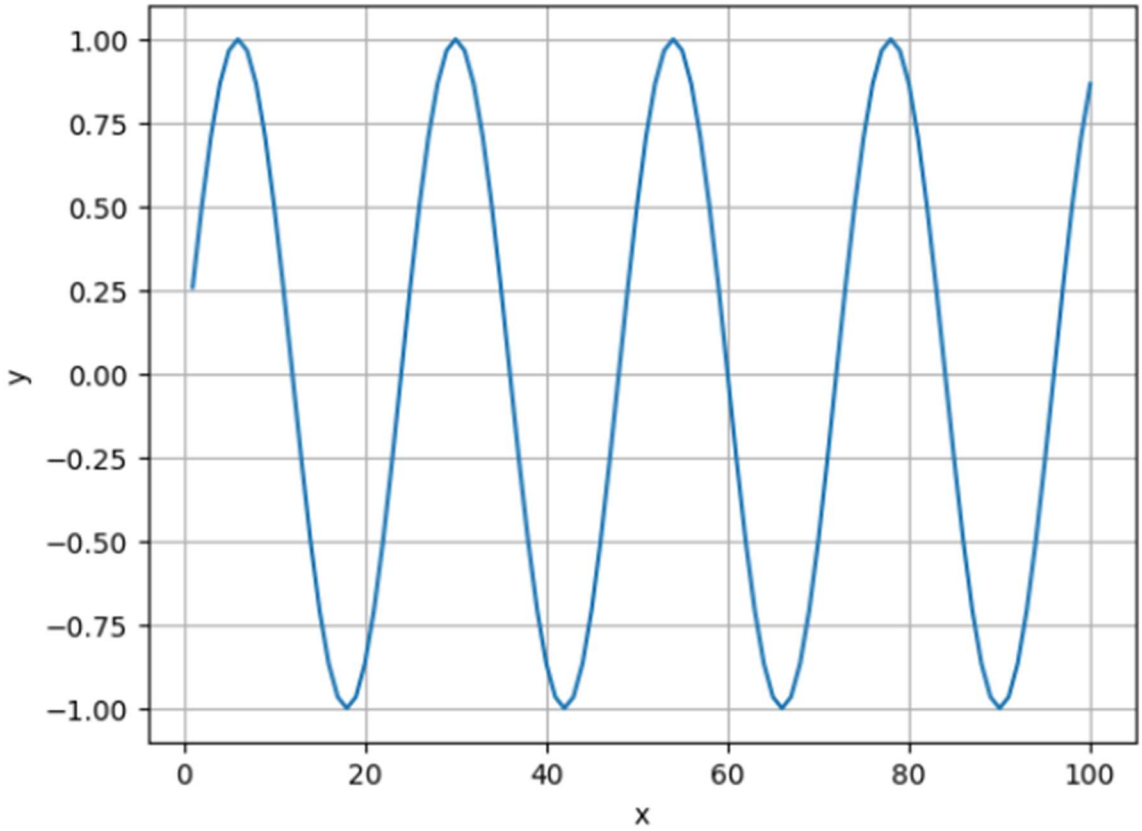


*Рис. 4.18*. График функции синуса

Осталось рассчитать АКФ y_1 и вывести полученные данные в график.

```
1  plot(lags(y_1), auto_corelation(y_1))
```

*Рис.* 4.19. Вызов графика функции АКФ с расчётом

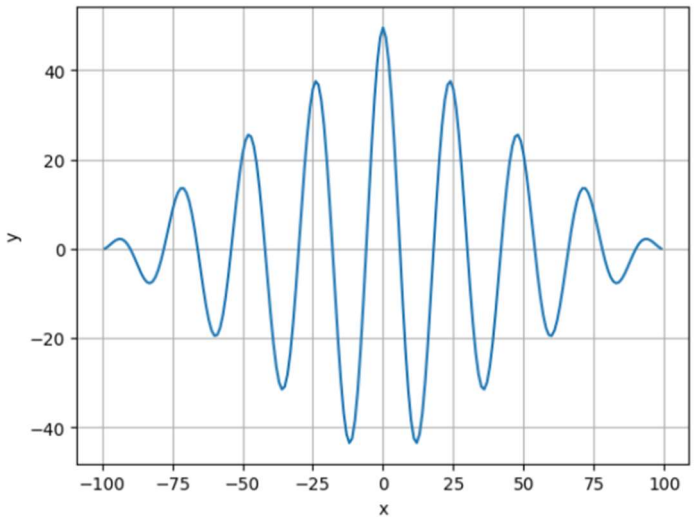


*Рис. 4.20*. График АКФ

**АКФ и ВКФ непрерывных сигналов**

Теперь рассмотрим возможность нахождения аналитического выражения АКФ для простых примеров. В нашем случае АКФ будет вычисляться согласно выражению:

$$\int_0^{10} y(x)\, y(x-T)dx,$$

где бесконечные пределы заменены на интервал [0;10] для простоты вычислений.

Обратите внимание, для самостоятельной работы при работе с АКФ сигналы в таблице с вариантами обозначаются как y(x) и y2(x).

Реализуем функцию contigious_corelation. Обратите внимание, теперь функции не являются массивами данных, теперь это Callable объект, иными словами, сама функция без её вызова. Помимо этого, теперь в нашу функцию передаём переменные a и b – это пределы интегрирования. Также стоит упомянуть внутреннюю функцию mul_func, для интегрирования нам не нужен результат работы функции, нам нужна сама функция, поэтому при помощи внутренней функции mul_func, которая обладает областью видимости данных до входных переменных от contigious_corelation указываем их перемножение при вызове mul_func, указывая во входных параметр x и параметр смещения T. Для интегрирования используем функцию quad, во входных параметрах передаём функцию mul_func, пределы интегрирования и в дополнительный аргументах args выставляем смещение. Функция quad возвращает кортеж из двух элементов, а именно значение интеграла и абсолютную ошибку. Абсолютная ошибка нам не нужна, поэтому пропускаем её, выставляя прочерк вторым аргументом

```
def contigious_corelation(
    func_1: Callable,
    func_2: Callable,
    laggs: np.ndarray,
    a: int,
    b: int,
) -> list:
    def mul_func(x, T):
        return func_1(x) * func_2(x - T)
    result = []
    for lag in laggs:
        integral, _ = quad(mul_func, a, b, args=(lag,))
        result.append(integral)
    return result
```

*Рис. 4.21.* Функция ВКФ для непрерывного сигнала

Создаём вторую функцию, от которой будем брать ВКФ.

```
def func_2(x):
    return math.sin(x) + 1
```

*Рис. 4.22*. Вторая функция для ВКФ

Теперь осталось задать смещения вышеупомянутой функцией np.linespace вызвать реализованную функцию contigious_corelation и отобразить график результата.

```
laggs = np.linspace(-50, 50, 100)
cc = contigious_corelation(func_1, func_2, laggs, 0, 10)
plot(laggs, cc)
```

*Рис.* 4.23. Вызов ВКФ для непрерывного сигнала и графика для него

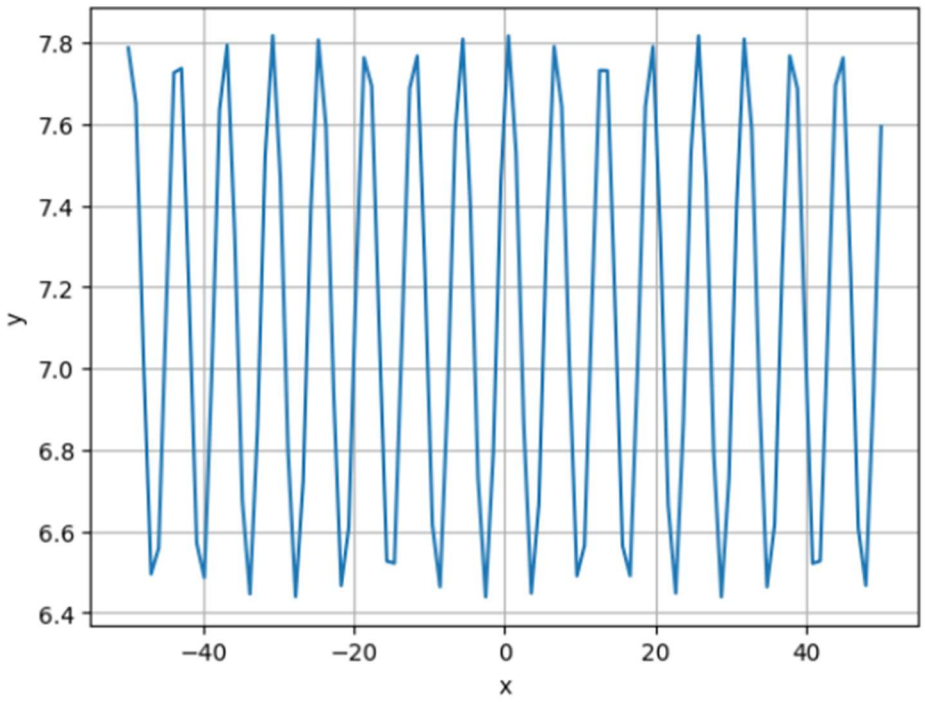


*Рис. 4.24*. График ВКФ

По аналогии с дискретным сигналом получить АКФ сигнала можно вызвав ВКФ с одинаковыми функциями. Реализуем данный подход и получим метод auto_contigious_corelation.

```
def auto_contigious_corelation(
    func: Callable,
    laggs: np.ndarray,
    a: int,
    b: int
) -> list:
    return contigious_corelation(func, func, laggs, a, b)
```

*Рис.* 4.25. АКФ
для непрерывного сигнала

Как и до этого создаём смещения, вызываем функцию АКФ и строим график.

```
laggs = np.linspace(-50, 50, 100)
cc = auto_contigious_corelation(func_1, laggs, 0, 10)
plot(laggs, cc)
```

*Рис. 4.26*. Вызов АКФ

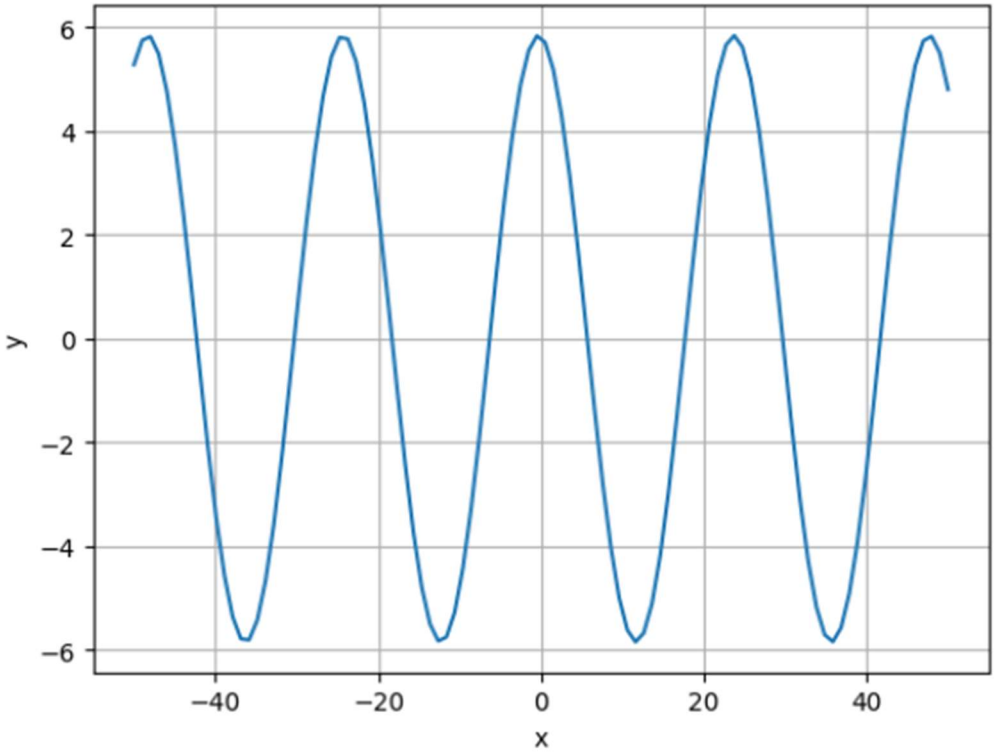


*Рис. 4.27.* График АКФ для непрерывной функции

**Задание**

Указанные значения заменяют собой соответствующие параметры для раздела, посвященного дискретным отсчетам и АКФ непрерывной функции.

*Таблица 4.1*

Задания для второй работы

| **Вар.** | **1,6,11, 16,21,26** | **2,7,12, 17,22,27** | **3,8,13, 18,23,28** | **4,9,14, 19,24,29** | **5,10,15, 20,25,30** |
|---|---|---|---|---|---|
| **y(x)** | 2*(Sin(x/10)) | 2*(Cos(x/10)) | 0.5*(Sin(x/7)) | 0.5*(Cos(x/7)) | 1.3*(Sin(x/3)) |
| **s1** | 0.5,0,1,2,3,0 | 3,2.5,0,1,-1,-3 | 1,2,1,-1,1,-1 | 0.5,1,-1,1,1,-1 | 1,2,-2,5,4.5,4 |
| **s2** | 7,-7,6,-6,5,-5 | 1,3,5,7,9,7 | 0,1,0,2,0,3 | -1,2,-1,1,-1,0 | 0,4,2,1,3,1 |
| **y2(x)** | y(x-3) | y(x-1) | y(x-5) | y(x-4) | y(x+2) |