## АННОТАЦИЯ

Белоглазов Н.А., Алгоритмы интерполяции функций. Создание библиотеки для языка Clojure.: Дипломная работа/ Минск: БГУ, 2013. – 35 с.

В данной работе рассматриваются алгоритмы интерполяции функций 1 и 2 переменных, а также их реализация и создание библиотеки на языке Clojure.

## ANNOTATION

Belahlazau M, Functions interpolation algorithms. Creation of a library for Clojure programming language./ Minsk: Belarusian State University, 2013. – 35 p.

Algorithms for interpolation 1 and 2 variable functions are described in this work along with their implementation and usage in Clojure programming language.

## АНАТАЦЫЯ

Белаглазаў М.А., Алгарытмы інтэрпаляцыі функцый. Стварэнне бібліятэкі дзеля мовы праграмавання Clojure. / Мінск: БГУ, 2013. – 35с .

У дадзенай працы разглядаюца алгарытмы інтэрпаляцыі функцый 1 і 2 пераменных, а таксама іх рэалізацыя на мове праграмавання Clojure.

## РЕФЕРАТ

Дипломная работа, 35 страниц, 14 рисунков, 28 формул, 8 источников

**Ключевые слова**: ИНТЕРПОЛЯЦИЯ, B-СПЛАЙН, КУБИЧЕСКИЙ СПЛАЙН, СРЕДНЕКВАДРАТИЧНОЕ ПРИБЛИЖЕНИЕ, CLOJURE, INCANTER, СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ,

**Объект исследования** - методы интерполяции функций 1 и 2 переменных.

**Цель работы** - создание библиотеки различных методов интерполяции на языке Clojure для СКА Incanter.

**Результат работы** - библиотека интерполяции.

**Область применения** - научные вычисления при помощи СКА Incanter.

# Оглавление

# 1 Введение

В последнее время бурное развитие получило новое, актуальное научное направление – компьютерная математика. Ее можно определить как совокупность теоретических, алгоритмических, аппаратных и программных средств, предназначенных для эффективного решения на компьютерной технике всех видов математических задач, включая символьные преобразования и вычисления с высокой степенью визуализации всех видов вычислений. Применение компьютерной математики существенно расширяет возможности автоматизации всех этапов математического моделирования. Существует множество различных систем компьютерной алгебры, реализующие данные возможности. К ним можно отнести Matlab, Mathematica, Maple, R, Sage, Incanter и другие.

Вышеперечисленные системы включают в себя различные подсистемы для работы с линейной алгеброй, графикой, статистикой, символьными вычислениями и другими областями вычислительной математики. Большинство таких систем представляют функции, позволяющие интерполировать наборы данных. Многим из тех, кто сталкивается с научными и инженерными расчётами 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.

Incanter - система компьютерной алгербы, написанная на языке Clojure. В данной системе отсутствуют инструменты для интерполяции функций. Целью данной работы является изучение различных алгоритмов интерполяции функций одной и двух переменных, их реализация и добавление в систему Incanter.

# 2 Язык программирования Clojure

Clojure [7] - 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'а 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

```clojure
(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: $f' \approx \frac{f(x+h)-f(x)}{h}$

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

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

erivative f1 0) ; f1'(0) = 1
erivative f2 0) ; f2'(0) = 0.00001
erivative f2 4) ; f2'(4) = 8.00001
```

# 3 Платформа для статистической обработки данных и работы с графикой Incanter

Incanter [8]- 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)))
```

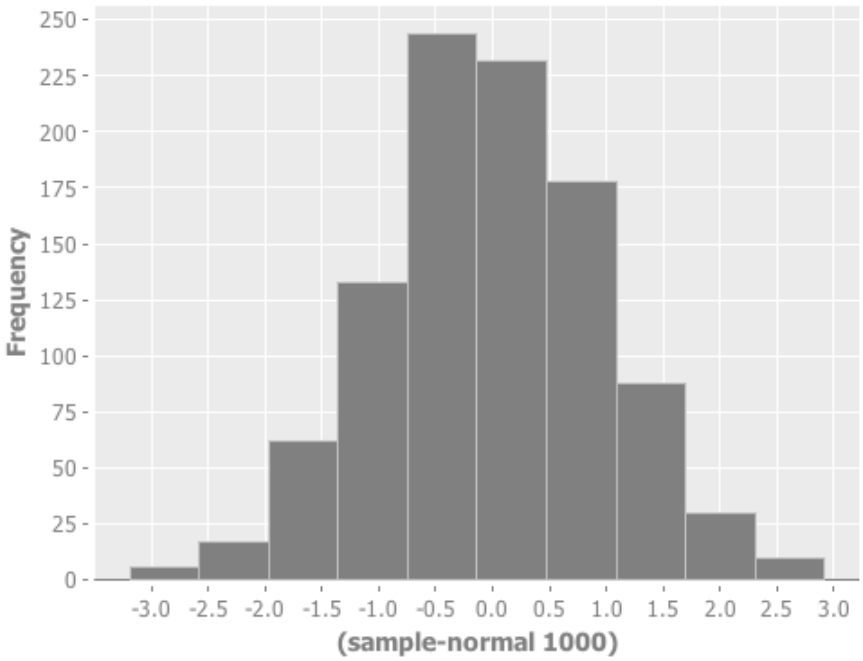


Рисунок 1 – Пример гистограммы нормального распределения

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

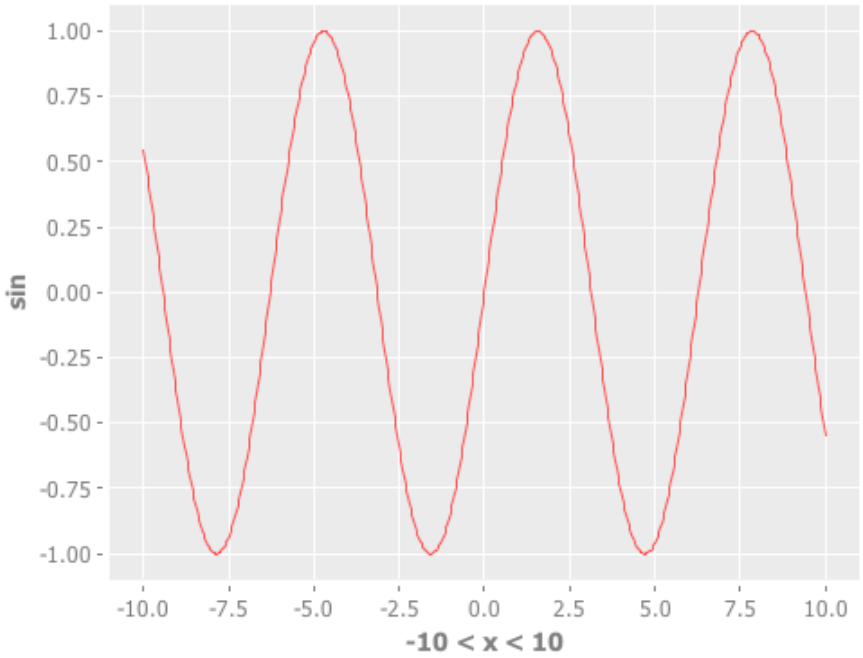


Рисунок 2 – Пример построение графика функции синус

# 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 $f: [a,b] \to \mathbb{R}$: 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 $\varphi$: $[a,b] \to \mathbb{R}$, такую что выполняются следующие условия: $\varphi(x_i) = y_i$. Т.е. построенная функция $\varphi$ должна совпадать с неизвестной функцией $f$ в заданном наборе узлов. Далее будут рассмотрены 3 способа построения функции $\varphi$: линейный, полиномиальный и кубическая интерполяция. Также будут рассмотрены среднеквадратичное приближение и B-сплайны, данные методы не интерполируют функцию, а приближают её. Методы реализованы на основе лекций Б.В. Фалейчика [1] [2].

## 5.1 Линейная интерполяция

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

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

Данный метод является наиболее быстрым методом из всех представленных. Но полученная функция не является непрерывно-дифференциируемой.

**Временная сложность метода:**

Построение $\varphi$: $O(n \log n)$ - 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))
```

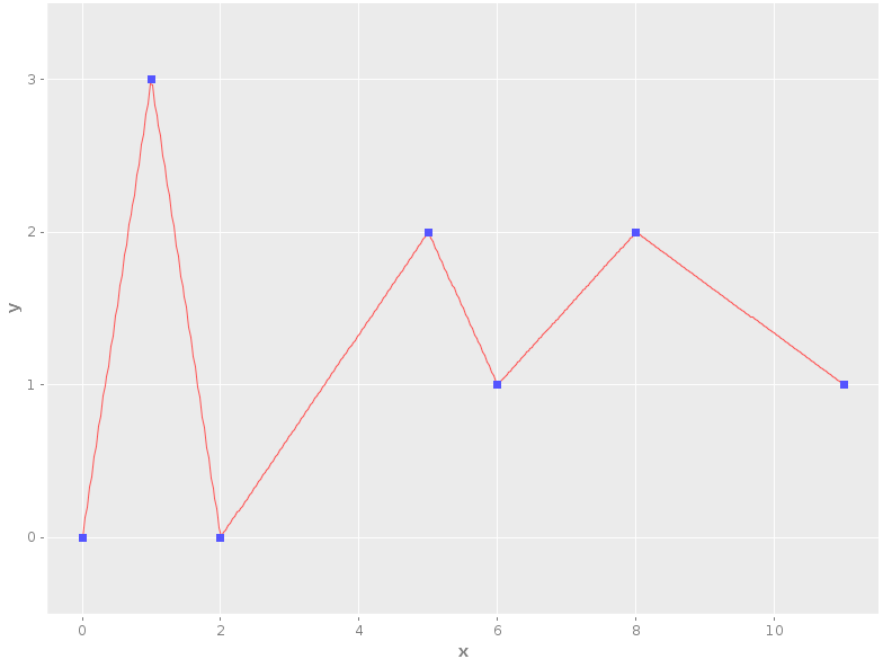


Рисунок 3 – Линейная интерполяция

## 5.2 Полиномиальная интерполяция в форме Ньютона

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

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

Результатом алгоритма является многочлен, функция дифференциируемая бесконечная число раз. Недостатком данного метода является то, что многочлен может иметь большую погрешность интерполирования, которая очень сильно зависит от выбора узлов $\{x_i\}$. Особенно это проявляется при больших $n$. Другим недостаком является низкая скорость построения многочлена и вычисления его значения в точке.

**Временная сложность метода:**

Построение $\varphi$: $O(n^2)$
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))
```

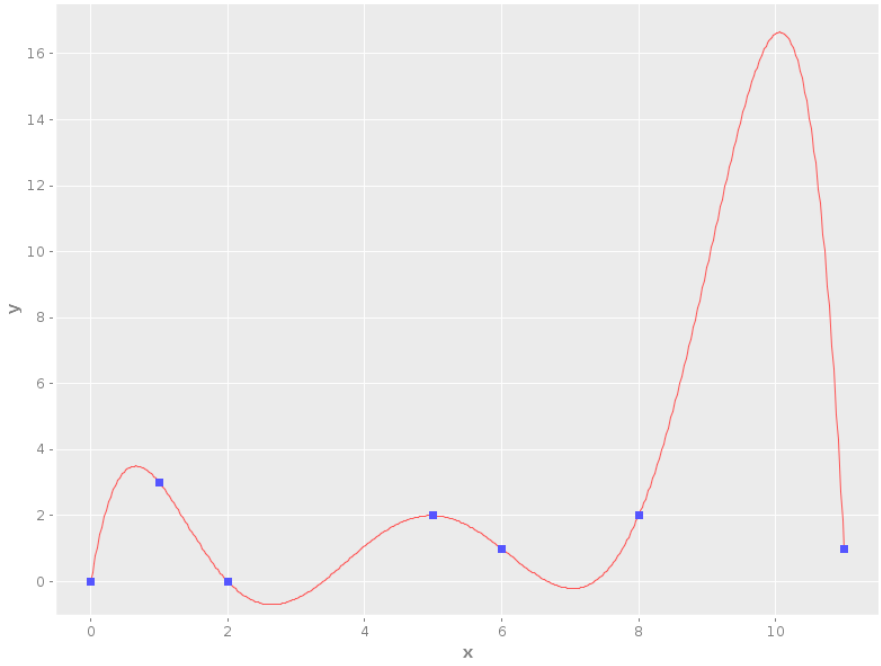


Рисунок 4 – Полиномиальная интерполяция

## 5.3 Полиномиальная интерполяция в барицентрической форме

Другой способ построения полиномиальных интерполяционных многочленов состоит в использовании барицентрической формы записи многочлена Лагранжа [3]. Барицентрическая интерполяционная формула выглядит следующим образом:

$$\varphi(x) = \frac{\sum_{i=0}^{n} y_i \frac{v_i}{x - x_i}}{\sum_{i=0}^{n} \frac{v_i}{x - x_i}} \tag{3}$$

где

$$v_i = \frac{1}{\prod_{j \neq i}(x_i - x_j)}, \quad i = \overline{0, n} \tag{4}$$

Одна из особенностей барицентрической формы состоит в том, что для построение многочлена для нового набора $y_i$ не требует пересчёта коэффициентов, как в форме Ньютона. Данная особенность хорошо подходит для построение параметрических кривых, когда узлы параметра $t$ фиксируются, вычисляются $v_i$, а далее поочередно подставляются наборы $x_i, y_i, z_i$ и получаются многочлены по каждому из наборов.

Если использовать узлы специального вида, то вычисление $v_i$ может свестись к более простой форме. Например если $x_i$ - чебышевские узлы второго рода:

$$x_i = \cos\frac{i\pi}{n}, \quad i = \overline{0, n} \tag{5}$$

то коэфициенты $v_i$ имеют следующий вид:

$$v_i = (-1)^i \delta_i, \quad \delta_i = \begin{cases} 1/2, & \text{если } i = 0 \text{ или } i = n, \\ 1, & \text{иначе.} \end{cases} \tag{6}$$

Барицентрическая форма использовалась для параметрической интерполяции, когда пользователь не задаёт узлы, а задаёт только точки, через которые должна проходить кривая и интервал параметризации. В качестве узлов использовались чебышевские узлы второго рода.

**Временная сложность метода:**

Построение $\varphi$: $O(n)$ - при условии использовании специальных узлов и быстрого вычисления $v_i$ за $O(1)$.

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 barycentric (interpolate-parametric points :polynomial

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

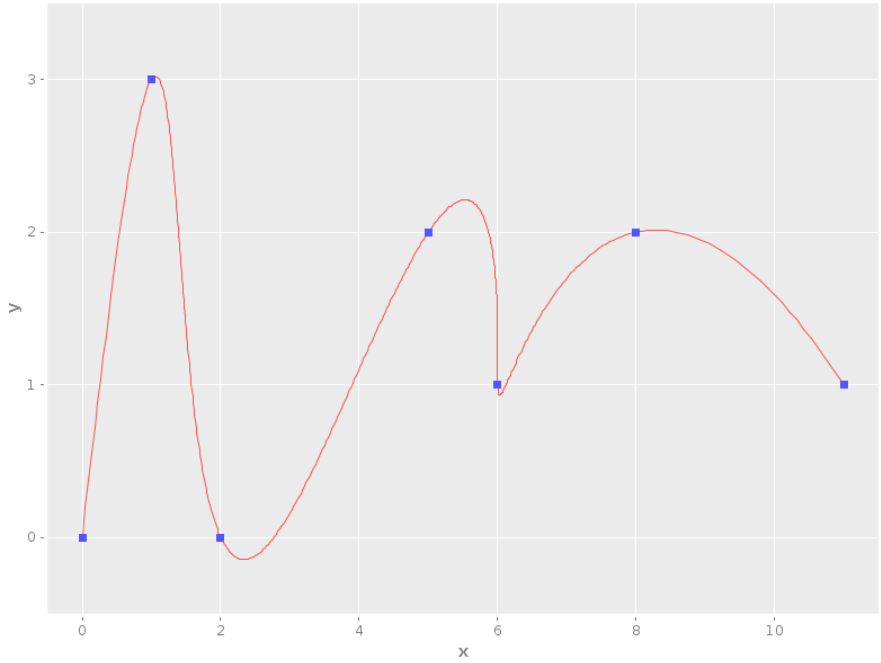


Рисунок 5 – Полиномиальная интерполяция с использованием узлов Чебышева

## 5.4 Кубический сплайн

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

$$\varphi(x) = s_i(x) = \alpha_i + \beta_i(x - x_i) + \frac{\delta_i}{2}(x - x_i)^2 + \frac{\gamma_i}{6}(x - x_i)^3, x \in [x_{i-1}, x_i] \quad (7)$$

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

$$\begin{aligned} s'_{i-1}(x_{i-1}) &= s'_i(x_{i-1}), i = \overline{2,n} \\ s''_{i-1}(x_{i-1}) &= s''_i(x_{i-1}), i = \overline{2,n} \end{aligned} \quad (8)$$

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

$$\begin{aligned} h_i &= x_i - x_{i-1}, i = \overline{1,n} \\ \alpha_i &= y_i, i = \overline{0,n} \\ \beta_i &= \frac{y_i - y_{i-1}}{h_i} + \frac{2\gamma_i + \gamma_{i-1}}{6}, i = \overline{1,n} \\ \delta_i &= \frac{\gamma_i - \gamma_{i-1}}{h_i}, i = \overline{2,n} \end{aligned} \quad (9)$$

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

$$h_i\gamma_{i-1} + 2(h_i + h_{i+1})\gamma_i + h_{i+1}\gamma_{i+1} = 6(\frac{y_{i+1} - y_i}{h_{i-1}} - \frac{y_i - y_{i-1}}{h_i}), i = \overline{i, n-1} \quad (10)$$

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 $\varphi'(x_0) = \varphi'(x_n)$, $\varphi$
   $\varphi''(x_n)$.

**Временная сложность метода:**

Построение $\varphi$: $O(n)$
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 :boundaries :closed))

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

## 5.5 Кубический эрмитов сплайн

Кусочно-полиномиальная функция $\varphi$ называется эрмитовым сплайном третьей степени для $f \in C^1[a, b]$, если

$$\begin{gathered} s\mid_{x\in\Delta_i} = s_i \in \mathbb{P}_3 \\ s(x_i) = f(x_i) \text{ и } s'(x_i) = f'(x_i) \ \ \forall i = \overline{0, n} \end{gathered} \quad (11)$$

Каждая функция $s_i$ является интерполяционным многочленом Эрмита третьей степени. Вычислять его будем используя форму Ньютона:

$$s_i(x) = \sum_{j=0}^{3} \alpha_{ij}\omega_{ij}(x) \quad (12)$$

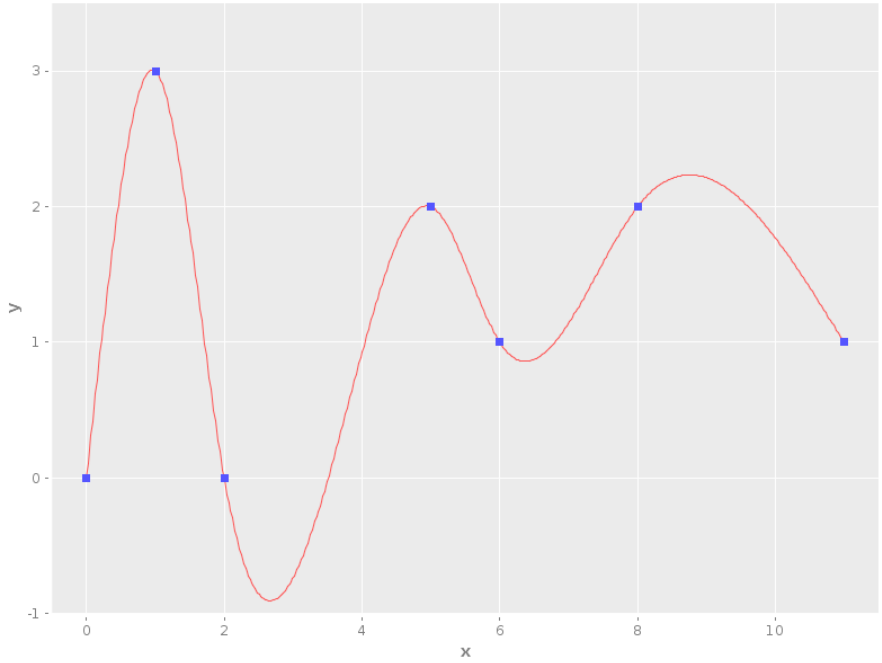


Рисунок 6 – Кубический сплайн с естественными граничными условиями

Где

$$\begin{array}{ll} \alpha_{i0} = f(x_{i-1}) & \omega_{i0}(x) = 1 \\ \alpha_{i1} = f'(x_{i-1}) & \omega_{i1}(x) = x - x_{i-1} \\ \alpha_{i2} = f[x_{i-1}, x_{i-1}, x_i] & \omega_{i2}(x) = (x - x_{i-1})^2 \\ \alpha_{i3} = f[x_{i-1}, x_{i-1}, x_i, x_i] & \omega_{i3}(x) = (x - x_{i-1})^2(x - x_i) \end{array} \tag{13}$$

Первые производные функции в узлах $x_i$ приближались используя конечные разности:

$$f'(x_i) \approx \begin{cases} \frac{f(x_1)-f(x_0)}{x_1-x_0} & i = 0 \\ \frac{1}{2}\left(\frac{f(x_i)-f(x_{i-1})}{x_i-x_{i-1}} + \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}\right) & i = \overline{1, n-1} \\ \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} & i = n \end{cases} \tag{14}$$

**Временная сложность метода:**

Построение $\varphi$: $O(n)$
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 hermite (interpolate points :cubic-hermite))
```

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

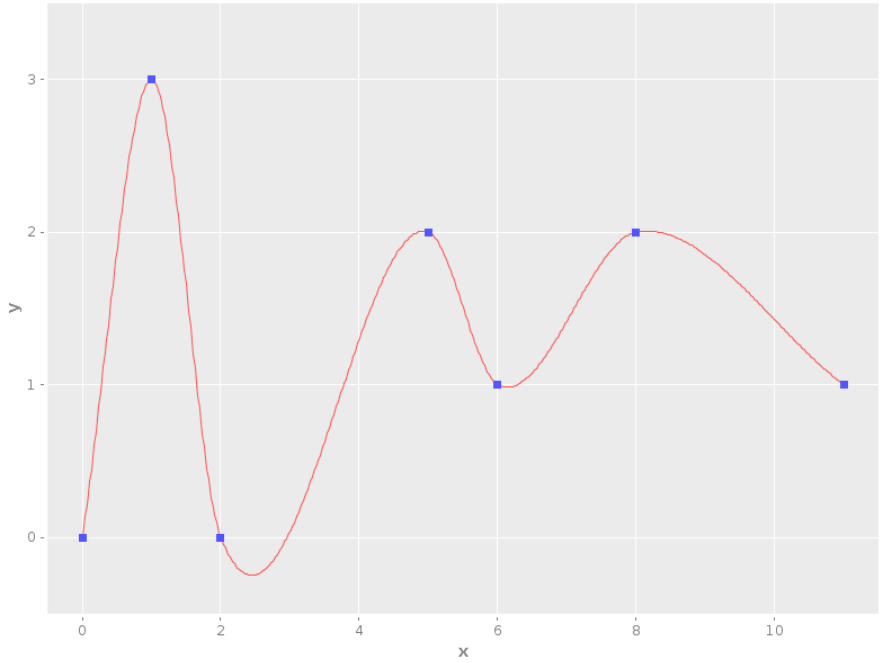


Рисунок 7 – Кубический Эрмитов сплайн

## 5.6 B-сплайн

Пусть задана бесконечная сетка узлов $x_i$:

$$\ldots < x_{-1} < x_0 < x_1 < \ldots$$

Тогда B-сплайном порядка $m \geq 1$ называется функция $N_i^m$, определяемая соотношением:

$$\begin{gathered} N_i^m = V_i^m N_i^{m-1} + (1 - V_{i+1}^m) N_{i+1}^{m-1}, \forall i \in \mathbb{Z} \\ V_i^m(x) = \frac{x - x_i}{x_{i+m} - x_i} \end{gathered} \tag{15}$$

B-сплайны имеют небольшой носитель: $suppN_i^m = [x_i, x_{i+m+1}]$. Таким образом, если мы будем строить функцию, являющуюся линейной комбинацией B-сплайнов, то изменения коэффициента при одном из сплайнов повлияет лишь не небольшую часть функцию, изменение будет локальным.

При переходе от бесконечной сетки $\{x_i\}$ к конечной сетке, заданной на отрезке $[a, b]$ мы сталкиваемся с проблемой, что носитель некоторых базисных функций не входит полностью в $[a, b]$. Чтобы решить эту проблему, добавим виртуальные узлы, равные $a$ слева и $b$ справа. В результате в изменённой сетке

появились кратные узлы, чтобы избежать деления на 0 добавим условие, что значение $V_i^m(x)$ будет равно 0, если знаменатель равен 0.

Рассмотрим приложение B-сплайнов к задаче интерактивного дизайна кривой.

Пусть $\{q_i\}_{i=0}^M$ - контрольные точки на плоскости. Построим обобщение кривой Безье для этих контрольных точек, используя в качестве базиса B-сплайны. В качестве отрезка параметризации возьмём $[a, b] = [0, 1]$. Требуется $M + 1$ линейно независимых на этом отрезке базисных спалйнов порядка $m$. Для этого возьмём сетку равномерно расположенных узлов с добавленными виртуальными узлами:

$$X = \{\underbrace{0, \dots, 0}_{m+1}, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, \underbrace{1, \dots, 1}_{m+1}\} \tag{16}$$

Зададим функцию $\varphi(x)$ в виде линейной комбинации B-сплайнов, построенных по заданным узлам:

$$\varphi(t) = \sum_{i=0}^{M} q_i N_i^m(t) \tag{17}$$

**Временная сложность метода:**

Построение $\varphi$: $O(n)$
Вычисление $\varphi(t)$: $O(m^2)$, где $m$ - 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 :deg

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

## 5.7 Среднеквадратичное приближение

Среднеквадратичное приближение, как и B-сплайны, апроксимирует заданные узлы, а не интерполирует их. Задача дискретного среднеквадратичного приближения для данного набора точек $(x_k, y_k), k = \overline{0, N}$, заключается в построении функции $\varphi$ вида:

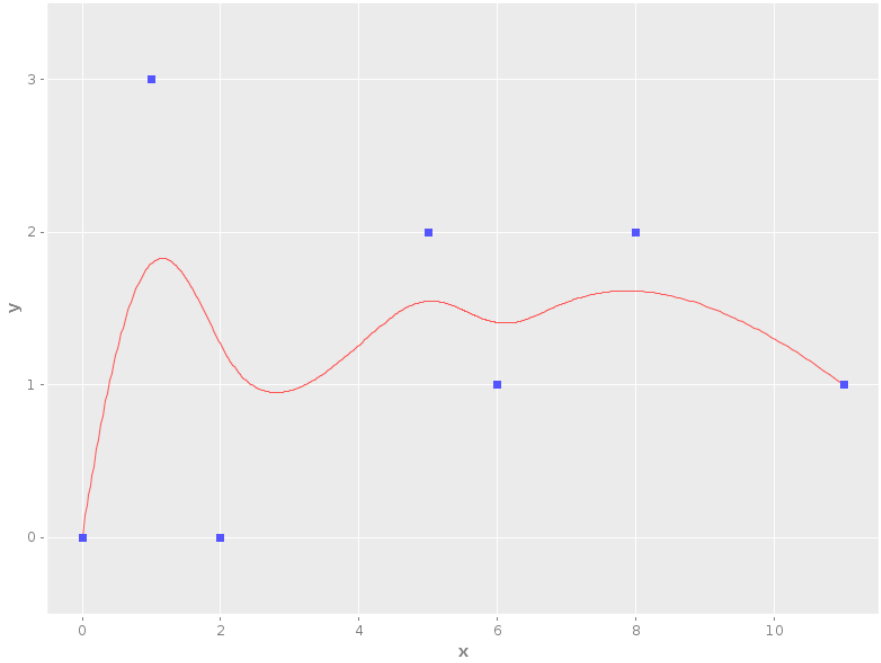


Рисунок 8 – B-сплайн 3 степени

$$\varphi(x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x) \tag{18}$$

для которой выражение

$$\sum_{k=0}^{N} (y_k - \varphi(x_k))^2 \tag{19}$$

принимает минимальное возможное значение. Здесь $\{\varphi_i(x)\}_{i=0}^{n}$ - некоторый заданный базис.

Раскрыв данную формулу получаем квадратичную форму. После минимизации полученной квадратичной формы мы приходим к тому, что коэффициенты $\{\alpha_i\}$ можно найти решив систему линейных уравнений:

$$\sum_{j=0}^{n} \gamma_{lj} \alpha_j = \beta_j, \quad l = \overline{0, n} \tag{20}$$

где:

$$\begin{gathered} \gamma_{ij} = \sum_{k=0}^{N} \varphi_i(x_k) \varphi_j(x_k) \\ \beta_i = \sum_{k=0}^{N} y_k \varphi_i(x_k) \end{gathered} \tag{21}$$

Решая данную систему мы получаем коэффициенты $\{\alpha_i\}$ и таким образом получаем функцию $\varphi$.

**Временная сложность метода:**

Построение $\varphi$: $O(n^2N + n^3)$ - построение матрицы коэффициентов $\gamma_{ij}$ и решение СЛАУ.

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 approximation function
(def lls (interpolate points :linear-least-squares
                             :basis :polynomial :n 3))

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

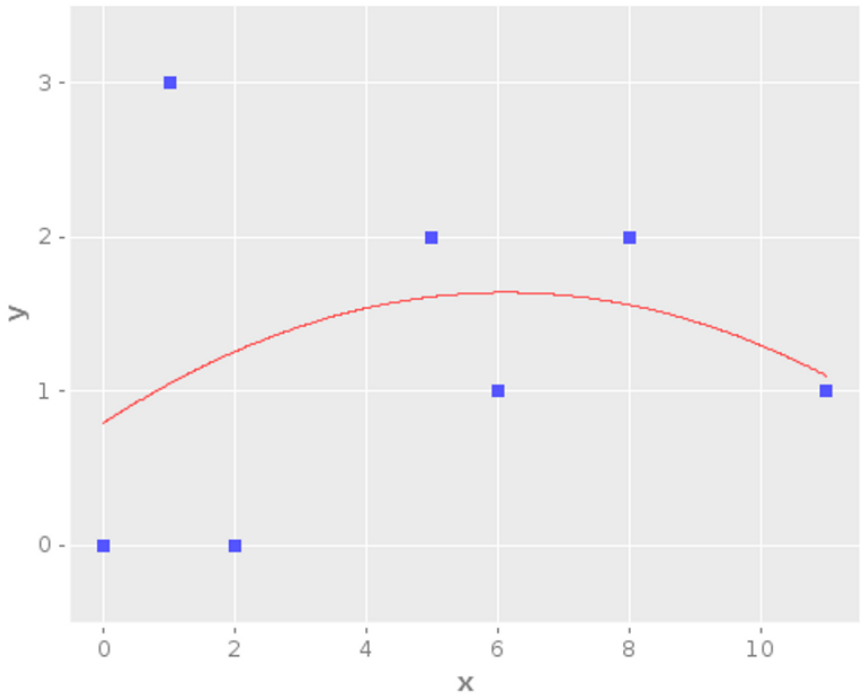


Рисунок 9 – Среднеквадратичное приближение по базису $1, x, x^2$

# 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. Методы реализованы на основе лекций Б.В. Фалейчика [1] [2].

## 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

$$\begin{aligned} \varphi(x, y_j) &= z_{i,j} + (z_{i+1,j} - z_{i,j})\frac{x - x_i}{x_{i+1} - x_i} \\ \varphi(x, y_{j+1}) &= z_{i,j+1} + (z_{i+1,j+1} - z_{i,j+1})\frac{x - x_i}{x_{i+1} - x_i} \\ \varphi(x, y) &= \varphi(x, y_j) + (\varphi(x, y_{j+1}) - \varphi(x, y_j))\frac{y - y_j}{y_{j+1} - y_j} \end{aligned} \tag{22}$$

**Временная сложность метода:**

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

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

**Пример:**

```
; define grid
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

; build interpolation function
(interpolate-grid grid :bilinear)
```

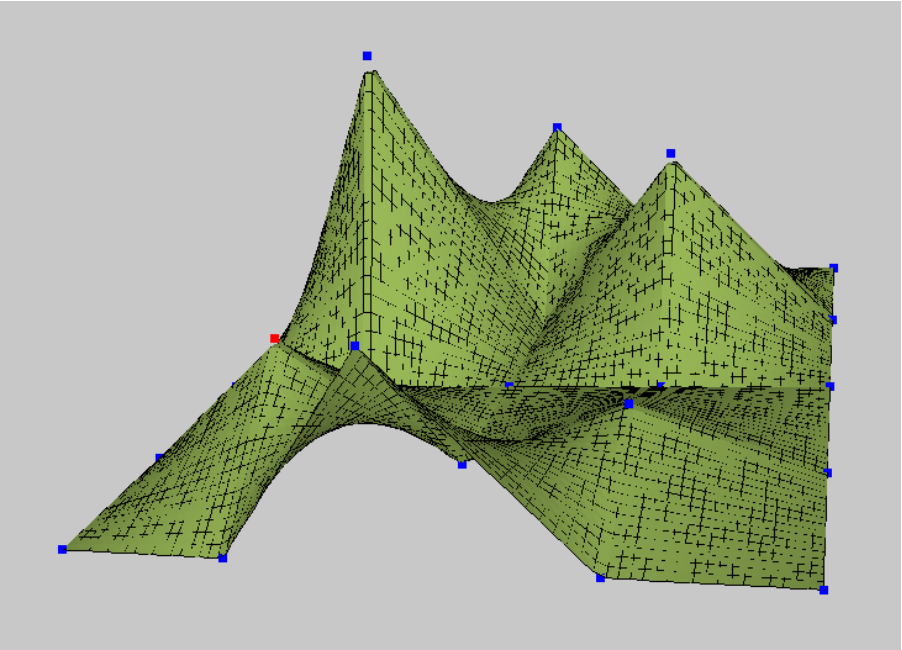

Рисунок 10 – Билинейная интерполяция

## 6.2 Полиномиальная интерполяция

Полиномиальная интерполяция функции 2 переменных была реализована по алгоритму, предложенному Varsamis и Karampetaki [4]. Матрица разделённых разностей $P$ вычисляется по следующей формуле:

$$P_{i,j}^{(k)} = \begin{cases} f(x_i, y_j) & \text{if } k = 0 \\ \dfrac{P_{i,j}^{(k-1)} - P_{i-1,j}^{(k-1)}}{x_i - x_{i-k}} & \text{if } (j \leq k-1 \wedge i > k-1) \\ \dfrac{P_{i,j}^{(k-1)} - P_{i,j-1}^{(k-1)}}{y_j - y_{j-k}} & \text{if } (i \leq k-1 \wedge j > k-1) \\ \dfrac{P_{i,j}^{(k-1)} + P_{i-1,j-1}^{(k-1)} - P_{i,j-1}^{(k-1)} - P_{i-1,j}^{(k-1)}}{(x_i - x_{i-k})(y_j - y_{j-k})} & \text{if } (i > k-1 \wedge j > k-1) \end{cases} \tag{23}$$

Интерполяционный многочлен вычисляется по формуле:

$$\varphi(x, y) = Y^T P X \tag{24}$$

где

$$X = \begin{pmatrix} 1 \\ x - x_0 \\ \vdots \\ (x - x_0)(x - x_1) \dots (x - x_{n-1}) \end{pmatrix}$$
$$Y = \begin{pmatrix} 1 \\ y - y_0 \\ \vdots \\ (y - y_0)(y - y_1) \dots (y - y_{m-1}) \end{pmatrix} \quad (25)$$

**Временная сложность метода:**

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

Вычисление $\varphi(x, y)$: $O(nm)$

**Пример:**

```
; define grid
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

; build interpolation function
(interpolate-grid grid :polynomial)
```

## 6.3 Бикубический сплайн

Бикубический сплайн является аналогом кубического сплайна. Будем искать функцию $\varphi(x, y)$ будем искать в виде:

$$\varphi(x, y) = \alpha_i(y) + \beta_i(y)(x - x_i) + \frac{\delta_i(y)}{y}(x - x_i)^2 + \frac{\delta_i(y)}{6}(x - x_i)^3, x \in [x_{i-1}, x_i] \quad (26)$$

Здесь коэффициенты $\alpha_i(y), \beta_i(y), \delta_i(y), \gamma_i(y), i = \overline{1, n}$ являются одномерными кубическими сплайнами. Для построения функции $\varphi$ необходимо построить данные одномерные сплайны. Но для их построения не хватает значений $\alpha_i(y), \beta_i(y), \delta_i(y), \gamma_i(y)$ в узлах $y_j, j = \overline{1, m}$.

Чтобы решить эту проблему зафиксируем $y = y_0$ в (26) и получаем одномерный кубический сплайн, вычисляя коэффициенты которого находим $\alpha_i(y_0), \beta_i(y$ $\overline{1, n}$. Таким образом фиксируя $y$ можно получить значения коэффициентов по

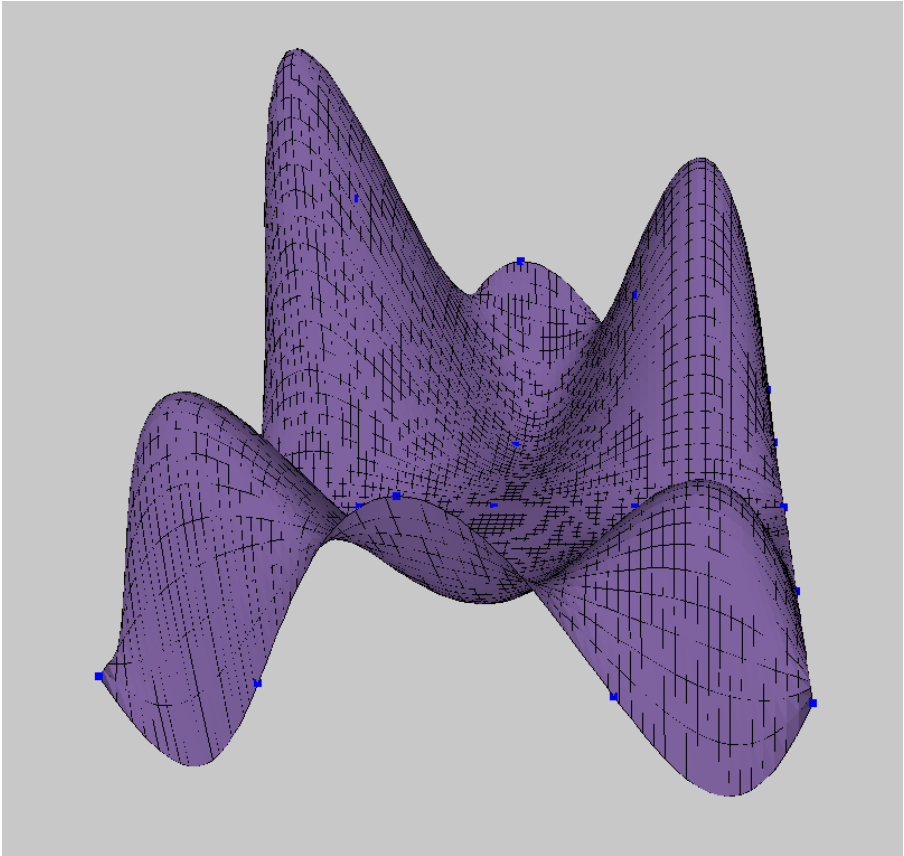

Рисунок 11 – Полиномиальная интерполяция

всех остальных узлах. Далее по значениям коэффициентов в узлах строим сплайны $\alpha_i(y), \beta_i(y), \delta_i(y), \gamma_i(y)$.

Для вычисления значения $\varphi$ в точке $(x, y)$ необходимо по $x$ определить отрезок $[x_{i-1}, x_i]$, которому принадлежит $x$. Далее требуется посчитать значения сплайнов $\alpha_i, \beta_i, \delta_i, \gamma_i$ в точке $y$. Полученные коэффициенты использовать в формуле (26) для получения конечного результата.

**Временная сложность метода:**

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

**Пример:**

```
; define grid
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

; build interpolation function
(interpolate-grid grid :bicubic)
```

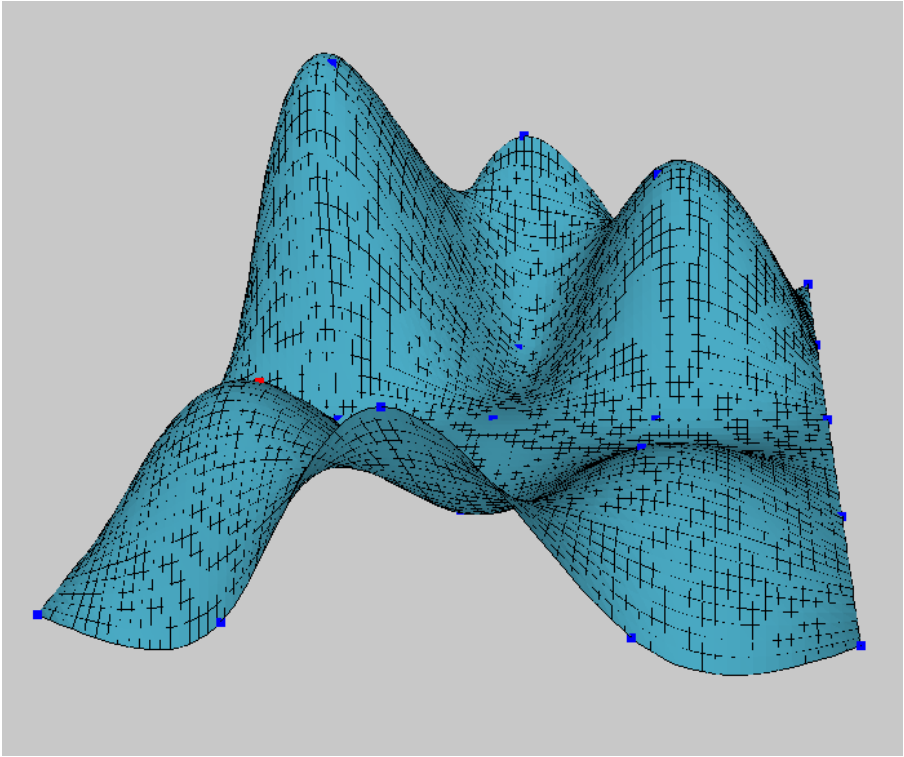

Рисунок 12 – Бикубический сплайн

## 6.4 Средневадратическое приближение

Среднеквадратическое приближение функции 2 переменных очень сильно похоже на среднеквадратическое приближения функции 1 переменной. Задаётся базис функций 2 переменных и вычисляются коэффициенты $\{\alpha_i\}$ при которых сумма расстояний от полученной поверхности до заданных точек минимальна. Положим $N \times M$ - размеры сетки, $n$ - число базисных функций. Изменяются только формулы вычисления $\gamma_{ij}$ и $\beta_i$:

$$\begin{aligned} \gamma_{ij} &= \sum_{k=0}^{N} \sum_{l=0}^{M} \varphi_i(x_k, y_l)\varphi_j(x_k, y_l) \\ \beta_i &= \sum_{k=0}^{N} \sum_{l=0}^{M} y_{kl}\varphi_i(x_k, y_l) \end{aligned} \tag{27}$$

**Временная сложность метода:**

Построение $\varphi$: $O(n^2NM + n^3)$ - построение матрицы коэффициентов $\gamma_{ij}$ и решение СЛАУ.

Вычисление $\varphi(x)$: $O(n)$.

**Пример:**

```
; define grid
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

; basis consists of 7 funtions: 1, x, y, sin x, cos x, sin
(defn basis [x y] [1 x y
                   (Math/sin x) (Math/cos x)
                   (Math/sin y) (Math/cos y)])

; build interpolation function
(interpolate-grid grid :linear-least-squares
                       :basis basis)
```

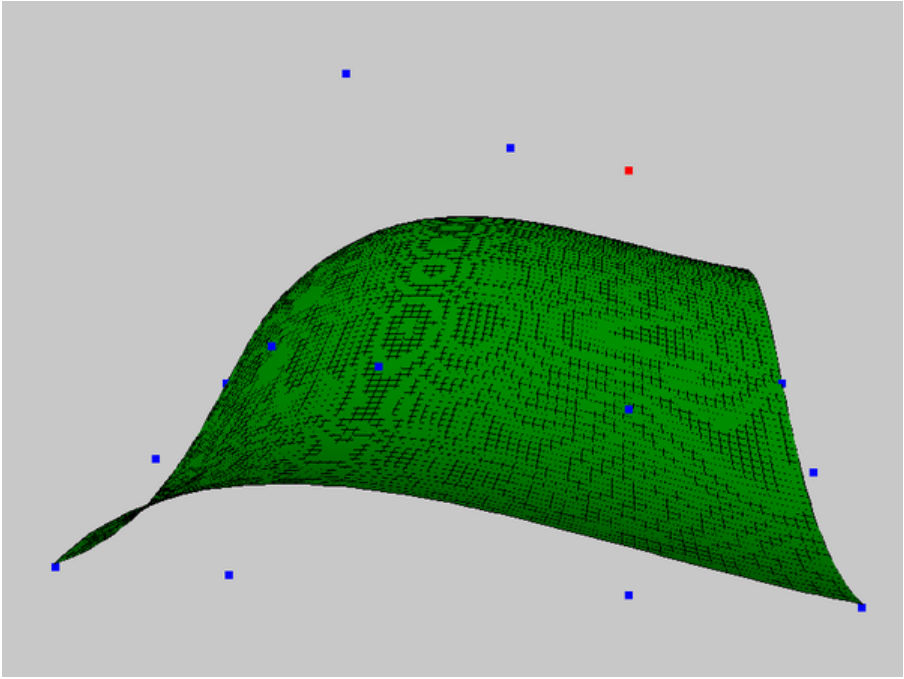

Рисунок 13 – Среднеквадратичное приближение по базису $1, x, y, sin(x), cos(x), sin(y), cos(y)$

## 6.5 B-сплайновая поверхность

B-сплайновая поверхность для функции двух переменных строится используя тензорное произведение B-сплайнов для одномерного случая. Формула $\varphi(u, v)$ выглядит следующим образом:

$$\varphi(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} q_{ij} N_i^d(u) N_j^d(v) \tag{28}$$

**Временная сложность метода:**

Построение $\varphi$: $O(nm)$
Вычисление $\varphi(u,v)$: $O(d^2)$, где $d$ - степень одномерных сплайнов.

**Пример:**

```
; define grid
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

; build interpolation function
(interpolate-grid grid :b-surface)
```

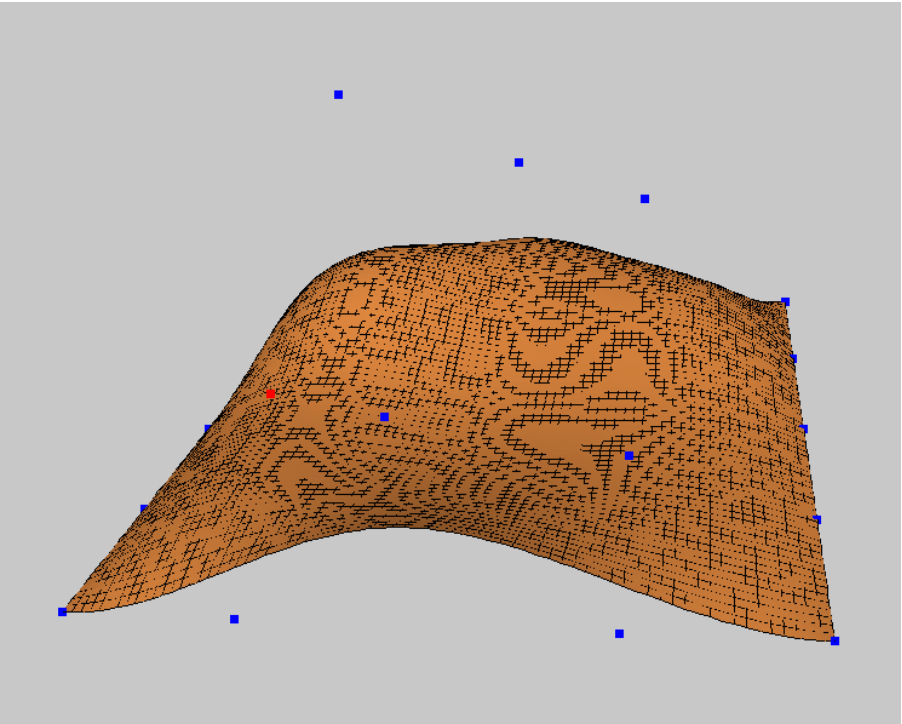

Рисунок 14 – B-сплайновая поверхность

# 7 Интерфейс библиотеки

Разработанная библиотека предоставляет 3 функции:

```
(interpolate points type & options)

(interpolate-parametric points type & options)

(interpolate-grid grid type & options)
```

Все вышеперечисленные функции имеют сходную сигнатуру. Они принимают следующие параметры:

- `points` или `grid` - набор точек или сетка, которые требуется интерполировать;
- `type` - тип интерполяции;
- `options` - дополнительные опции, специфичные для каждой функции и типа интерполяции;

Все функции возвращают новую функцию - интерполирующую функцию, с помощью которой находятся значения в интересующих точках.

## 7.1 Функция `interpolate`

Данная функция позволяет построить интерполирующую функцию $f(x) = y$ по заданному набору точек $(x_i, y_i)$. Точки могут задаваться в любом порядке, перед использованием они будут автоматически отсортированы по координате $x$.

Данная функция поддерживает следующие типы интерполяции: линейная, полиномиальная, кубический сплайн, кубический Эрмитов сплайн, среднеквадратичное приближение. Сооветствующие аргументы для параметра `type`: `:linear`, `:polynomial`, `:cubic`, `:cubic-hermite`, `:linear-least-squares`.

**Дополнительные опции:**

- `:boundaries` - граничные условия для кубического сплайна. Поддерживаются 2 вида условий: естественные (`:natural`) и замкнутые (`:closed`
- `:derivatives` - производные для кубического Эрмитова сплайна;
- `:basis`, `:n`, `:degree` - опции настройки среднеквадратичного приближения. Позволяют задать базис, произвольный или один из 2 встроенных (полиномиальный и B-сплайны); число функций в базисе, если выбран встроенный; степень B-сплайнов;

**Пример:**

Построение кубического сплайна с замкнутыми граничными условиями по точкам $(0,0),(1,3),(2,0),(4,2)$:

```
(def points [[0 0] [1 3] [2 0] [4 2]])

(def cubic (interpolate points :cubic :boundaries :closed))

(cubic 0) ; 0.0
(cubic 1) ; 3.0
(cubic 3) ; -1.2380952380952381
```

## 7.2 Функция `interpolate-parametric`

Данная функция строит интерполирующую параметрическую функцию $f(t) = (x^1, x^2, \ldots, x^n)$ определённую на отрезке $[0,1]$ и проходящую через заданные пользователем точки $(x_i^1, x_i^2, \ldots, x_i^n)$. Особенность данной функции заключается в том, что пользователь не задаёт узлы $t_i$, соответствующие каждой точке, данные узлы они выбираются алгоритмом. Эта особенность используется в полиномиальной интерполяции, для которой очень важен выбор хороших узлов: для параметрической полиномиальной интерполяции используются узлы Чебышева.

Данная функция поддерживает такие же типы интерполяции, как `interpolate`, а также B-сплайн (`:b-spline`).

**Дополнительные опции:**

Поддерживается тот же набор дополнительных опций, которые поддерживает функция `interpolate`, а также добавлены 2 новые опции:

- `:degree` - задаёт степень B-сплайна, если выбрана аппроксимация B-сплайнами;
- `:range` - отрезок, на котором будет определена интерполирующая функция, по умолчанию это $[0,1]$;

**Пример:**

Построение параметрической интерполирующей функции на отрезке $[-1,1]$ по точкам $(0,0),(1,3),(2,0),(4,2)$:

```
(def points [[0 0] [1 3] [2 0] [4 2]])
```

```
(def linear (interpolate-parametric points :linear
:range [-1 1]))

(linear -1) ; (0.0 0.0)
(linear  1) ; (4.0 2.0)
(linear  0) ; (1.5 1.5)
```

## 7.3 Функция `interpolate-grid`

Данная функция строит интерполирующую функцию 2 переменных $f(x, y) = z$ по прямоугольной сетке точек. По умолчанию, пользователь задаёт сетку значений $\{z_{ij}\}, i = \overline{1, n}, j = \overline{1, m}$, далее автоматически строится равномерная сетка узлов размера $n \times m$ заданная на области $[0, 1] \times [0, 1]$. В конце полученная сетка узлов и значений интерполируется и строится интерполирующая функция.

Функция поддерживает следующие типы интерполяций: билинейная (`:bilinear`), полиномиальная (`:polynomial`), бикубический сплайн (`:bicubi` бикубический Эрмитов сплайн (`:bicubic-hermite`), среднеквадратичное приближение (`:linear-least-squares`), B-сплайновая поверхность (`:b-surface`).

**Дополнительные опции:**

- `:boundaries` - граничные условия для бикубического сплайна. Поддерживаются 2 вида условий: естественные (`:natural`) и замкнутые (`:closed`);
- `:basis`, `:n` - опции настройки среднеквадратичного приближения. Позволяют задать базис, произвольный или встроенный полиномиальныйи число функций в базисе, если выбран встроенный;
- `:degree` - задаёт степень B-сплайна, если выбрана аппроксимация B-сплайновой поверхностью;
- `:x-range`, `:y-range` - область, на которой задана интерполирующая функция, по умолчанию это $[0, 1] \times [0, 1]$;
- `:xs`, `:ys` - явное задание узлов интерполяции;

**Пример:**

Интерполирование сетки значений $\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$ с использованием билинейной интерполяции:

```
(def grid [[0 1 2]
           [3 4 5]
           [6 7 8]])

(def bilinear (interpolate-grid grid :bilinear))

(bilinear    0   0) ; 0.0
(bilinear    1   1) ; 8.0
(bilinear  0.5 0.5) ; 4.0
(bilinear 0.25   1) ; 6.5
```

# 8 Заключение

Разработана библиотека для интерполяции и апроксимации функций, реализующая популярные методы, такие как линейная интерполяция, полиномиальная интерполяций, кубический и кубический Эрмитов сплайны, B-сплайны и среднеквадратичное приближение. Интерфейс библиотеки был реализован с упором на простоту и не требует особой математической подготовки, таким образом библиотека может быть быть использована широким классом пользователей, как для каких-либо математических вычислений, так и для графических и визуальных целей. В то же время библиотека предоставляет небольшой набор дополнительных опций, которые могут быть использованы для более точной настройки нужных методов под конкретные требования.

На данный момент (25 мая 2013) код библиотеки добавлен в основную ветку проекта Incanter, и библиотека будет доступна всем желающим после выхода новой версии Incanter: 1.5.

# 9 Литература


[1] Б.В.Фалейчик, Методы численного анализа // конспект лекций, Часть 1, 2010

[2] Б.В.Фалейчик, Методы вычислений // курс лекций, 2012

[3] Jean-Paul Berrut, Lloyd N. Trefethen, Barycentric Lagrange Interpolation // SIAM REVIEW Vol. 46, No. 3, pp. 501–517, 2004

[4] Dimitris N. Varsamis, Nicholas P. Karampetaki, On the Newton multivariate polynomial interpolation with applications // 7th International Workshop on Multidimensional Systems, 2011

[5] Neidinger R. D., Multivariable Interpolating Polynomials in NewtonForms // Joint Mathematics Meetings 2009, 2009.

[6] Hoi Sub Kim, The computation of multivariate B-splines with application to surface approximations // J. Ksiam Vol.3, No. 1, 1999

[7] Clojure [Electronic resource]. - http://clojure.org

[8] Incanter [Electronic resource]. - http://incanter.org