Оглавление

1 Введение

С развитием компьютеров и языков программирования они всё чаще используются для научных и статистических расчётов. Для этих целей были созданы специальные системы компьютерной математики (СКМ) такие как Mathemacia, Matlab, Maple, R, Sage. Incanter - одна из таких систем написанных на языке Clojure.

Во время различных вычислений иногда возникает необходимость в интерполяции: построении непрерывных функций по заданным значениям в определённых узлах. Во всех крупных СКМ присутстсвуют специальные модули для интерполяции функций. Задачей данной работы является добавление такого модуля для интерполяции функций 1 и 2 переменных в систему Incanter.

В отчёте кратко описан язык Clojure, система Incanter, рассмотрены 4 метода интерполяции функций 1 и 2 переменных.

2 Язык программирования Clojure

Clojure - современный диалект языка программирования Lisp. Clojure - язык программирования общего назначения с поддержкой разработки в интерактивном режиме, поощряющий функциональное программирование, и упрощающий поддержку многопоточности. Clojure работает на платформе JVM (Java Virtual Machine), что предоставляет доступ ко множеству библиотек написанных для данной платформы.

2.1 Основные возможности языка

Clojure является функциональным языком программирования с поддержкой функций в качестве объектов первого класса (first class objects) и неизменяемыми (за исключением специальных случаев) данными, включая поддержку "ленивых"коллекций данных. От Lisp'a Clojure "унаследовал"макросы, мультиметоды и интерактивный стиль разработки, а JVM дает переносимость и доступ к большому набору библиотек, созданных для этой платформы.

Неизменность структур данных позволяет использовать их в разных потоках выполнения программы, что упрощает многопоточное программирование. Однако не все структуры являются неизменяемыми — в нужных случаях программист может явно использовать изменяемые структуры данных, используя Software Transactional Memory (STM), что обеспечивает надежную работу в многопоточной среде.

За счет того, что Clojure был спроектирован для работы на базе JVM, обеспечивается доступ к большому набору библиотек, существующих для данной платформы. Взаимодействие с Java реализуется в обе стороны — как вызов кода, написанного на Java, так и реализация классов, которые доступны как для вызова из Java, так и из других языков, существующих для JVM, например, Scala.

В clojure присутствуют макросы. Макросы — это мощное средство уменьшения сложности кода, позволяющие строить проблемно-ориентированную среду на основе базового языка. Макросы активно используются в Clojure, и множество конструкций, составляющих язык, определены как макросы

на основе ограниченного количества специальных форм и функций, реализованных в ядре языка. Макросы в Clojure смоделированы по образцу макросов в Common Lisp, и являются функциями, которые выполняются во время компиляции кода. В результате выполнения этих функций должен получиться код, который будет подставлен на место вызова макроса.

2.2 Примеры

Определение нормы трёхмерного вектора

```
(defn norm [[x y z]]
(Math/sqrt (+ (* x x) (* y y) (* z z))))

(norm [1 2 3]) ; 3.7416573867739413
(norm [1 0 1]) ; 1.4142135623730951
```

Приближённое вычисление производной в точке по формуле: $f' \approx \frac{f(x+h)-f(x)}{h}$

```
(defn derivative [f x]
 (let [h 0.00001]
   (/ (- (f (+ x h))
         (f x))
      h)))

(defn f1 [x] x)        ; f1(x) = x
(defn f2 [x] (* x x)) ; f2(x) = x * x

(derivative f1 0) ; f1'(0) = 1
(derivative f2 0) ; f2'(0) = 0.00001
(derivative f2 1) ; f2'(0) = 8.00001
```

3 Платформа для статистической обработки данных и работы с графикой Incanter

Incanter - математический пакет для алгебраических и статистических расчётов на языке Clojure. Он был разработан под влиянием языка R, таким образом многие функции имеют структуру, похожую на аналогичные функции в R. Данный пакет является крупнейшим математическим пакетом для Clojure. Для реализации многих частей, таких как линейная алгебра, графика используются популярные Java библиотеки, например Parallel Colt (линейная алгебра, статистика), JFreeChart (библиотека для построение различных графиков).

Пакет включает в себя следующие основные модули:

- core - основные математические функцие, функции для работы с матрицами и векторами;
- charts - функции для построение и отображения различных графиков и гистограмм, также есть возможность создания динамических графиков;
- stats - функции для произведение статистических расчётов;
- distributions - функции для моделирование различных видов распределения случайной величины;
- latex - функции для отображения математических формул, применяется для добавление подписей к графикам;
- mongodb - функции для работы с базой данных MongoDB;

3.1 Примеры

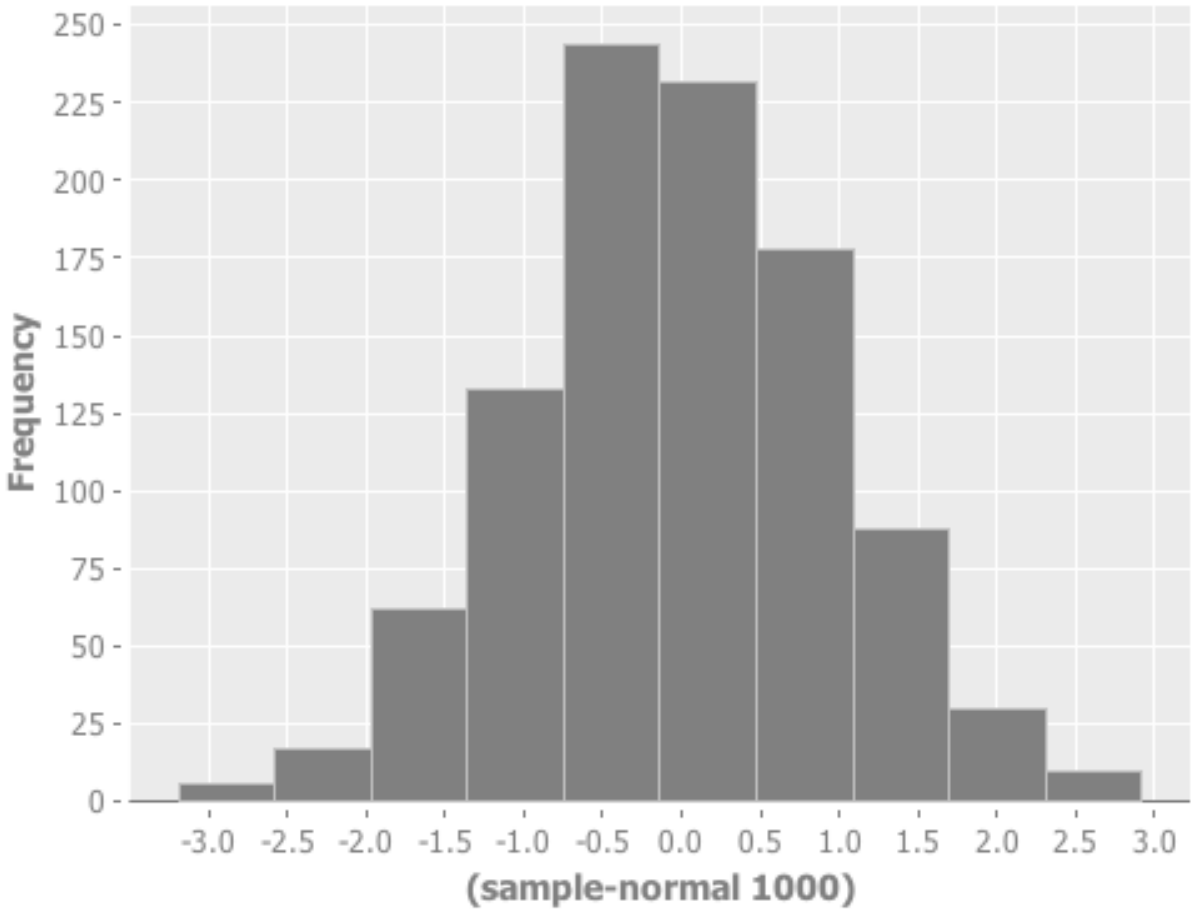
Работа с матрицами

```
(def a (matrix [[1 2 3] [4 4 6] [7 8 9]])) ; matrix 3x3
```

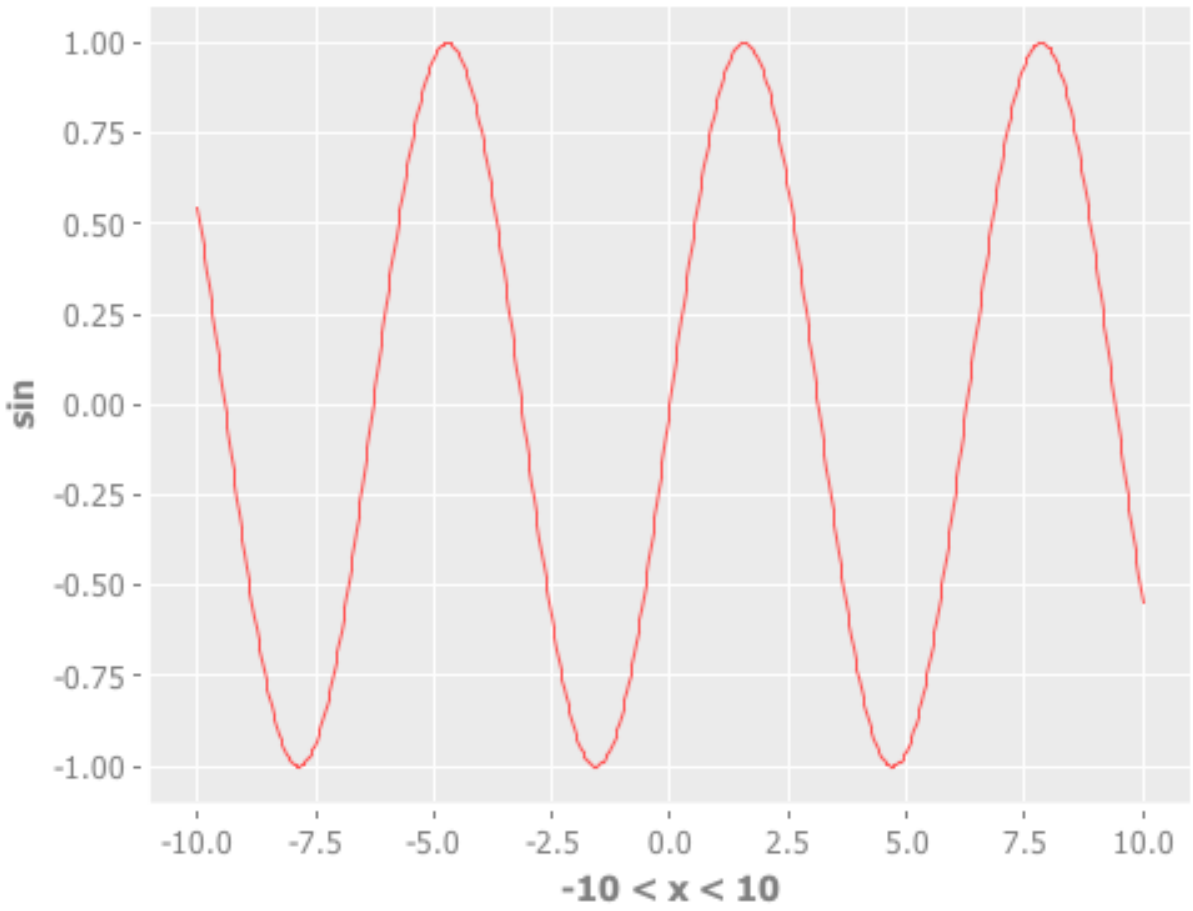
```
(def b (matrix [1 2 3])) ; vector of 3 elements

(solve a) ; finds inverse matrix of a

(solve a b) ; solves system a x = b

(mmult a a) ; multiplies a by itself
```

Построение графиков

```
(view (histogram (sample-normal 1000)))
```




```
(view (function-plot sin -10 10))
```


1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
sin
-10.0
-7.5
-5.0
-2.5
0.0
2.5
5.0
7.5
10.0
-10 < x < 10

4 Интерполяция

Интерполяция - способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. В научных и инженерных расчётах часто приходится оперировать наборами значений, полученных опытным путём или методом случайной выборки. Как правило, на основании этих наборов требуется построить функцию, на которую могли бы с высокой точностью попадать другие получаемые значения. Такая задача называется аппроксимацией. Интерполяцией называют такую разновидность аппроксимации, при которой кривая построенной функции проходит точно через имеющиеся точки данных.

Интерполяция может применяться для различных целей:

- масштабирование изображений;
- генерация гладких кривых и поверхностей в 3D графике;
- цифровая обработка сигналов;

Задачей данной работы является создание модуля interpolation для платформы Incanter, который будет содержать функции для интерполяции функций 1 и 2 переменных.

5 Интерполяция функции одной переменной

Рассмотрим набор попарно различных точек $\{x_i\}_{i=0}^n, x_i \in [a,b]$. Пусть $\{y_i\}_{i=0}^n$- значения некоторой функции $f\colon [a,b] \to \mathbb{R}$: в этих точках: $y_i = f(x_i)$. Предполагается, что сама функция f не известна, а известны только её значения в точках x_i. Задача интерполяции функции 1 переменной - построить функцию $\varphi\colon [a,b] \to \mathbb{R}$, такую что выполняются следующие условия: $\varphi(x_i) = y_i$. Т.е. построенная функция φ должна совпадать с неизвестной функцией f в заданном наборе узлов. Далее будут рассмотрены 3 способа построения функции φ: линейный, полиномиальный и кубическая интерполяция. Также будут рассмотрены B-сплайны, которые не интерполируют функцию, а приближают её.

5.1 Линейная интерполяция

Линейная интерполяция - наиболее простой способ интерполяция, при котором φ является кусочно-линейной функцией. При таком способе интерполяции соседние узлы соединены прямой линией. Интерполирующая функция φ имеет следующий вид:

$$\varphi(x) = y_i + (y_{i+1} - y_i)\frac{x - x_i}{x_{i+1} - x_i}, x \in [x_i, x_{i+1}] \tag{1}$$

Преимущества:

- простота реализации;
- высокая скорость построения φ;
- высокая скорость вычисления $\varphi(x)$;

Недостатки:

- φ не является непрерывно-дифферецируемой;

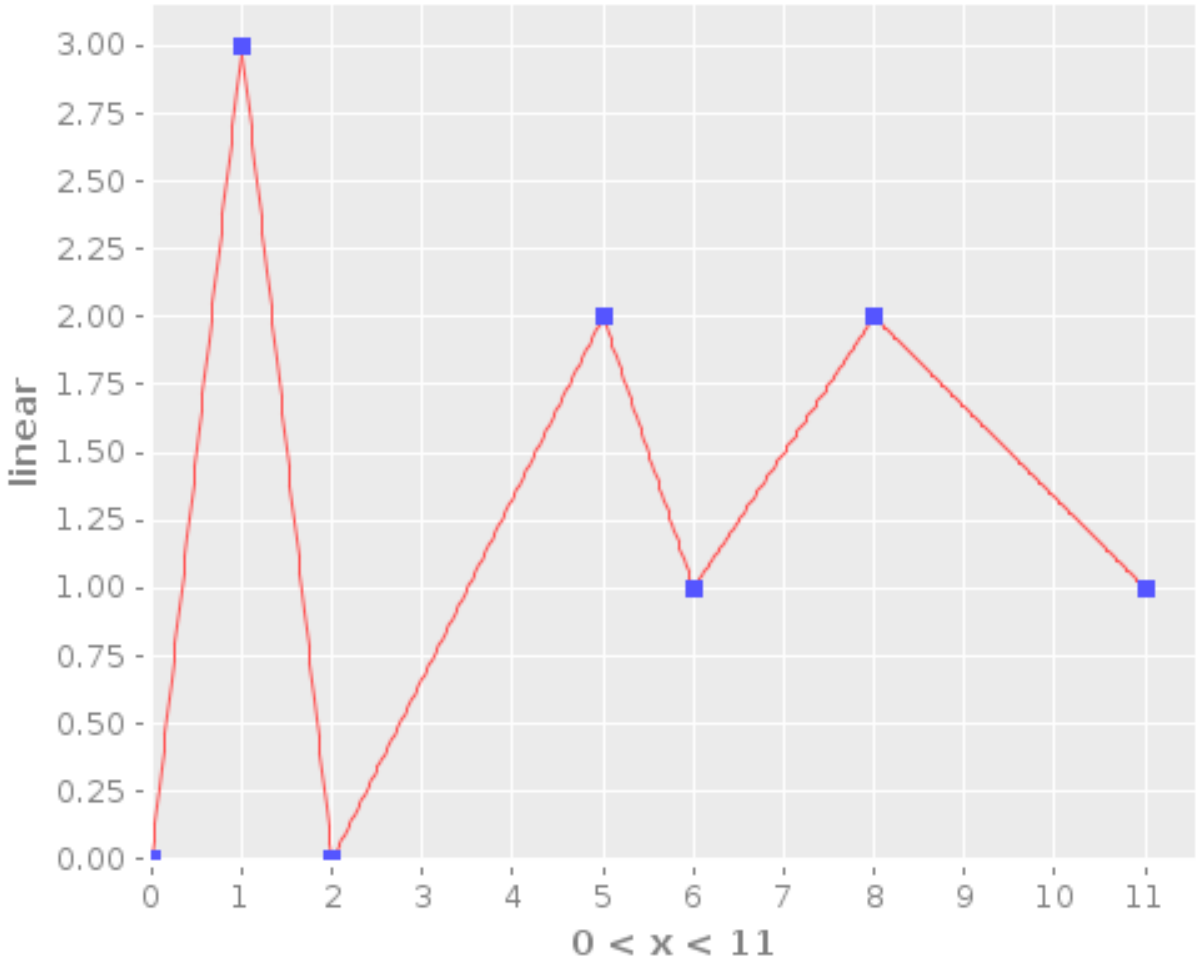
Временная сложность метода:

Построение φ: $O(n \log n)$ - требуется отсортировать все узлы.

Вычисление $\varphi(x)$: $O(\log n)$ - поиск соответствующего узла.

Пример:

```
; define points
(def points [[0 0] [1 3] [2 0] [5 2] [6 1] [8 2] [11 1]])

; build interpolation function
(def lin (interpolate points :linear))

; view plot on [0, 11]
(view (function-plot lin 0 11))
```



5.2 Полиномиальная интерполяция

В данном способе интерполяции φ является многочленом степени n. Для построение φ используется формула Ньютона с разделёнными разностями. Вычисления производятся по следующим формулам:

$$\begin{gathered} \varphi(x) = \sum_{i=0}^{n} f[x_0, \ldots, x_i]\omega_i(x) \\ \omega_i(x) = (x - x_0)(x - x_1)\ldots(x - x_i) \\ f[x_0, \ldots, x_i] = \frac{f[x_1, \ldots, x_i] - f[x_0, \ldots, x_{i-1}]}{x_i - x_0}, \forall i \geq 1 \\ f[x_j] = f(x_j), \forall j \end{gathered} \tag{2}$$

Преимущества:

- φ имеет производную любого порядка;

Недостатки:

- при больших n интерполяционный многочлен будет иметь большую погрешность интерполирования;
- низкая скорость построенире φ;
- низкая скорость вычисления $\varphi(x)$;

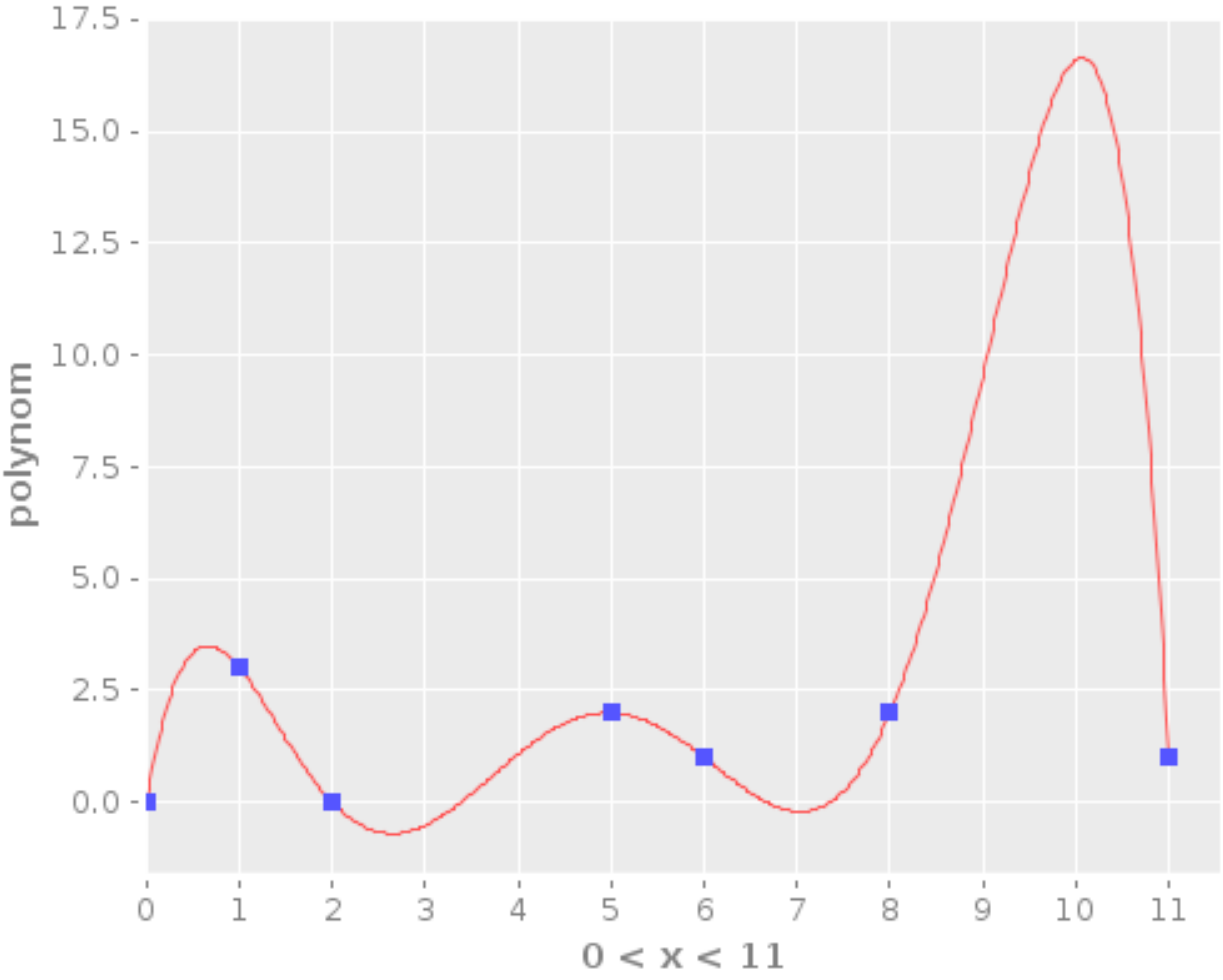
Временная сложность метода:

Построение φ: $O(n^2)$
Вычисление $\varphi(x)$: $O(n)$

Пример:

```
; define points
(def points [[0 0] [1 3] [2 0] [5 2] [6 1] [8 2] [11 1]])

; build interpolation function
(def polynom (interpolate points :polynomial))
```

```
; view plot on [0, 11]
(view (function-plot polynom 0 11))
```



5.3 Кубический сплайн

Функция φ является кусочной и на каждом отрезке $[x_{i-1}, x_i], i = \overline{1,n}$ задаётся отдельным кубическим многочленом:

$$\varphi(x) = s_i(x) = \alpha_i + \beta_i(x - x_i) + \frac{\delta_i}{2}(x - x_i)^2 + \frac{\delta_i}{6}(x - x_i)^3, x \in [x_{i-1}, x_i] \quad (3)$$

Также накладываются требования наличия непрерывной первой и второй производной φ, из чего получаем дополнительные условия:

$$\begin{aligned} s'_{i-1}(x_{i-1}) &= s'_i(x_{i-1}), i = \overline{2,n} \\ s''_{i-1}(x_{i-1}) &= s''_i(x_{i-1}), i = \overline{2,n} \end{aligned} \quad (4)$$

Используя эти условия и условия интерполяции получаем следующие формулы для вычисления коэффициентов $\{\alpha_i\}$, $\{\beta_i\}$, $\{\delta_i\}$:

$$\begin{gathered} h_i = x_i - x_{i-1}, i = \overline{1,n} \\ \alpha_i = y_i, i = \overline{0,n} \\ \beta_i = \frac{y_i - y_{i-1}}{h_i} + \frac{2\gamma_i + \gamma_{i-1}}{6}, i = \overline{1,n} \\ \delta_i = \frac{\gamma_i - \gamma_{i-1}}{h_i}, i = \overline{2,n} \end{gathered} \tag{5}$$

Коэффициенты $\{\delta_i\}$ можно получить, решив следующую 3-диагональную систему линейных уравнением:

$$h_i\gamma_{i-1} + 2(h_i + h_{i+1})\gamma_i + h_{i+1}\gamma_{i+1} = 6(\frac{y_{i+1} - y_i}{h_{i-1}} - \frac{y_i - y_{i-1}}{h_i}), i = \overline{i, n-1} \tag{6}$$

Данная система имеет $n-2$ уравнений и n неизвестных. Задавая различные граничные условия можно получить недостающие уравнения и решить систему, тем самым получая коэффициенты для кубического сплайна. Были реализованы 2 вида граничных условий:

1. Естественные граничные условия. Полагают $\varphi''(x_0) = \varphi''(x_n) = 0$.
2. Периодические (замкнутые) граничные условия. Полагают $\varphi'(x_0) = \varphi'(x_n)$, $\varphi''(x_0) = \varphi''(x_n)$.

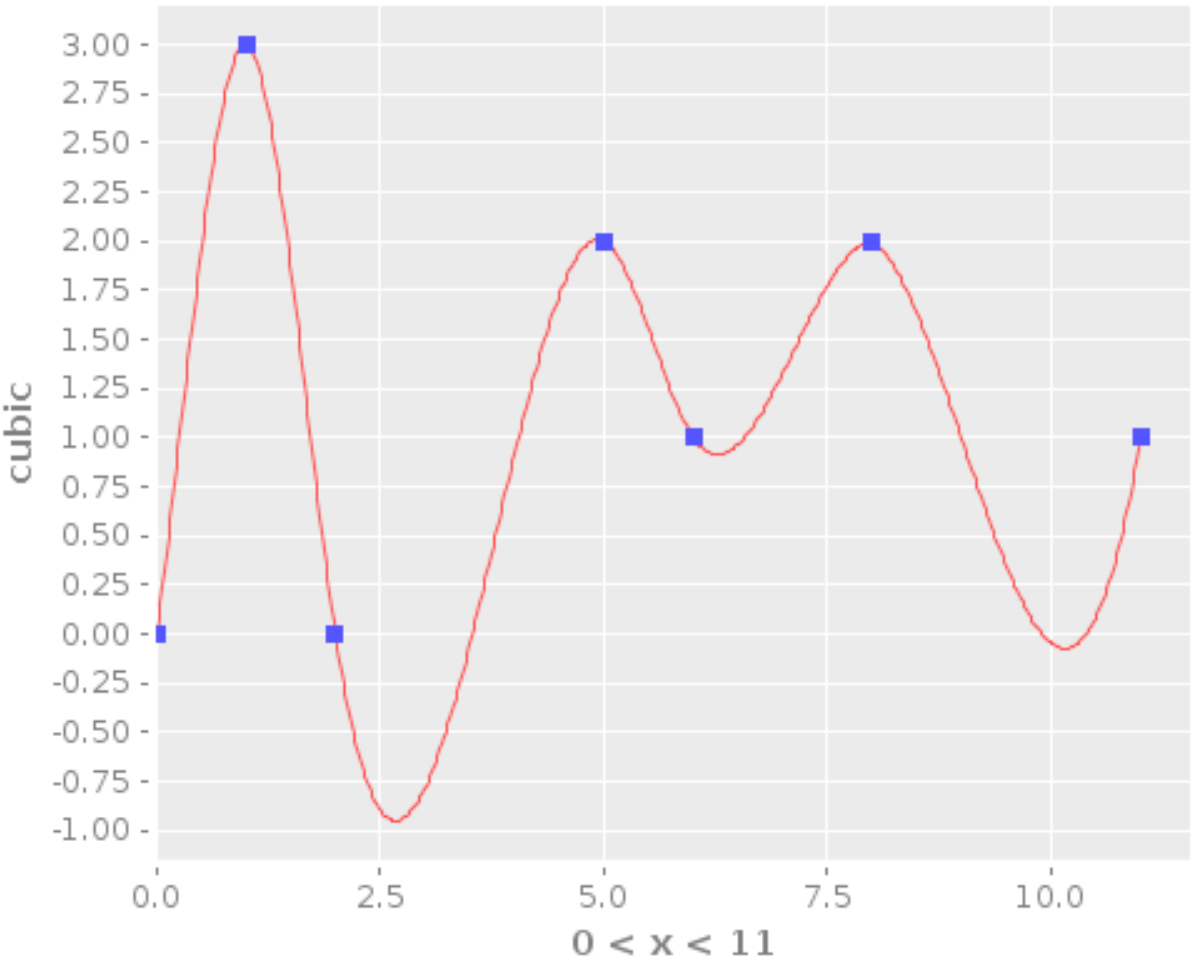
Временная сложность метода:

Построение φ: $O(n)$
Вычисление $\varphi(x)$: $O(\log n)$ - поиск соответствующего промежутка.

Пример:

```
; define points
(def points [[0 0] [1 3] [2 0] [5 2] [6 1] [8 2] [11 1]])

; build interpolation function
```

```
(def cubic (interpolate points :cubic-spline :boundaries :closed))

; view plot on [0, 11]
(view (function-plot cubic 0 11))
```



5.4 B-сплайн

B-сплайны не интерполируют узлы, как ранее расмотренные методы, а приближают их. При построении B-сплайновой кривой узлы $\{x_i\}$ обычно не задаются, они вычисляются в процессе построения кривой. Задаются только значения $\{y_i\}$.

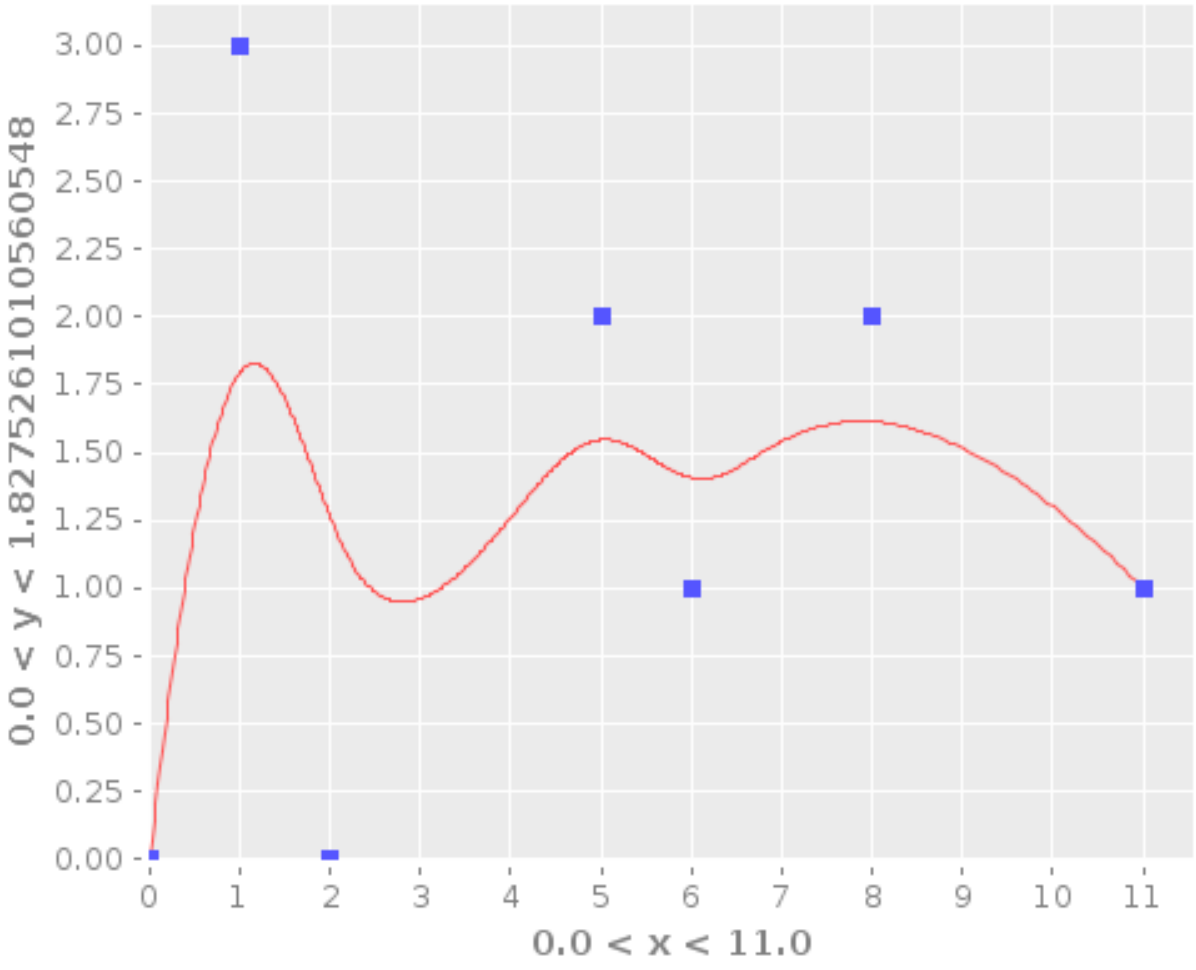
Временная сложность метода:

Построение φ: $O(n)$

Вычисление $\varphi(t)$: $O(d^2)$, где d - степень сплайна

Пример:

```
; define points
(def points [[0 0] [1 3] [2 0] [5 2] [6 1] [8 2] [11 1]])

; build approximation function
(def b-spline (interpolate-parametric points :b-spline :degree 3))

; view plot on [0, 1]
(view (parametric-plot cubic 0 1))
```



6 Интерполяция функции двух переменных

Задача приближения функции одной переменной, которая рассматривалась до сих пор, естественным образом обобщается на случай функций нескольких переменных. Рассмотрим задачу интерполяции функций двух переменных на прямоугольной области. Дано: набор узлов $(x_i, y_j), i = \overline{0, n}, j = \overline{0, m}$ и значения неизвестной функции f в этих узлах: $f(x_i, y_j) = z_{ij}$. Требуется построить функцию удовлетворяющую условиям интерполяции: $(x_i, y_j) = z_{ij}$ Существует и более сложная версия интерполяции функции 2 переменных, в которой узлы задаются не в виде сетки, а произвольным образом. Но решение такой задачи в данной работе рассматриваться не будет.

6.1 Билинейная интерполяция

Ключевая идея заключается в том, чтобы провести обычную линейную интерполяцию сначала по одной переменной, затем по другой.
Пусть $x \in [x_i, x_{i+1}], y \in [y_j, y_{j+1}]$. Тогда

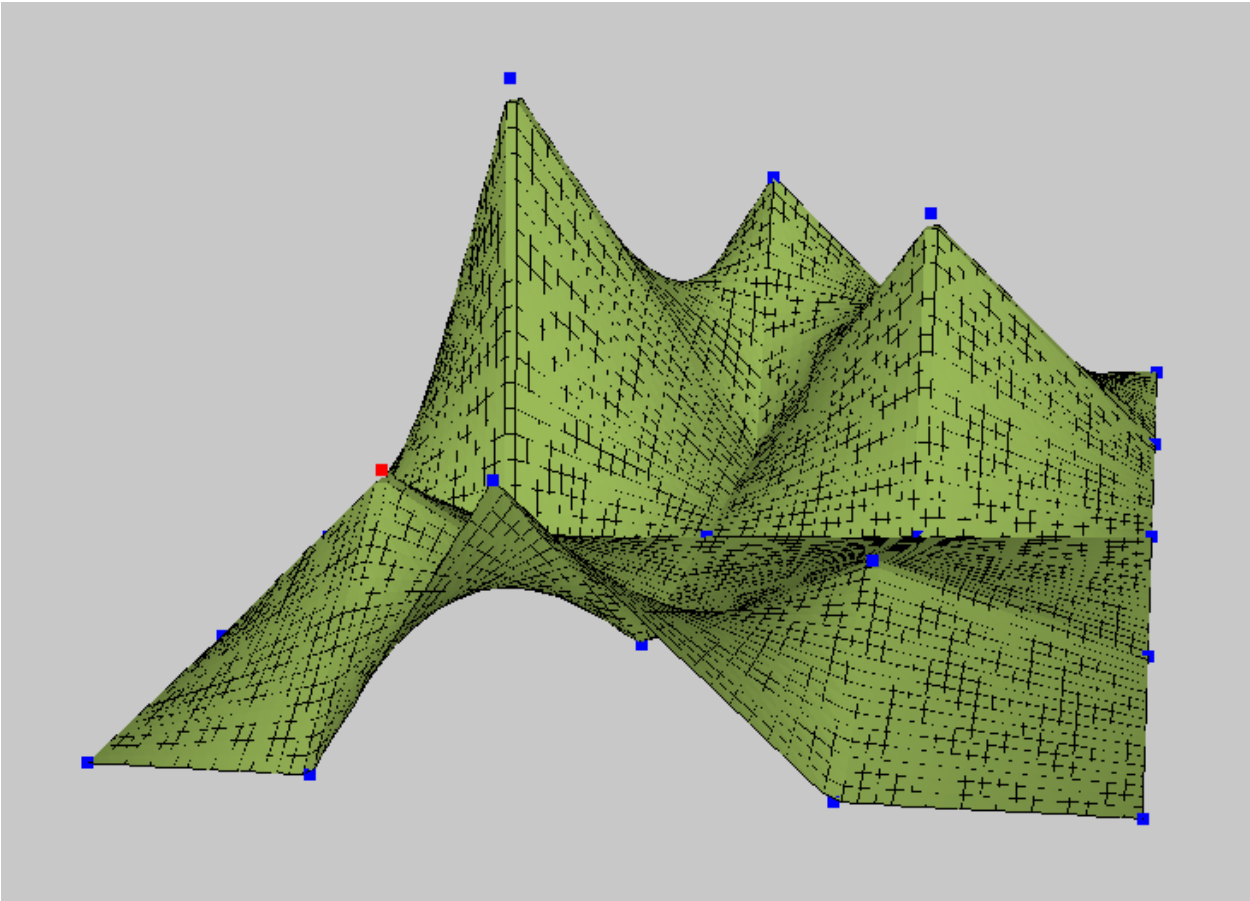
$$\begin{aligned} \varphi(x, y_j) &= z_{i,j} + (z_{i+1,j} - z_{i,j})\frac{x - x_i}{x_{i+1} - x_i} \\ \varphi(x, y_{j+1}) &= z_{i,j+1} + (z_{i+1,j+1} - z_{i,j+1})\frac{x - x_i}{x_{i+1} - x_i} \\ \varphi(x, y) &= \varphi(x, y_j) + (\varphi(x, y_{j+1}) - \varphi(x, y_j))\frac{y - y_j}{y_{j+1} - y_j} \end{aligned} \tag{7}$$

Временная сложность метода:

Построение φ: $O(nm)$ - требуется преобразовать входную сетку для возможности быстрого поиска нужного сегмента

Вычисление $\varphi(x, y)$: $O(\log n + \log m)$ - поиск сегмента

Пример:



6.2 Полиномиальная интерполяция

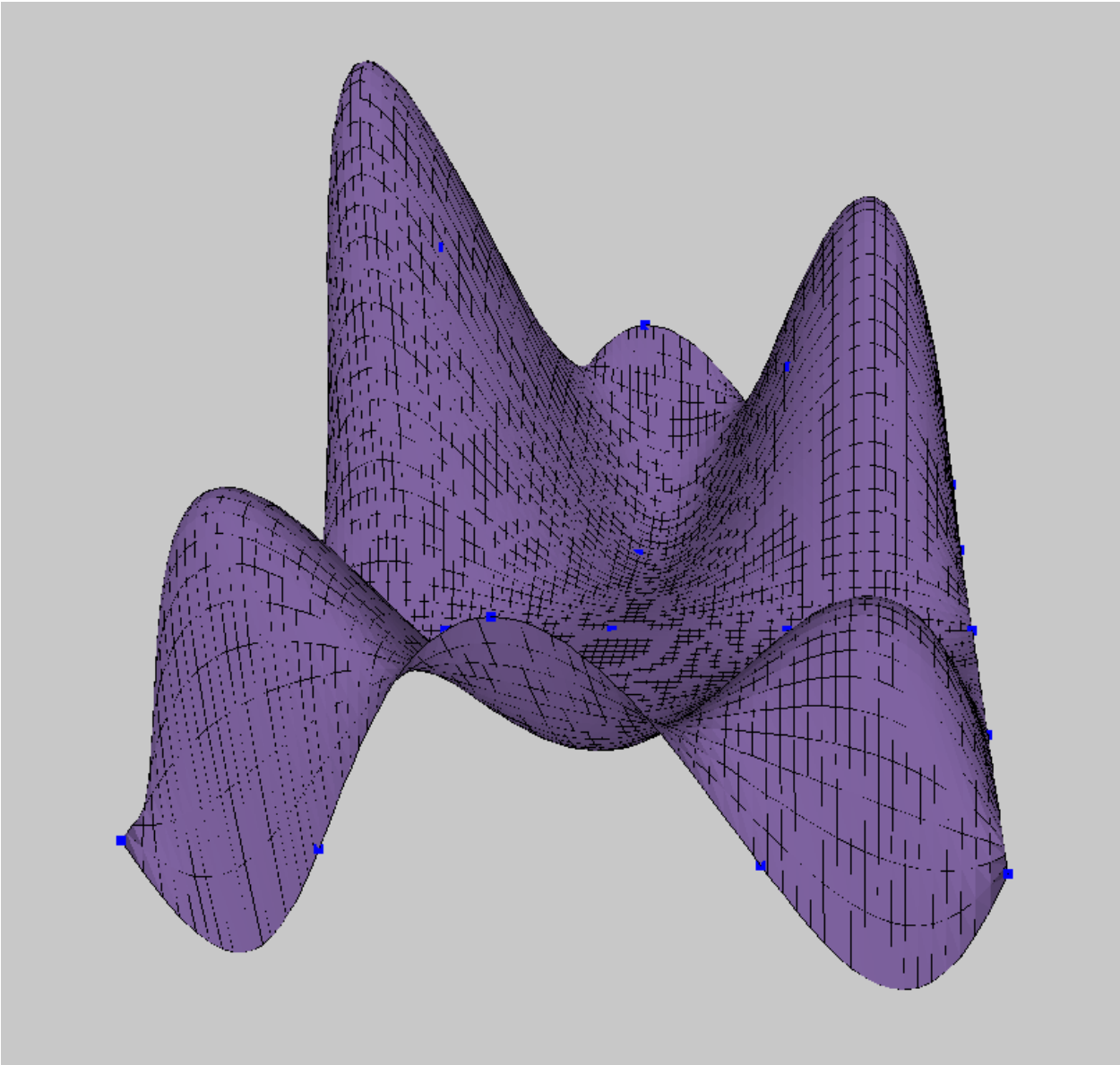
Полиномиальная интерполяция функции 2 переменных, как и для 1-мерного случая, была реализована в форме Ньютона.

Временная сложность метода:

Построение φ: $O(nm \max(n, m))$ - вычисление разделённых разностей для двумерного случая.

Вычисление $\varphi(x, y)$: $O(nm)$

Пример:



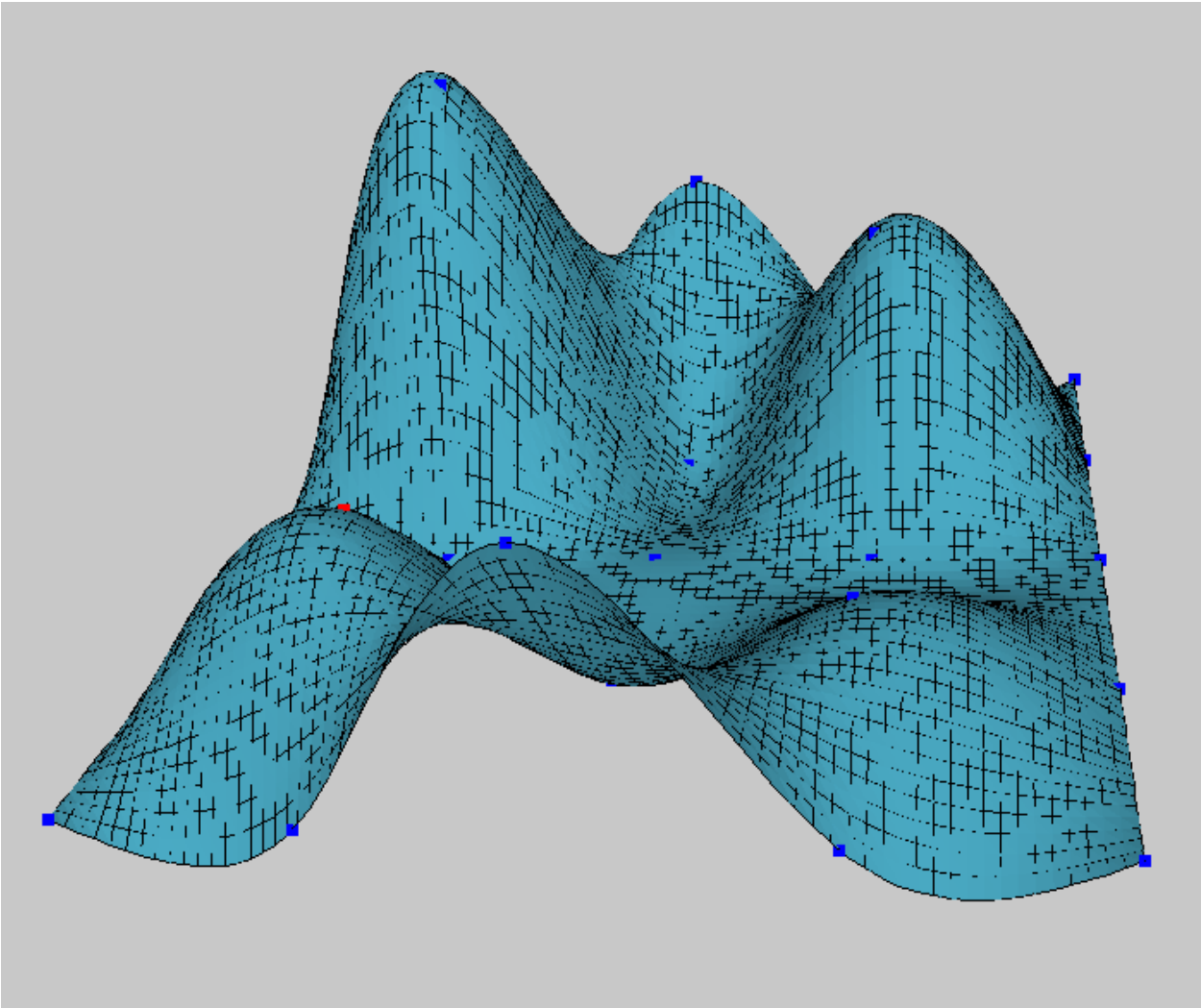
6.3 Бикубический сплайн

Аналог кубического сплайна. Основная идея заключается в том, что фиксируется одна из переменных, например y,и вычисляются коэффициенты кубических сплайнов по x. Далее по полученным коэффициентам строятся кубические сплайны, фиксируя уже переменную x. Полученные сплайны и используются для вычисления функции.

Временная сложность метода:

Построение φ: $O(nm)$
Вычисление $\varphi(x, y)$: $O(\log n + \log m)$

Пример:



6.4 B-поверхность

B-поверхность для функции двух переменных строится используя тензорное произведение B-сплайнов для одномерного случая.

Временная сложность метода:

Построение φ: $O(nm)$

Вычисление $\varphi(x, y)$: $O(d^2)$, где d - степень одномерных сплайнов.

Пример: