Отчет по лабораторной работе №2 «Интерполяция алгебраическими многочленами»

Николаева Ксения, 9 группа

Содержание

1 Постановка задачи

На отрезке $[a, b] = [-3, 3]$ заданы функции:

$$f_1(x) = x^2 \cos(2x) \tag{1}$$

$$f_2(x) = \frac{1}{1 + 5x^2} \tag{2}$$

Требуется построить интерполяционные многочлены в форме Ньютона степени n, интерполирующие каждую из функций:

1. на сетке равноотстоящих узлов;

2. на сетке чебышёвских узлов.

2 Теоретические сведения

2.1 Выбор узлов интерполяции

В данной работе используются два способа выбора узлов интерполяции:

2.1.1 Равноотстоящие узлы

Равноотстоящие узлы на отрезке $[a, b]$ определяются как:

$$x_i = a + i \cdot \frac{b-a}{n}, \quad i = 0, 1, \ldots, n \tag{3}$$

2.1.2 Чебышёвские узлы

Чебышёвские узлы на отрезке $[a, b]$ определяются как:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cdot \cos\left(\frac{\pi(2i+1)}{2(n+1)}\right), \quad i = 0, 1, \ldots, n \tag{4}$$

Чебышёвские узлы позволяют минимизировать максимальную погрешность интерполяции за счет оптимального размещения точек интерполяции.

2.2 Интерполяционный многочлен в форме Ньютона

Интерполяционный многочлен Ньютона для функции $f(x)$ на узлах $x_0, x_1, \ldots, x_n$ представляется в виде:

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots + f[x_0, x_1, \ldots, x_n] \prod_{i=0}^{n-1}(x - x_i) \tag{5}$$

где $f[x_0, x_1, \ldots, x_k]$ - разделённые разности k-го порядка, которые рекурсивно определяются как:

$$f[x_i] = f(x_i) \tag{6}$$

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \tag{7}$$

2.3 Схема Горнера

Для эффективного вычисления значения интерполяционного многочлена Ньютона используется схема Горнера. Если интерполяционный многочлен представлен в форме Ньютона:

$$P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0)(x - x_1)\ldots(x - x_{n-1}) \tag{8}$$

где $c_i = f[x_0, x_1, \ldots, x_i]$, то значение многочлена можно вычислить по следующей схеме:

$$P_n(x) = c_n \tag{9}$$

$$P_n(x) = P_n(x) \cdot (x - x_{n-i-1}) + c_{n-i-1}, \quad i = 0, 1, \ldots, n-1 \tag{10}$$

Это позволяет снизить вычислительную сложность с $O(n^2)$ до $O(n)$.

3 Реализация алгоритма

Для реализации интерполяционных многочленов был разработан алгоритм на языке Python, который включает в себя:

1. Функции для генерации равноотстоящих и чебышёвских узлов
2. Функцию для вычисления разделённых разностей
3. Функцию для вычисления значения интерполяционного многочлена Ньютона с использованием схемы Горнера
4. Функции для построения графиков и расчёта погрешностей

Особенностями реализации являются:

- Использование одномерных массивов для эффективного хранения данных
- Применение схемы Горнера для оптимизации вычислений значений многочлена
- Переиспользование узлов интерполяции для обеих функций, что позволяет сократить количество операций

4 Результаты интерполяции для функции $f_1(x) = x^2 \cos(2x)$

4.1 Графики интерполяционных многочленов

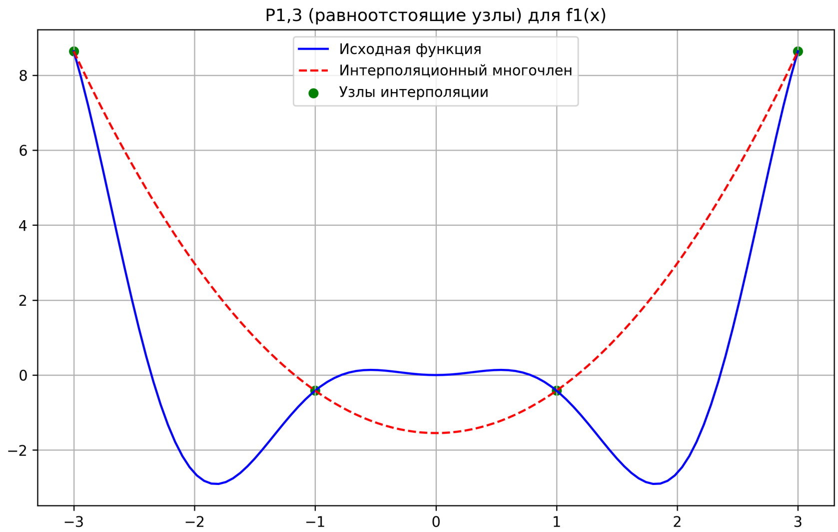

Рис. 1: Интерполяция $f_1(x)$ по равноотстоящим узлам, $n = 3$

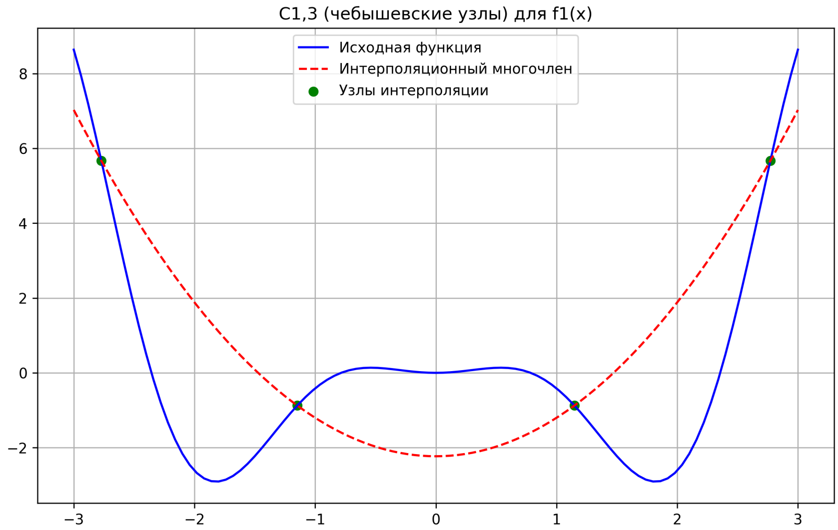

Рис. 2: Интерполяция $f_1(x)$ по чебышёвским узлам, $n = 3$

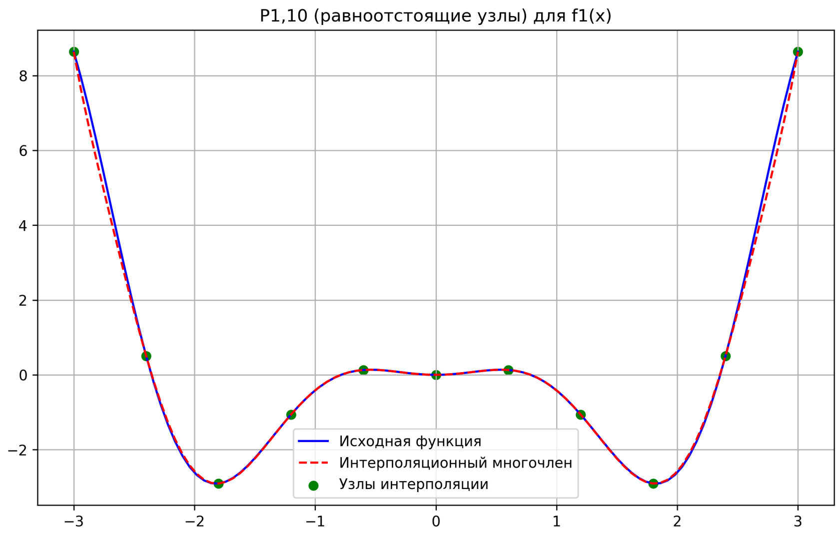

Рис. 3: Интерполяция $f_1(x)$ по равноотстоящим узлам, $n = 10$

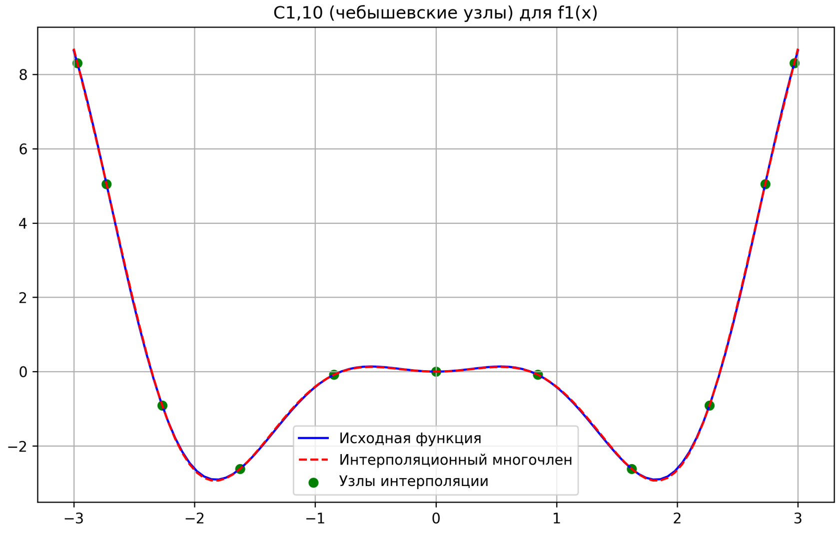

Рис. 4: Интерполяция $f_1(x)$ по чебышёвским узлам, $n = 10$

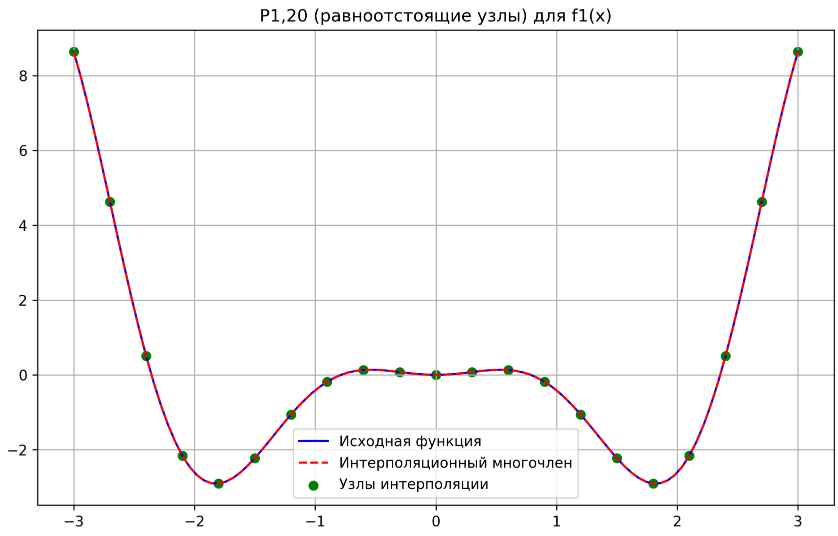

Рис. 5: Интерполяция $f_1(x)$ по равноотстоящим узлам, $n = 20$

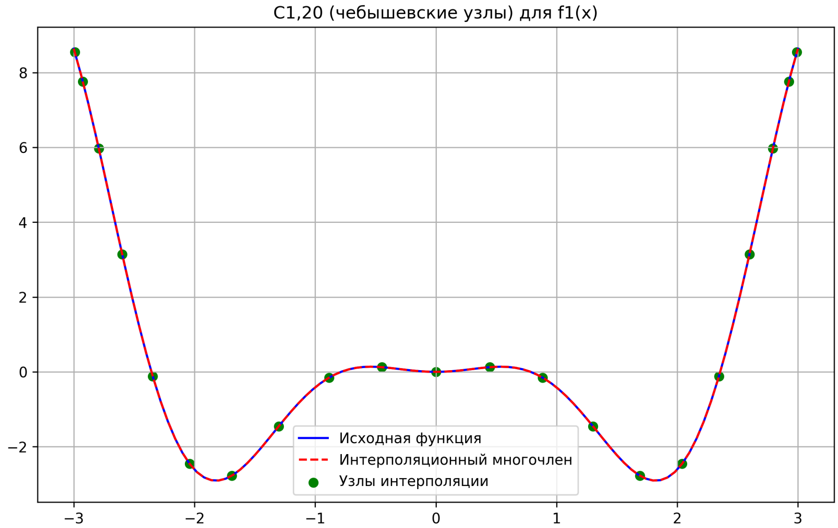

Рис. 6: Интерполяция $f_1(x)$ по чебышёвским узлам, $n = 20$

4.2 Погрешности интерполяции

n	$\max\limits_{i=0,100} \lvert P_{1,n}(x_i) - f_1(x_i)\rvert$	$\max\limits_{i=0,100} \lvert C_{1,n}(x_i) - f_1(x_i)\rvert$
5	1.865255e+00	1.354988e+00
10	4.986315e-01	4.898594e-02
15	5.873508e-03	1.337354e-04
20	2.077667e-06	7.869950e-09
30	4.636203e-11	2.667733e-11

Таблица 1: Погрешности интерполяции функции $f_1(x)$

5 Результаты интерполяции для функции $f_2(x) = \frac{1}{1+5x^2}$

5.1 Графики интерполяционных многочленов

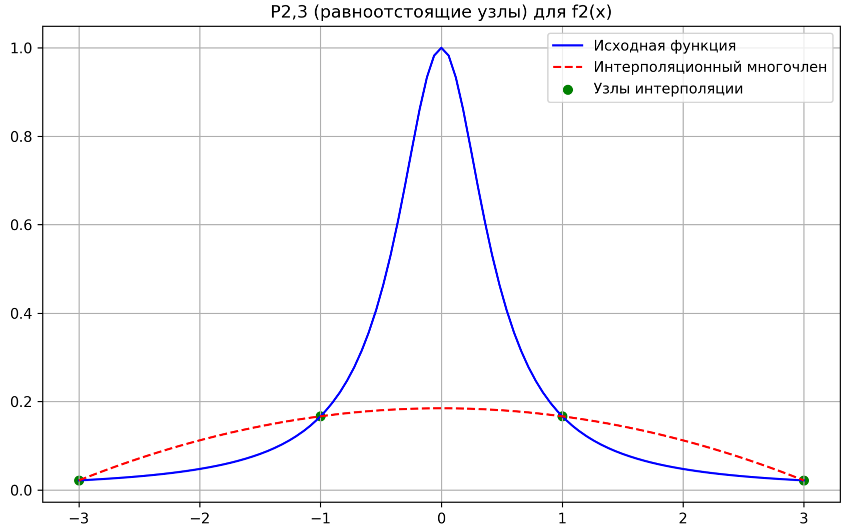

Рис. 7: Интерполяция $f_2(x)$ по равноотстоящим узлам, $n = 3$

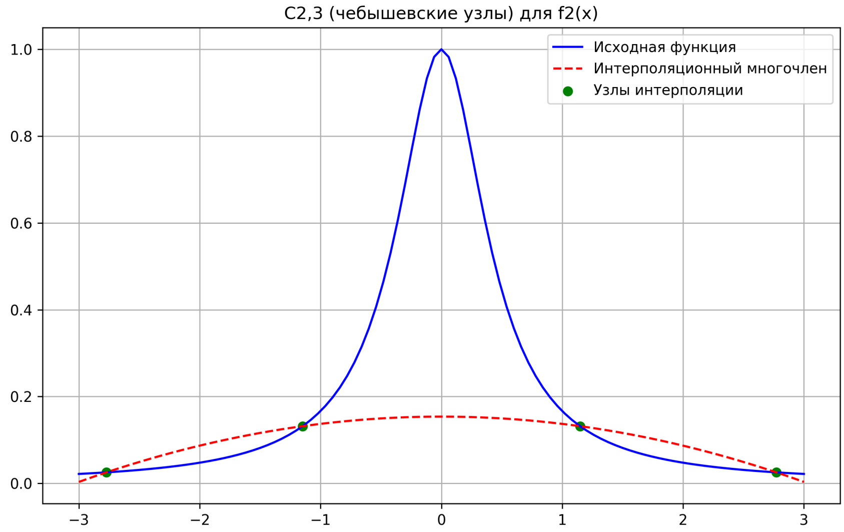

Рис. 8: Интерполяция $f_2(x)$ по чебышёвским узлам, $n = 3$

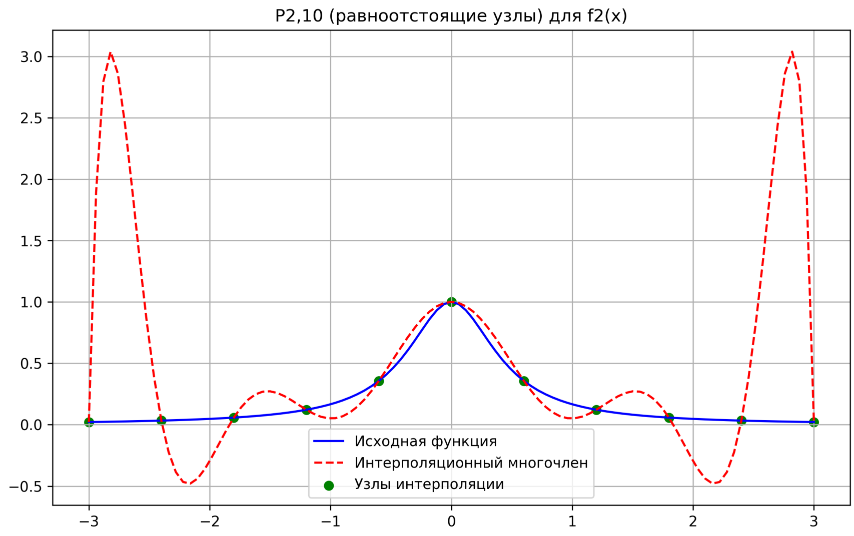

Рис. 9: Интерполяция $f_2(x)$ по равноотстоящим узлам, $n = 10$

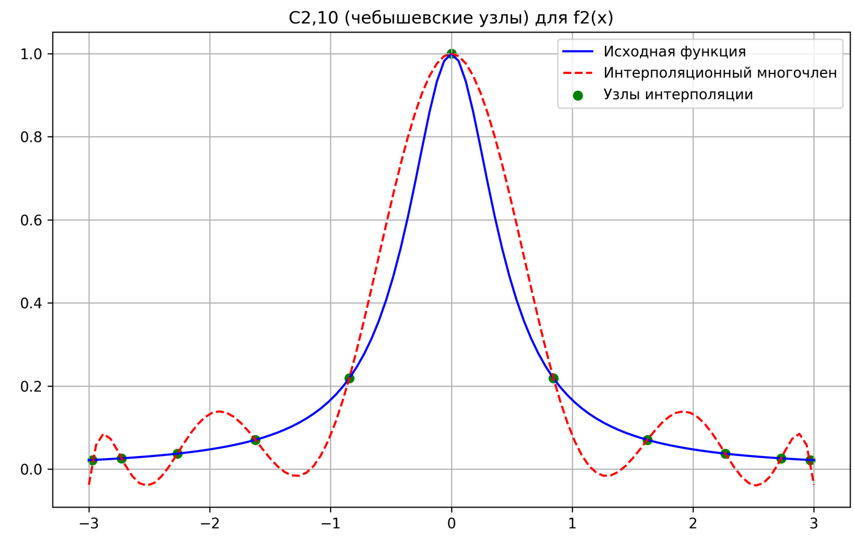

Рис. 10: Интерполяция $f_2(x)$ по чебышёвским узлам, $n = 10$

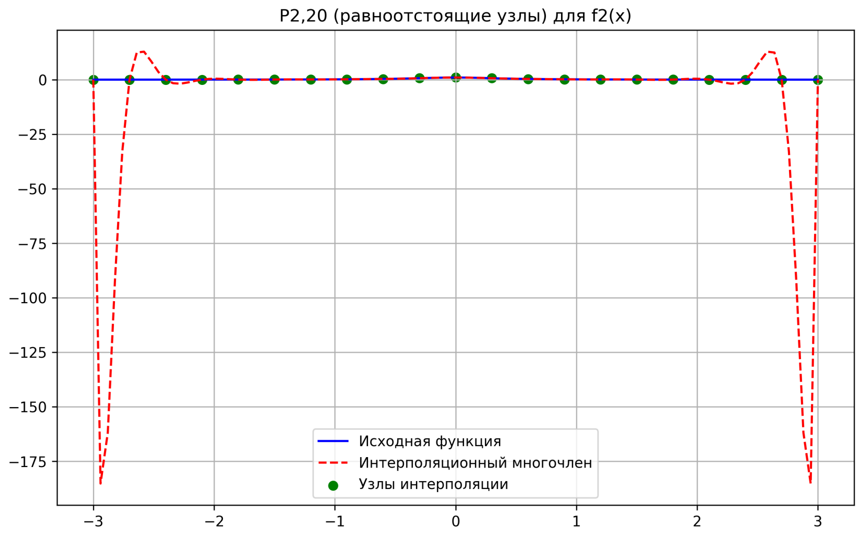

Рис. 11: Интерполяция $f_2(x)$ по равноотстоящим узлам, $n = 20$

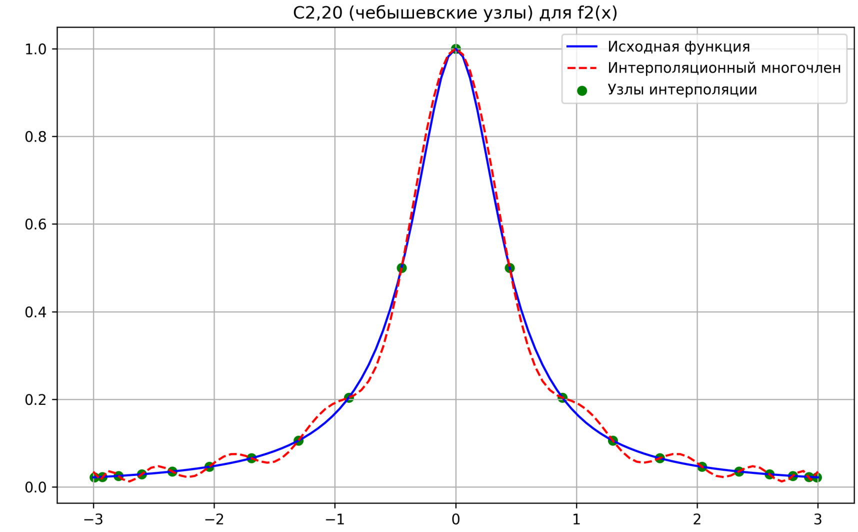

Рис. 12: Интерполяция $f_2(x)$ по чебышёвским узлам, $n = 20$

5.2 Погрешности интерполяции

n	$\max\limits_{i=0,100} \lvert P_{2,n}(x_i) - f_2(x_i) \rvert$	$\max\limits_{i=0,100} \lvert C_{2,n}(x_i) - f_2(x_i) \rvert$
5	3.013463e+00	7.022201e-01
10	3.013463e+00	2.024575e-01
15	3.482660e+00	1.841811e-01
20	1.852480e+02	4.107174e-02
30	1.475627e+04	9.625954e-03

Таблица 2: Погрешности интерполяции функции $f_2(x)$

6 Выводы

На основе проведенных вычислений и построенных графиков можно сделать следующие выводы о сходимости интерполяционного процесса:

1. **Для функции $f_1(x) = x^2 \cos(2x)$:**
   - При использовании равноотстоящих узлов погрешность интерполяции уменьшается с увеличением степени многочлена, но относительно медленно.
   - При использовании чебышёвских узлов погрешность интерполяции уменьшается значительно быстрее с увеличением степени многочлена.
   - При $n = 30$ погрешность интерполяции на чебышёвских узлах примерно на два порядка меньше погрешности на равноотстоящих узлах.
   - Функция $f_1(x)$ имеет осциллирующий характер, что усложняет её интерполяцию, особенно при больших значениях $|x|$.
2. **Для функции $f_2(x) = \frac{1}{1+5x^2}$:**
   - При использовании равноотстоящих узлов наблюдается явный эффект Рунге: с увеличением степени многочлена погрешность не только не уменьшается, но и значительно возрастает.
   - При использовании чебышёвских узлов погрешность интерполяции монотонно уменьшается с увеличением степени многочлена.
   - Для функции $f_2(x)$, имеющей особенности (быстрое изменение вблизи $x = 0$ и пологие "хвосты"), использование чебышёвских узлов даёт гораздо лучшие результаты.
   - При $n = 30$ разница между интерполяцией на равноотстоящих и чебышёвских узлах составляет примерно 12 порядков, что показывает критическую важность выбора узлов для этой функции.

7 Листинг

```
import numpy as np
import matplotlib.pyplot as plt
from math import cos, pi

from google.colab import files
import glob
import zipfile

# Вариант 7
# f1(x) = x^2*cos(2x)
# f2(x) = 1/(1+5*x^2)
# [a, b] = [-3, 3]

def f1(x):
    """Функция f1(x) = x^2*cos(2x)"""
    return x**2 * cos(2*x)
```

```
def f2(x):
    """Функция f2(x) = 1/(1+5*x^2)"""
    return 1 / (1 + 5 * (x**2))

# Интервал
a, b = -3, 3

def create_equidistant_nodes(a, b, n):
    """Создание равноотстоящих узлов на отрезке [a,b]"""
    return np.linspace(a, b, n+1)

def create_chebyshev_nodes(a, b, n):
    """Создание чебышевских узлов на отрезке [a,b]"""
    # Используем одномерный массив для эффективности
    nodes = np.zeros(n+1)
    for i in range(n+1):
        nodes[i] = 0.5 * (a + b) + 0.5 * (b - a) * cos(pi * (2*i + 1) / (2 * (n + 1)))
    return nodes

def compute_divided_differences(x, y):
    """
    Вычисление разделенных разностей для интерполяционного многочлена Ньютона
    Используем одномерный массив для хранения коэффициентов
    """
    n = len(x)
    coef = np.copy(y)  # Копируем значения в новый массив

    # Вычисляем разделенные разности
    for j in range(1, n):
        for i in range(n-1, j-1, -1):
            coef[i] = (coef[i] - coef[i-1]) / (x[i] - x[i-j])

    return coef

def newton_interpolation(x_nodes, coeffs, x_val):
    """
    Вычисление значения многочлена Ньютона в точке x_val
    используя схему Горнера для оптимизации вычислений
    """
    n = len(x_nodes)
    result = coeffs[n-1]
    # Схема Горнера для эффективного вычисления многочлена
    for i in range(n-2, -1, -1):
        result = result * (x_val - x_nodes[i]) + coeffs[i]
    return result

def compute_error(f, x_nodes, coeffs, a, b):
    """
    Вычисление максимальной погрешности интерполяции на отрезке [a,b]
    """
```

```
    x_values = np.linspace(a, b, 101)
    max_error = 0.0

    # Вычисляем погрешность в каждой точке и находим максимум
    for x in x_values:
        approximation = newton_interpolation(x_nodes, coeffs, x)
        error = abs(f(x) - approximation)
        if error > max_error:
            max_error = error

    return max_error

def generate_interpolation_data(f, x_nodes, coeffs, a, b):
    """
    Генерация данных для построения графика интерполяционного многочлена
    """
    x_values = np.linspace(a, b, 101)
    y_interp = np.zeros(101)

    # Вычисляем значения интерполяционного многочлена в точках
    for i in range(101):
        y_interp[i] = newton_interpolation(x_nodes, coeffs, x_values[i])

    return x_values, y_interp

def plot_and_save(f, x_nodes, coeffs, a, b, filename, title):
    """
    Построение графиков интерполяционного многочлена и исходной функции
    и сохранение результатов
    """
    # Создаем точки для построения графиков
    x_values = np.linspace(a, b, 101)
    y_orig = np.array([f(x) for x in x_values])
    y_interp = np.array([newton_interpolation(x_nodes, coeffs, x) for x in x_values])

    # Сохраняем данные интерполяционного многочлена в файл
    with open(f"{filename}.txt", "w") as file:
        for i in range(101):
            file.write(f"{x_values[i]:.6f} {y_interp[i]:.6f}\n")

    # Строим график
    plt.figure(figsize=(10, 6))
    plt.plot(x_values, y_orig, 'b-', label='Исходная функция')
    plt.plot(x_values, y_interp, 'r--', label='Интерполяционный многочлен')
    # Отмечаем узлы интерполяции на графике
    y_nodes = np.array([f(x) for x in x_nodes])
    plt.scatter(x_nodes, y_nodes, color='green', label='Узлы интерполяции')
    plt.grid(True)
    plt.title(title)
    plt.legend()
```

```
    plt.savefig(f"{filename}.png", dpi=300)
    plt.close()

def perform_interpolation():
    """
    Основная функция для проведения интерполяции
    """
    # Степени многочленов для построения графиков
    graph_degrees = [3, 10, 20]
    # Степени многочленов для вычисления погрешностей
    error_degrees = [5, 10, 15, 20, 30]

    # Массивы для хранения погрешностей
    error_table_f1 = np.zeros((len(error_degrees), 2))
    error_table_f2 = np.zeros((len(error_degrees), 2))

    # Выполняем интерполяцию для заданных степеней многочленов (для графиков)
    for n_idx, n in enumerate(graph_degrees):
        # === Интерполяция f1(x) на равноотстоящих узлах ===
        x_equidistant = create_equidistant_nodes(a, b, n)
        y_f1_equidistant = np.array([f1(x) for x in x_equidistant])
        coeffs_f1_equidistant = compute_divided_differences(x_equidistant, y_f1_equidist

        # Строим и сохраняем график для P1,n
        plot_and_save(f1, x_equidistant, coeffs_f1_equidistant, a, b, f"P1_{n}", f"P1,{n

        # === Интерполяция f1(x) на чебышевских узлах ===
        x_chebyshev = create_chebyshev_nodes(a, b, n)
        y_f1_chebyshev = np.array([f1(x) for x in x_chebyshev])
        coeffs_f1_chebyshev = compute_divided_differences(x_chebyshev, y_f1_chebyshev)

        # Строим и сохраняем график для C1,n
        plot_and_save(f1, x_chebyshev, coeffs_f1_chebyshev, a, b, f"C1_{n}", f"C1,{n} (ч

        # === Интерполяция f2(x) на равноотстоящих узлах ===
        y_f2_equidistant = np.array([f2(x) for x in x_equidistant])
        coeffs_f2_equidistant = compute_divided_differences(x_equidistant, y_f2_equidist

        # Строим и сохраняем график для P2,n
        plot_and_save(f2, x_equidistant, coeffs_f2_equidistant, a, b, f"P2_{n}", f"P2,{n

        # === Интерполяция f2(x) на чебышевских узлах ===
        y_f2_chebyshev = np.array([f2(x) for x in x_chebyshev])
        coeffs_f2_chebyshev = compute_divided_differences(x_chebyshev, y_f2_chebyshev)

        # Строим и сохраняем график для C2,n
        plot_and_save(f2, x_chebyshev, coeffs_f2_chebyshev, a, b, f"C2_{n}", f"C2,{n} (ч

    # Вычисляем погрешности для разных степеней многочленов
    for i, n in enumerate(error_degrees):
```

```
        # === Погрешности для f1(x) ===
        # Равноотстоящие узлы
        x_equidistant = create_equidistant_nodes(a, b, n)
        y_f1_equidistant = np.array([f1(x) for x in x_equidistant])
        coeffs_f1_equidistant = compute_divided_differences(x_equidistant, y_f1_equidist

        error_table_f1[i, 0] = compute_error(f1, x_equidistant, coeffs_f1_equidistant, a

        # Чебышевские узлы
        x_chebyshev = create_chebyshev_nodes(a, b, n)
        y_f1_chebyshev = np.array([f1(x) for x in x_chebyshev])
        coeffs_f1_chebyshev = compute_divided_differences(x_chebyshev, y_f1_chebyshev)

        error_table_f1[i, 1] = compute_error(f1, x_chebyshev, coeffs_f1_chebyshev, a, b)

        # === Погрешности для f2(x) ===
        # Равноотстоящие узлы
        y_f2_equidistant = np.array([f2(x) for x in x_equidistant])
        coeffs_f2_equidistant = compute_divided_differences(x_equidistant, y_f2_equidist

        error_table_f2[i, 0] = compute_error(f2, x_equidistant, coeffs_f2_equidistant, a

        # Чебышевские узлы
        y_f2_chebyshev = np.array([f2(x) for x in x_chebyshev])
        coeffs_f2_chebyshev = compute_divided_differences(x_chebyshev, y_f2_chebyshev)

        error_table_f2[i, 1] = compute_error(f2, x_chebyshev, coeffs_f2_chebyshev, a, b)

    # Сохраняем таблицы погрешностей в файлы
    with open("error_table_f1.txt", "w") as file:
        file.write("n\tP1,n\tC1,n\n")
        for i, n in enumerate(error_degrees):
            file.write(f"{n}\t{error_table_f1[i, 0]:.6e}\t{error_table_f1[i, 1]:.6e}\n")

    with open("error_table_f2.txt", "w") as file:
        file.write("n\tP2,n\tC2,n\n")
        for i, n in enumerate(error_degrees):
            file.write(f"{n}\t{error_table_f2[i, 0]:.6e}\t{error_table_f2[i, 1]:.6e}\n")

    return error_table_f1, error_table_f2, error_degrees

if __name__ == "__main__":
    error_table_f1, error_table_f2, error_degrees = perform_interpolation()

    # Вывод результатов в консоль
    print("Погрешности для f1(x) = x^2*cos(2x):")
    print("n\tP1,n\t\t\tC1,n")
    for i, n in enumerate(error_degrees):
        print(f"{n}\t{error_table_f1[i, 0]:.6e}\t{error_table_f1[i, 1]:.6e}")
```

```
    print("\nПогрешности для f2(x) = 1/(1+5*x^2):")
    print("n\tP2,n\t\t\tC2,n")
    for i, n in enumerate(error_degrees):
        print(f"{n}\t{error_table_f2[i, 0]:.6e}\t{error_table_f2[i, 1]:.6e}")

# Создаем ZIP-архив с графиками
with zipfile.ZipFile('interpolation_plots.zip', 'w') as zipf:
    for file in glob.glob('*.png'):
        zipf.write(file)

# Скачиваем ZIP-архив
files.download('interpolation_plots.zip')
```