Отчет по вычислительному практикуму

Шилов Максим

Численное методы решения нелинейных уравнений

Метод секущих

Идея интерполяционных методов состоит в том, что задача, нахождения корней уравнения

$$f(x) = 0 \quad (1)$$

заменяется задачей нахождения корней интерполяционного полинома $P_n(x)$, построенного для $f(x)$.

Рассмотрим случай, когда для $f(x)$ строится интерполяционный полином первого порядка $P_1(x)$ — интерполяционный метод первого порядка. Пусть известны приближения x_{n-1} и x_n к кор- ню x^* уравнения (1). Выбрав в качестве узлов интерполяции x_n и x_{n-1}, построим для $f(x)$ интерполяционный полином в форме Ньютона:

$$P_1(x) = f(x_n) + f(x_n, x_{n-1})(x - x_n). \quad (2)$$

Заменяя в уравнении (1) функцию $f(x)$ интерполяционным полиномом $P_1(x)$ из (2), получаем линейное уравнение

$$f(x_n) + f(x_n, x_{n-1})(x - x_n) = 0$$

Принимая его решение за новое приближение x_{n-1}, приходим к интерполяционному методу первого порядка:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n, n_{n-1})}.$$

Подставляя сюда выражение для разделенной разности первого порядка функции $f(x)$, окончательно получаем

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n). \quad (3)$$

Геометрическая интерпретация рассмотренного метода состоит в следующем. Через точки $(x_{n-1}, f(x_{n-1}))$ и $(x_n, f(x_n))$ проводится прямая - секущая графика функции $y = f(x)$. Ее уравнение имеет вид

$$y = f(x_n) + \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}(x - x_n)$$

Полагая в этом равенстве $y = 0$, находим точку пересечения секущей с осью Ox, значение которой в точности совпадает с x_{n+1}, вычисляем по формуле (3). Благодаря такой геометрической интерпретации интерполяционный метод первого порядка называют методом секущих.

Метод хорд

$$x_0 = a$$

$$x_{n+1} = x_n - \frac{x_n - b}{f(x_n) - f(b)} f(x_n).$$

$$y = f(x_n) + \frac{f(x_n) - f(b)}{x_n - b}(x - x_n)$$

Вычисления

$f(x) = 3x - \cos x - 1 = 0 \quad \Rightarrow \quad 3x = \cos x + 1 \quad \Rightarrow \quad x^* \in [0, \frac{2}{3}]$

т.к. производная $f(x)$ существует и сохраняет знак внутри интервала $[0, \frac{2}{3}]$, то корень x^* заведомо будет единственным.

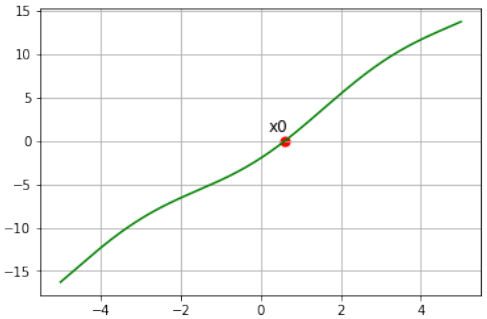
Метод секущих

Точность:	$x_0 \in [0, \frac{2}{3}]$	Кол-во итераций:
10^{-3}	0.6071898315409672	2
10^{-6}	0.6071016710657215	3
10^{-9}	0.6071016481033554	4
10^{-12}	0.6071016481033554	4

Метод хорд

Точность:	$x_0 \in [0, \frac{2}{3}]$	Кол-во итераций:
10^{-3}	0.6069837096982954	2
10^{-6}	0.6071016288617436	4
10^{-9}	0.6071016483488866	5
10^{-12}	0.6071016481031627	7

Метод хорд имеет меньшую скорость сходимости.



Код программ (Python 3.7)

Преамбула

```
In []: import numpy as np
       import matplotlib.pyplot as plt
```

1 Метод секущих

```
In []: def secant(inter, e):

            iterat = []

            x = []

            for i in inter:
                j = 0
                x0 = i[1]
                x1 = 1
                while True:
                    j += 1
                    xn = x1 - ((x1-x0)/(function(x1)-function(x0)))*function(x1)
                    if function(xn) == 0:
                        x.append(xn)
                        break
                    if np.abs(function(xn)) < e:
                        x.append(xn)
                        break
                    x0 = x1
                    x1 = xn
                iterat.append(j)

            return (x, iterat)

        def function(x):
            return (3*x - np.cos(x) - 1)

        inter = [[0,1/2]]
        e = 1e-12

        roots, iteration = secant(inter, e)

        print(roots, iteration)
```

2 Метод хорд

```
In []: def chord(inter, e):

            iterat = []

            x = []

            for i in inter:
                j = 0
                x0 = i[0]
                x1 = i[1]
                while True:
                    j += 1
                    xn = x0 - ((x0-x1)/(f(x0)-f(x1)))*f(x0)
                    if f(xn) == 0:
                        x.append(xn)
                        break
                    if np.abs(f(xn)) < e:
                        x.append(xn)
                        break
                    x0 = xn
                iterat.append(j)

            return (x, iterat)

        def f(x):
            return (3*x - np.cos(x) - 1)

        inter = [[0,1/2]]
        e = 1e-12

        x, iteration = chord(inter, e)

        print(x, iteration)
```