Федеральное государственное бюджетное образовательное
учреждение высшего образования
Московский государственный университет
имени М. В. Ломоносова

Механико-математический факультет

Кафедра вычислительной математики

Курсовая работа

Тема: *Задача о наибольшей общей подпоследовательности.*

Выполнил:
студент 3 курса 332 группы
Шерстобитов Андрей Сергеевич

Научный руководитель:
Валединский Владимир Дмитриевич

Москва
2021

Содержание

1 Задача с целыми числами

1.1 Постановка задачи

Для данных последовательностей найти самую длинную общую подпоследовательность, при этом подпоследовательность может быть разрывна в данной последовательности.

Пример: Пусть даны две последовательности:

$$S1 = \{B, C, D, A, A, C, D\}$$
$$S2 = \{A, C, D, B, A, C\}$$

Тогда их общие подпоследовательности это

$$\{B, C\},\ \{C, D, A, C\},\ \{D, A, C\},\ \{A, A, C\},\ \{A, C\},\ \{C, D\},\ ...$$

Среди этих подпоследовательностей мы найдем самую длинную, а именно $\{C, D, A, C\}$

1.2 Введение в алгоритм

Прежде чем переходить непосредственно к алгоритму, рассмотрим несколько понятий:

Определение 1. *Назовем Ω множество элементов, из которых мы будем составлять наши последовательности. В данном разделе легче представлять это множество, как множество букв английского алфавита, а последовательности - соответственно - строки.*

Определение 2. *Знаком & будем обозначать стандартную конкатенацию последовательностей. Например: $\{a, b, c\}\ \&\ \{c, b, a\}\ =\ \{a, b, c, c, b, a\}$.*

Определение 3. *Префиксом последовательности $S = \{x_1, \ldots, x_n\}$ длины $0 < k \leq n$ обозначим $S_k = \{x_1, \ldots, x_k\}$*

Определение 4. *Назовем $\mathrm{LCS}(S1, S2)$ функцию, которая принимает на вход две последовательности $S1 = \{x_1, \ldots, x_n\}$ и $S2 = \{y_1, \ldots, y_n\}$ и возвращает наибольшую общую подпоследовательность C.*

Докажем вспомогательную лемму:

Лемма 1. *Функция $\mathrm{LCS}(S1, S2)$ обладает следующими свойствами:*

1. $\forall\ A \in \Omega$
$$\mathrm{LCS}(S1\ \&\ A,\ S2\ \&\ A) = \mathrm{LCS}(S1,\ S2)\ \&\ A$$

2. $\forall A \neq B \in \Omega$
$$\mathrm{LCS}(S1\ \&\ A, S2\ \&\ B) = \max\{\mathrm{LCS}(S1\ \&\ A, S2), \mathrm{LCS}(S1, S2\ \&\ B)\}$$

Доказательство. 1. Представим нахождение общей последовательности как последовательное сопоставление элементов последовательностей друг за другом без пересечений.

Рассмотрим $S1 = \{a, b, c, d, e, f\}$, $S2 = \{d, a, c, e, q\}$ и уже найденную для них общую подпоследовательность $C = \{a, c, e\}$, тогда

$$\begin{array}{cccccc} a & b & c & d & e & f \\ d & a & c & e & q & \end{array}$$

Добавив в конец каждой последовательности по одному элементу (пусть p), мы не нарушим наш алгоритм, так как новых пересечений быть не может.

$$\begin{array}{ccccccc} \mathrm{a} & \mathrm{b} & \mathrm{c} & \mathrm{d} & \mathrm{e} & \mathrm{f} & \mathrm{p} \\ \mathrm{d} & \mathrm{a} & \mathrm{c} & \mathrm{e} & \mathrm{q} & \mathrm{p} & \end{array}$$

2. В противном случае, если последние элементы последовательностей $S1 = \{x_1, \ldots, x_n\}$, $S2 = \{y_1, \ldots, y_m\}$ различны ($x_n \neq y_m$), то какой-либо из них не входит в наибольшую общую подпоследовательность, иначе могут появиться самопересечения. Значит нужно рассмотреть последовательности $S1 = \{x_1, \ldots, x_n\}$, $S2' = \{y_1, \ldots, y_{m-1}\}$ и $S1' = \{x_1, \ldots, x_{n-1}\}$, $S2 = \{y_1, \ldots, y_m\}$ и среди них выбрать максимальную общую подпоследовательность.

□

Таким образом для последовательностей $X = (x_1, \ldots, x_n), Y = (y_1, \ldots, y_m)$, их префиксов $X_{1,2,\ldots,n}, Y_{1,2,\ldots,m}$ наша функция $\mathrm{LCS}(X_i, Y_j)$ принимает вид:

$$\mathrm{LCS}(X_i, Y_j) = \begin{cases} \emptyset & \text{если } i = 0 \text{ или } j = 0 \\ \mathrm{LCS}(X_{i-1}, Y_{j-1}) \& x_i & \text{если } i, j > 0 \text{ и } x_i = y_j \\ \max\{\mathrm{LCS}(X_i, Y_{j-1}), \mathrm{LCS}(X_{i-1}, Y_j)\} & \text{если } i, j > 0 \text{ и } x_i \neq y_j \end{cases}$$

1.3 Алгоритм

Рассмотрим алгоритм для последовательностей $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_m)$:

1. Создаем таблицу размера $(n+1) \times (m+1)$, где первая строка и первый столбец заполнены нулями. Это нужно для случая, когда одна из последовательностей пустая.

2. Заполняем эту таблицу следующим образом:

   (a) Если элемент на текущей строке и текущей колонке совпадают, то по свойству 1 заполняем ячейку числом на 1 больше, чем в левой верхней диагональной ячейке. Проводим стрелку к этой же ячейке от текущей.

   (b) Иначе по свойству 2 берем максимальное значение с предыдущей колонки и предыдущей строки и заполняем текущую ячейку. Проводим стрелку к максимальному элементу. Если они равны, то к любому из них.

3. Повторяем предыдущий шаг до заполнения таблицы.

4. Получаем в правой нижней ячейке таблицы длину максимальной общей подпоследовательности.

5. Чтобы распечатать искомую подпоследовательность, начинаем с правого нижнего элемента и двигаемся по стрелкам. Добравшись до первой строки или столбца получим нашу последовательность.

1.4 Пример работы алгоритма

Рассмотрим две последовательности $X = \{C, B, D, A\}, Y = \{A, C, A, D, B\}$ и выполним для них алгоритм.

1. Составляем требуемую таблицу.

	$\emptyset$	C	B	D	A
$\emptyset$	0	0	0	0	0
A	0				
C	0				
A	0				
D	0				
B	0				

2. Заполним первую строку соответственно свойствам нашей функции LCS(S1, S2)

	$\emptyset$	C	B	D	A
$\emptyset$	0	0	0	0	0
A	0	0	0	0	1
C	0				
A	0				
D	0				
B	0				

3. Повторим для последующих строк.

	$\emptyset$	C	B	D	A
$\emptyset$	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

4. Значение в последней ячейке - длина наибольшей общей последовательности

	$\emptyset$	C	B	D	A
$\emptyset$	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

5. Находим общие подпоследовательности, двигаясь по стрелочкам.

	$\emptyset$	C	B	D	A
$\emptyset$	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

Таким образом, получившиеся наибольшие общие подпоследовательности:

$$\{C, B\}, \{C, A\}, \{C, D\}$$

Замечание. *Данный пример показал, что наибольшая общая подпоследовательность не единственна.*

1.5 Реализация алгоритма на C++

```cpp
using Sequence = std::vector<int>;

class IntegralTable {
 public:
    IntegralTable(const Sequence& S1,
                  const Sequence& S2)
        : _rows(S1.size() + 1), _columns(S2.size() + 1)
        , _table(std::make_unique<uint32_t[]>(_rows * _columns)) {
        size_t i, j;
        for (i = 0u; i <= _rows - 1; ++i) {
            for (j = 0u; j <= _columns - 1; ++j) {
                if (i * j == 0u)
                    at(i, j) = 0u;
                else if (S1[i - 1] == S2[j - 1])
                    at(i, j) = at(i - 1, j - 1) + 1;
                else
                    at(i, j) = std::max(at(i - 1, j), at(i, j - 1));
            }
        }
    }

    uint32_t& at(size_t i, size_t j) {
        return _table.get()[i * _columns + j];
    }

 private:
    size_t _rows;
    size_t _columns;
    std::unique_ptr<uint32_t[]> _table;
};

Sequence find(const Sequence& S1, const Sequence& S2) {
    const IntegralTable table(S1, S2);
    size_t index = table.at(S1.size(), S2.size());
    Sequence lcs(index);

    size_t i = S1.size(), j = S2.size();
    while (i > 0 && j > 0) {
        if (S1[i - 1] == S2[j - 1]) {
            lcs[index - 1] = S1[i - 1];
            i--, j--, index--;
        } else if (table.at(i - 1, j) > table.at(i, j - 1)) {
            i--;
        } else {
            j--;
        }
    }

    return lcs;
}
```

Сложность работы: $\mathcal{O}(mn)$, где m и n - длины последовательностей.
Сложность по памяти: $\mathcal{O}(mn)$, где m и n - длины последовательностей.

1.6 Результаты

Сборка программы проводится с помощью gcc. Флаги компиляции: -O3 -Werror - Wextra -Wall

1. Две последовательности: $X = \{'A',' C',' B'\},\ Y = \{'C',' B',' E'\}$
   Результирующая таблица:

	$\emptyset$	A	C	B
$\emptyset$	0	0	0	0
C	0	0	0	1
B	0	1	1	1
E	0	1	2	2

   Получившийся результат $\{'C',' B'\}$
   Затраченное время: 1us.

2. $X = \underbrace{\{'a', \ldots,' a'\}}_{2048}\ Y = \underbrace{\{'a', \ldots,' a'\}}_{2048}$
   Результат: $\underbrace{\{'a', \ldots,' a'\}}_{2048}$
   Затраченное время: 17662us (17ms)

3. $X = \underbrace{\{'b', \ldots, \underbrace{'a', \ldots,' a'}_{48}, \ldots,' b'\}}_{2048}\ Y = \underbrace{\{'a', \ldots,' a'\}}_{2048}$
   Результат: $\underbrace{\{'a', \ldots,' a'\}}_{48}$
   Затраченное время: 21400us (21ms)

4. $X = \underbrace{\{'a',' b',' a',' b', \ldots,' b'\}}_{2048}\ Y = \underbrace{\{'a', \ldots,' a'\}}_{2048}$
   Результат: $\underbrace{\{'a', \ldots,' a'\}}_{1024}$
   Затраченное время: 20886us (20ms)

5. $X = \underbrace{\{'a',' b',' a',' b', \ldots,' b'\}}_{2048}\ Y = \underbrace{\{'a', \ldots,' a'\}}_{2048}$
   Результат: $\underbrace{\{'a', \ldots,' a'\}}_{1024}$
   Затраченное время: 21584us (21ms)

2 Задача с действительными числами

Задачу нахождения общей подпоследовательности для двух последовательностей $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$ с плавающей точкой можно поставить по-разному.

Самый наивный вариант это получить от пользователя некую погрешность ε, с помощью которой мы должны будем определить является ли каждый i-ый элемент членом общей подпоследовательности или нет.

Пытаясь решить эту задачу аналогично задаче с целыми числами мы столкнемся с несколькими трудностями:

1. Нужно определиться как работать с данной погрешностью, ведь просто напросто сравнивать разницу i-ого и j-ого элементов последоватностей с ε и строить таблицу аналогично первой задаче не получится.

   Например: X и Y такие, что $\|X\|_2 < \varepsilon$ и $\|Y\|_2 < \varepsilon$. Что это значит для нас в таком случае? Это значит, что эти две последовательности будут совпадать, так как $\forall i, j\ |X_i - Y_j| < \varepsilon$, что не будет правильным ответом.

2. Пусть мы придумали какой-то функционал $\phi(X_i, Y_j)$, который позволит нам как-то правильно заполнить таблицу. Возникает следующая проблема. Как с помощью это функционала понимать какой элемент является очередным элементом искомой подпоследовательности?

Посмотрим на нашу задачу под другим углом:

Давайте предположим, что одна из последовательностей уже является наибольшей общей подпоследовательностью. Остается придумать как понять верна ли наша гипотеза. Попытаемся формализовать такую постановку.

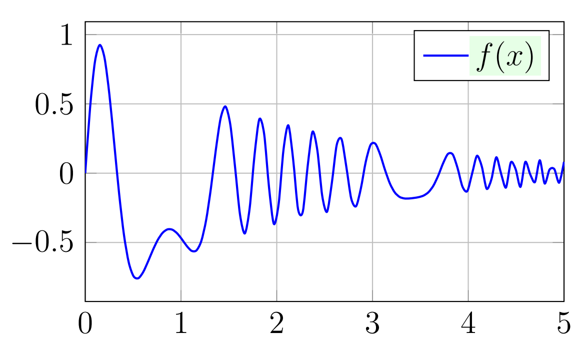
2.1 Постановка задачи

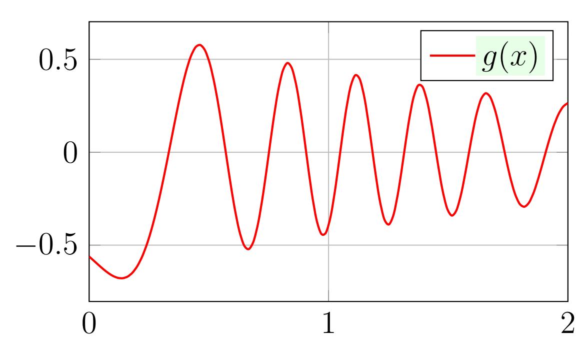
Для двух заданных последовательностей понять, можно ли сопоставить одну последовательность с подмножеством значений другой последовательности? Дать какое-либо определение сопоставлению и показать качество получившегося сопоставления.

Пояснительный примеры:

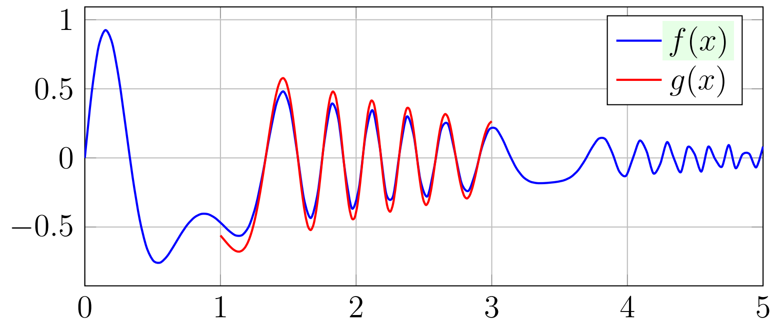
1. Интерпретируем задачу через непрерывные функции:
   Рассмотрим две непрерывные функции одной переменной $f(x)$ на отрезке $[a, b]$ и $g(x)$ на отрезке $[c, d]$.
   Вопрос: Можно ли утверждать, что значения $g(x)$ на отрезке $[c, d]$ являются подмножеством значений функции $f(x)$ на отрезке $I \subset [a, b]$?

   Возьмем две функции: $f(x)$ на отрезке $[0, 5]$ и $g(x)$ на отрезке $[0, 2]$.





   Тогда наилучшим соответствим между f и g будет:



Как качество нашего сопоставления можно выбрать число:

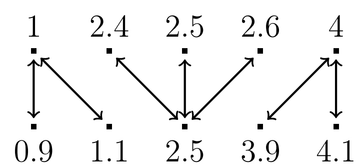
$$\int_1^3 (f(x) - g(k(x)))^2 dx,$$

где $k(x)$ - линейная функция от x, соответсвующая сдвигу.

2. (a) Даны две последовательности

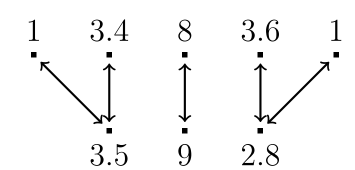
$$S1 = \{1, 2.4, 2.5, 2.6, 4\},\ S2 = \{0.9, 1.1, 2.5, 3.9, 4.1\}$$

Тогда их наилучшим сопоставлением будет отображение:



(b)

$$S1 = \{1,\ 3.4,\ 8,\ 3.6,\ 1\},\ S2 = \{3.5,\ 9,\ 2.8\}$$



2.2 Введение в алгоритм

Мы будем искать наилучшее сопоставление используя динамическое программирование аналогично и первой задаче с целыми числами.

В качестве функционала качества рассмотрим функцию

$$M(X_i, Y_j) = \frac{1}{k} \sum_{i=1}^{k} (x_i - y_{j(i)})^2$$

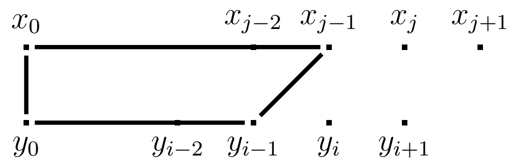
где X_i, Y_j - префиксы соответственно последовательностей X, Y, а k - количество сопоставлений в нашем алгоритме.

Вся задача теперь сводится к тому, чтобы отобразить одну последовательность в другую так, чтобы функционал был минимальным.

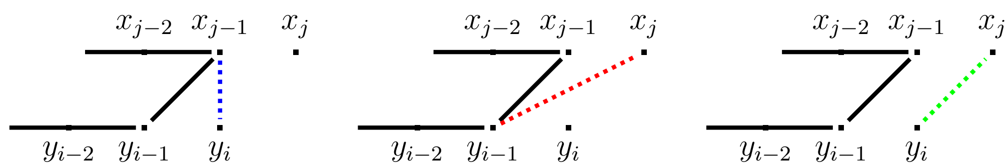
$$M \xrightarrow[\text{задача}]{} \min$$

Рассмотрим поподробнее как происходит заполнение таблицы и соответственно сопоставление i-го элемента первой последовательности с j-ым элементом второй.

Представим, что для $(i-1)$-го и $(j-1)$-го элементов мы уже проделали алгоритм и имеем некоторые сопоставления.



Тогда для j-го и i-го элемента могут быть следующие представления.



Как это выглядит на нашей таблице? Построим таблицу и посмотрим подробнее

	$\dots$	x_{j-1}	x_j	x_{j+1}
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
y_{i-1}	$\dots$	$M_{j-1,i-1}$	$M_{j,i-1}$	
y_i	$\dots$	$M_{j-1,i}$	$M_{j,i}$	
y_{i+1}				

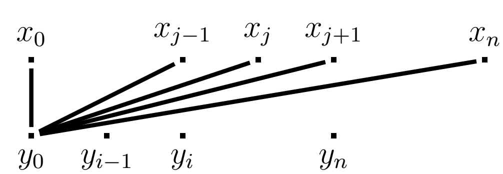
Среди этих трех вариантов мы будем выбирать тот, который даст меньшее вложение в нашу сумму итогового функционала.

Замечание: Наименьшему функционалу могут соответствовать разные сопоставления.

1. Следующие значения в таблице описывают такое соответствие:

	x_0	$\dots$	$\dots$	x_n
y_0	$M_{0,0}$	$\dots$	$\dots$	$M_{n,0}$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

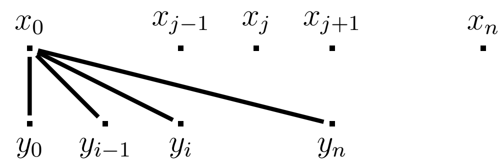
$\Leftrightarrow$



2. Аналогично столбцу:

	x_0	$\dots$
y_0	$M_{0,0}$	$\dots$
$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$
y_n	$\dots$	$\dots$

$\Leftrightarrow$



2.3 Алгоритм

Рассмотрим алгоритм для последовательностей $X = (x_1, \dots, x_n)$, $Y = (y_1, \dots, y_m)$.

1. Создаем таблицу размера $n \times m$.
2. Заполняем таблицу значениями функционала, вычисляя каждое значение ячейки аналогично замечанию. То есть в ячейку (i, j) кладем значение $m_{ij} = (x_i - y_j)^2$.
3. Далее обходим таблицу следующим образом:
   (a) Запоминаем позиции i, j текущей ячейки.
   (b) Для текущей ячейки (i, j) рассматриваем три следующие ячейки $(i + 1, j + 1)$, $(i + 1, j)$ и $(i, j + 1)$. Выбираем среди них ту, чье значение $m_{i'j'}$ дает минимальный вклад в наш функционал, то есть такую ячейку, что $M(X_i, Y_j) + m_{i'j'} \to \min$. Затем переходим на выбранную ячейку.
   (c) Если $i == n$, переходим на следующий столбик, аналогично при $j == m$ переходим на следующую строчку.
   (d) Если $i == n$ и $j == m$ добавляем элемент.
4. Повторяем предыдущий шаг до полного обхода таблицы.
5. Получаем поэлементное отражение элементов одной последовательности в другую.

Чтобы получить итоговое значение $M(X_n, Y_m)$ во время обхода суммириуем значения текущей ячейки в конечную переменную и считаем количество обойденных ячеек, а затем делим первое на второе.

Чтобы получить график функционала можем также запоминать не только позиции текущих ячеек, но и их значения.

2.4 Пример работы алгоритма

Рассмотрим работу алгоритма на следующих последовательностях:

$$S1 = \{1, 2.4, 2.5, 2.6, 4\},\ S2 = \{0.9, 1.1, 2.5, 3.9, 4.1\}$$

1. Создаем требуемую таблицу.

	1	2.4	2.5	2.6	4
0.9					
1.1					
2.5					
3.9					
4.1					

2. Заполняем таблицу, используя m

	1	2.4	2.5	2.6	4
0.9	0.01	2.25	2.56	2.89	9.61
1.1	0.01	1.69	1.96	2.25	8.41
2.5	2.25	0.01	0	0.01	2.25
3.9	8.41	2.25	1.96	1.69	0.01
4.1	9.61	2.89	2.56	2.25	0.01

3. Запоминаем позицию первой ячейки

	1	2.4	2.5	2.6	4
0.9	0.01	2.25	2.56	2.89	9.61
1.1	0.01	1.69	1.96	2.25	8.41
2.5	2.25	0.01	0	0.01	2.25
3.9	8.41	2.25	1.96	1.69	0.01
4.1	9.61	2.89	2.56	2.25	0.01

4. Смотрим на следующие ячейки и выбираем подходящую

	1	2.4	2.5	2.6	4
0.9	0.01	2.25	2.56	2.89	9.61
1.1	0.01	1.69	1.96	2.25	8.41
2.5	2.25	0.01	0	0.01	2.25
3.9	8.41	2.25	1.96	1.69	0.01
4.1	9.61	2.89	2.56	2.25	0.01

5. Обходим всю таблицу и получаем отображение

	1	2.4	2.5	2.6	4
0.9	0.01	2.25	2.56	2.89	9.61
1.1	0.01	1.69	1.96	2.25	8.41
2.5	2.25	0.01	0	0.01	2.25
3.9	8.41	2.25	1.96	1.69	0.01
4.1	9.61	2.89	2.56	2.25	0.01

Итоговое отображение

$$\{\{0,0\},\{1,0\},\{2,1\},\{2,2\},\{2,3\},\{3,4\},\{4,4\}\}$$

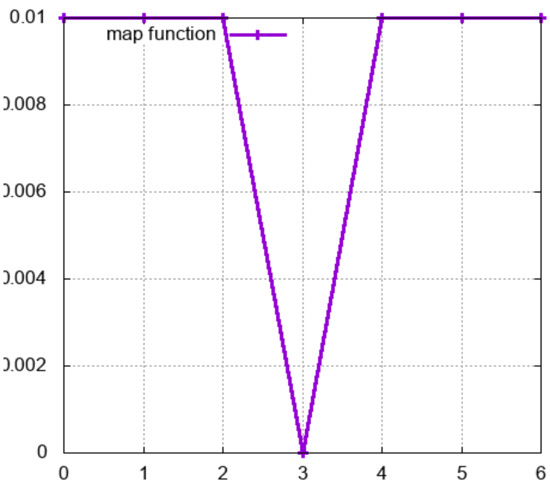
2.5 Реализация алгоритма на C++

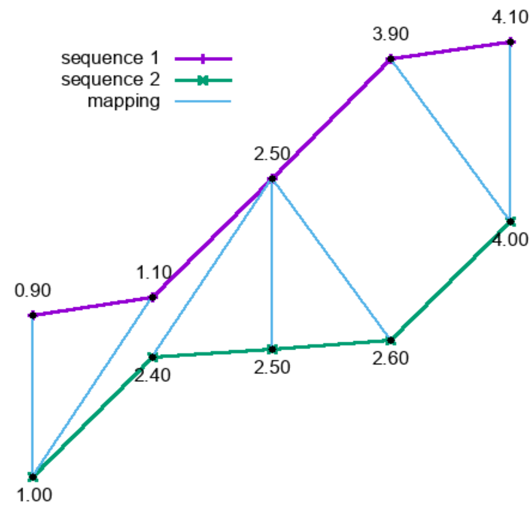
```
using Sequence = std::vector<double>;

class FloatingTable {
 public:
    FloatingTable(const Sequence& S1,
                  const Sequence& S2)
        : _rows(S1.size()), _columns(S2.size())
        , _table(std::make_unique<uint32_t[]>(_rows * _columns)) {
        for (i = 0u; i < _rows; ++i) {
            for (j = 0u; j < _columns; ++j) {
                at(i, j) = std::pow(S1[i] - S2[j], 2);
            }
        }
    }

    size_t nrows() const { return _rows; }
    size_t ncols() const { return _columns; }

    double& at(size_t i, size_t j) {
        return _table.get()[i * _columns + j];
    }

 private:
    size_t _rows;
    size_t _columns;
    std::unique_ptr<uint32_t[]> _table;
};

using mapping_t = std::vector<std::pair<size_t, size_t>>;

mapping_t map(const FloatingTable& table) {
    mapping_t result; std::array<double, 3> next;
    for (size_t i=0, j=0; i<table.nrows() && j<table.ncols();) {
        result.emplace_back(i, j);
        if (i+1 == table.nrows()) {
            j++; continue;
        }
        if (j+1 == table.ncols()) {
            i++; continue;
        }
        next = {table.at(i+1,j+1), table.at(i+1,j), table.at(i,j+1)};
        auto min_pos = std::min_element(next.begin(), next.end());
        if (min_pos == next.begin())
            i++, j++;
        else if (min_pos == std::next(next.begin()))
            i++;
        else
            j++;
    }
    return mapping_t;
}

```

Сложность работы: $\mathcal{O}(mn)$, где m и n - длины последовательностей.
Сложность по памяти: $\mathcal{O}(mn)$, где m и n - длины последовательностей.

2.6 Результаты

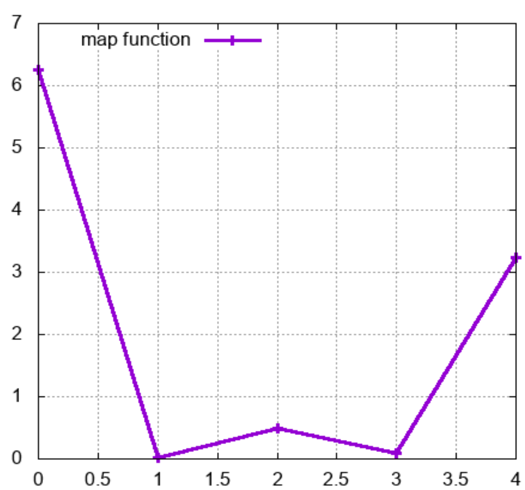
1. $X = \{1,\ 2.4,\ 2.5,\ 2.6,\ 4\}$, $Y = \{0.9,\ 1.1,\ 2.5,\ 3.9,\ 4.1\}$

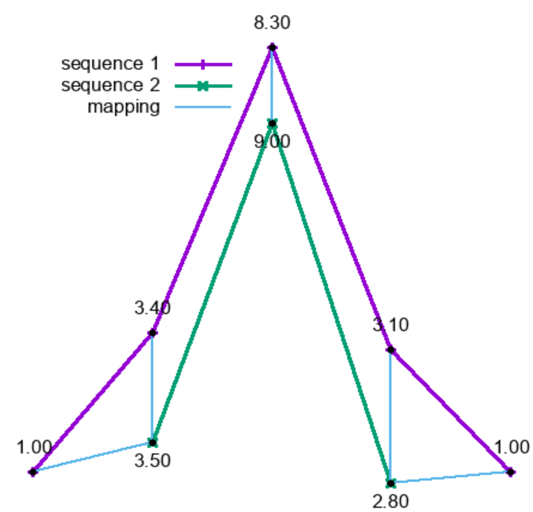




$$M(X, Y) = 0.00857143$$

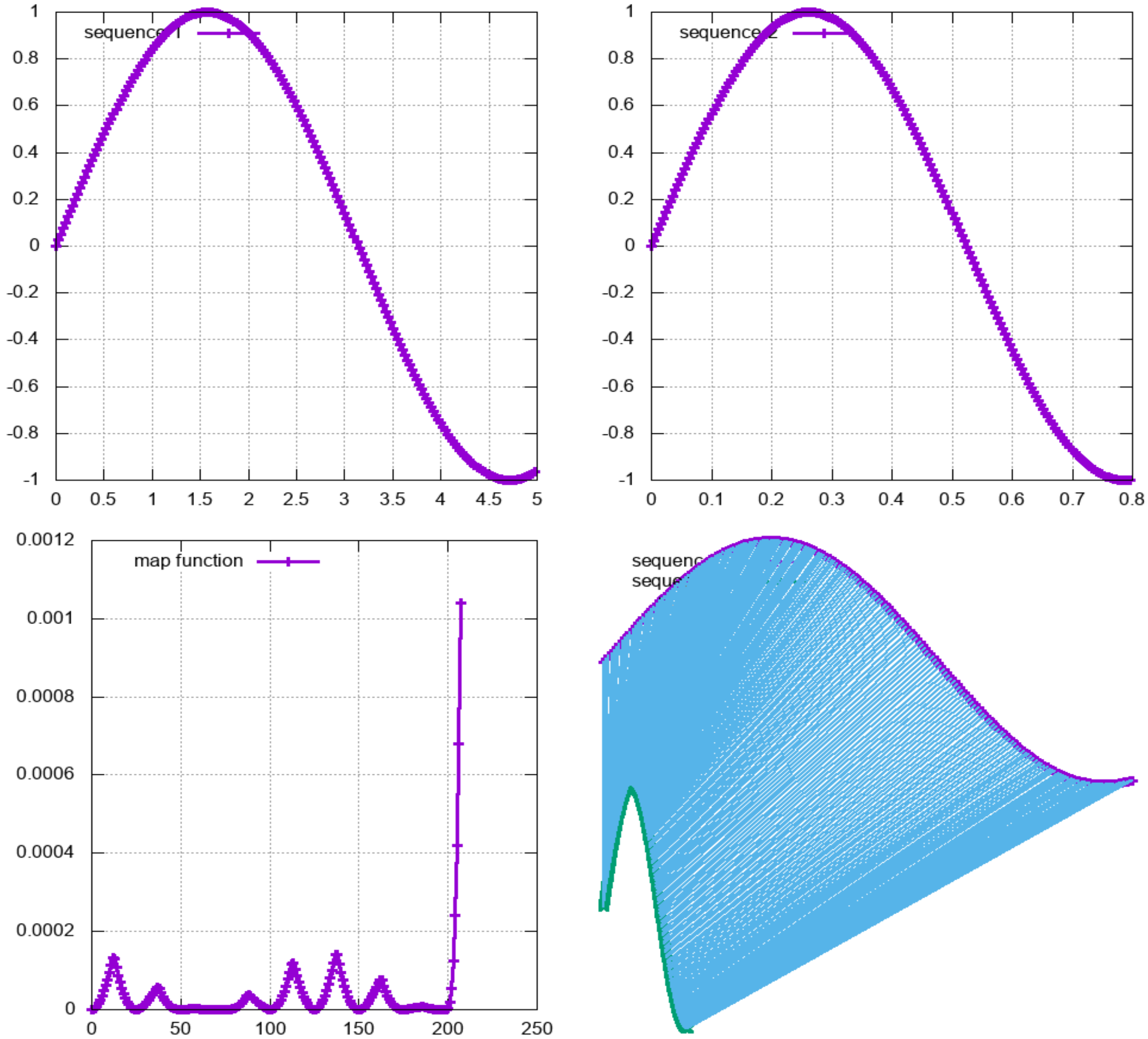
2. $X = \{1,\ 3.4,\ 8,\ 3.6,\ 1\}$, $Y = \{3.5,\ 9,\ 2.8\}$





$$M(X, Y) = 2.016$$

3. $X = \sin(x)$ на отрезке $[0,\ 5]$, $Y = \sin(6x)$ на отрезке $[0,\ 1]$



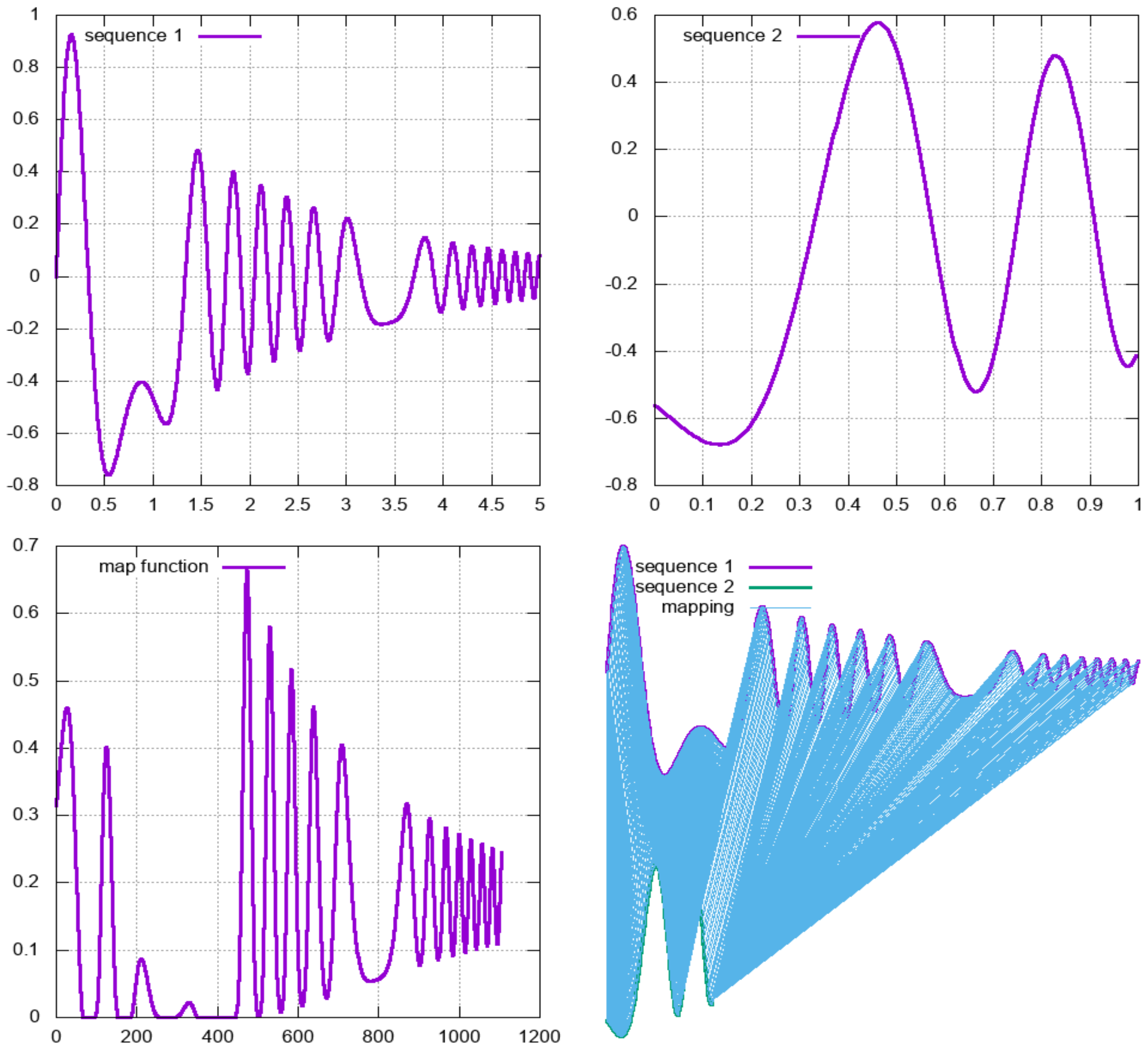
$$M(X, Y) = 3.69459e - 05$$

Заметим резкое возрастание функционала в конце. Это связано с тем, что после того как нашлось идеальное сопоставление n точек первой последовательности в $k < n$ точек второй последовательности все остальные $n - k$ точек первой последовательности начали сопоставляться с последней точкой второй. Рассмотрим эту проблему далее в разделе Проблемы.

4.

$$X = \exp(-\frac{x}{2}) * (\sin(10 * x * \cos(x))) \text{ на отрезке } [0, 5]$$

$$Y = 1.2 \exp(-\frac{x+1}{2}) * (\sin(10 * (x+1) * \cos(x+1))) \text{ на отрезке } [0, 1]$$



$$M(X, Y) = 0.145174$$

На данном примере наш алгоритм показал не совсем ожидаемый результат. В самом начале образовался “веер“. То есть большая часть первой последовательности отобразилась в одну точку второй. Иначе говоря $y_j \mapsto x_i, \ldots, x_{i+k}$. Почему так произошло? Посмотрим подробнее в следующем разделе.

2.7 Проблемы

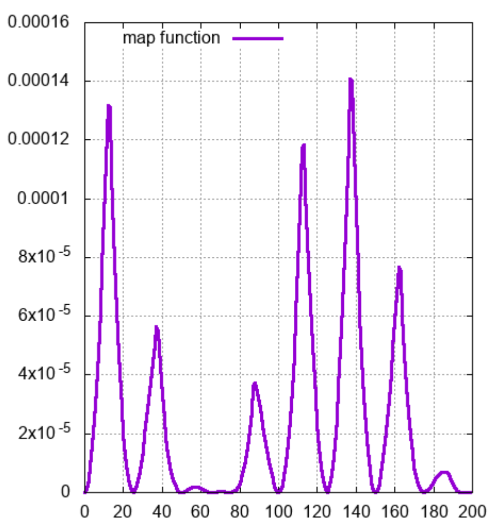
2.7.1 Разные длины последовательностей

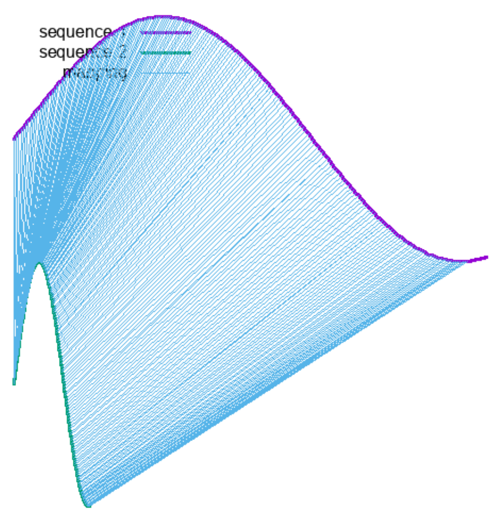
В третьем примере мы столкнулись с тем, что если в одной из последовательностей, допустим $X = (x_1, \ldots, x_n)$, количество элементов больше, чем в другой $Y = (y_1, \ldots, y_m)$, то есть $n > m$, то все $n - m$ элементов первой последовательности будут отражаться в последний элемент второй последовательности.

Возможное решение: Довольно простое. Вместо того, чтобы идти по краю таблицы в правый нижний угол, будем обрывать алгоритм как только достигнем конца одной из последовательности.

Изменение в реализации:

```
mapping_t map(const FloatingTable& table) {
    mapping_t result; std::array<double, 3> next;
    for (size_t i=0, j=0; i<table.nrows() && j<table.ncols();) {
        result.emplace_back(i, j);
        if (i+1 == table.nrows()) break;
        if (j+1 == table.ncols()) break;
        next = {
            table.at(i+1,j+1), table.at(i+1,j), table.at(i,j+1)
        };
        auto min_pos = std::min_element(next.begin(), next.end());
        if (min_pos == next.begin())
            i++, j++;
        else if (min_pos == std::next(next.begin()))
            i++;
        else
            j++;
    }
    return mapping_t;
}

```

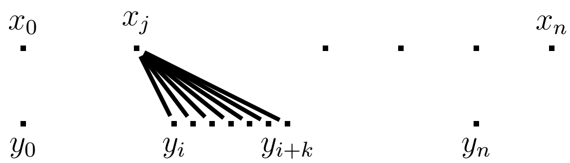




$$M(X, Y) = 2.5505e - 05$$

2.7.2 Множественные сопоставления одной точке

Посмотрим чему соответствует “веер“, опираясь на замечание из Введения в алгоритм:



$\Leftrightarrow$

	$\dots$	x_j	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$
y_i	$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$
y_{i+k}	$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$

Вывод: алгоритм жадно находит подходящее значение в таблице, которое дает наименьший вклад в функционал и не пытается смотреть на другие, возможно более подходящие значения.

Возможное решение: Введем штрафы для нашего алгоритма таким образом:

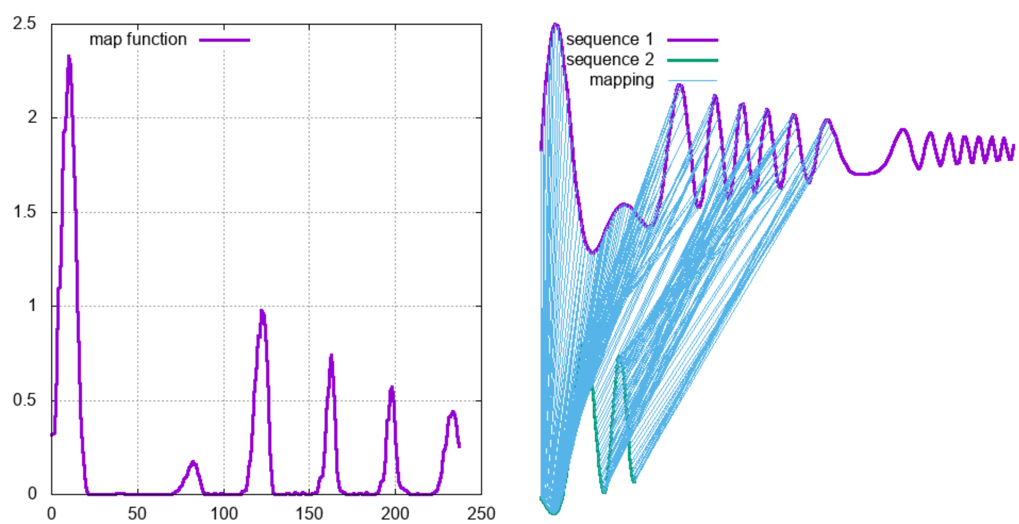
1. Штраф за передвижение по столбцу α и строке β равен 1.

2. При выборе позиции перехода рассматриваем следующие значения: $\alpha * c_{i,j+1}$, $c_{j+1,i+1}$, $\beta * c_{i+1,j}$, где $c_{i',j'}$ - значение в ячейке (i', j').

3. Если подходящий элемент находится на следующей строке, то инкрементируем β, если на следующем столбце, то инкрементируем α, если подходящий элемент лежит на диагонали, то $\alpha = \beta = 1$.

Изменение в реализации:

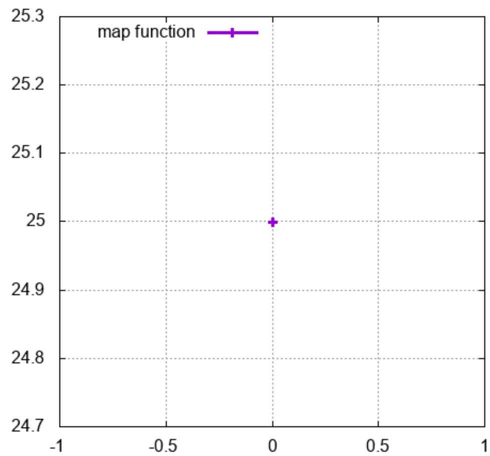
```
mapping_t map(const FloatingTable& table) {
    mapping_t result; std::array<double, 3> next;
    size_t row_penalty = 1, column_penalty = 1;
    for (size_t i=0, j=0; i<table.nrows() && j<table.ncols();) {
        result.emplace_back(i, j);
        if (i+1 == table.nrows()) break;
        if (j+1 == table.ncols()) break;
        next = {
            table.at(i+1,j+1),
            row_penalty * table.at(i+1,j),
            column_penalty * table.at(i,j+1)
        };
        auto min_pos = std::min_element(next.begin(), next.end());
        if (min_pos == next.begin())
            row_penalty = column_penalty = 1, i++, j++;
        else if (min_pos == std::next(next.begin()))
            i++, row_penalty++;
        else
            j++, column_penalty++;
    }
    return mapping_t;
}
```

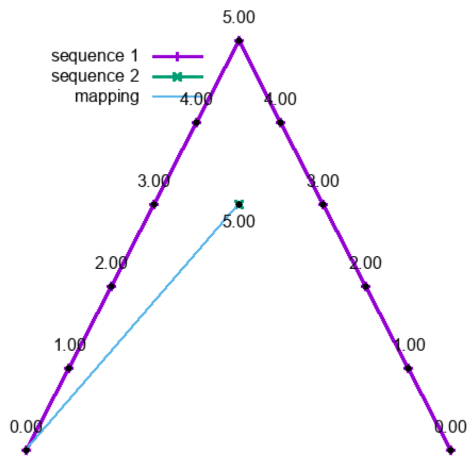


$$M(X, Y) = 0.190666$$

2.7.3 Добавление первого элемента таблицы

Рассмотрим следующий пример





$$M(X,Y) = 25$$

Из-за того, что наш алгоритм начинает обход по таблице с левого верхнего угла и заканчивает работу тогда, когда хотя бы по одной из последовательностей закончится обход, мы получаем большое значение функционала. Нам бы хотелось, чтобы точка 5 второй последовательности из данного примера сопоставилась с соответсвующей ей точкой 5 из первой последовательности.

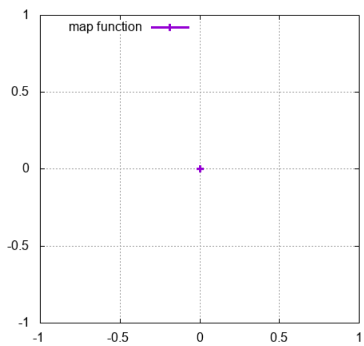
Возможное решение: Давайте попробуем начинать наш алгоритм с поиска элемента второй последовательности такого, что наш функционал является минимальным.

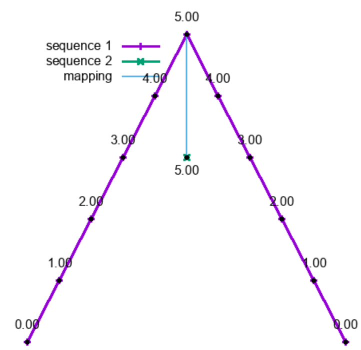
Изменение в реализации:

```
class FloatingTable {
 public:
    FloatingTable(const Sequence& BiggerSequence,
                  const Sequence& SmallerSequence)
        : _rows(BiggerSequence.size())
        , _columns(SmallerSequence.size())
        , _table(std::make_unique<uint32_t[]>(_rows * _columns)) {
        if (BiggerSequence.size() < SmallerSequence.size())
            throw std::runtime_error("Incorrect sizes");
        /* ... */
    }

    const double* begin() const
        { return _table.get(); }
    const double* end()   const
        { return _table.get() + _rows * _columns; }

 /* ... */
};
```

```cpp
mapping_t map(const FloatingTable& table) {
    /* ... */
    auto pos = std::min_element(
        table.begin(), table.begin() + table.ncolumns()
    );
    size_t i = 0, j = (pos - table.begin());

    for (; i<table.nrows() && j<table.ncols();) {
        /* ... */
    }
    return result;
}

```

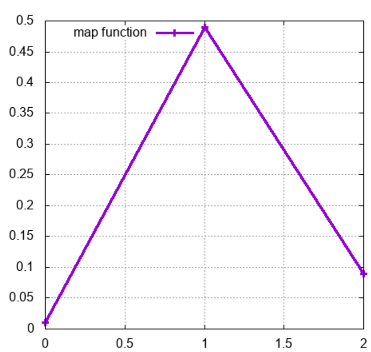
Посмотрим снова на приведенный пример:

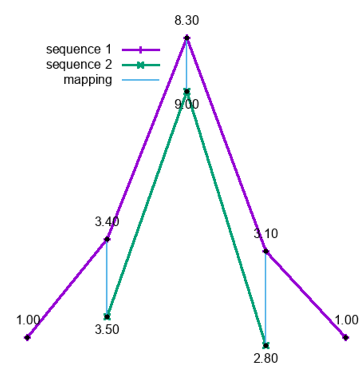




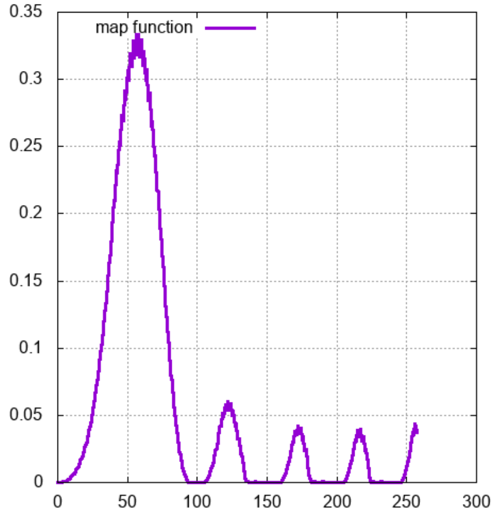
$$M(X,Y) = 0$$

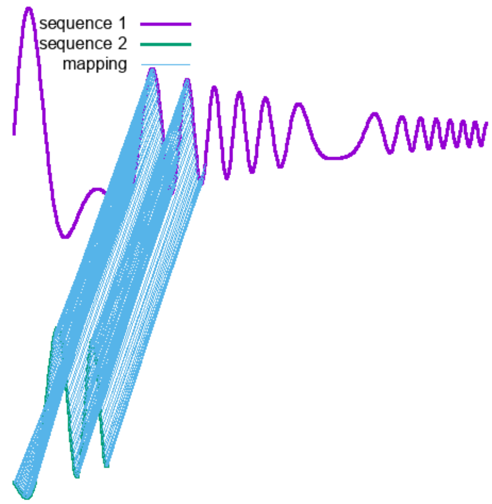
Заметно улучшение. Давайте посмотрим на другие примеры:





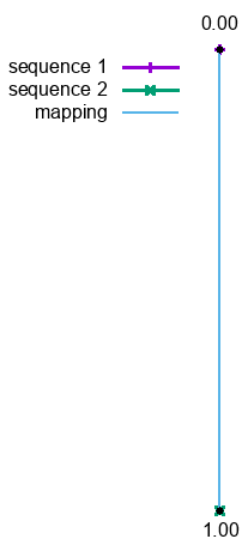
$$M(X,Y) = 0.196667$$

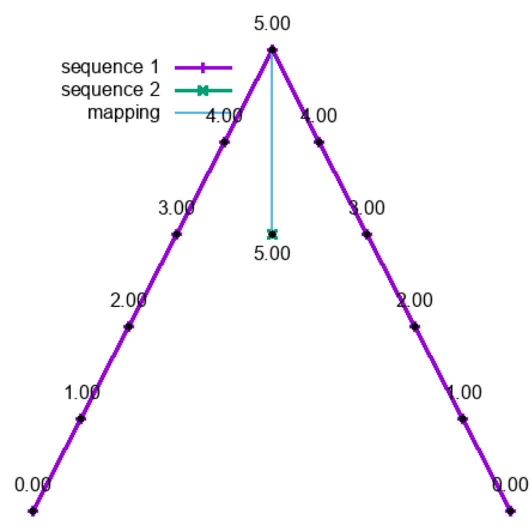


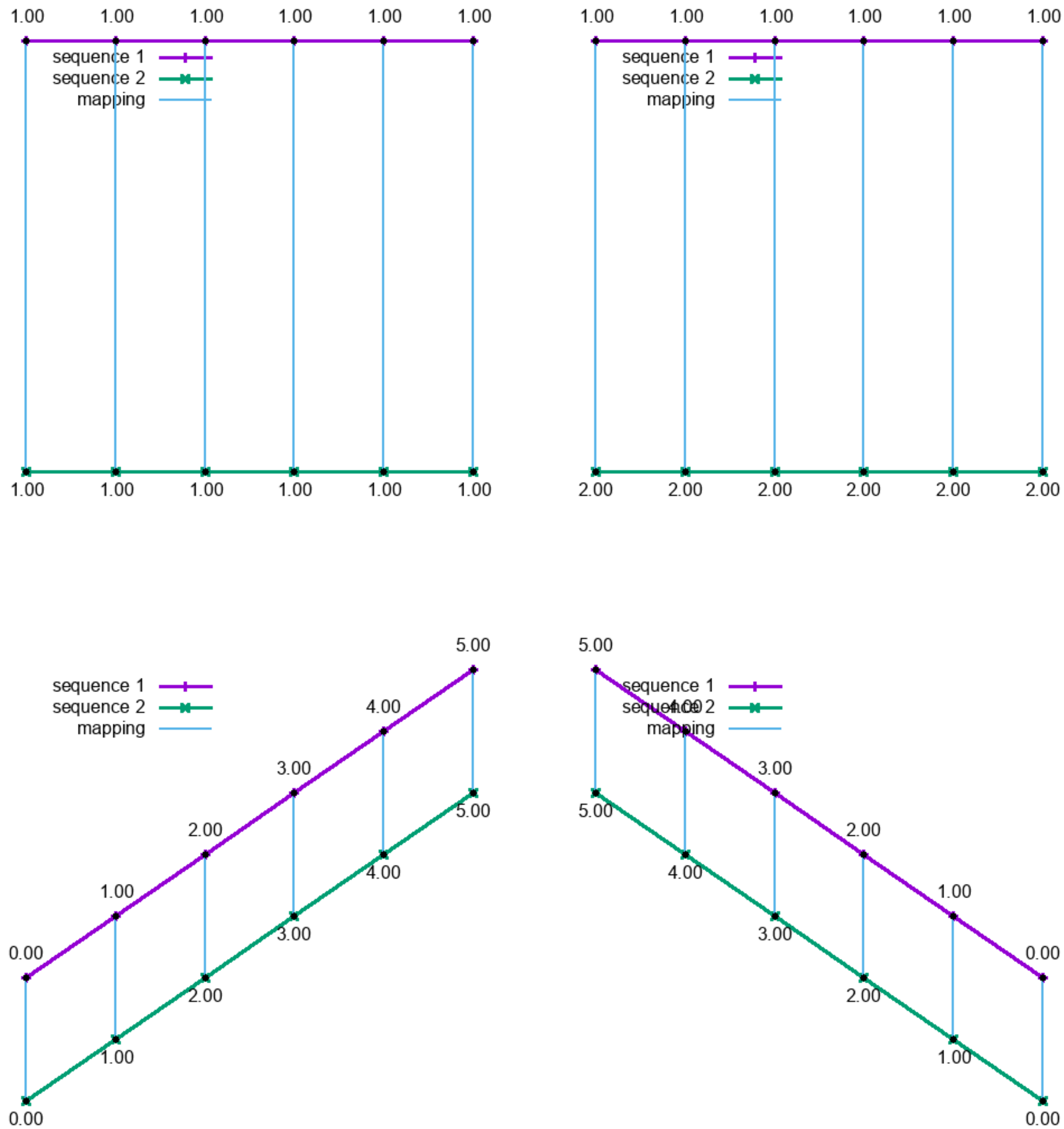


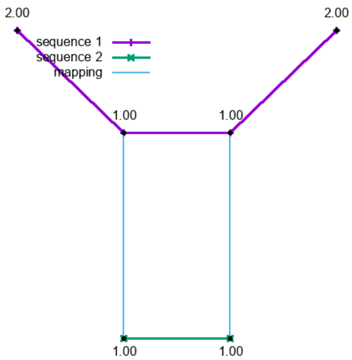
$$M(X, Y) = 0.0575981$$

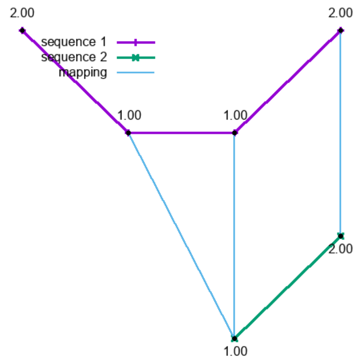
2.8 Тестирование

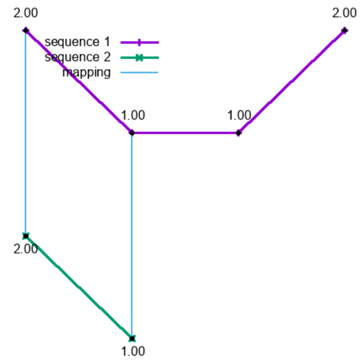


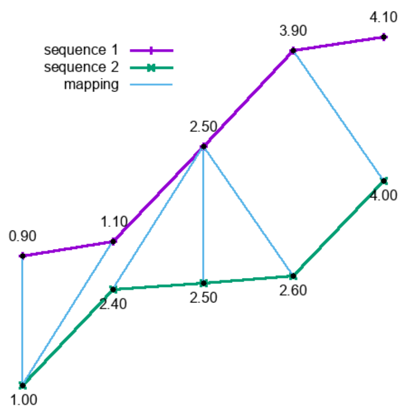


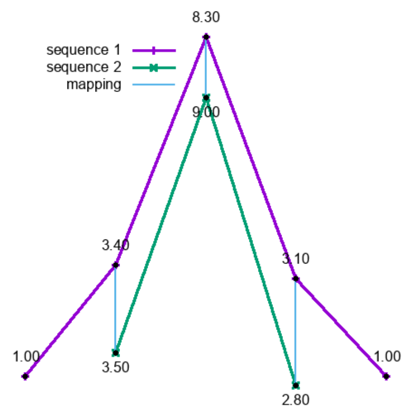

1.00 1.00 1.00 1.00 1.00 1.00
sequence 1
sequence 2
mapping
1.00 1.00 1.00 1.00 1.00 1.00
1.00 1.00 1.00 1.00 1.00 1.00
sequence 1
sequence 2
mapping
2.00 2.00 2.00 2.00 2.00 2.00
sequence 1
sequence 2
mapping
0.00 1.00 2.00 3.00 4.00 5.00
0.00 1.00 2.00 3.00 4.00 5.00
5.00 4.00 3.00 2.00 1.00 0.00
sequence 1
sequence 2
mapping
5.00 4.00 3.00 2.00 1.00 0.00

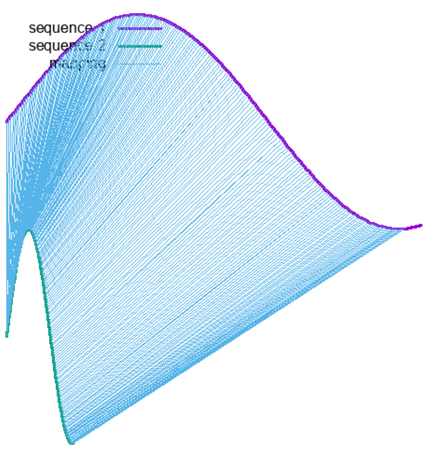

sequence 1
sequence 2
mapping
2.00
2.00
1.00
1.00
1.00
1.00

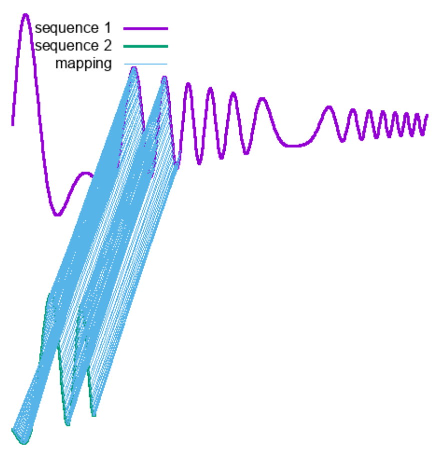

sequence 1
sequence 2
mapping
2.00
2.00
1.00
1.00
2.00
1.00


sequence 1
sequence 2
mapping
2.00
2.00
1.00
1.00
2.00
1.00


sequence 1
sequence 2
mapping
4.10
3.90
2.50
1.10
0.90
4.00
2.40
2.50
2.60
1.00


sequence 1
sequence 2
mapping
8.30
9.00
3.40
3.10
1.00
1.00
3.50
2.80


sequence 1
sequence 2
mapping


sequence 1
sequence 2
mapping