Оглавление

1 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.

1.1 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:

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

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

Временная сложность метода:

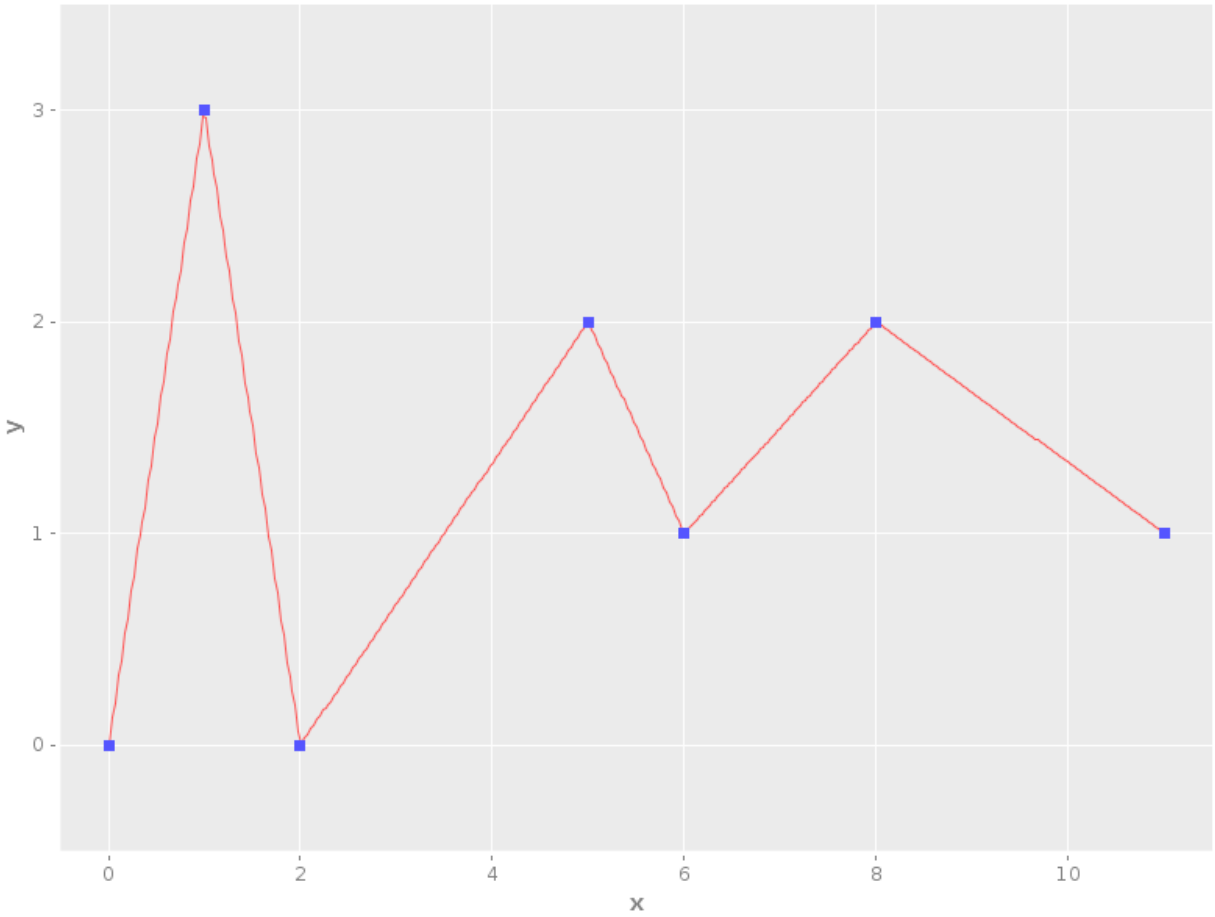
Построение φ: $O(n \log n)$ - требуется отсортировать все узлы.
Вычисление $\varphi(x)$: $O(\log n)$ - поиск соответствующего узла.

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))
```



1.2 Полиномиальная интерполяция в форме Ньютона

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

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

Результатом алгоритма является многочлен, функция дифференцируемая 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:

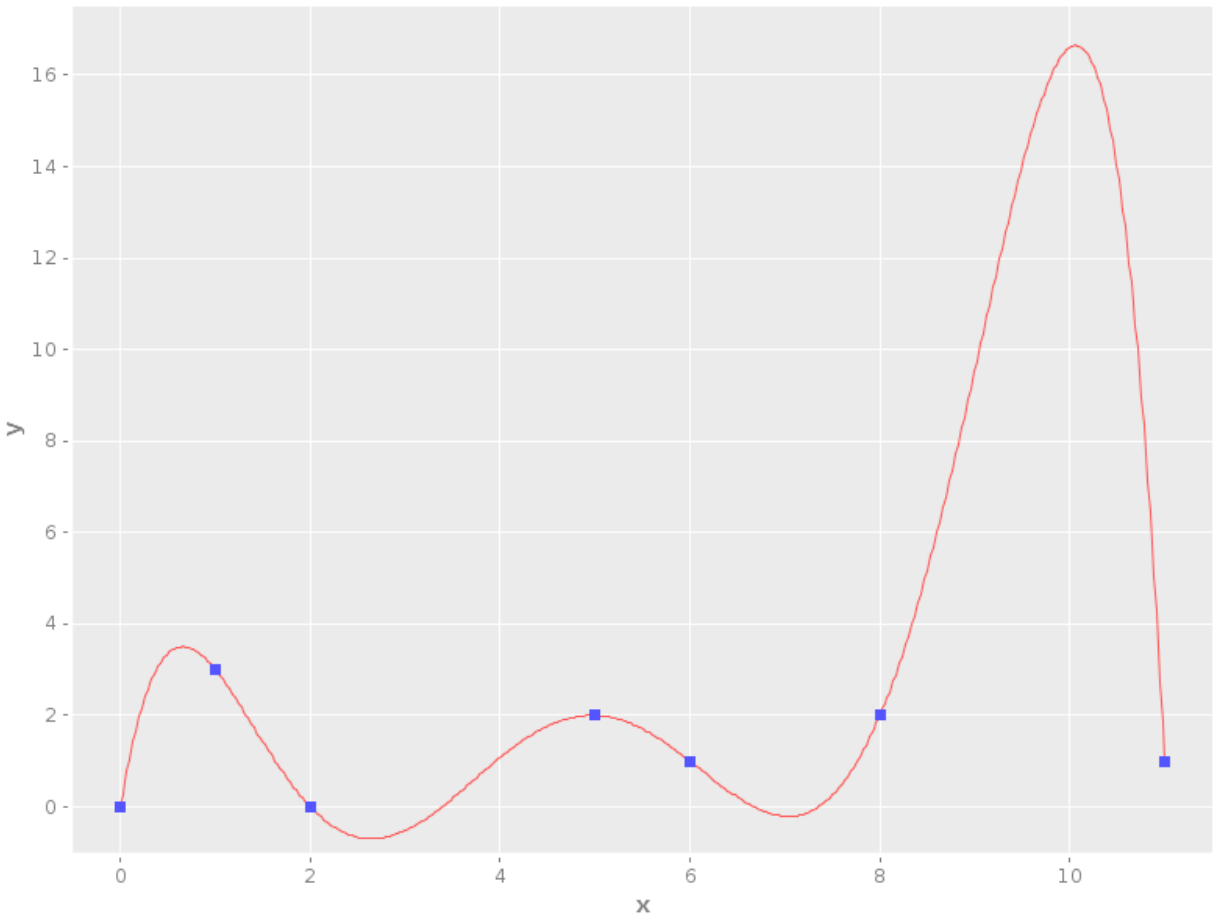
Построение φ: $O(n^2)$
Вычисление $\varphi(x)$: $O(n)$

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))
```



1.3 Полиномиальная интерполяция в барицентрической форме

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

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

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

Вычисление $\varphi(x)$: $O(n)$

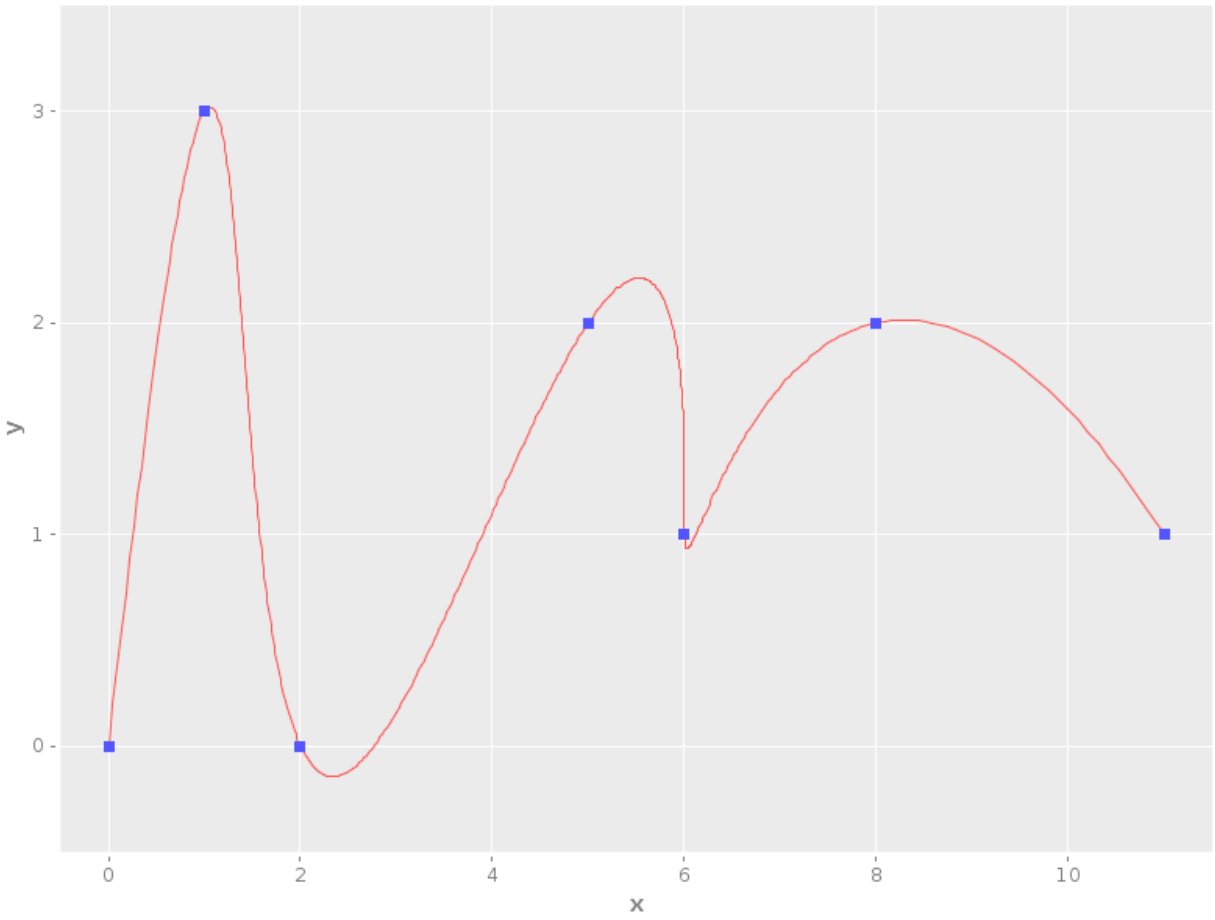
Пример:

```
; 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))
```



1.4 Кубический сплайн

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

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

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

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

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

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

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

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

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

Временная сложность метода:

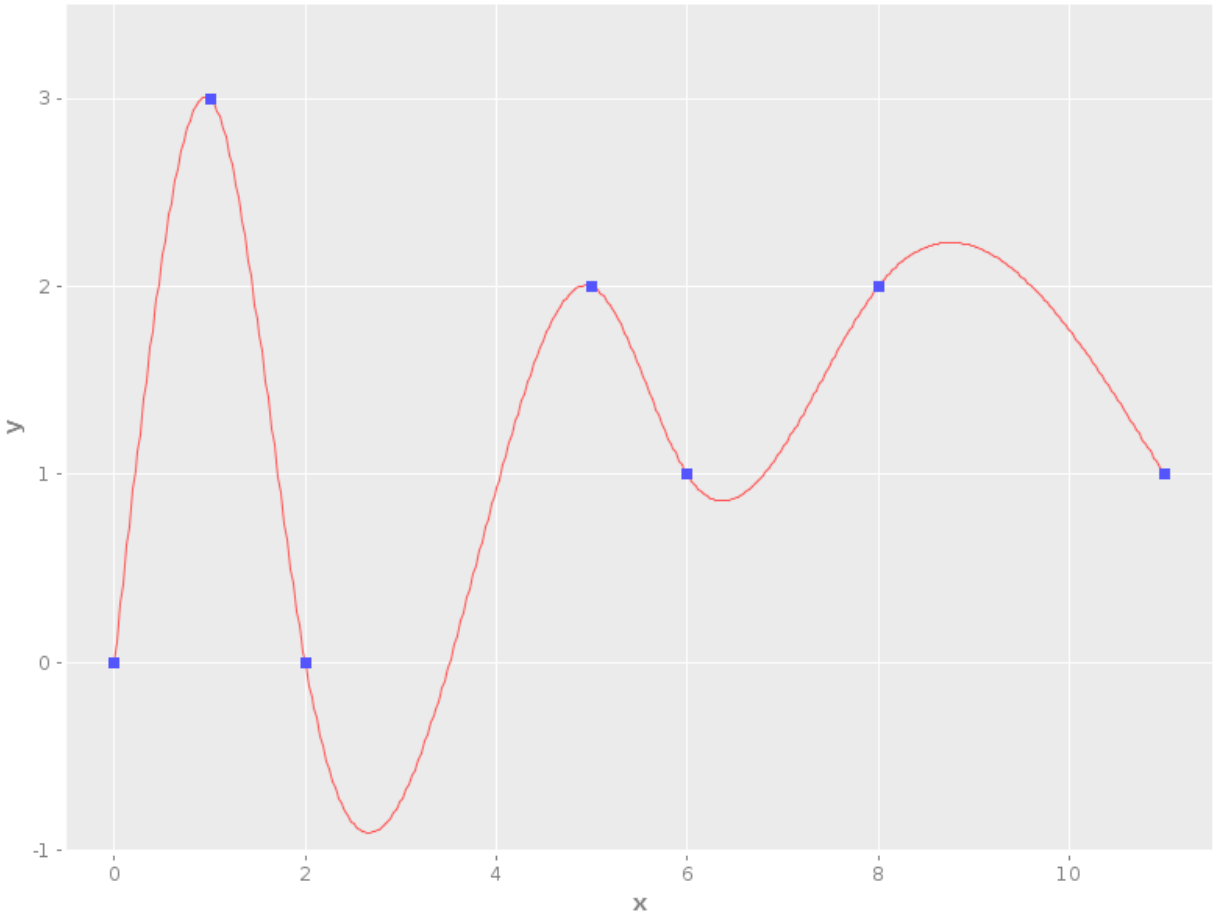
Построение φ: $O(n)$
Вычисление $\varphi(x)$: $O(\log n)$ - поиск соответствующего промежутка.

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 :boundaries :closed))
```

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



1.5 Кубический эрмитов сплайн

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

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

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

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

Где

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

Первые производные функции в узлах x_i приближались используя конечные разности:

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

Временная сложность метода:

Построение φ: $O(n)$

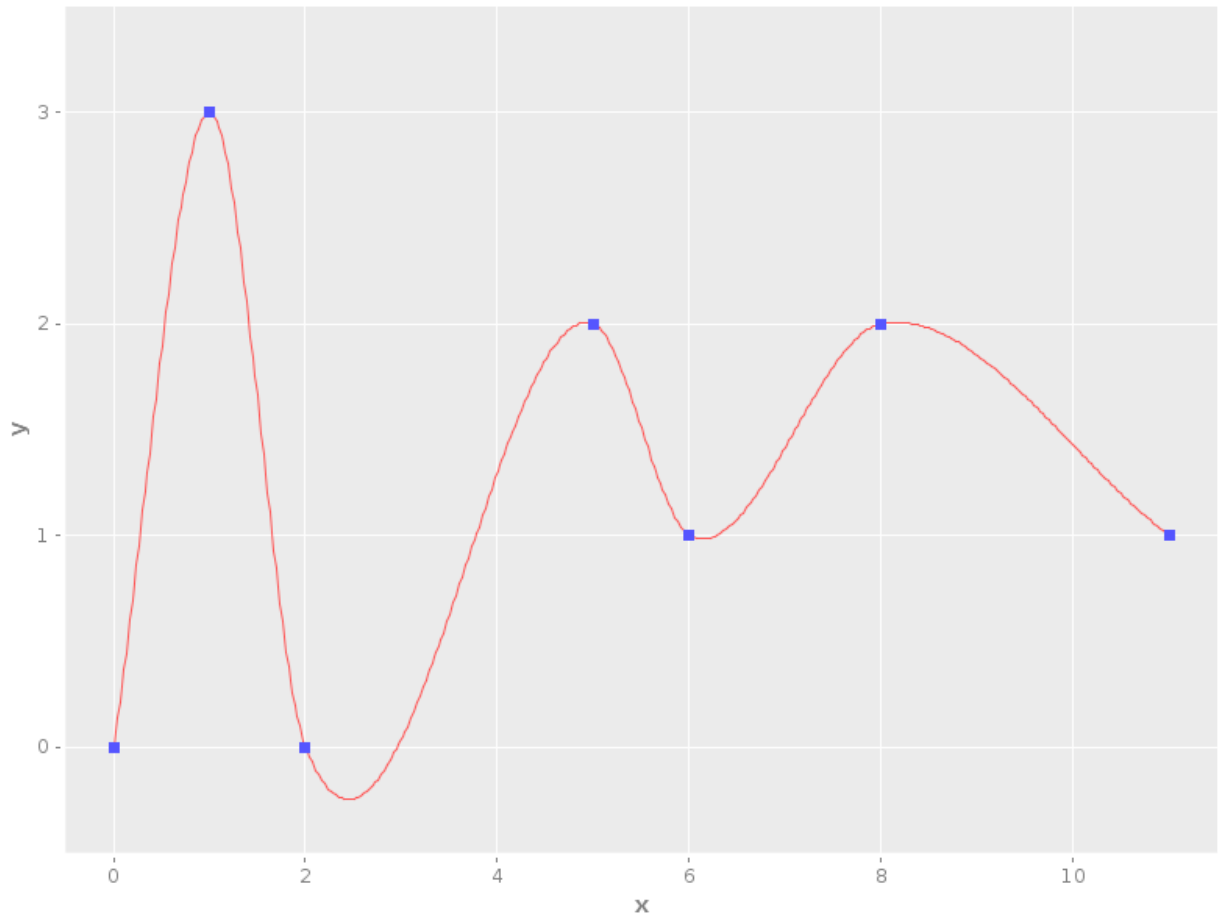
Вычисление $\varphi(x)$: $O(\log n)$ - поиск соответствующего промежутка.

Пример:

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

; build interpolation function
(def hermite (interpolate points :cubic-hermite))

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



1.6 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:

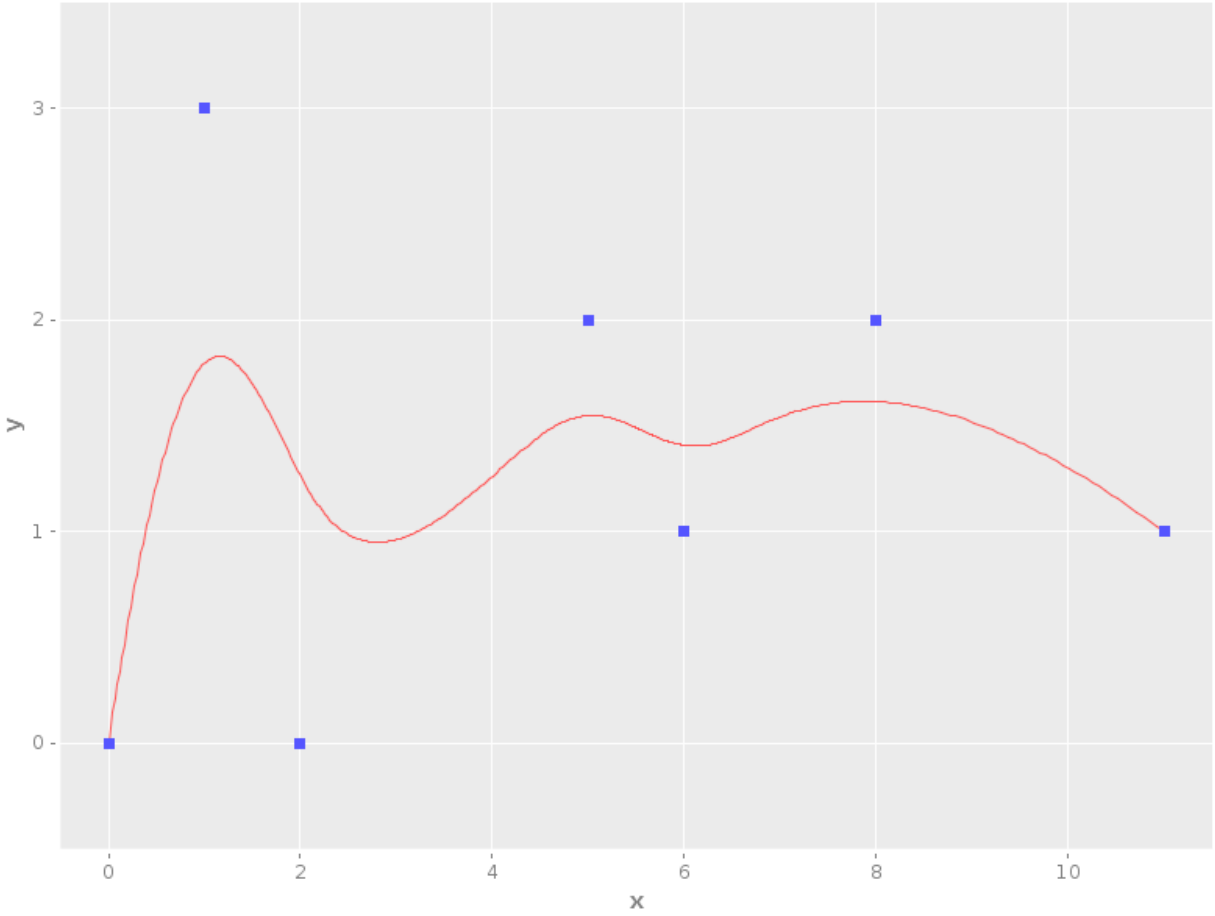
Построение φ: $O(n)$
Вычисление $\varphi(t)$: $O(m^2)$, где m - степень сплайна

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 :degree 3))
```

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



1.7 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 φ вида:

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

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 φ.

Временная сложность метода:

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

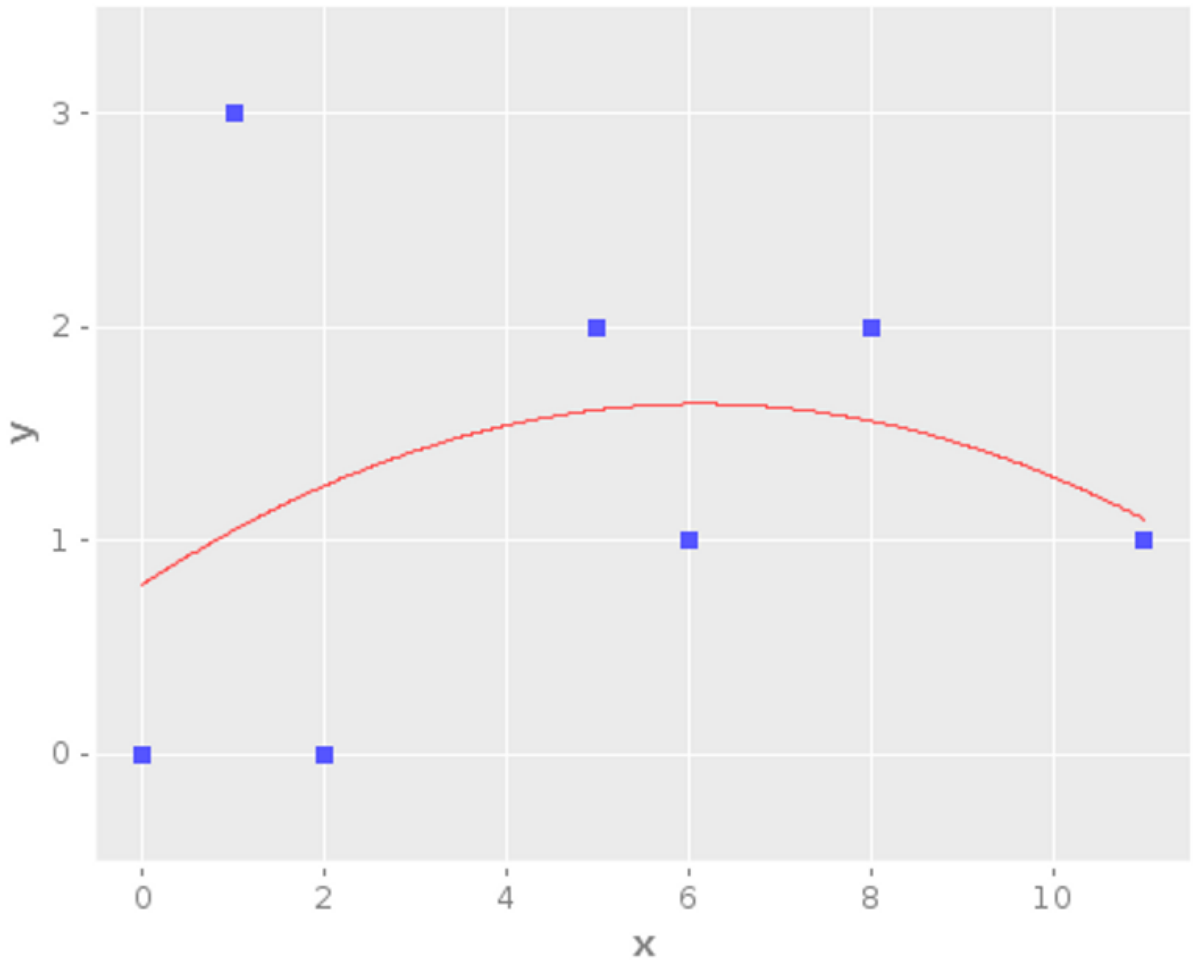
Вычисление $\varphi(x)$: $O(n)$.

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))
```


3
2
1
0
y
0
2
4
6
8
10
x