


ГУАП

КАФЕДРА № 41

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

ассистент

должность, уч. степень, звание

 24.09.20

подпись, дата

А.К. Акопян .

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



подпись, дата

Хасанов Б.Р.

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 процессе математического моделирования приходится вычислять значения функций, входящих в математическое описание модели. Для сложных моделей такие расчеты могут оказаться трудоемкими даже при использовании ЭВМ. Также в математических моделях часто используются функциональные зависимости, аналитические выражения для которых неизвестны. Примером таких зависимостей, широко используемых в электротехнике, могут служить зависимости между индукцией магнитного поля  $B$  и напряженностью магнитного поля  $H$  для специальных электротехнических сталей. Для каждого сорта стали эти зависимости получают опытным путем, изменяя с определенным шагом напряженность  $H$  и измеряя полученное значение индукции  $B$ . По результатам опытов строят таблицы значений  $H$  и  $B$ . Данные таблицы применяются для расчетов электрических машин и аппаратов на производстве и при математическом моделировании данных устройств [4].

Основной недостаток табличного задания функциональных зависимостей по сравнению с аналитическим - отсутствие информации о значении функции в промежуточных точках. Сказанное выше можно отнести ко всем имеющим дискретный характер зависимостям, например, к результатам численного моделирования, так как расчет всегда ведется с определенным шагом аргумента. При использовании дискретных зависимостей встают по крайней мере, две проблемы - определение значения функции в промежуточных точках и экономия оперативной памяти ЭВМ при их использовании в математическом моделировании [4].

Данные проблемы решают следующим образом. Функцию  $f(x)$ , известную аналитически или таблично, заменяют более простой функцией  $\varphi(x)$ , которую нетрудно вычислять при любом значении аргумента  $x$  в заданном интервале его изменения. Приближение функции  $f(x)$  более простой функцией  $\varphi(x)$  называется аппроксимацией. Частным случаем аппроксимации является интерполяция. При интерполяции значения  $\varphi(x)$  совпадают со значениями аппроксимируемой функции  $f(x)$  в узлах таблицы [4].

### 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 дискретизированного сигнала с помощью всё той же функции `signal_generate`. В данном случае.  $M$  – количество точек в дискретизированном сигнале. Именно его нам нужно будет менять по заданию, чтобы получить требуемое значение среднеквадратической ошибки интерполяции:

```
M = 50
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 М. 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 интерполянт Лагранжа, результат на рисунке 4.4

```
delta_lagrange_interpolation = generated_signal - lagrange_interpolation(n)
epsilon_lagrange = stdev(delta_lagrange_interpolation)/stdev(generated_signal)
plt.figure()
plt.plot(delta_lagrange_interpolation)
plt.text(0,0, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_lagrange))
```

Продолываем всё то же самое для кубического интерполянта, результат на рисунке 4.5.

```
delta_spline_interpolation = generated_signal - cubic_spline_interpolation(n)
epsilon_spline = stdev(delta_spline_interpolation)/stdev(generated_signal)
plt.figure()
plt.plot(delta_spline_interpolation)
plt.text(0,0, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_spline))
```

## 4 Полученные графики

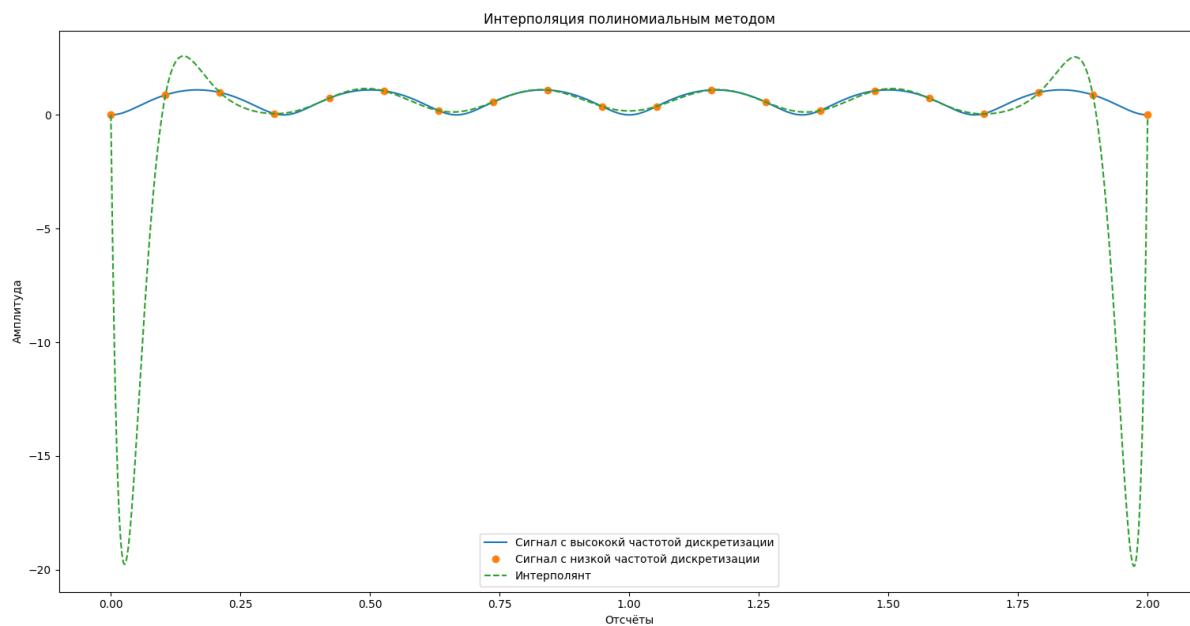


Рисунок 4.1 — Интерполяция методом Лагранжа при количестве точек 20

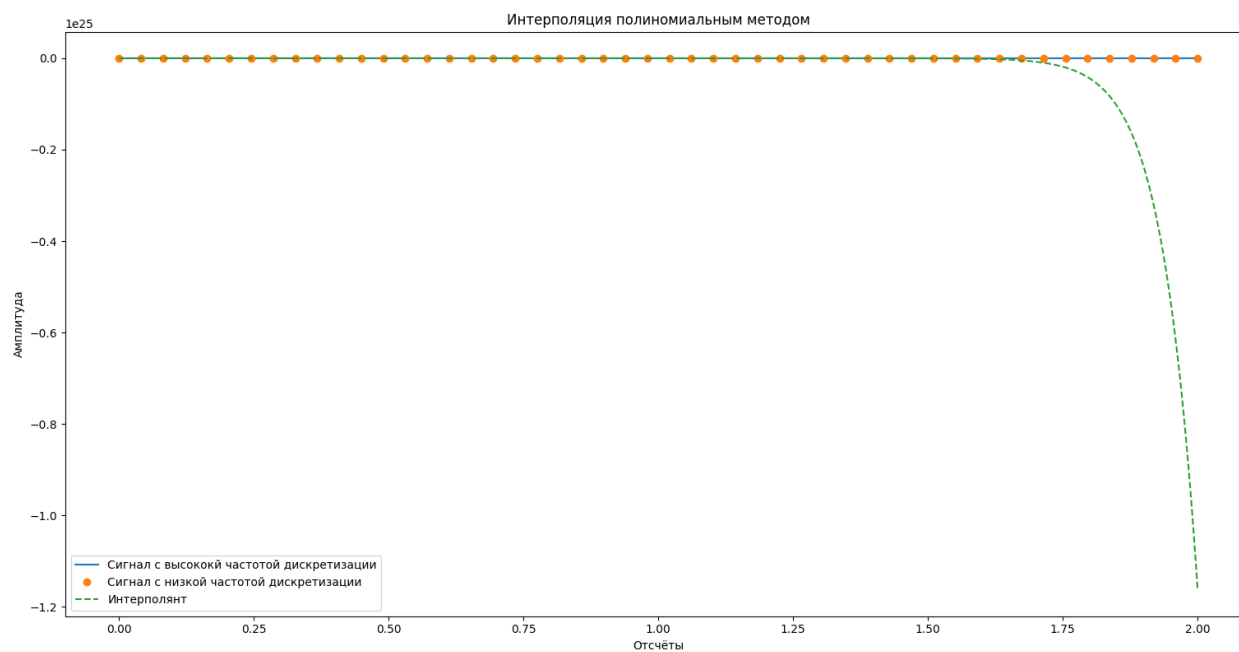


Рисунок 4.2 — Интерполяция методом Лагранжа при количестве точек 50

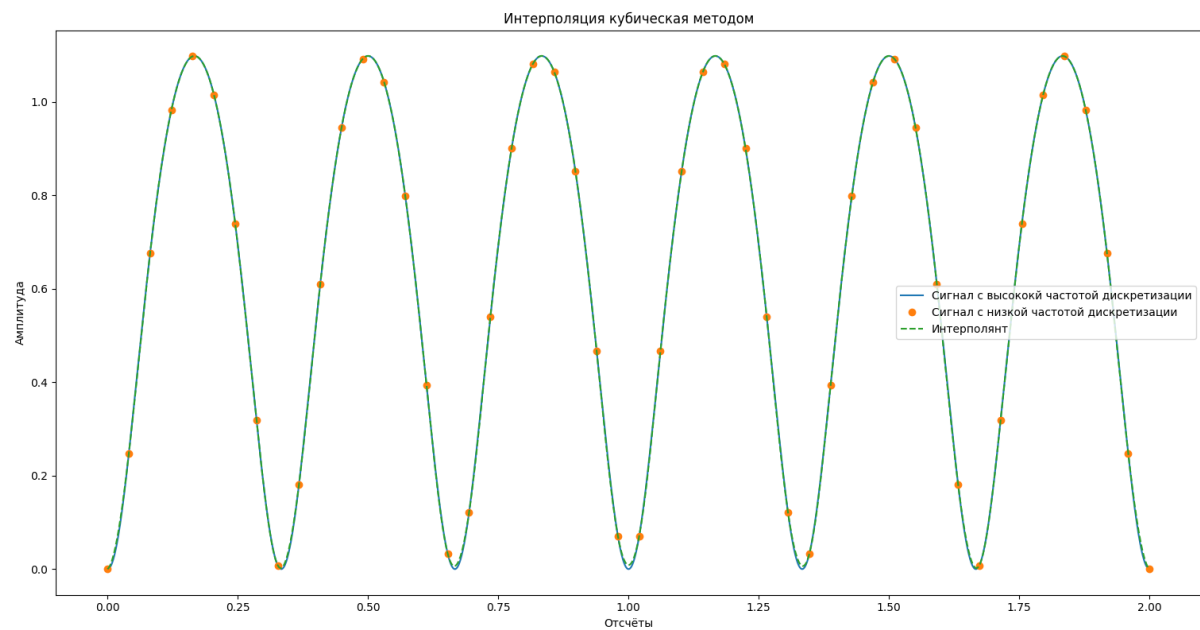


Рисунок 4.3 — Результат кубической интерполяции при 50 точках

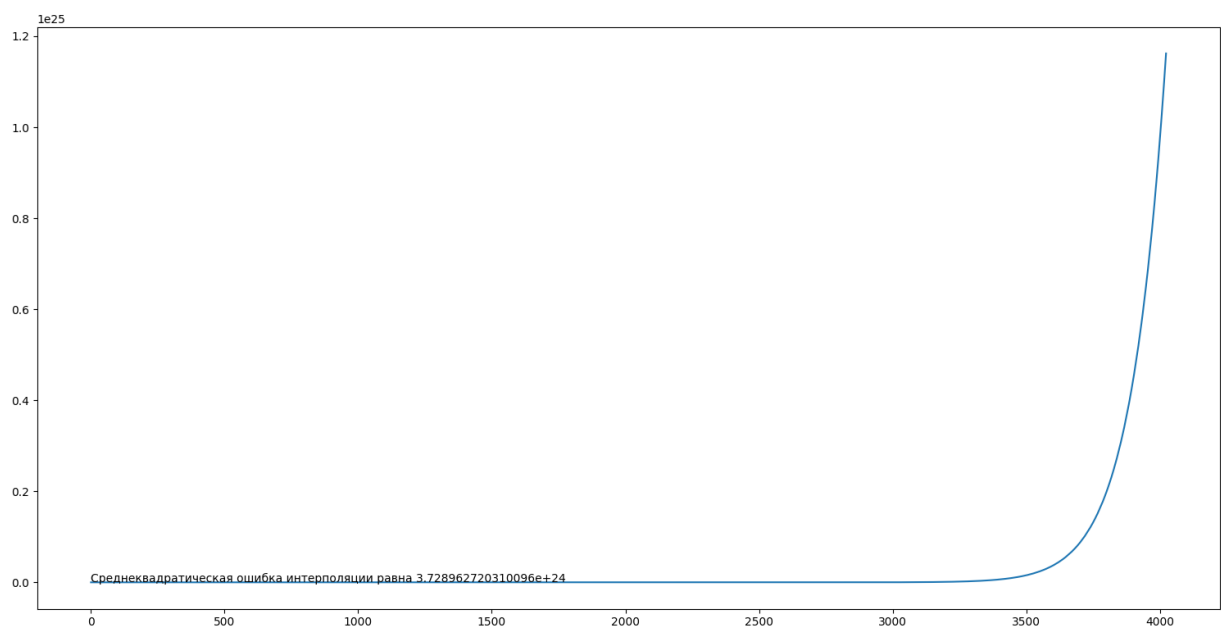


Рисунок 4.4 – График разности между заданной функцией и интерполантом полученным методом Лагранжа, при количестве точек 50. В нижнем левом углу выведена среднеквадратическая ошибка интерполяции

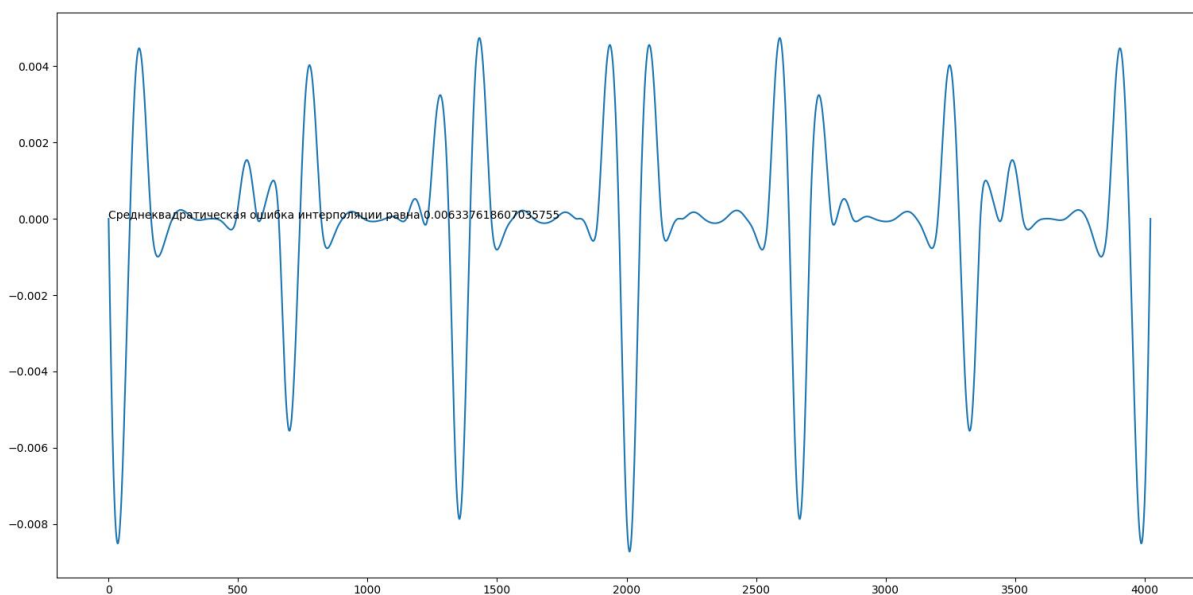


Рисунок 4.5 – График разности между заданной функцией и интерполянтom кубического сплайна, при количестве точек 50. Слева по середине выведена среднеквадратическая ошибка интерполяции

#### 4 Выводы

В результате работы получили два массива значений ошибок: для полиномиальной и кубической интерполяций. Сравнение двух методов удобнее проводить на примере рисунка 4.5.1, так как это график зависимости величины относительной среднеквадратической ошибки интерполяции от количества узлов. Из рисунка мы видим, что полиномиальная интерполяция первые 32 отсчета не совпадает требованию, которое изложено в задании: ошибка не должна быть более 0,02. При количестве узлов равном 33 полиномиальная интерполяция начинает удовлетворять требованиям: ошибка с увеличением числа узлов стремительно уменьшается. Кубическая интерполяция же только при  $M=36$  начинает удовлетворять требованию. При этом значение ошибки падает медленно, почти линейно. На основании этого можно предположить, что кубическая интерполяция более предсказуема, плавная. У нее не замечено резких скачков, как у полиномиальной. Но у последней есть неоспоримое преимущество – низкое значение ошибки при больших  $M$ . Каким методом пользоваться, как всегда, решает пользователь, ибо каждая техническая задача по-своему уникальна. Но использовать кубическую интерполяцию, как следует из полученных графиков, следует при малых  $M$ , когда количество узлов не велико. Казалось бы, что полиномиальная тогда наоборот должна выигрывать при больших, даже слишком больших  $M$ . Но при построении графика, представленного на рисунке 4.6 мы видим, что с

повышением числа  $M$  ошибка у полиномиальной интерполяции начинает расти. Это наталкивает нас на мысль: либо мы используем кубический метод на достаточно широком диапазоне числа  $M$ , но довольствуемся достаточно средними показателями ошибки, хотя и удовлетворяющими требованиям, либо используем на узком диапазоне числа  $M$  полиномиальную интерполяцию, но получаем высокую точность, очень низкий уровень ошибки.

Кроме того, опыты показали, что период дискретизации не влияет на характер линий на графиках ошибок, что отчетливо видно на рисунках 4.5.1-4.5.4, а также рисунках 4.6 и 4.7.



### **Список источников**

1. Цифровая обработка сигналов: учебник для ВПО /С.Н. Воробьев. - М.: Академия, 2013. - 320 с. [библиотечный шифр 621.391 В75]
2. Цифровая обработка сигналов: учебное пособие / В.А. Сериков, В.Р. Луцев; С. - Петерб. гос. ун-т аэрокосм. приборостроения. - СПб: Изд-во ГУАП, 2014. – 110 с. [библиотечный шифр 621.391 С32]
3. 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 11.09.2020г.).
4. Аппроксимация и интерполяция в электротехнике. URL: [https://studopedia.ru/9\\_211032\\_approksimatsiya-i-interpolyatsiya-v-elektrotehnike.html](https://studopedia.ru/9_211032_approksimatsiya-i-interpolyatsiya-v-elektrotehnike.html) (Дата обращения 11.09.2020)

## 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 высокочастотой 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 дискретизацию сигнала
    M = 50
    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)
plt.figure()
plt.plot(delta_lagrange_interpolation)
plt.text(0,0, "Среднеквадратическая ошибка интерполяции равна { }".format(epsilon_lagrange))

# Для кубической интерполяции
delta_spline_interpolation = generated_signal - cubic_spline_interpolation(n)
epsilon_spline = stdev(delta_spline_interpolation)/stdev(generated_signal)
plt.figure()
plt.plot(delta_spline_interpolation)
plt.text(0,0, "Среднеквадратическая ошибка интерполяции равна { }".format(epsilon_spline))

plt.show()

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

