Практикум по вычислительным методам: Вычисление функций

Блинов Иван Сергеевич

01 марта 2019

Приближенное решение уравнения $f(x) = 0$ методом деления отрезка пополам

Описание метода

Для работы метода нам нужно знать отрезок [a, b], такой что выполняется теорема Больцано-Коши $f(a) * f(b) < 0$. В таком случае на этом отрезке $\exists c : f(c) = 0, c \in (a, b)$. Мы будем строить последовательность отрезков $\{[a_n, b_n] : [a_n, b_n] \subset [a_{n-1}, b_{n-1}] \subset [a, b]\}$, на концах которой функция принимает значения разных знаков. На каждом шаге итерации мы вычисляем значение $\xi = \frac{a_n + b_n}{2}$ и значение функции $f(\xi)$ в этой точке. После мы проверяем является ли ξ корнем нашего уравнения и если не является то мы добавляем в нашу последовательность отрезков один из отрезков $[a_n, \xi]$ или $[\xi, b_n]$ (выбираем из них тот на концах которого функция имеет разные знаки)

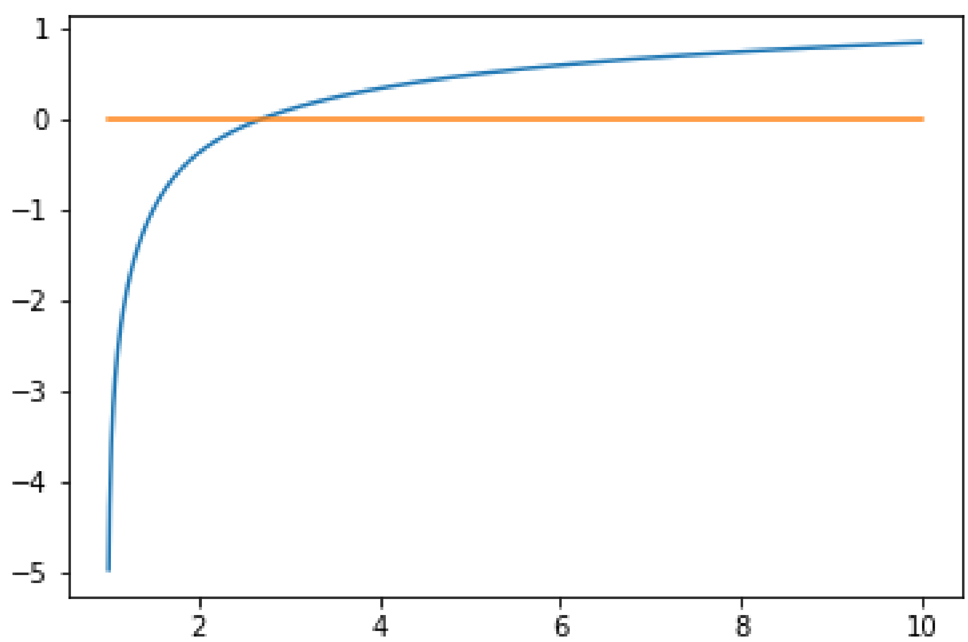
```
In [2]: from matplotlib import pyplot as plt
        import numpy as np
        from scipy import optimize
```

$f(x) = lnln(x) - e^{-x^2}$

```
In [3]: f = lambda x: np.log(np.log(x))-np.exp(-x**2)
```

Строим график для визуального определения отрезка

```
In [4]: x = np.arange(1.01, 10, 0.01)
        plt.plot(x, f(x), x, np.zeros(len(x)))
        plt.show()
```



In [5]:
```
def bisect(f,a,b,eps):
    k=0
    psi = (b+a)/2
    an = a
    bn = b
    while (bn - an > 2 * eps) and (f(psi)!=0):
        psi = (bn + an) / 2
        if f(bn)*f(psi)<0:
            an = psi
        else:
            bn = psi
        k += 1
    return psi, k
```

Мой ответ

In [6]:
```
my_ans, k = bisect(f, 2.0, 6.0, 1e-6)
my_ans, k
```

Out[6]:
```
(2.7199459075927734, 21)
```

Ответ, полученный встроенными методами языка

In [31]:
```
scipy_ans = optimize.root_scalar(f, bracket=[2.0, 6.0], method='bisect').root
scipy_ans
```

Out[31]:
```
2.719947541330839
```

Разница ответов

In [32]:
```
scipy_ans-my_ans
```

Out[32]:
```
1.6337380657205358e-06
```

Метод простых итераций решения уравнения $f(x) = 0$

In [3]:
```
from matplotlib import pyplot as plt
import numpy as np
from scipy import optimize as opt
```

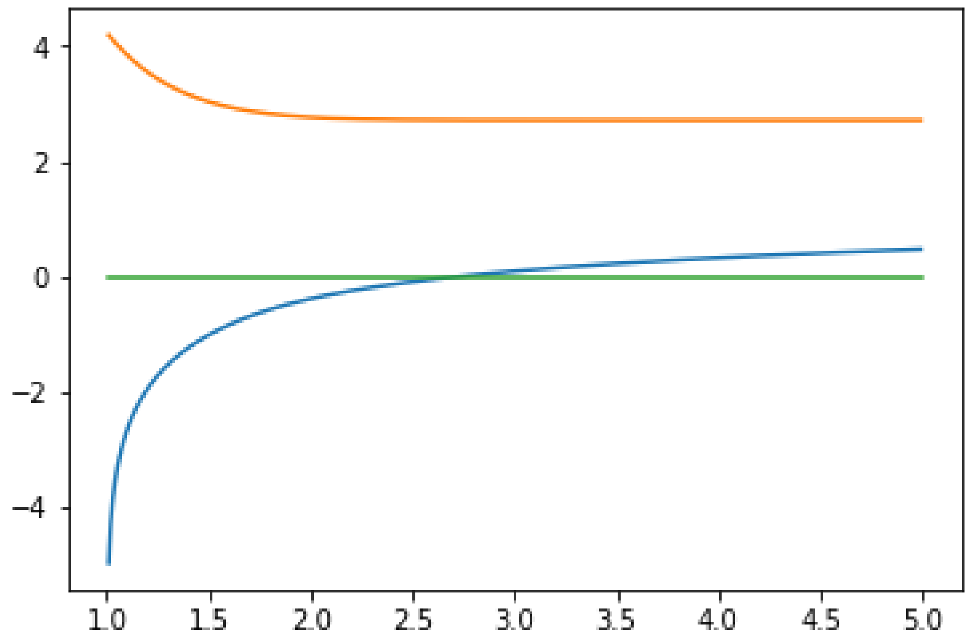
In [41]:
```
f = lambda x: np.log(np.log(x))-np.exp(-x**2)
phi = lambda x: np.exp(np.exp(np.exp(-x**2)))
```

$f(x) = lnln(x) - e^{-x^2}$

$\phi(x) = e^{e^{e^{-x^2}}}$

$\phi'(x) = -2xe^{-x^2+e^{e^{-x^2}}+e^{-x^2}}$

In [42]:
```
x = np.arange(1.01, 5, 0.01)
plt.plot(x, f(x), x, phi(x), x, np.zeros(len(x)))
plt.show()
```




```
In [51]: def simple_iter(f, x0, eps):
             x = x0
             k = 0
             while True:
                 y = f(x)
                 k += 1
                 if abs(y - x) <= eps:
                     return y, k
                 else:
                     x = y
```

Мой ответ

```
In [52]: my_sol, k = simple_iter(phi, 3, 1e-6)
         print(my_sol, k, sep='\n')

2.719947542325148
4
```

Ответ, полученный методами языка

```
In [49]: sc_sol = opt.root_scalar(f, method='brentq', bracket=[2,4]).root
         sc_sol

Out[49]: 2.7199475413303658
```

Разница ответов

```
In [50]: sc_sol-my_sol
```

```
Out[50]: -9.947820345246328e-10
```

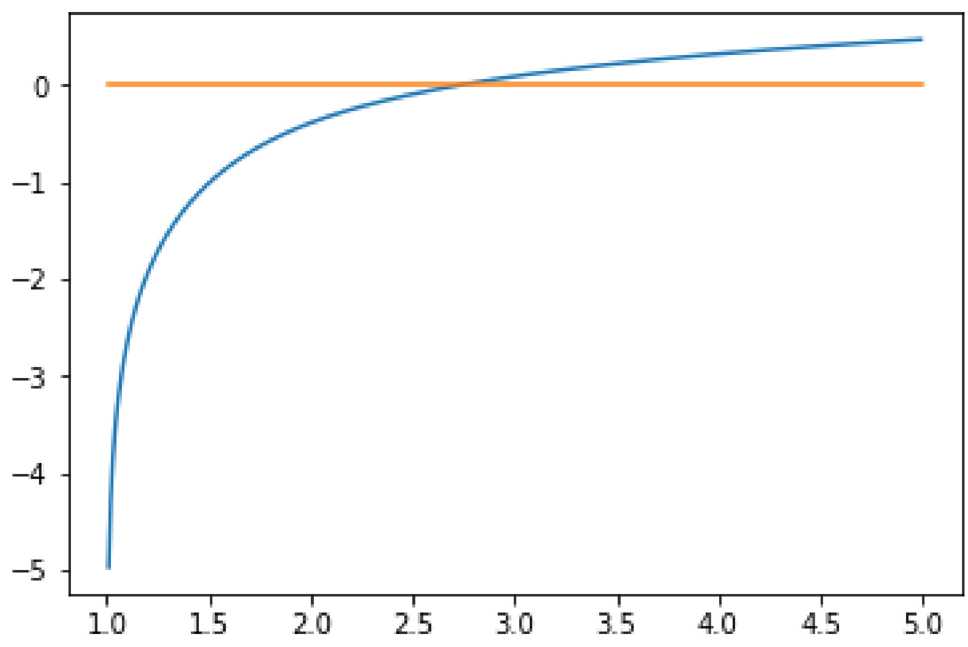
Приближенное решение уравнения $f(x) = 0$ методом Ньютона

```
In [11]: from matplotlib import pyplot as plt
         import numpy as np
         from scipy import optimize as opt
         from math import sqrt
```

$f(x) = lnlnx - e^{-x^2}$
$f'(x) = 2xe^{-x^2} + \frac{1}{xlnx}$
$f''(x) = e^{-x^2}(2 - 4x^2) - \frac{1}{x^2 ln^2(x)} - \frac{1}{x^2 ln(x)}$

```
In [3]: f = lambda x: np.log(np.log(x))-np.exp(-x**2)
        der_f = lambda x: 2*x*np.exp(-x**2)+1/(x*np.log(x))
        der2_f = lambda x: np.exp(-x**2)*(2-4*x**2) - 1/(x**2 * np.log(x)**2) - 1/(x**2 * np.log
```

```
In [5]: x = np.arange(1.01, 5, 0.01)
        plt.plot(x, f(x), x, np.zeros(len(x)))
        plt.show()
```



Функция $f(x)$ дважды дифференциируема на $[1.5, 5.0]$ и $f(1.5) * f(5) < 0$, следовательно выполняются условия сходимости метода Ньютона

Выберем начальное приближение $x0 = 2.5$ Проверим, будет ли выполняться $f(x)f''(x) > 0$

In [6]:
```
f(2.5)*der2_f(2.5)
```

Out[6]: 0.036597429339471584

Определим значения m_1 и M_2

In [9]:
```
m1 = min([abs(der_f(x)) for x in np.arange(1.5,5, 0.01)])
m2 = max([abs(der2_f(x)) for x in np.arange(1.5, 5, 0.01)])
print(m1, m2)
```

```
0.12467109925328225 4.53733067455812
```

По заданному $\varepsilon_0 = 1e - 6$ найдем $\varepsilon = \sqrt{2m_1\varepsilon_0/M_2}$

In [13]:
```
eps = sqrt(2*m1*10**-6/m2)
eps
```

Out[13]: 0.0002344216274173975

In [21]:
```
def newton(f, der_f, x0, eps):
    k = 0
    xc = x0
    while True:
        k += 1
        xn = xc - f(xc)/der_f(xc)
        if abs(xn-xc)<eps:
            return(xn, k)
        xc = xn
```

In [22]:
```
my_sol, k = newton(f, der_f, 2.5, eps)
my_sol, k
```

Out[22]: (2.7199475325339133, 3)

In [17]:
```
sc_sol = opt.root_scalar(f, method='brentq', bracket=[1.5,5]).root
sc_sol
```

Out[17]: 2.719947541330329

In [18]:
```
sc_sol - my_sol
```

Out[18]: 8.796415595924145e-09

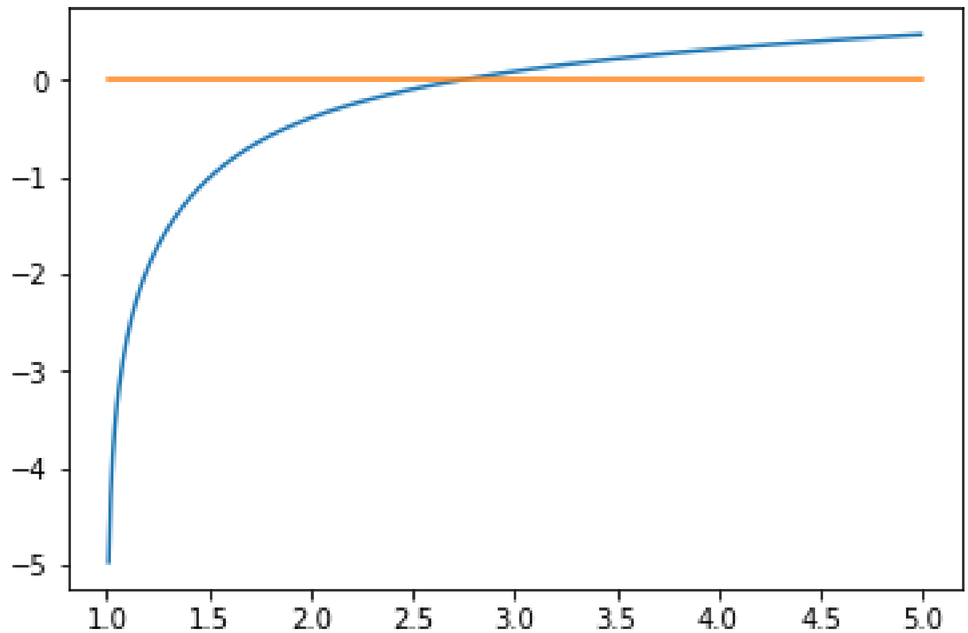
Метод хорд и касательных

In [1]:
```
from matplotlib import pyplot as plt
import numpy as np
from scipy import optimize as opt
```

$f(x) = lnlnx - e^{-x^2}$

In [2]:
```
f = lambda x: np.log(np.log(x))-np.exp(-x**2)
```

In [3]:
```
x = np.arange(1.01, 5, 0.01)
plt.plot(x, f(x), x, np.zeros(len(x)))
plt.show()
```



In [4]:
```
def tangent(f, x, eps):
    k = 0
    x, x_prev = x, 2 * x

    while abs(x - x_prev) > eps:
        x, x_prev = x - f(x) / (f(x) - f(x_prev)) * (x - x_prev), x
        k += 1


    return x, k
```

In [6]:
```
my_sol, k = tangent(f, 2, 10**-6)
print(my_sol, k, sep='\n')
```

```
2.7199475413303302
7
```

In [7]:
```
sc_sol = opt.root_scalar(f, method='brentq', bracket=[1.5,5]).root
sc_sol
```

```
Out[7]: 2.719947541330329

In [8]: sc_sol-my_sol

Out[8]: -1.3322676295501878e-15
```