

Оглавление

| | | |
|-----|--|----|
| 1 | Интерполяция функции одной переменной | 2 |
| 1.1 | Линейная интерполяция | 2 |
| 1.2 | Полиномиальная интерполяция в форме Ньютона . . | 4 |
| 1.3 | Полиномиальная интерполяция в барицентрической форме | 5 |
| 1.4 | Кубический сплайн | 7 |
| 1.5 | Кубический эрмитов сплайн | 10 |
| 1.6 | В-сплайн | 11 |

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 $f: [a, b] \rightarrow \mathbb{R}$: 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 $\varphi: [a, b] \rightarrow \mathbb{R}$, 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 В-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:

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

Преимущества:

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

Недостатки:

- φ не является непрерывно-дифференцируемой;

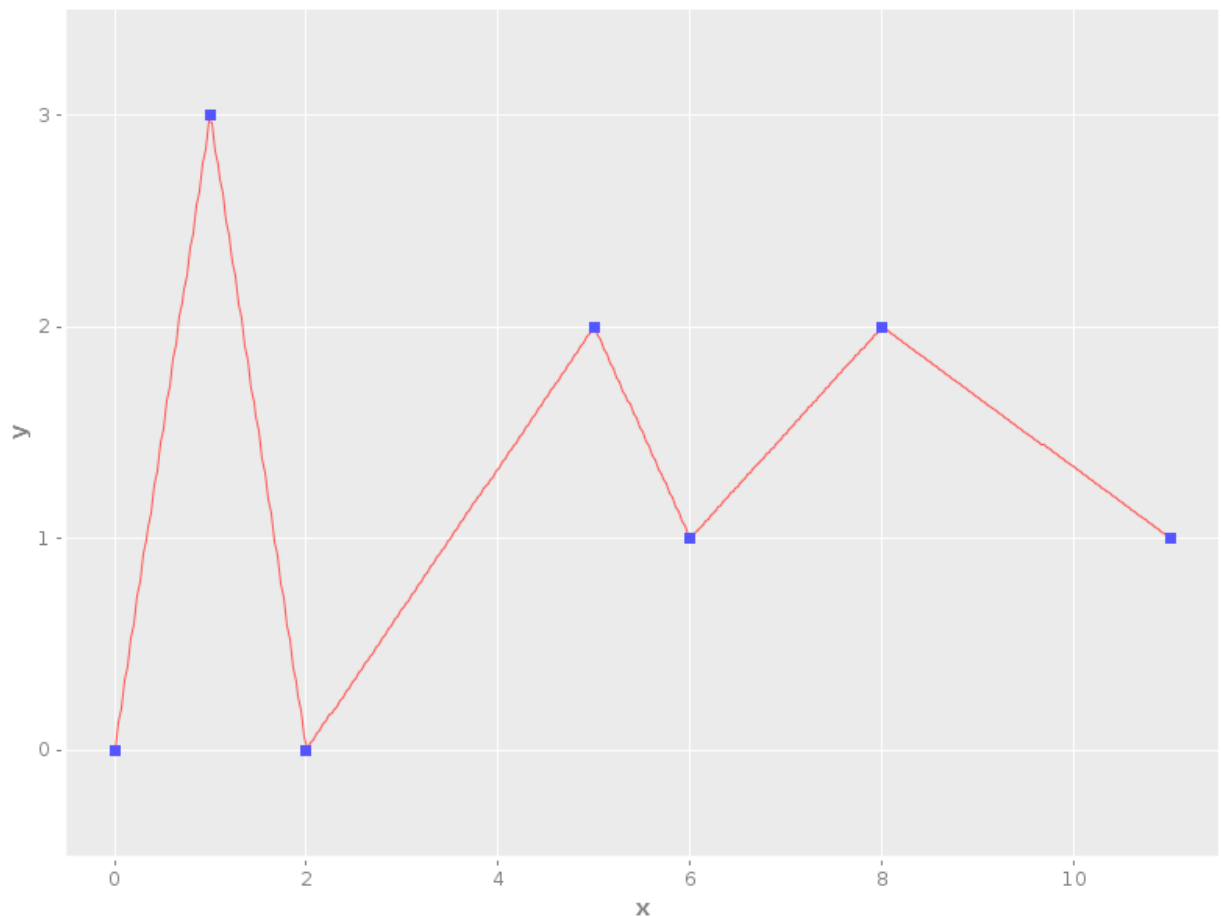
Временная сложность метода:

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 разно-

стями. Вычисления производятся по следующим формулам:

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

Преимущества:

- φ имеет производную любого порядка;

Недостатки:

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

Временная сложность метода:

Построение φ : $O(n^2)$

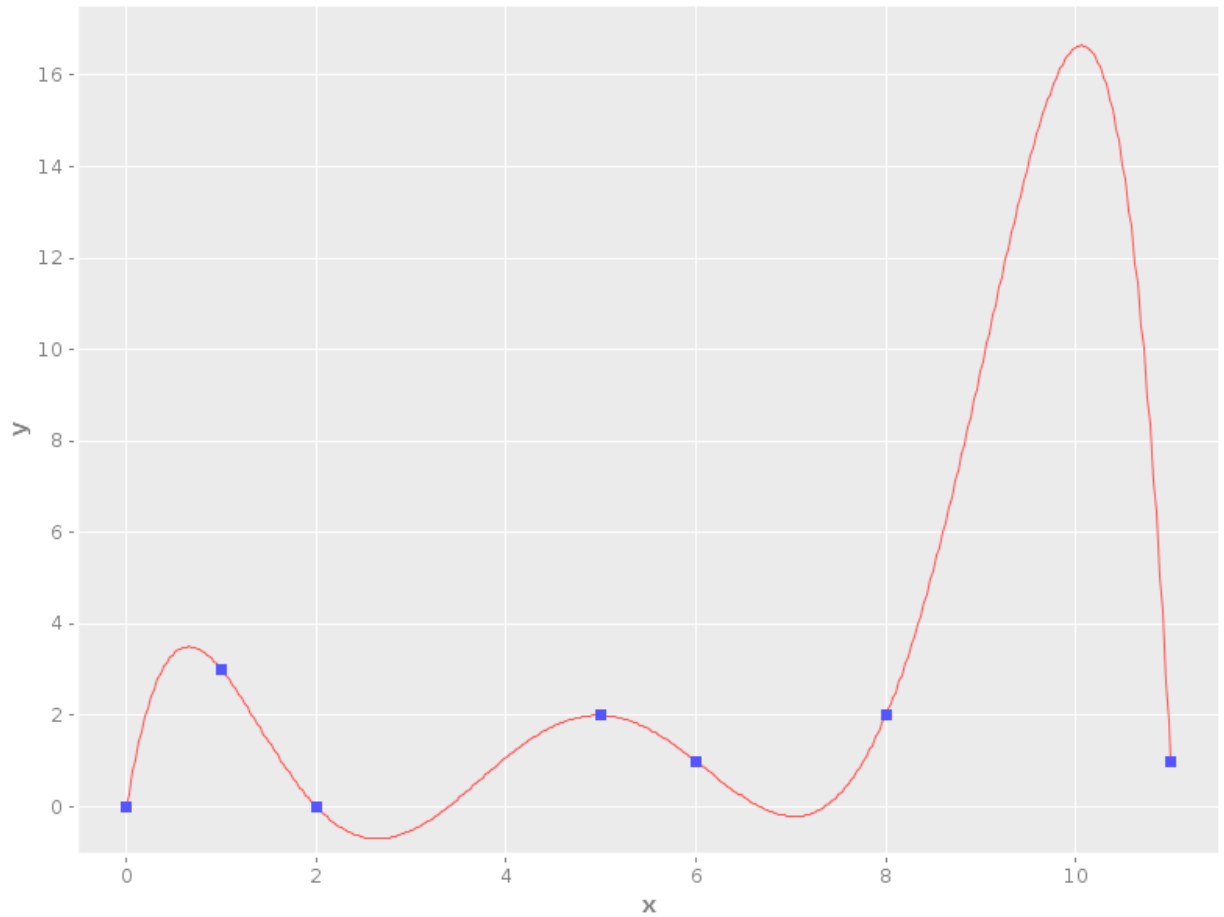
Вычисление $\varphi(x)$: $O(n)$

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))
```



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:

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

где

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

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 построения 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:

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

то коэффициенты v_i имеют следующий вид:

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

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 $\varphi(x): O(n)$

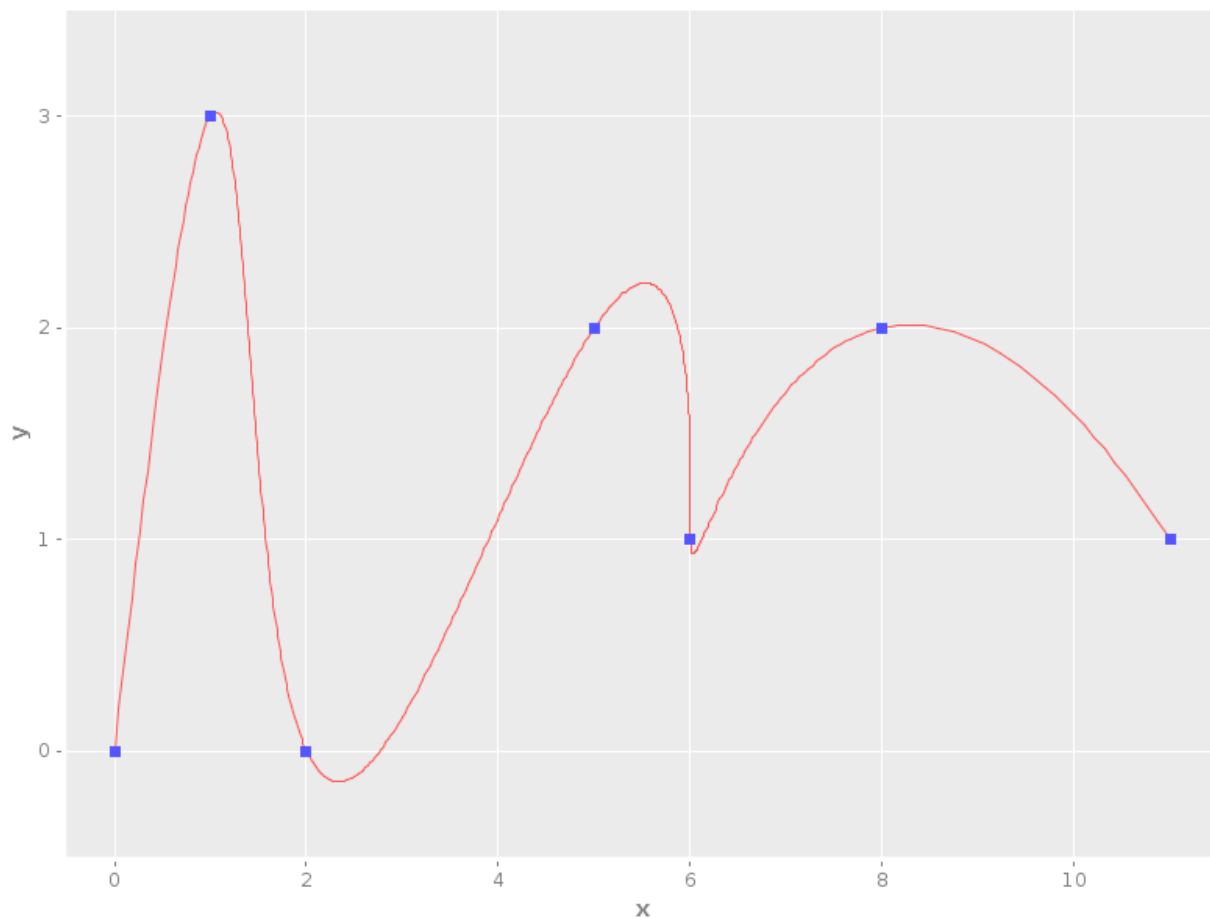
Пример:

```
; 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))



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 $\{\alpha_i\}$, $\{\beta_i\}$, $\{\delta_i\}$:

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

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

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

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:

```

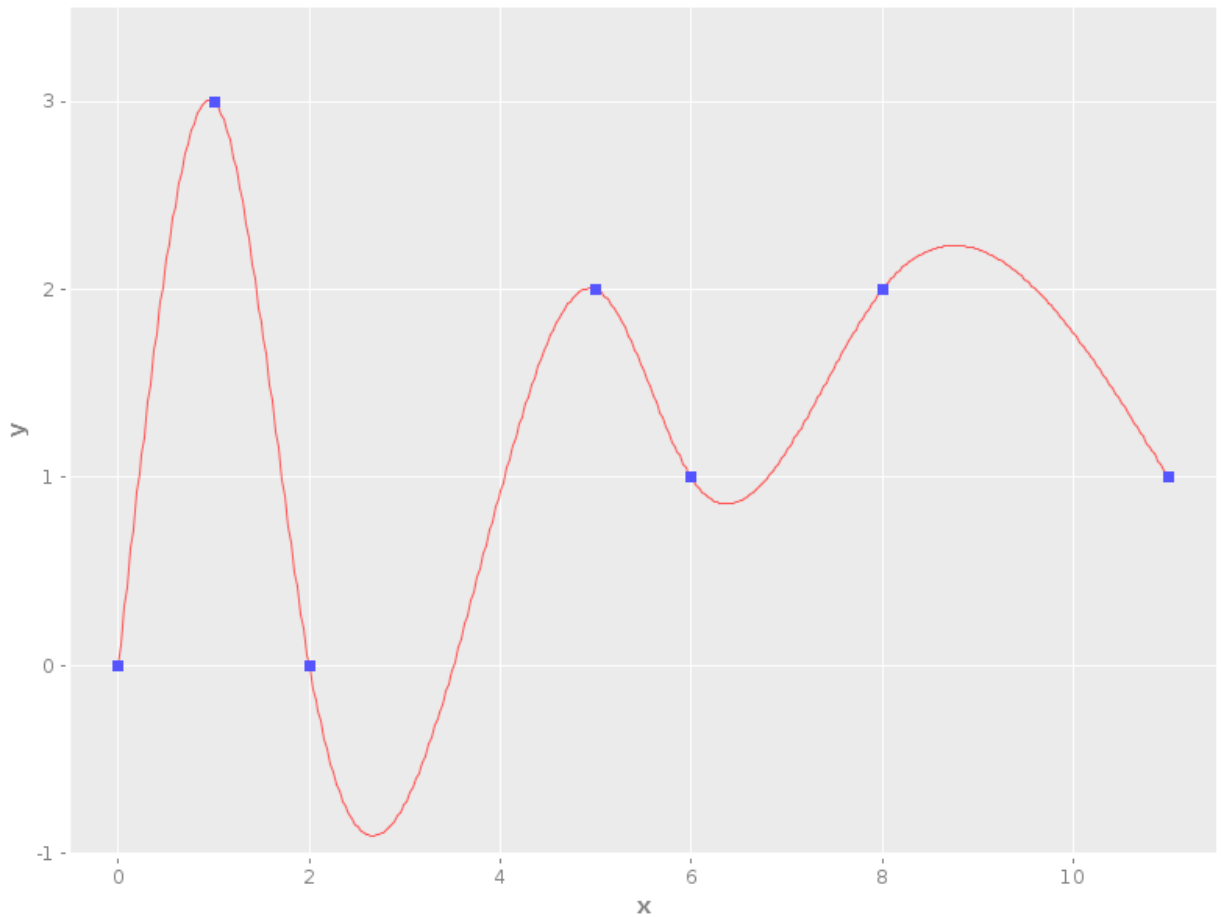
; define points
(def points [[0 0] [1 3] [2 0] [5 2] [6 1] [8 2] [11 1]])

; build interpolation function
(def cubic (interpolate points :cubic-spline :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

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

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

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

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:

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

Временная сложность метода:

Построение φ : $O(n)$

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

Пример:

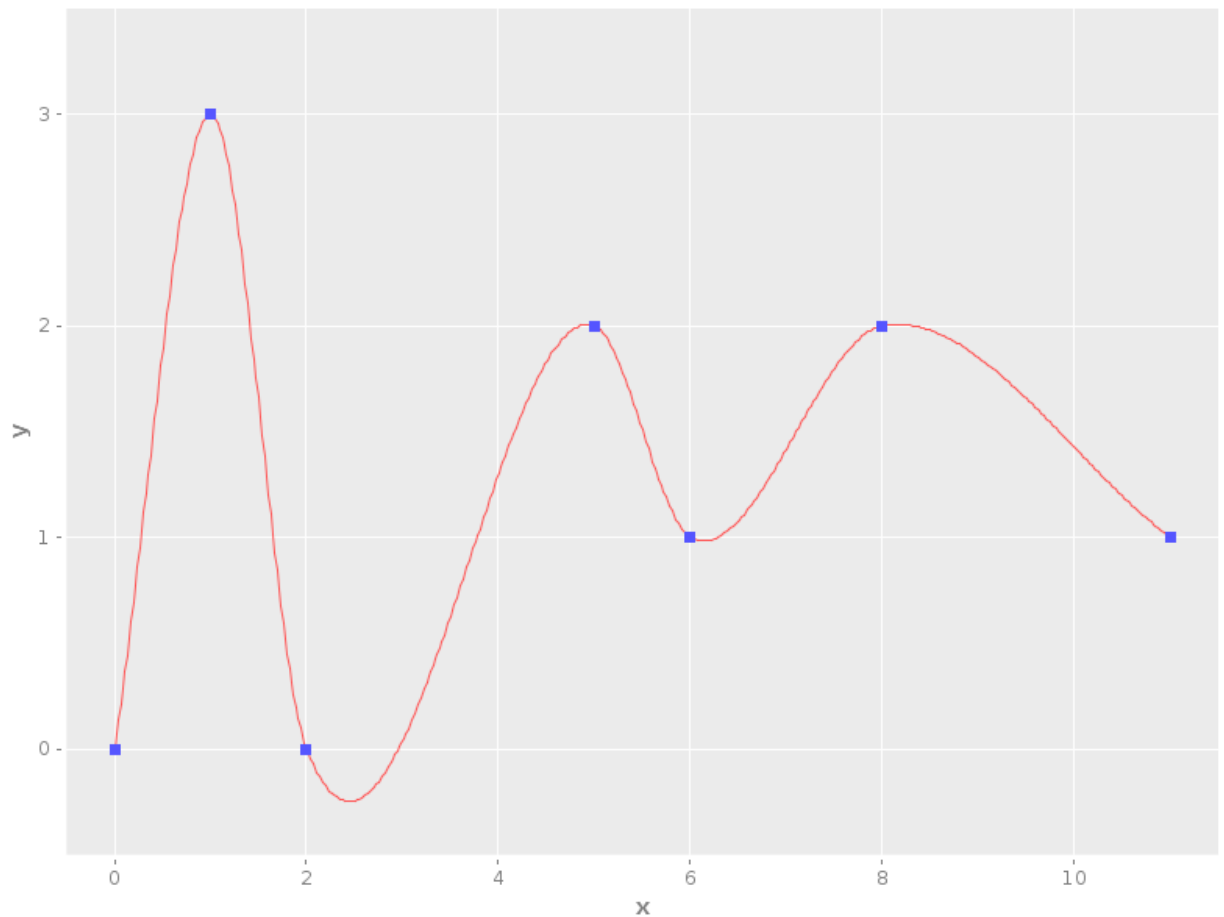
```

; 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-spline :boundaries :

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

```



1.6 В-сплайн

В-сплайны не интерполируют узлы, как ранее рассмотренные методы, а приближают их. При построении В-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))
```

