Отчет по лабораторной работе №2: Поиск корней нелинейных функций

Чигарев Дмитрий 381807-1

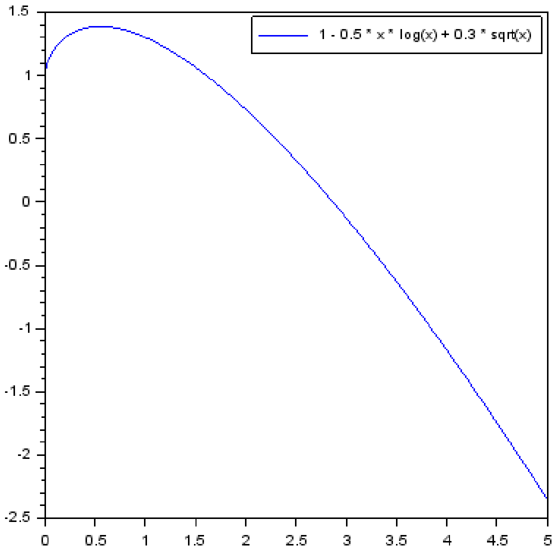
1, 2. Графическое исследование

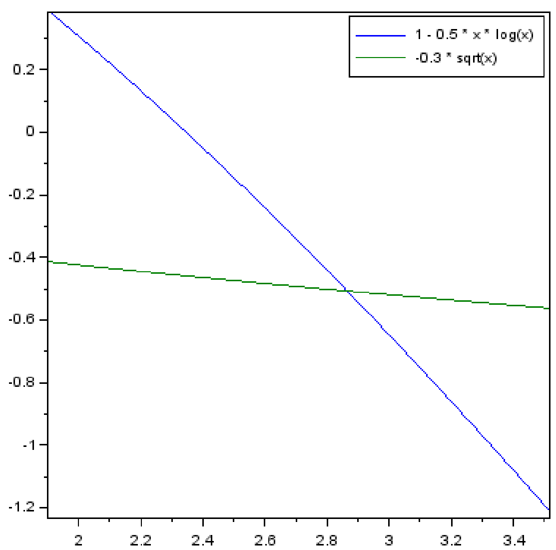
Функция для поиска корней:

$$f(x) = 1 - 0.5xlog(x) + 0.3\sqrt{x}$$

Локализуем корни визуально, нарисовав функцию, а также её разбиение в виде:

$$f_1(x) = 1 - 0.5xlog(x)$$
$$f_2(x) = -0.3\sqrt{x}$$





Из графиков видно, что как минимум один корень расположен на отрезке $x \in [2.5, 3.5]$. Проверим, могут ли быть ещё корни вне рассматриваемого отрезка.

Возьмем производную функции и увидим, что при $x \in [2, \infty)$ значение производной гарантировано меньше нуля:

$$f'(x) = \frac{0.15}{\sqrt{x}} - 0.5\log(x) - 0.5 < 0.15 - 0.5\log(x) < 0, \qquad \text{оценки верны при } x > 2$$

Соответственно, после того как функция пересечет ось OX в рассматриваемой области, она так и продолжит бесконечно убывать, следовательно никогда не сможет пересечь OX вновь, и следовательно корень только один.

3. МПИ

Итерационный процесс решения нелинейных уравнений вида $f(x) = 0$ представляется как:

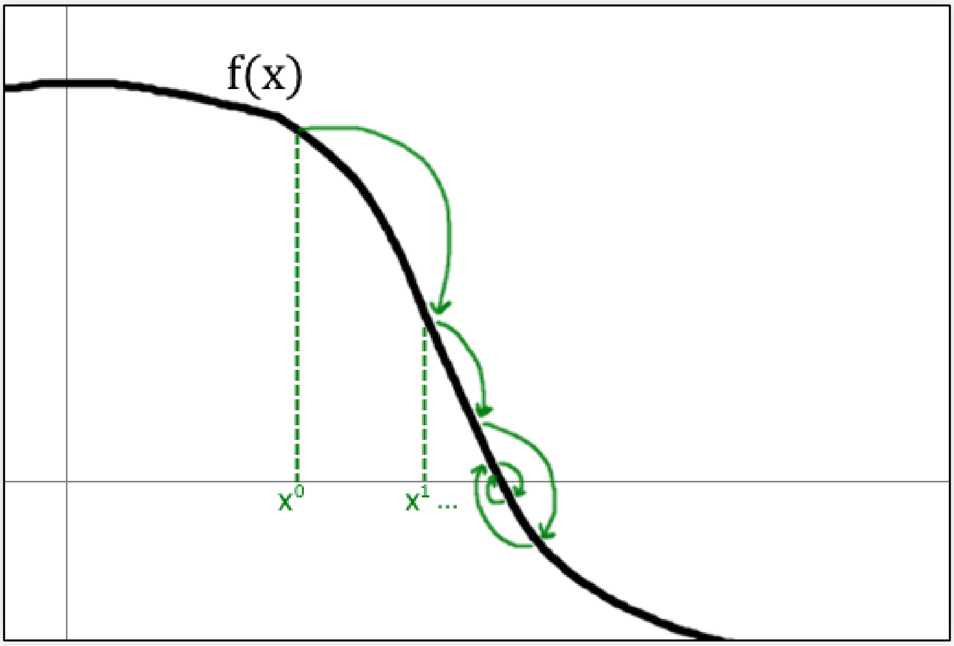
$$x^{k+1} = s(x^k, f)$$
$$\lim_{k\to\infty} f(x^k) = 0$$

Определение самой итерационной функции s остается на плечах реализации конкретного метода.

Метод простых итераций предлагает следующее определение:

$$s(x, f) = x + p(x) * f(x)$$

В случае, если функция убывает в окрестности корня, то положительное значение функции $f(x)$, будет говорить о том, что её корень находится правее, а отрицательное – левее, таким образом, прибавляя к текущему x значение функции в точке, мы будет двигать x в сторону нуля функции.



Чем ближе x^k к корню, тем меньше значение $f(x^k)$, соответственно шаги постепенно будут становиться всё меньше и меньше, в результате x^k должно сойтись к нулю функции. Погрешность на текущем шаге метода простой итерации оценивается следующим образом:

$$|x^{n+1} - x^*| \leq \frac{x^{n+1} - x^n}{1 - q^n}$$

$$q^n = \frac{x^{n+1} - x^n}{x^n - x^{n-1}} - \text{сжатие на данном шаге}$$

Функция $p(x)$ выполняет роль коэффициента, регулирующего величину шага. При слишком большом шаге есть вероятность выхода за пределы окрестности корня и по итогу невозможность сойтись к решению, при слишком маленьком – неоправданно занизить скорость сходимости.

Часто в качестве функции $p(x)$ выбирается константа, при этом очевидным образом вытекают следующие условия:

$$\begin{matrix} p > 0, \text{если } f'(x) < 0 \\ p < 0, \text{если } f'(x) > 0 \end{matrix} \quad x \in S - \text{область поиска}$$

Абсолютное же значение p можно взять как обратное к среднему значению производной функции в некоторой окрестности корня. Можно использовать следующую аппроксимация:

$$0 < m < |f'(x)| < M$$

$$p = \left(\frac{m + M}{2}\right)^{-1}$$

Достаточным условием сходимости метода, при таком выборе p, будет выбор следующего начального приближения:

$$x^0 \in S,$$
$$|s'(x)| < 1, \text{где } x \in S,$$
$$S - \text{окрестность корня}$$

Теперь определим $s(x)$ для нашей функции предложенным выше способом. Для этого нужно определить верхнюю и нижнюю границы для $f'(x)$ на отрезке $x \in [2.5, 3.5]$.

Рассмотрев $f''(x)$ обнаружим, что $f'(x)$ – монотонно убывает на заданном отрезке, а значит её границами будут крайние точки отрезка:

$$f''(x) = -\frac{0.075}{x^{3/2}} - \frac{0.5}{x} < 0$$
$$\downarrow$$
$$m = |f'(2.5)|$$
$$M = |f'(3.5)|$$
$$\downarrow$$
$$p = \frac{2}{|f'(2.5)| + |f'(3.5)|}$$

Окончательной записью для $s(x)$ будет:

$$s(x) = x + \frac{2f(x)}{|f'(2.5)| + |f'(3.5)|}$$

Отыщем корень $f(x)$ методом простых итераций, используя найденную функцию $s(x)$:

```
>>> p = 2 / (abs(df(2.5)) + abs(df(3.5)))
>>> function [res]=s(x), res=x + p * f(x) endfunction
>>> x = simple_iter(f, s=s, eps=0.001, x0=2.5, verbose=%T, check_convergence_cond=%T)
Шаг 1: x= 2.844573610D+00 | f(x)= 1.909853727D-02 | error= 3.445736103D-01 |
Шаг 2: x= 2.864577523D+00 | f(x)= 3.812894159D-04 | error= 2.123679610D-02 |
Шаг 3: x= 2.864976888D+00 | f(x)= 6.836186946D-06 | error= 4.075000941D-04 |
Шаг 4: x= 2.864984048D+00 | f(x)= 1.222903511D-07 | error= 7.290980704D-06 |
Шаг 5: x= 2.864984176D+00 | f(x)= 2.187524051D-09 | error= 1.304206388D-07 |
```

4. Метод Ньютона

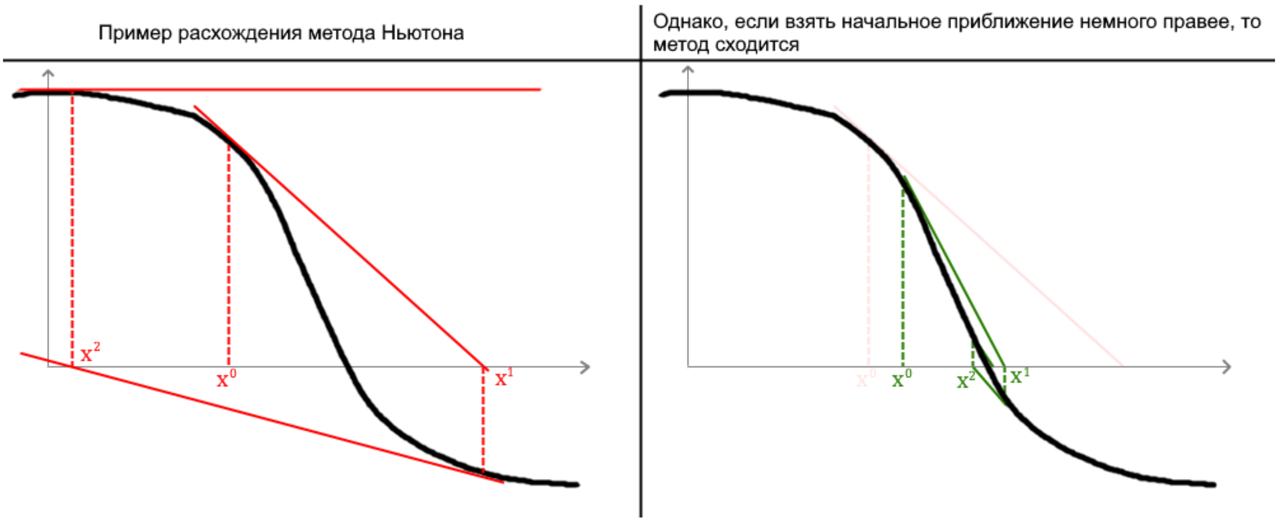
Метод Ньютона – частный случай метода простой итерации, гарантирующий квадратичную сходимость, при выполнении некоторых условий. Однако в обмен на быструю сходимость мы получаем необходимость вычисления производной исходной функции на каждом шаге итерационного процесса. Также метод очень чувствителен к точности задания начального приближения.

Итерационная функция имеет следующий вид:

$$s(x) = x - \frac{f(x)}{f'(x)}$$

Геометрически, x^{k+1} будет представлять точку пересечения касательной к функции в точке x^k с осью OX.

Метод требует хорошего начального приближения и может не сойтись, в случае недостаточно точного задания x^0, особенно если функция и её производная меняют своё поведение в области поиска корня.



Из ограничения $|s'(x)| < 1$ для МПИ, можно вывести следующее достаточное условие сходимости метода Ньютона, накладываемое на x^0:

$$\frac{f'(x)}{f''(x)} \geq 0$$
$$x \in S - \text{окрестность корня}, \qquad x^0 \in S$$

Помимо ограничений на x^0, метод накладывает следующие ограничения на саму функцию $f(x)$:

1. Ограниченность $f''(x), x \in S$
2. $f'(x) \neq 0, x \in S$
3. $f''(x)$ сохраняет знак на S

Представленная реализация метода может проверить все эти условия для исследуемой функции, если передать соответствующий флаг. Здесь вычислим лишь начальное приближение x^0.

Из достаточных условий сходимости следует, что множество удовлетворяющих нас x^0 это решение следующего неравенства:

$$\frac{f'(x)}{f''(x)} \geq 0$$

Как было показано в п. 3, вторая производная всюду отрицательна, а первая отрицательна при $x > 2$. Соответственно, множество x^0 для которых метод Ньютона гарантированно сойдется есть $x^0 \in [2, \infty]$, при этом мы получим квадратичную сходимость.

Пусть $x^0 = 2.5$, найдем корень $f(x)$ методом Ньютона:

```
>>> x = newtoon_method(f, x0=2.5, eps=1.e-6, verbose=%T, check_convergence_cond=%T)
Шаг 1: x= 2.881080720D+00 | f(x)=-1.511767563D-02 | f'(x)=-9.407109684D-01 | error= 3.810807197D-01 |
Шаг 2: x= 2.865010241D+00 | f(x)=-2.443775780D-05 | f'(x)=-9.376666920D-01 | error= 1.542019673D-02 |
Шаг 3: x= 2.864984179D+00 | f(x)=-6.452405277D-11 | f'(x)=-9.376617405D-01 | error= 2.610464287D-05 |
Шаг 4: x= 2.864984179D+00 | f(x)= 1.110223025D-16 | f'(x)=-9.376617405D-01 | error= 6.881402521D-11 |
```

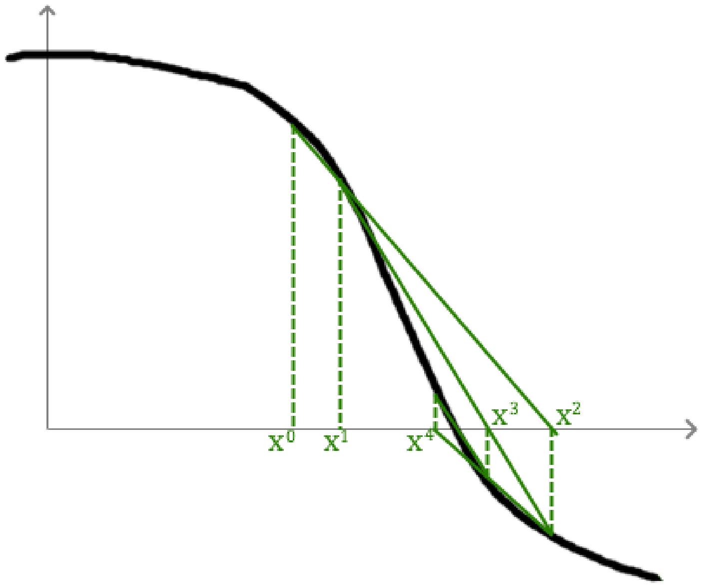
5. Метод секущих

Метод секущих – двух-шаговый итерационный метод, который, в сравнении с Ньютоном, не требует вычисления производной, но обеспечивает лишь сверхлинейную сходимость.

Итерационная функция имеет вид:

$$s(x^k, x^{k-1}) = x^k - \frac{f(x^k) * (x^k - x^{k-1})}{f(x^k) - f(x^{k-1})}$$

Геометрически, x^{k+1} будет точкой пересечения прямой, проходящей через две предыдущие точки, с осью OX:



Найдем корень $f(x)$ методом секущих:

```
>>> x = secant_method(f, x0=2.5, x1=3.5, eps=1.e-6, verbose=%T)
Шаг 1: x= 2.842662523D+00 | f(x)= 2.088270245D-02 | error= 3.426625234D-01 |
Шаг 2: x= 2.865888175D+00 | f(x)=-8.477206312D-04 | error= 2.491434500D-02 |
Шаг 3: x= 2.864982124D+00 | f(x)= 1.926026746D-06 | error= 8.720320141D-04 |
Шаг 4: x= 2.864984178D+00 | f(x)= 1.763553747D-10 | error= 2.049241166D-06 |
Шаг 5: x= 2.864984179D+00 | f(x)= 2.220446049D-16 | error= 1.880969985D-10 |
```

6. SciLab fsolve

Функция $fsolve$ находит решение систем линейных уравнений используя некоторую модификацию метода Пауэлла (какая именно модификация используется не задокументировано).

Найдем корень $f(x)$:

```
>>> [x, _, info] = fsolve([2.5], f, tol=1.e-6)
>>> printf("Метод завершил работу с кодом: %i", info)
Метод завершил работу с кодом: 1 (успех)
>>> printf("%e | f(x) = %e", x, f(x))
x = 2.8649841785208250e+00 | f(x) = 2.2204460492503131e-16
```

7. Сравнение исследованных методов

Метод	x	f(x)	Оценка погрешности	Сходимость
МПИ	2.864984176D+00	2.187524051D-09	1.304206D-07	Линейная
Метод Ньютона	2.864984179D+00	1.110223025D-16	6.881402D-11	Квадратичная
Метод секущих	2.864984179D+00	2.220446049D-16	1.880969D-10	Линейная
fsolve	2.531785392D+00	2.220446049D-16	-	Вероятно линейная

8. Приложения

1. Ссылка на репозиторий с исходным кодом: https://github.com/proxodilka/numerical-analysis-labs/blob/master/lab2_non_linear_roots/lab2.sce