БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Лабораторная работа 2

Метод Ньютона решения нелинейного уравнения

Вариант 7

Выполнил:
Журик Никита Сергеевич
2 курс, 6 группа
Преподаватель:
Будник Анатолий Михайлович

Минск, 2019

Содержание

1. Постановка задачи

1. Отделить корень и определить отрезок $[a;b]$.
2. Проверить условия теоремы о сходимости метода Ньютона.
3. Решить нелинейное уравнение $f(x) = 0$ методом Ньютона с точностью $\epsilon = 10^{-8}$.
4. Найти априорную и фактическую оценки количества итераций.
5. Вычислить невязку решения.
6. Проанализировать полученные результаты и сравнить с методом простой итерации.

2. Алгоритм решения

- Сперва отделим корни уравнения
$$f(x) = 0 \tag{1}$$
при помощи таблицы значений. Таким образом, для каждого корня получим некоторый отрезок, содержащий сам корень, причём на этом отрезке функция в левой части монотонна.

- В отличие от метода простой итерации метод Ньютона не требует представления уравнения в каноническом виде. Однако при этом он накладывает на функцию исходного уравнения более строгое ограничение, а именно, функция в левой части исходного уравнения должна быть дифференцируема. Тогда итерационный процесс для нахождения решения может быть построен следующим образом:
$$x^{k+1} = x^k - \frac{f(x)}{f'(x)}\bigg|_{x^k} \tag{2}$$

- Справделива следующая теорема:
Теорема о сходимости метода Ньютона.
Пусть
  - $f(x) \in C^{(2)}(S_0)$, где $S_0 = [x^0; x^0 + 2h_0]; h_0 = -\frac{f(x)}{f'(x)}\Big|_{x^0}$, причём на концах отрезка S_0 $f(x) \neq 0$ и $f'(x) \neq 0$;
  - Для начального приближения справедливо $2|h_0|M \leq |f'(x^0)|$, где $M = \max\limits_{x \in S_0} |f''(x)|$.

  Тогда:
  - Внутри S_0 лежит единственный корень уравнения (1);
  - Последовательность приближений x^k может быть построена по x^0 с использованием (2);
  - Последовательность x^k сходится к корню x^*;
  - Скорость сходимости характеризуется неравенством:
$$|x^* - x^{k+1}| \leq |x^{k+1} - x^k| \leq \frac{M}{2|f'(x^k)|}|x^k - x^{k-1}|^2, k = 1, 2, ... \tag{3}$$

  Из последнего неравенства следует априорная оценка числа итераций:
$$k \geq log_2 \frac{ln(\alpha\epsilon)}{ln(\alpha|x^1 - x^0|)}, \alpha = \max_{x \in S_0} \left|\frac{f''(x)}{2f'(x)}\right| \tag{4}$$

Решение конкретного уравнения

- Рассмотрим уравнение $3ln^2x + 6lnx - 5 = 0$. Вычислим производную функции $f(x)$ в явном виде:
$$f'(x) = \frac{6lnx}{x} + \frac{6}{x} \tag{5}$$

- Тогда итерационный процесс будет выглядеть следующим образом:
$$x^{k+1} = x^k - \frac{3ln^2x + 6lnx - 5}{\frac{6}{x}(lnx + 1)}\bigg|_{x^k}; k = 0, 1, 2, ...; x^0 \tag{6}$$

- В качестве начальных приближений к корням используем найденные при решении методом простой итерации отрезки.
- Проверим условия теоремы о сходимости метода Ньютона.

$$f''(x) = -6\frac{lnx}{x^2}$$
$$f'''(x) = \frac{12lnx - 6}{x^3}$$
$$f'''(x) = 0 \iff lnx = \frac{1}{2} \implies x = 1.6487212707$$

Так как функция трижды дифференцируема на $(0; +\infty)$, она дважды непрерывно дифференцируема на любом отрезке, вложенном в него.

1. Рассмотрим отрезок, содержащий меньший корень $[0.01; 0.1673684210526316]$.

$$h_0 = -0.12942998486317597$$

Так как $x^0 + 2h_0 < 0$, $f(x)$ не определена на S_0. Следовательно, условия теоремы не выполняются для меньшего корня.

2. Теперь рассмотрим отрезок $[1.7410526315789476; 1.8984210526315792]$, содержащий больший корень:

$$h_0 = -0.00022113804402974617$$

Проверим второе условие:

$$\begin{aligned} f'''(x) \neq 0 \ \forall x \in S_0 &\implies \\ M &= \max\{|f''(1.88301784772424)|, |f''(1.8834601238122997)|\} \\ &= 1.0709293413775358 \\ 2|h_0|M &= 0.00047364643969258517 \\ |f'(x^0)| &= 5.202479912227185 \end{aligned}$$

Легко видеть, что и второе условие выполнено в окрестности большего корня. Таким образом, для него справедлива теорема и возможна априорная оценка числа итераций:

$$log_2\frac{ln(\alpha\epsilon)}{ln(\alpha|x^1 - x^0|)} = \begin{bmatrix} \frac{f''(x)}{f'(x)} = -\frac{lnx}{x(lnx+1)} \implies \\ \alpha = \max_{x \in S_0}\left|\frac{f''(x)}{f'(x)}\right| \\ = 0.20583104079469631 93 \end{bmatrix} = 1.0004713925101845 \tag{7}$$

Тогда

$$k_{aprior} = 2 \tag{8}$$

3. Листинг программы

Для реализации алгоритма был использован Python и библиотеки numpy и matplotlib.

```
#Common.py

#!/usr/bin/env python
# coding: utf-8

#Plot the function

import math
import numpy as np
import matplotlib.pyplot as plt

samples = 20
left_border = 0.01
```

```python
right_border = 3
delta = (right_border - left_border) / samples
eps = 10 ** -5

def f(x):
    l = math.log(x)
    return 3 * l ** 2 + 6 * l - 5

def f_prime(x):
    l = math.log(x)
    return 6 * (l + 1) / x

def phi(x):
    return math.e ** ((-3 * math.log(x) ** 2 + 5) / 6)

def phi_prime(x):
    return -phi(x) * math.log(x) / x

x_values = np.linspace(left_border, right_border, samples)
f_values = np.zeros(np.shape(x_values))

for i in range(samples):
    f_values[i] = f(x_values[i])

#Root separation

def separate_roots(x_values, f_values):
    intervals = [ ]
    for i in range(samples - 1):
        if (f_values[i + 1] * f_values[i] < 0):
            intervals = np.append(intervals, i)
    return intervals

def dichotomy(init_intervals):

    def dichotomy_single_root(interval):
        if (interval[1] - interval[0] < delta):
            return interval
        center = (interval[0] + interval[1]) / 2
        if (f(interval[0]) * f(center) < 0):
            return dichotomy_single_root([interval[0], center])
        else:
            return dichotomy_single_root([center, interval[1]])

    for i in range(len(init_intervals)):
        init_intervals[i] = dichotomy_single_root(init_intervals[i])
    return init_intervals

#Newton.py

#!/usr/bin/env python
# coding: utf-8

from Common import *

intervals = separate_roots(x_values, f_values)
print(intervals)

#Solve the equation using Newton method

def find_root(interval):
    x_left = interval[0]
    x_right = interval[1]
    lam = f(x_left) / f(x_right)
    old_x = (x_left - lam * x_right) / (1 - lam)
    new_x = old_x - f(old_x) / f_prime(old_x)
```

```
    iter_num = 1
    while (abs(old_x - new_x) >= eps):
        old_x = new_x
        new_x = old_x - f(old_x) / f_prime(old_x)
        iter_num += 1
    return (old_x, new_x, iter_num)

for interval in intervals:
    (x_k, x_k1, iter_num) = find_root(interval)
    print("Interval: [{}; {}]\nx^0 = {}\nRoot: x* = {}; f(x*) = {}\n|x^(k+1) - x^k| = {}\nNumber
         of iterations: {}\n"
          .format(interval[0], interval[1], x_0, x_k1, f(x_k1), abs(x_k - x_k1), iter_num))
```

4. Вывод программы

```
Interval: [0.01; 0.1673684210526316]
x^0 = 0.14134905726406316
Root: x* = 0.07186304228921059; f(x*) = 3.552713678800501e-15
|x^(k+1) - x^k| = 2.6333352165508472e-11
Number of iterations: 8

Interval: [1.7410526315789476; 1.8984210526315792]
x^0 = 1.8834601238122997
Root: x* = 1.8832389908008633; f(x*) = 0.0
|x^(k+1) - x^k| = 5.032593453080381e-09
Number of iterations: 2
```

5. Выводы

- Как следует из результата выполнения программы, метод сошёлся для меньшего корня к решению уравнения, несмотря на то, что условия теоремы не были выполнены. Из числа итераций видно, что сходился метод достаточно медленно (по сравнению со вторым корнем), но требуемая точность была достигнута.
- В окрестности второго корня метод сошёлся за две итерации, что совпадает с априорной оценкой в силу того, что оценённое количество итераций мало. Выбранная точность $\epsilon = 10^{-8}$ привела к тому, что невязка решения оказалась равной нулю.
- Ниже приведены результаты лишь для большего корня, так как метод простой итерации не сошёлся к меньшему:

$$
\begin{aligned}
k_{Newton} &= 2 \\
k_{Simple} &= 24 \\
r_{Newton} &= 0.0 \\
r_{Simple} &= 1.9699690767538414e-08
\end{aligned}
$$

  Сравнивая метод Ньютона с методом простой итерации, можно заметить, что метод Ньютона сходится к решению уравнения значительно быстрее, а само приближение к решению точнее, чем в методе простой итерации. Это обусловлено квадратичной скоростью сходимости метода Ньютона в сравнении с линейной скоростью сходимости метода простой итерации, что делает его очень полезным при решении уравнений с непрерывно дифференцируемой функцией.