ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ 10/10

ПРЕПОДАВАТЕЛЬ

ассистент	24.09.20	А.К. Акопян
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

ИЗУЧЕНИЕ МЕТОДОВ ИНТЕРПОЛЯЦИИ

по курсу: методы и устройства цифровой обработки сигналов

Вариант 10

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР 4711		Хасанов Б.Р.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1 Цель работы

Изучить основы методов интерполяции. Осуществить сравнение двух методов интерполяции: метод полиномиальной интерполяции и метод кубической сплайн-интерполяции.

2 Краткие теоретические сведения

В рамках данной лабораторной работы будут рассмотрены два метода интерполяции. Интерполяция, интерполирование (от лат. inter-polis – «разглаженный, подновлённый, обновлённый; преобразованный») – в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений[3]

Первый метод интерполяции – полином Лагранжа. Он имеет следующую формулу:

$L_n(x)=\sum_{i=0}^{n} y_i \left(\prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j} \right)$ и позволяет проводить интерполяцию при количеству узлов не превышающему 20. Так же, данный метод интерполяции, как видно по формуле, прост в реализации.

Принципиальное отличие идеи сплайн-интерполяции от интерполяции полиномом состоит в том, что полином один, а сплайн состоит из нескольких полиномов, а именно их количество равно количеству инервалов, внутри которых мы производим интерполяцию.[3]

Вот формула кубического сплайна: $S_i(x)=a\,x^3+b\,x^2+c\,x+d$

3 Программа, в которой представлена последовательность и результаты обработки сигналов, с необходимыми комментариями

Программа написана на языке программирования python 3

Стандартно в начале главной функции импортируются нужные библиотеки

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange, CubicSpline
from statistics import stdev
```

Затем, генерируем сигнал данный нам по заданию:

```
n = np.linspace(0, 2, N)
signal_function = 'log(2 - cos(2 * pi * 3 * time))'
generated_signal = signal_generate(n, signal_function)
```

При генерации сигнала используется функция signal_generate, вот её содержание ниже. Функция eval исполняет переданную ей строку, как часть кода

```
def signal_generate(time, function):
    """
    Генерирует сигнал с заданными параметрами

    :time: Отсчёты сигнала
    :function: Функция по которой строиться график
    :returns: Сигнал с заданными параметрами
    """
    from numpy import log, pi, cos

    generated_signal = eval(function)

    return generated_signal
```

Возвращаемся обратно в main(). Генерируем отсчёты дискретезированного сигнала с помощью всё той же функции signal_generate. В данном случае. M – количество точек в дискретезированном сигнале. Именно его нам нужно будет менять по заданию, чтобы получить требуемое значение среднеквадратической ошибки интерполяции:

```
M = 46
sd = np.linspace(0, 2, M)
points_of_generated_signal = signal_generate(sd, signal_function)
```

Далее, проводим полиномиальную интерполяцию с помощью функции lagrange из библиотеки scipy и записываем её в переменную lagrange_interpolation:

```
lagrange_interpolation = lagrange(sd, points_of_generated_signal)
```

Теперь, чтобы интерполировать сигнал методом Лагранжа достаточно просто вызвать lagrange_interpolation как функцию в которую передаются отсчёты, что мы и сделаем в следующей строчке передавая в функцию interpolation_signal_plot сигналы для построения сравнительного графика на котором будут показаны сигнал с высокой частотой дискретизации(исходный), с низкой частотой дискретизации, отмеченный точками и сигнал полученный путём интерполяции методом Лагранжа. Результат выполнения строчки ниже на отображён на рисунке 4.1 и 4.2 при разных M. Приводить саму функцию interpolation_signal_plot нет смысла, т. к. там никаких расчётов не происходит, просто выводятся графики.

```
interpolation_signal_plot(generated_signal, n, points_of_generated_signal, sd,
lagrange_interpolation(n), "полиномиальным")
```

Далее проделываем всё то же самое только теперь используя кубическую интерполяцию с помощью функции CubicSpline из библиотеки scipy. Результат на рисунке 4.3

```
cubic_spline_interpolation = CubicSpline(sd, points_of_generated_signal)
interpolation_signal_plot(generated_signal, n, points_of_generated_signal, sd,
cubic_spline_interpolation(n), "кубическая")
```

С помощью следующих строчек выясняем относительную среднеквадратическую ошибку интерполяции для интерполяции Лагранжа и выводим график разности между заданной функцией и интерполянтом Лагранжа, результат на рисунках 4.4 и 4.5 при разных М.

```
delta_lagrange_interpolation = generated_signal - lagrange_interpolation(n)
epsilon_lagrange = stdev(delta_lagrange_interpolation)/stdev(generated_signal)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(delta_lagrange_interpolation)
ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_lagrange), transform=ax.transAxes, fontsize=15)
```

Проделываем всё то же самое для кубического интерполянта, результат на рисунке 4.6 и 4.7 при разных М.

```
delta_spline_interpolation = generated_signal - cubic_spline_interpolation(n)
epsilon_spline = stdev(delta_spline_interpolation)/stdev(generated_signal)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(delta_spline_interpolation)
ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_spline), transform=ax.transAxes, fontsize=15)
```

4 Полученные графики

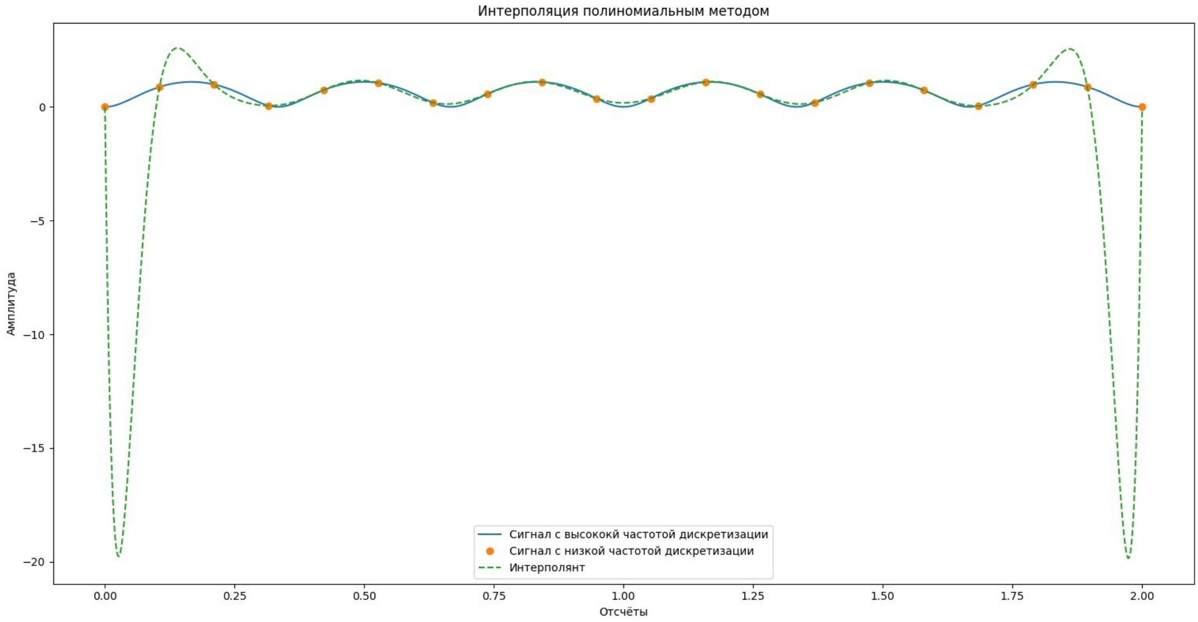

Рисунок 4.1 — Интерполяция методом Лагранжа при количестве точек 20

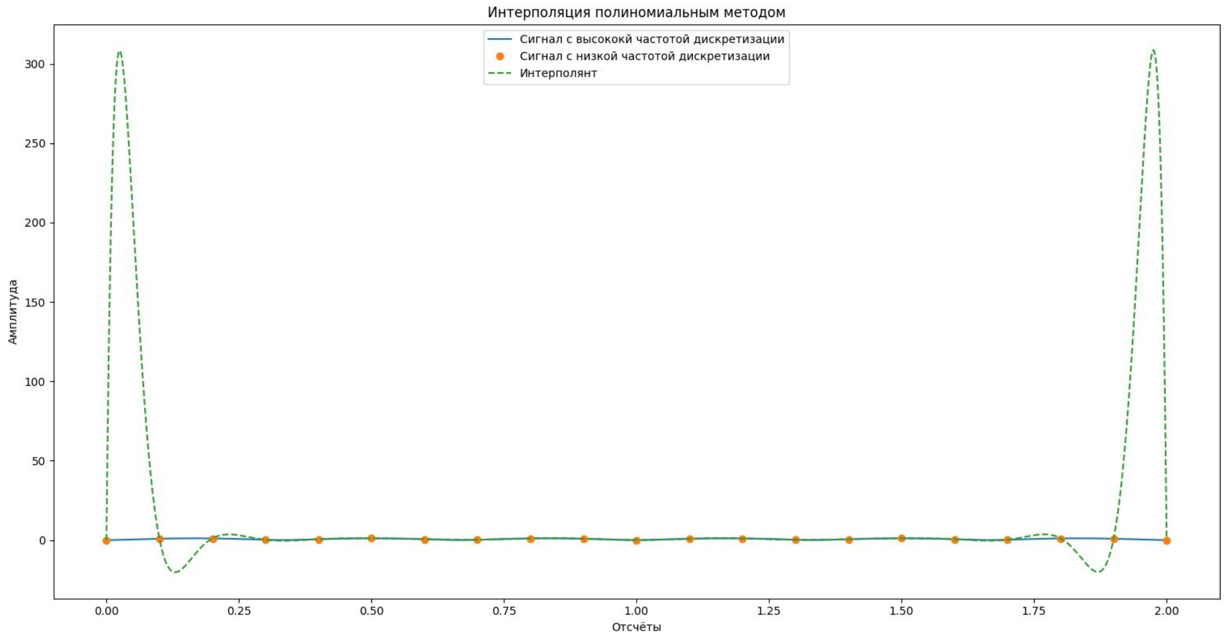

Рисунок 4.2 — Интерполяция методом Лагранжа при количестве точек 21

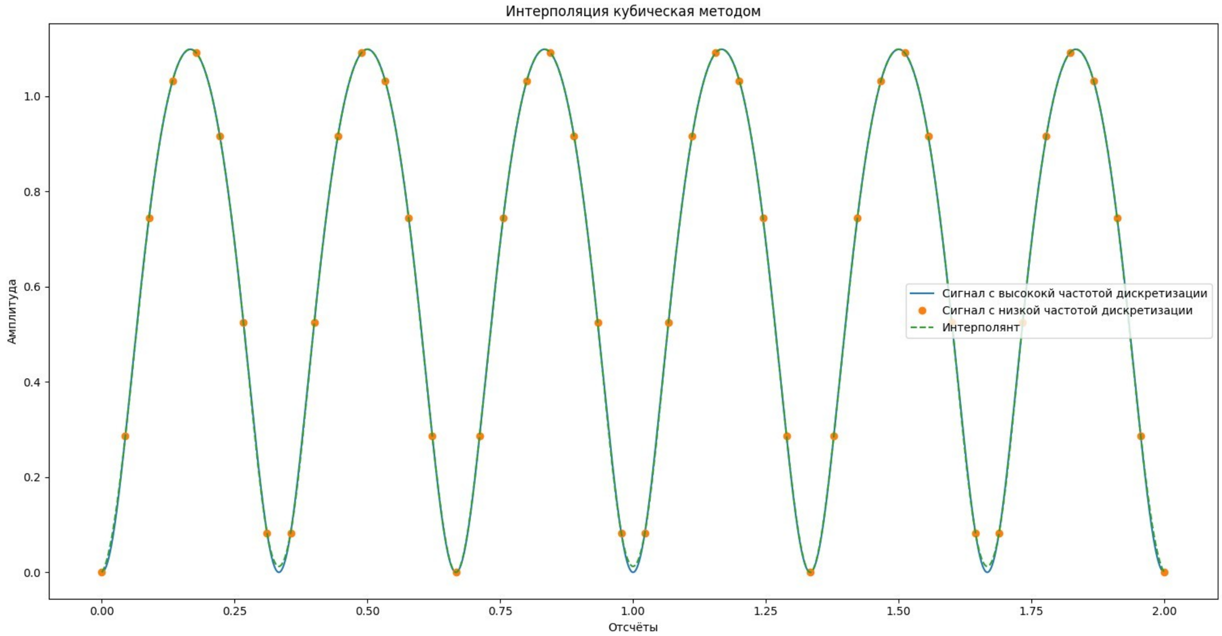

Рисунок 4.3 — Результат кубической интерполяции при 46 точках

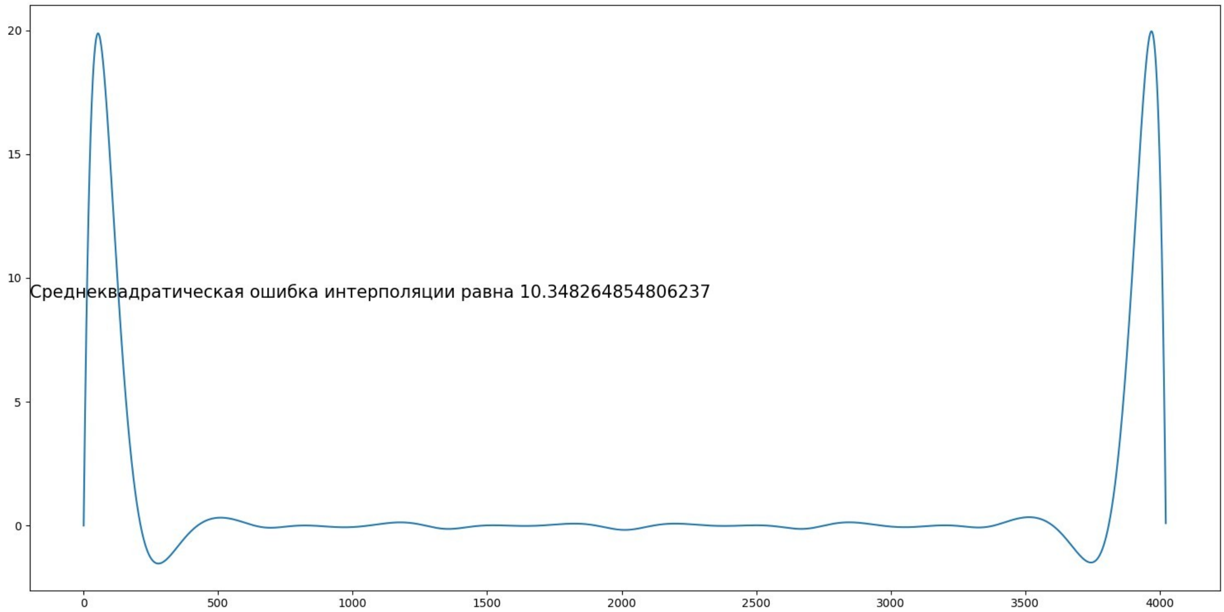

Рисунок 4.4 – График разности между заданной функцией и интерполянтом полученным методом Лагранжа, при количестве точек 20. Среднеквадратическая ошибка интерполяции выведена в виде текста в середину

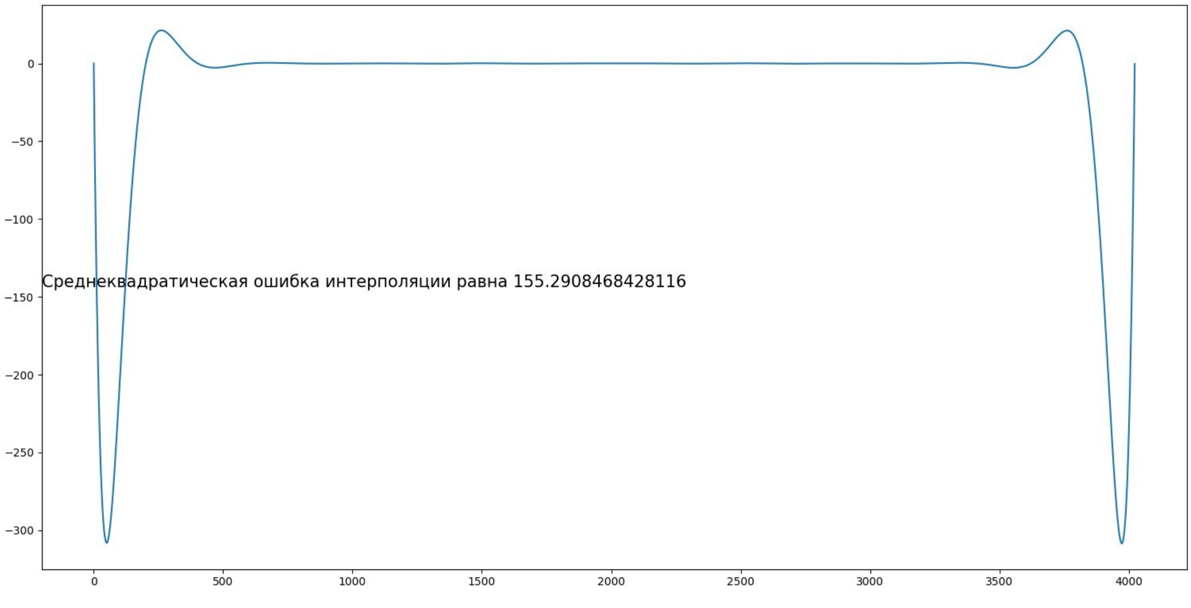

Рисунок 4.5 – График разности между заданной функцией и интерполянтом полученным методом Лагранжа, при количестве точек 21. Среднеквадратическая ошибка интерполяции выведена в виде текста в середину

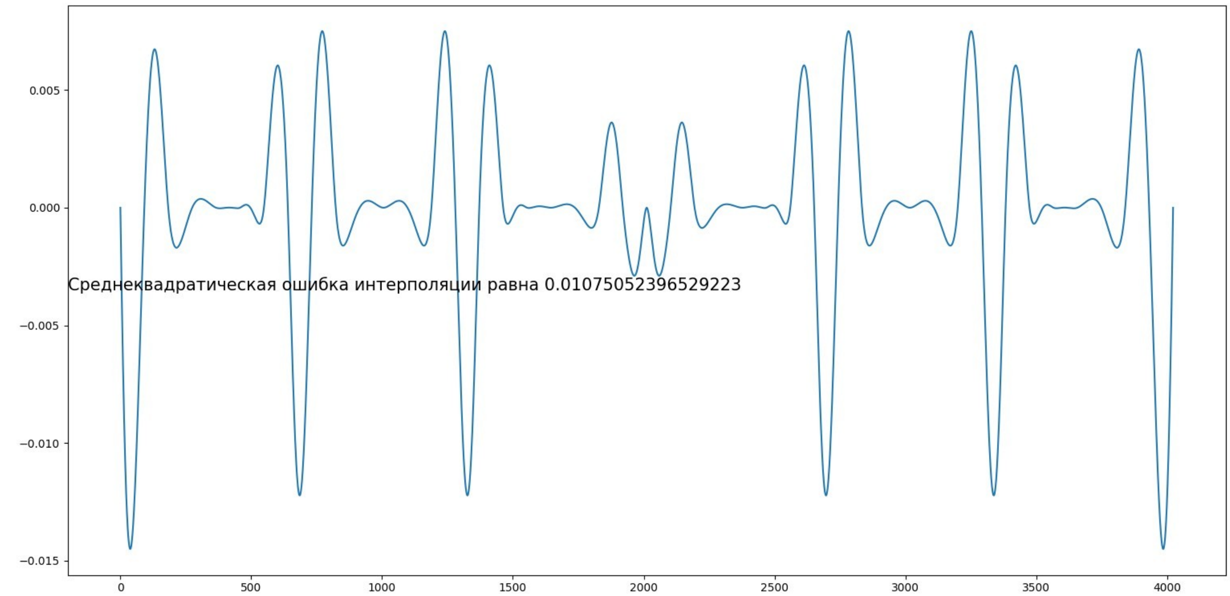

Рисунок 4.6 – График разности между заданной функцией и интерполянтом кубического сплайна, при количестве точек 45. Слева по середине выведена среднеквадратическая ошибка интерполяции

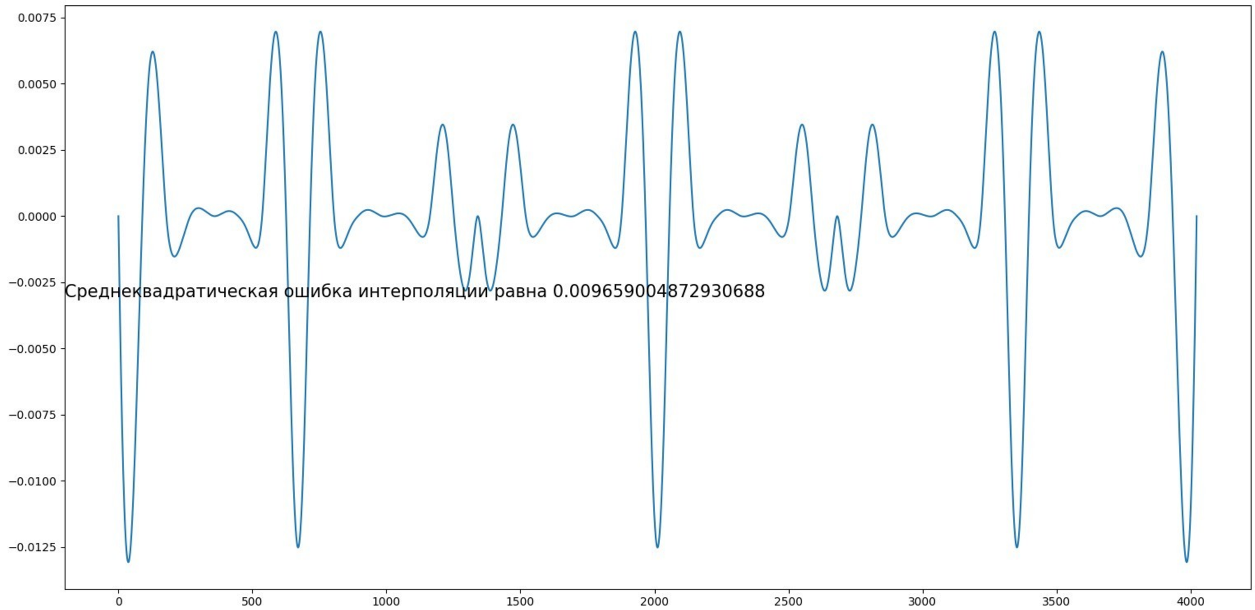

Рисунок 4.6 – График разности между заданной функцией и интерполянтом кубического сплайна, при количестве точек 46. Слева по середине выведена среднеквадратическая ошибка интерполяции

4 Выводы

В рамках данной работы я на практике проверил работоспособность интерполяций, как методом Лагранжа, так и кубическим сплайном. В результате убедился, что метод Лагранжа при количестве узлов больше 20 проводит интерполяцию неверно и приводит к большим искажениям, называемым краевыми эффектами, именно их и можно заметить на рисунках 4.1 и 4.2, на рисунке 4.2 краевые эффекты становятся слишком заметными, а среднеквадратическая ошибка становиться намного (в несколько раз) заметнее с каждым новым узлом, это видно на рисунках 4.4 и 4.5. Интерполяция кубическим методом при увеличении количества точек на заданном отрезке даёт более точные результаты. Это показано на рисунках 4.6 и 4.7, где среднеквадратическая ошибка уменьшается с каждым новым узлом.

Из этого можно сделать вывод, что методом Лагранжа подходит для менее точной интерполяции (при соблюдении условия в 20 узлов) в ситуации, когда мы ограничены в вычислительных мощностях. Тогда как кубический метод позволяет наращивать точность до требуемых величин, усложняя расчёт.

Список источников

1. Цифровая обработка сигналов: учебное пособие / В.А. Сериков, В.Р. Луцив; С.-Петерб. гос. ун-т аэрокосм. приборостроения. - СПб: Изд-во ГУАП, 2014. – 110 с. [библиотечный шифр 621.391 С32]

2. Лекция 9: Методы интерполяции и их применение в ЦОС. О.О. Жаринов, ГУАП, 9 сентября 2020г. [Электронный ресурс]. – Режим доступа: https://bbb4.guap.ru/playback/presentation/2.0/playback.html?meetingId=52ed67f6ad8cd06c8d3c56a487d54eb4466bbaa8-1599652620665. – Загл. с экрана. (Дата обращения 25.09.2020г.).

3. Теоретические основы сплайн-интерполяции или почему IQ тесты не имеют решения. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/323442/ – Загл. с экрана. (Дата обращения 25.09.2020г.).

Приложение А – Программа

```
# Const
N = 4024


def signal_generate(time, function):
    """
    Генерирует сигнал с заданными параметрами

    :time: Отсчёты сигнала
    :function: Функция по которой строиться график
    :returns: Сигнал с заданными параметрами
    """
    from numpy import log, pi, cos

    generated_signal = eval(function)

    return generated_signal


def interpolation_signal_plot(high_discretization_signal, high_discretization_numbers, 
low_discretization_signal,
                              low_discretization_numbers, interpolated_signal, interpolation_method):
    """
    Строит три графика с заданной дискретизацией

    :high_discretization_signal: Сигнал с высокой частотой дискретизации
    :high_discretization_numbers: Отсчёты сигнала с высокой частотой дискретизации
    :low_discretization_signal: Сигнал с низкой частотой дискретизации
    :low_discretization_numbers: Отсчёты сигнала с низкой частотой дискретизации
    :input_signal: Сигнал с низкой частотой дискретизации прошедший интерполяцию
    :interpolation_method: Метод интерполяции
    """
    import matplotlib.pyplot as plt
    import numpy as np

    plt.figure()
    plt.title("Интерполяция {} методом".format(interpolation_method))
    plt.xlabel('Отсчёты')
    plt.ylabel('Амплитуда')
    plt.plot(high_discretization_numbers, high_discretization_signal, label="Сигнал с высококй 
частотой дискретизации")
    plt.plot(low_discretization_numbers, low_discretization_signal, "o", label="Сигнал с низкой 
частотой дискретизации")
    plt.plot(high_discretization_numbers, interpolated_signal, "--", label="Интерполянт")
    plt.legend()


def main():
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.interpolate import lagrange, CubicSpline
```

```
    from statistics import stdev

    # Генерируем сигнал
    n = np.linspace(0, 2, N)
    signal_function = 'log(2 - cos(2 * pi * 3 * time))'
    generated_signal = signal_generate(n, signal_function)

    # Генерируем дискритизацию сигнала
    M = 46
    sd = np.linspace(0, 2, M)
    points_of_generated_signal = signal_generate(sd, signal_function)

    # Интерполяция методом лагранжа
    lagrange_interpolation = lagrange(sd, points_of_generated_signal)
    interpolation_signal_plot(generated_signal, n, points_of_generated_signal, sd,
lagrange_interpolation(n), "полиномиальным")

    # Кубическая интерполяция
    cubic_spline_interpolation = CubicSpline(sd, points_of_generated_signal)
    interpolation_signal_plot(generated_signal, n, points_of_generated_signal, sd,
cubic_spline_interpolation(n), "кубическая")

    # Определяем величину относительной среднеквадратической ошибки интерполяции
    # Для полиномиальной интерполяции методом лагранжа
    delta_lagrange_interpolation = generated_signal - lagrange_interpolation(n)
    epsilon_lagrange = stdev(delta_lagrange_interpolation)/stdev(generated_signal)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(delta_lagrange_interpolation)
    ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна
{}".format(epsilon_lagrange), transform=ax.transAxes, fontsize=15)

    # Для кубической интерполяции
    delta_spline_interpolation = generated_signal - cubic_spline_interpolation(n)
    epsilon_spline = stdev(delta_spline_interpolation)/stdev(generated_signal)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(delta_spline_interpolation)
    ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна
{}".format(epsilon_spline), transform=ax.transAxes, fontsize=15)

    plt.show()


if __name__ == "__main__":
    main()
```