Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №3

по дисциплине **«Функциональное программирование»**

Вариант: **ВЫЧМАТ**

Преподаватель:
Доморацкий Эридан Алексеевич

Выполнил: Беля Алексей
Группа: P3317

Санкт-Петербург, 2025

1. Задание

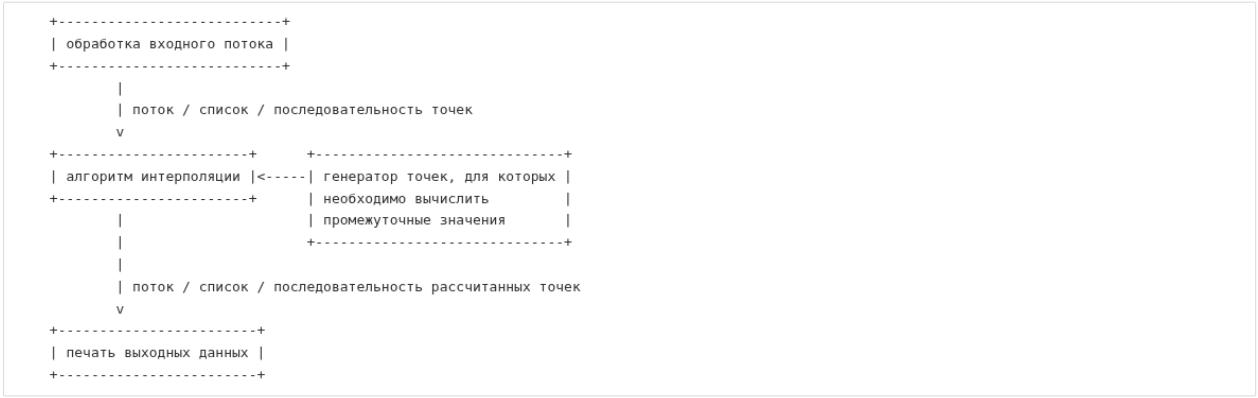
Лабораторная работа №3

Цель: получить навыки работы с вводом/выводом, потоковой обработкой данных, командной строкой.

В рамках лабораторной работы вам предлагается повторно реализовать лабораторную работу по предмету "Вычислительная математика" посвящённую интерполяции (в разные годы это лабораторная работа 3 или 4) со следующими дополнениями:

- обязательно должна быть реализована линейная интерполяция (отрезками, link);
- настройки алгоритма интерполяции и выводимых данных должны задаваться через аргументы командной строки:
  - какие алгоритмы использовать (в том числе два сразу);
  - частота дискретизации результирующих данных;
  - и т.п.;
- входные данные должны задаваться в текстовом формате на подобии ".csv" (к примеру `x;y\n` или `x\ty\n`) и подаваться на стандартный ввод, входные данные должны быть отсортированы по возрастанию x;
- выходные данные должны подаваться на стандартный вывод;
- программа должна работать в потоковом режиме (пример -- `cat | grep 11`), это значит, что при запуске программы она должна ожидать получения данных на стандартный ввод, и, по мере получения достаточного количества данных, должна выводить рассчитанные точки в стандартный вывод;

Приложение должно быть организовано следующим образом:



Потоковый режим для алгоритмов, работающих с группой точек должен работать следующим образом:

```
o o o o o o . . x x x
  x x x . . o . . x x x
    x x x . . o . . x x x
      x x x . . o . . x x x
        x x x . . o . . x x x
          x x x . . o . . x x x
            x x x . . o o o o o o EOF
```

где:

- каждая строка -- окно данных, на основании которых производится расчёт алгоритма;
- строки сменяются по мере поступления в систему новых данных (старые данные удаляются из окна, новые -- добавляются);
- `o` -- рассчитанные данные, можно видеть:
  - большинство окон используется для расчёта всего одной точки, так как именно в "центре" результат наиболее точен;
  - первое и последнее окно используются для расчёта большого количества точек, так лучших данных для расчёта у нас не будет.
- `.` -- точки, задействованные в рассчете значения `o`.
- `x` -- точки, расчёт которых для "окон" не требуется.

Пример вычислений (`my_lab3 --linear --step 0.7`, `<` -- ввод, `>` -- вывод):

```
< 0 0
< 1 1
> linear: 0 0
> linear: 0.7 0.7
< 2 2
> linear: 1.4 1.4
> 3 3
> linear: 2.1.4 2.1
> linear: 2.8 2.8
< EOF
> linear: 2.8 2.8
```

`my_lab3 --newton -n 4 --step 0.5` (интерполяция по 4 точкам):

```
< 0 0
< 1 1
< 2 2
> 3 3
> 4 4
> newton: 0 0
> newton: 0.5 0.5
> newton: 1 1
> newton: 1.5 1.5
> newton: 2 2
> newton: 2.5 2.5
> newton: 3 3
< 5 5
> newton: 3.5 3.5
> newton: 4 4
< 7 7
> newton: 4.5 4.5
> newton: 5 5
< 8 8
> newton: 5.5 5.5
> newton: 6 6
> newton: 6.5 6.5
> newton: 7 7
< EOF
> newton: 7.5
> newton: 8
```

Общие требования:

- программа должна быть реализована в функциональном стиле;
- ввод/вывод должен быть отделён от алгоритмов интерполяции;
- требуется использовать идиоматичный для технологии стиль программирования.

Содержание отчёта:

- титульный лист;
- требования к разработанному ПО, включая описание алгоритма;
- ключевые элементы реализации с минимальными комментариями;
- ввод/вывод программы;
- выводы (отзыв об использованных приёмах программирования).

2. Ключевые элементы реализации

Программа выполняет интерполяцию по потоку точек, поступающих на стандартный ввод, в функциональном стиле на OCaml. В основе лежит ленивый поток `Seq` и механизм скользящего окна фиксированного размера, что позволяет обрабатывать большие или бесконечные последовательности без предварительной загрузки всех данных в память.

1. Архитектура
- Модуль `Config` отвечает за парсинг аргументов командной строки (выбор методов интерполяции, шага по X, размера окна) и формирует иммутабельный рекорд конфигурации.
- Модуль `Io` реализует чтение точек из stdin в виде ленивой последовательности и вывод результатов в формате `method: x y`.
- Модуль `Stream_processor` содержит основную логику: скользящее окно, генерацию узлов интерполяции и применение алгоритмов ко всем выбранным методам.
- Алгоритмы интерполяции (линейный и Ньютона) вынесены в отдельный модуль `Interpolation_alg`.
- В `bin/main.ml` соединяются все части: чтение конфигурации, чтение потока, обработка и вывод с немедленным `flush stdout` для интерактивного режима.

2. Скользящее окно и потоковая обработка
- Входные данные читаются как `Seq.t`, поверх которого реализована функция `sliding_window size seq`. Она формирует ленивую последовательность окон: для каждого окна возвращается кортеж `(buffer, is_first, is_last)`.
- Окно двигается по потоку с шагом один элемент: при достижении нужного размера создаётся первое полное окно, затем при каждом новом элементе «съезжает» на один вправо.
- Конец потока обрабатывается аккуратно: если окно уже было полным, оно не дублируется, если данных мало — единственное окно помечается как первое и последнее одновременно.

3. Генерация X-координат и стратегии для методов
- Для каждого окна вычисляются границы `x_min` и `x_max`, а также генерируется список X-точек с заданным шагом `step` от общей базовой точки `base_x` (первая фактически прочитанная точка).
- Функция `get_interpolation_points` применяет стратегии для разных методов:
  - Для линейной интерполяции первые окна выводят все свои точки, а промежуточные — только X строго больше левой границы окна, чтобы не дублировать значения на стыках.
  - Для метода Ньютона первое окно выводит точки до последней точки окна включительно, последующие — только точки правее последней точки предыдущего окна, что устраняет дублирование интерполированных значений.

4. Кэширование базовой точки и минимальное использование мутабельности

- Чтобы поддержать ленивую обработку и интерактивный ввод, используется одна `ref` для кэширования `base_x` — X первой пришедшей точки. Это делается в момент первого прохода по потоку через `Seq.map`; далее `base_x` не меняется.
- Благодаря этому не требуется материализовывать весь поток (через `List.of_seq`), и программа начинает выдавать результат сразу после появления первых входных точек.

5. Запуск и качество кода

- Проект собирается через dune, есть отдельная директория `tests/` с тестами и примерами входных данных.
- В CI на GitHub Actions на каждом пуше выполняются сборка, тесты и проверка форматирования (ocamlformat), что помогает поддерживать единый стиль и корректность реализации.

Примеры ключевых фрагментов кода

1. Точка входа `bin/main.ml`

```
let () =
  (* Парсинг аргументов командной строки *)
  let config = Interpolation.Config.parse_args () in

  (* Читаем поток точек из stdin *)
  let input_seq = Interpolation.Io.read_points_lazy () in

  (* Обрабатываем поток с интерполяцией *)
  let output_seq = Interpolation.Stream_processor.process_stream
config input_seq in

  (* Выводим результаты построчно с немедленным flush *)
  Seq.iter (fun (method_type, point) ->
    Interpolation.Io.print_point (Interpolation.Config.method_name
method_type) point;
    flush stdout
  ) output_seq;

  (* Финальный flush на случай буферизации *)
  flush stdout
```

2. Скользящее окно `Stream_processor.sliding_window`

```
let sliding_window size seq =
  let rec aux buffer was_full seq_fun () =
    match seq_fun () with
    | Seq.Nil ->
        if was_full then
```

```
          Seq.Nil
        else if List.length buffer >= size then
          Seq.Cons ((buffer, false, true), fun () -> Seq.Nil)
        else if List.length buffer > 0 then
          Seq.Cons ((buffer, true, true), fun () -> Seq.Nil)
        else
          Seq.Nil
    | Seq.Cons (x, next) ->
        let new_buffer = buffer @ [x] in
        if List.length new_buffer < size then
          aux new_buffer false next ()
        else if List.length buffer < size then
          Seq.Cons ((new_buffer, true, false), aux new_buffer true
next)
        else
          let shifted_buffer = (List.tl buffer) @ [x] in
          Seq.Cons ((shifted_buffer, false, false), aux
shifted_buffer true next)
  in
  aux [] false seq
```

3. Кэширование базовой точки для ленивой обработки

```
let process_stream config input_seq =
  let methods = config.Config.methods in
  let step = config.Config.step in
  let window_size = config.Config.window_size in

  let base_x_ref = ref None in
  let cached_seq = Seq.map (fun point ->
    (match !base_x_ref with
    | None -> base_x_ref := Some point.Io.x
    | Some _ -> ());
    point
  ) input_seq in

  let windows = sliding_window window_size cached_seq in
  (* ... дальнейшая обработка окон ... *)
```

3. Результаты работы тестов

=== Тест 1: Линейная интерполяция с шагом 0.7 ===

Entering directory '/home/aeshabb/ITMO/fp/lab3'

Leaving directory '/home/aeshabb/ITMO/fp/lab3'

linear: 0 0

linear: 0.7 0.7

linear: 1.4 1.4

linear: 2.1 2.1

linear: 2.8 2.8

=== Тест 2: Линейная интерполяция с шагом 1.0 ===

Entering directory '/home/aeshabb/ITMO/fp/lab3'

Leaving directory '/home/aeshabb/ITMO/fp/lab3'

linear: 0 0

linear: 1 1

linear: 2 2

=== Тест 3: Интерполяция Ньютона (окно 4) ===

cat: tests/examples/newton_test.txt: Нет такого файла или каталога

Entering directory '/home/aeshabb/ITMO/fp/lab3'

Leaving directory '/home/aeshabb/ITMO/fp/lab3'

=== Тест 4: Квадратичная функция линейной интерполяцией ===

cat: tests/examples/complex_test.txt: Нет такого файла или каталога

Entering directory '/home/aeshabb/ITMO/fp/lab3'

Leaving directory '/home/aeshabb/ITMO/fp/lab3'

=== Тест 5: CSV формат с точкой с запятой ===

Entering directory '/home/aeshabb/ITMO/fp/lab3'

Leaving directory '/home/aeshabb/ITMO/fp/lab3'

newton: 0 0

newton: 0.5 0.25

newton: 1 1

newton: 1.5 2.25

newton: 2 4

newton: 2.5 6.25

newton: 3 9

✓ Все тесты завершены!

4. Листинг программы

https://github.com/aeshabb/lab3_fp

5. Вывод

В ходе работы была реализована и доведена до промышленного состояния потоковая система интерполяции в функциональном стиле на OCaml: вход обрабатывается лениво через Seq, вокруг него построен аккуратно реализованный механизм скользящего окна без дублирования точек на стыках, реализованы две стратегии интерполяции (линейная и Ньютона) с продуманной выдачей x-значений, проведена максимальная демутабельзация кода (за исключением строго необходимой ссылки для ленивого кэширования начальной точки), организованы тесты и демонстрационные скрипты, а также настроен CI с автоматической сборкой, тестированием и проверкой форматирования — в результате получилась удобная, расширяемая и хорошо документированная реализация, готовая для дальнейшего использования и развития.