Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования.
«Рязанский государственный радиотехнический университет имени В. Ф. Уткина»

Кафедра вычислительной и прикладной математики

Отчет
о лабораторной работе № 4
по дисциплине
«Вычислительные алгоритмы»
на тему
“Построение интерполяционных многочленов”

Выполнила: ст. гр. 343 Гаджиева А.В
Проверила:
доц. Проказникова Е.Н.
ас. Щенева Ю.Б.

Рязань 2025

Дата выполнения лабораторной работы: 6.04.2025

Задание

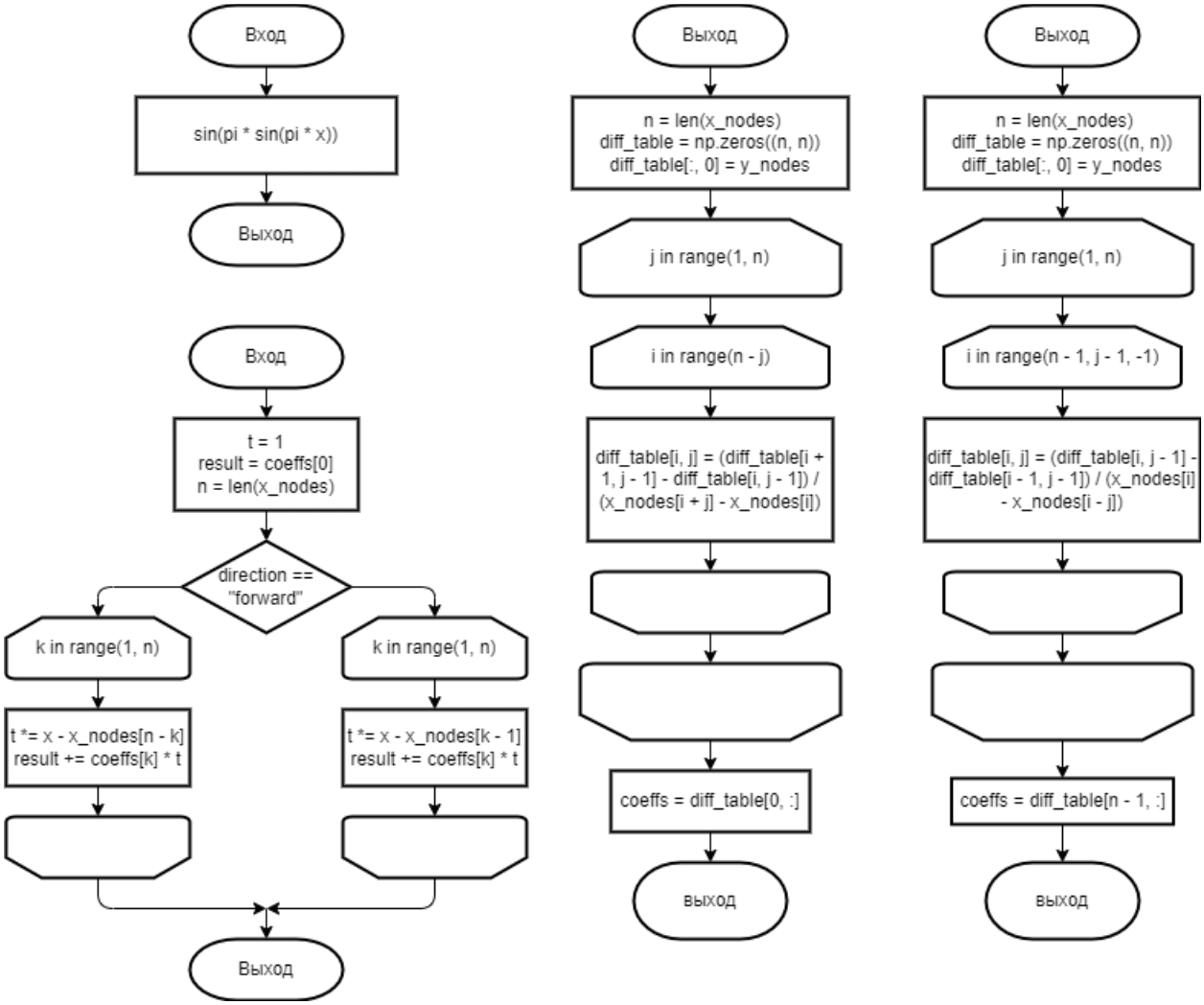
Для функции f(x), заданной своими значениями в точках x0,...,xn, построить интерполяционный многочлен Ньютона с использованием разделенных разностей. Для вариантов с четными номерами использовать формулу интерполирования вперед, для вариантов с нечетными номерами - фор мулу интерполирования назад.

Сравнить графики аппроксимируемой функции f(x) и интерполяционного многочлена внутри и за пределами отрезка интерполирования.

Построить график погрешности интерполяции. Качественно, с помощью графиков, оценить влияние на погрешность количества и расположения внутренних узлов интерполяции x1,...,xn–1.

0. $f(x) = \mathrm{Sin}(\pi\,\mathrm{Sin}(\pi\,x))$, $x_{\min}=0$, $x_{\max}=1$, $n=5$.

Схемы алгоритмов программы



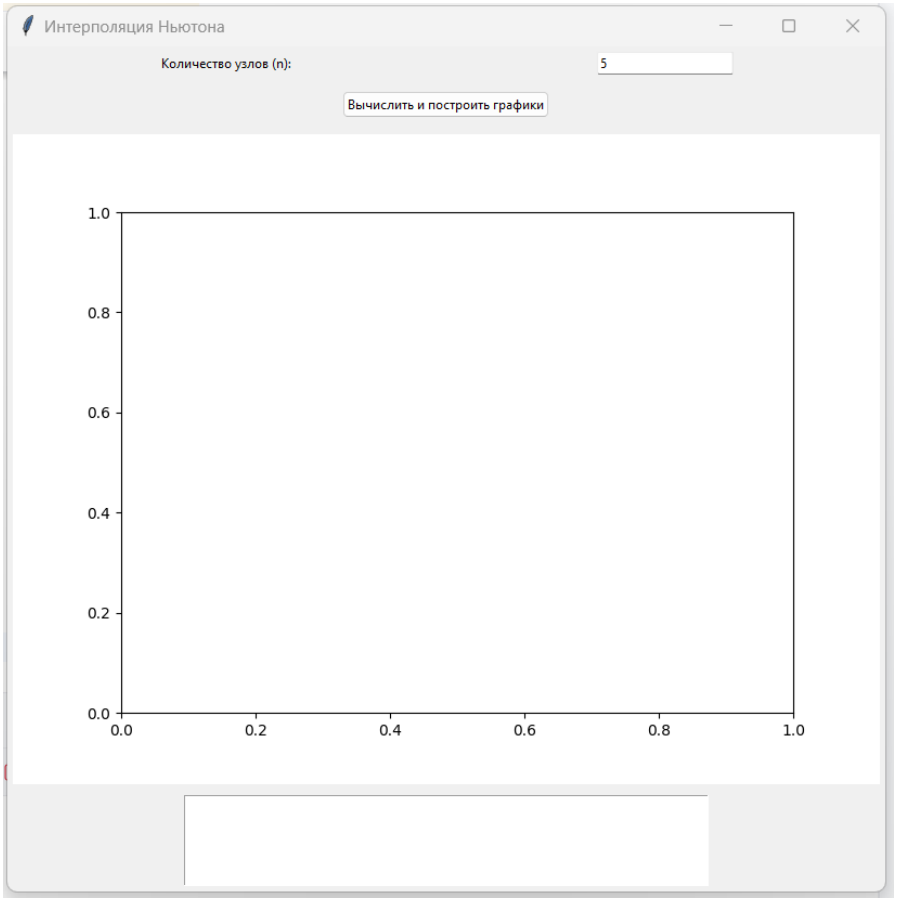
Листинг кода основных методов

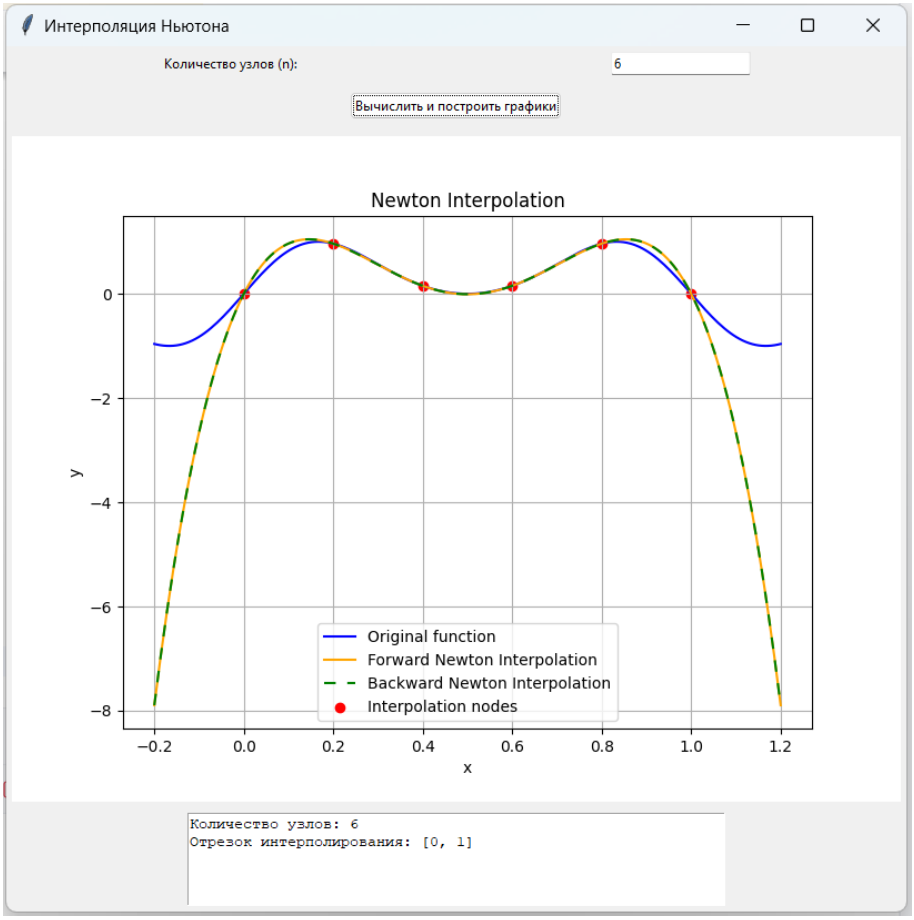
```
    def f(self, x):
        return np.sin(np.pi * np.sin(np.pi * x))

    def forward_newton(self, x_nodes, y_nodes):
        n = len(x_nodes)
        diff_table = np.zeros((n, n))

        diff_table[:, 0] = y_nodes
        for j in range(1, n):
            for i in range(n - j):
                diff_table[i, j] = (diff_table[i + 1, j - 1] - diff_table[i,
j - 1]) / (
                    x_nodes[i + j] - x_nodes[i]
                )

        coeffs = diff_table[0, :]

        def inner(x):
            t = 1
            result = coeffs[0]
```

```
            for k in range(1, n):
                t *= x - x_nodes[k - 1]
                result += coeffs[k] * t

            return result
        return inner


    def backward_newton(self, x_nodes, y_nodes):
        n = len(x_nodes)
        diff_table = np.zeros((n, n))

        diff_table[:, 0] = y_nodes
        for j in range(1, n):
            for i in range(n - 1, j - 1, -1):
                diff_table[i, j] = (diff_table[i, j - 1] - diff_table[i - 1,
j - 1]) / (
                    x_nodes[i] - x_nodes[i - j]
                )

        coeffs = diff_table[n - 1, :]

        def inner(x):
            t = 1
            result = coeffs[0]
            for k in range(1, n):
                t *= x - x_nodes[n - k]
                result += coeffs[k] * t

            return result

        return inner

    def calculate_and_plot(self):
        try:
            self.n = int(self.n_entry.get())
            if self.n <= 0:
                raise ValueError("Количество узлов должно быть положительным
числом.")

            x_nodes = np.linspace(self.a, self.b, self.n)
            y_nodes = self.f(x_nodes)

            forward = self.forward_newton(x_nodes, y_nodes)
            backward = self.backward_newton(x_nodes, y_nodes)


            xs = np.linspace(self.a - 0.2, self.b + 0.2, 200)  # Add more
points outside interval
            ys = self.f(xs)

            # Clear the previous plot
            self.ax.clear()

            self.ax.plot(xs, ys, "-", color=self.original_color,
label="Original function")
            self.ax.plot(xs, forward(xs), "-", color=self.forward_color,
label="Forward Newton Interpolation")
            self.ax.plot(xs, backward(xs), "--", color=self.backward_color,
dashes=(5, 5), label="Backward Newton Interpolation")
            self.ax.scatter(x_nodes, y_nodes, c=self.node_color,
label="Interpolation nodes")
            self.ax.legend()
```

```
            self.ax.set_title("Newton Interpolation")
            self.ax.set_xlabel("x")
            self.ax.set_ylabel("y")
            self.ax.grid(True)

            self.canvas.draw()

            # Display some results
            self.result_text.delete("1.0", tk.END)
            self.result_text.insert(tk.END, f"Количество узлов: {self.n}\n")
            self.result_text.insert(tk.END, f"Отрезок интерполирования:
[{self.a}, {self.b}]\n")


        except ValueError as e:
            tk.messagebox.showerror("Ошибка", str(e))


root = tk.Tk()
gui = NewtonInterpolatorGUI(root)
root.mainloop()
```

Графический интерфейс программы





Проверка правильности работы программы