Отчет по вычислительному практикуму

Шилов Максим

Алгебраические методы интерполирования

Опред. Алгебраический полином

$$P_m(x) = a_0 + a_1 x + \cdots + a_m x^m$$

называется интерполяционным полиномом для функции $f(x)$, заданной на отрезке $[a,b]$, по ее значениям $f(x_i)$ в $n+1$ попарно различных точках $x_i \in [a,b]$, $i = 0,1,\ldots,n$ (узлах интерполяции), если

$$P_m(x_i) = f(x_i), \quad i = 0,1,\ldots,n.$$

Теорема 1. Задача алгебраической интерполяции при $m = n$ имеет единственное решение.

Теорема 2. Алгебраическая интерполяция в форме Лагранжа:

$$P_n(x) = \sum_{k=0}^{n} \frac{w(x)}{(x-x_k)w'(x_k)} f(x_k) \quad ,$$

где

$w(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$ - полином степени n+1

Теорема 3. Алгебраическая интерполяция в форме Ньютона:

$$P_n(x) = \sum_{k=0}^{n} f(x_0,x_1,\ldots,x_k)(x-x_0)(x-x_1)\ldots(x-x_{k-1}) \quad ,$$

где

$f(x_0,x_1,\ldots,x_k)$ - разделенные разности k-го порядка

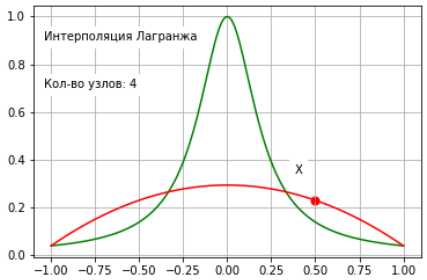
Вычисления

$$f(x) = \frac{1}{1+25x^2}, \quad f(0.5) = 0.137931\,, \quad [a,b] = [-1,1]$$

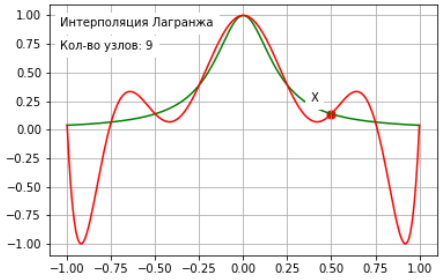
Узлы интерполяции равноотстоящие и упорядочены по величине, так что $x_{i+1} - x_i = h = \mathsf{const}$, то есть $x_i = x_0 + ih$.

Интерполяция в форме Лагранжа

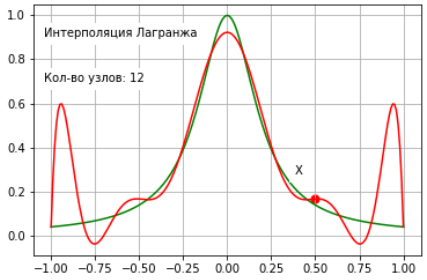
a) $f(-1.0) = 0.03846, \quad f(-0.33333) = 0.26470, \quad f(0.33333) = 0.26470, \quad f(1.0) = 0.03846$
4 узла $\Rightarrow$ $P_3(0.5) = 0.22935520, \quad |P_3(0.5) - f(0.5)| = 0.0914241$

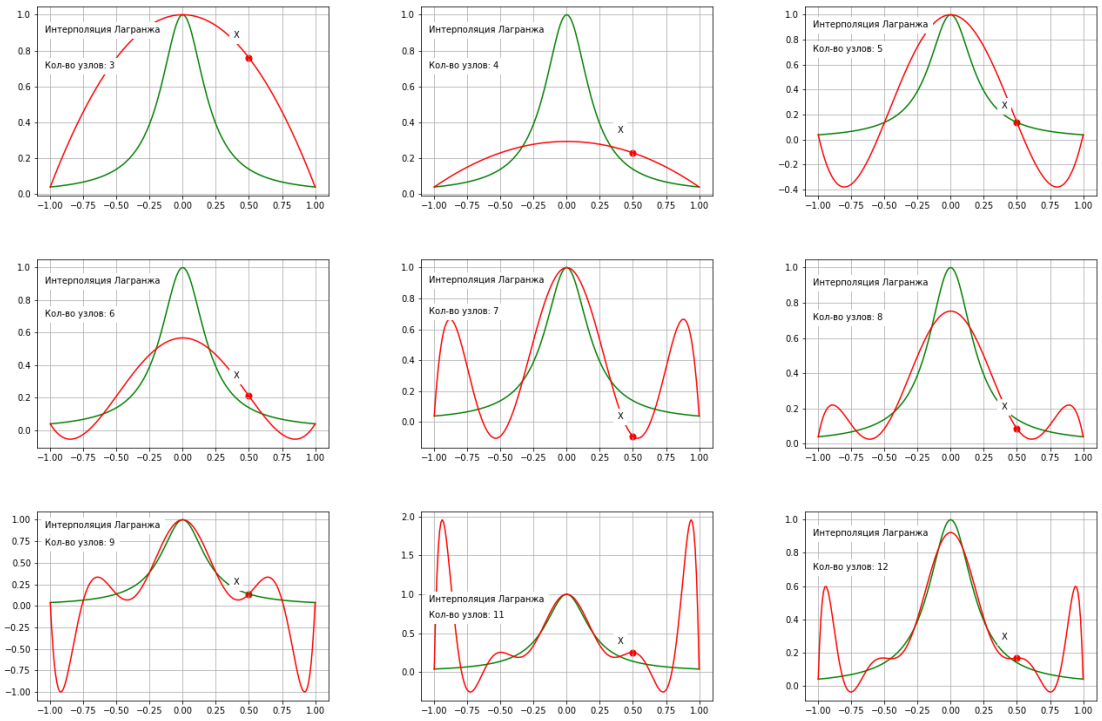


b) $f(-1.0) = 0.038461, \quad f(-0.75) = 0.066390, \quad f(-0.5) = 0.137931,$
$f(-0.25) = 0.390243, \quad f(0.0) = 1.0, \quad f(0.25) = 0.390243, \quad f(0.5) = 0.137931,$
$f(0.75) = 0.066390, \quad f(1.0) = 0.038461, \quad f(1.0) = 0.03846$
9 узлов $\Rightarrow$ $P_8(0.5) = 0.1379310, \quad |P_8(0.5) - f(0.5)| = 0.0$



c) $f(-1.0) = 0.0384615, \quad f(-0.818182) = 0.056384, \quad f(-0.636364) = 0.089896,$
$f(-0.454545) = 0.162198, \quad f(-0.272727) = 0.349711, \quad f(-0.0909091) = 0.828767,$
$f(0.0909091) = 0.828767, \quad f(0.272727) = 0.349711, \quad f(0.454545) = 0.162198,$
$f(0.636364) = 0.089896, \quad f(0.818182) = 0.056384, \quad f(1.0) = 0.0384615$
12 узлов $\Rightarrow$ $P_{11}(0.5) = 0.1656405, \quad |P_{11}(0.5) - f(0.5)| = 0.0277095$





Интерполяция в форме Ньютона

В силу теоремы 1 сам интерполяционный полином будет один и тот же.

Но для интерполяционного полинома, представленного в форме Лагранжа, переход от $P_n(x)$ к $P_{n+1}(x)$ черезвычайно трудоемок. При добавлении новой точки x_{n+1} необходимо пересчитать все "коэффициенты"

$$\frac{w(x)}{(x-x_k)w'(x_k)}.$$

Замечательность представления

$$P_n(x) = \sum_{k=0}^{n} f(x_0, x_1, \ldots, x_k)(x-x_0)(x-x_1)\ldots(x-x_{k-1})$$

состоит в том, что при добавлении или удалении последних по номеру узлов интерполяции перестройке должны подвергнуться лишь те последние слагаемые суммы из правой части, которые вовлекает эти изменяемые узлы. Первые слагаемые зависят только от первых узлов интерполяции и останутся неизменными.

$$\begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

Код программ (Python 3.7)

Преамбула

```
In [ ]: import numpy as np
```

1 Интерполяционный полином Ньютона

```
In [   ]: def fact(k):
              j = 1
              for i in range(2, k+1):
                  j = j*i
              return j

          def triangle(b, t, k):
              m = 0
              if k-t == 1:
                  m = b[t + 1] - b[t]
              else:
                  m = triangle(b, t + 1, k) - triangle(b, t, k - 1)
              return m

          def algoritm(a, b, x, n):
              Lx = 0
              q = (x - a[0]) / (a[n-1] - a[n-2])
              Lx = b[0]

              for i in range(1, n):
                  y = 1;
                  for j in range(i):
                      y = y*(q - j)
                  Lx = Lx + (y * triangle(b, 0, i)) / (fact(i))

              return Lx

          point = 0.5
          A = -1
          B = 1
          number_of_nodes = 12
          step = (np.abs(A) + np.abs(B))/(number_of_nodes-1)
          a = np.arange(A, B+step, step)
          b = 1/(1+25*a*a)

          n = len(a)

          Lx = algoritm(a, b, point, n)
          print(Lx)
```

2 Интерполяционный полином Лагранжа

```
In [   ]: def algoritm(l, a, b, x, n):
              Lx = 0;
              for i in range(n-1):
                  l[0] = l[0] * ((x - a[i + 1]) / (a[0] - a[i + 1]))

              for i in range(1, n):
                  for j in range(i):
                      l[i] = l[i] * ((x - a[j]) / (a[i] - a[j]))

              for i in range(1, n-1):
                  for j in range(i, n-1):
                      l[i] = l[i] * ((x - a[j + 1]) / (a[i] - a[j + 1]))

              for i in range(n):
                  Lx += b[i] * l[i]

              return Lx

          x = 0.5

          A = -1
          B = 1

          number_of_nodes = 12
          step = (np.abs(A) + np.abs(B))/(number_of_nodes-1)

          a = np.arange(-1, 1+step, step)
          b = 1/(1+25*a*a)

          l = np.ones(len(a))
          n = len(a)

          Lx = algoritm(l, a, b, x, n)

          print(Lx)
```