Отчет по Лабораторной работе №2
по курсу “Вычислительная математика”

Вариант №3

Выполнил:
Студент группы P32082
Дробыш Дмитрий Александрович

Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург, 2023

1. Задание лабораторной работы.

Лабораторная работа №2
Численное решение нелинейных уравнений и систем

Цель работы: изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

№ варианта определяется как номер в списке группы согласно ИСУ.

Лабораторная работа состоит из двух частей: вычислительной и программной.

1 Вычислительная реализация задачи:

Задание:

1. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 6)
2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью $\varepsilon=10^{-2}$.
4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.

5.1 Для метода половинного деления заполнить таблицу 1.

5.2 Для метода хорд заполнить таблицу 2.

5.3 Для метода Ньютона заполнить таблицу 3.

5.4 Для метода секущих заполнить таблицу 4.

5.5 Для метода простой итерации заполнить таблицу 5.

6. Заполненные таблицы отобразить в отчете.

Таблица 1

Уточнение корня уравнения методом половинного деления

№ шага	a	b	x	f(a)	f(b)	f(x)	|a-b|
1							
2							
3….							

Таблица 2

Уточнение корня уравнения методом хорд

№ шага	a	b	x	f(a)	f(b)	f(x)	$\lvert x_{k+1}- x_k \rvert$
1							
2							
3….							

Таблица 3

Уточнение корня уравнения методом Ньютона

№ итерации	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}	$\lvert x_{k+1}- x_k \rvert$
1					
2					
3…					

Таблица 4

Уточнение корня уравнения методом секущих

№ итерации	x_{k-1}	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1					
2					
3…					

Таблица 5

Уточнение корня уравнения методом простой итерации

№ итерации	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1				
2				
3…				

2 Программная реализация задачи:

Для нелинейных уравнений:

1. Все численные методы (см. табл. 8) должны быть реализованы в виде отдельных подпрограмм/методов/классов.
2. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.
3. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.
4. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные.
5. Для методов, требующих начальное приближение к корню (методы Ньютона, секущих, хорд с фиксированным концом), выбор начального приближения (a или b) вычислять в программе.
6. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале.
7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
8. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

Для систем нелинейных уравнений:

1. Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы).
2. Организовать вывод графика функций.
3. Начальные приближения ввести с клавиатуры.
4. Для метода простой итерации проверить достаточное условие сходимости.
5. Организовать вывод вектора неизвестных: $x_1,\ x_2$.
6. Организовать вывод количества итераций, за которое было найдено решение.
7. Организовать вывод вектора погрешностей: $|x_i^{(k)} - x_i^{(k-1)}|$
8. Проверить правильность решения системы нелинейных уравнений.

3 Оформить отчет, который должен содержать:

1. Титульный лист.
2. Цель лабораторной работы.
3. Порядок выполнения работы.
4. Рабочие формулы используемых методов.
5. Графики функций на исследуемом интервале.
6. Заполненные таблицы вычислительной части лабораторной работы (в зависимости от варианта: табл. 1 – 5).
7. Листинг программы, по крайней мере, коды используемых методов.
8. Результаты выполнения программы при различных исходных данных.
9. Выводы

Таблица 6

Вид нелинейного уравнения для вычислительной реализации

№ варианта	Функция	№ варианта	Функция
1	$2{,}74x^3 - 1{,}93x^2 - 15{,}28x - 3{,}72$	21	$1{,}8x^3 - 2{,}47x^2 - 5{,}53x + 1{,}539$
2	$-1{,}38x^3 - 5{,}42x^2 + 2{,}57x + 10{,}95$	22	$x^3 - 3{,}78x^2 + 1{,}25x + 3{,}49$
3	$x^3 + 2{,}84x^2 - 5{,}606x - 14{,}766$	23	$-x^3 + 5{,}67x^2 - 7{,}12x + 1{,}34$
4	$x^3 - 1{,}89x^2 - 2x + 1{,}76$	24	$x^3 - 2{,}92x^2 + 1{,}435x + 0{,}791$
5	$-2{,}7x^3 - 1{,}48x^2 + 19{,}23x + 6{,}35$	25	$x^3 - 2{,}56x^2 - 1{,}325x + 4{,}395$
6	$2x^3 + 3{,}41x^2 - 23{,}74x + 2{,}95$	26	$1{,}62x^3 - 8{,}15x^2 + 4{,}39x + 4{,}29$
7	$x^3 + 2{,}28x^2 - 1{,}934x - 3{,}907$	27	$2{,}335x^3 + 3{,}98x^2 - 4{,}52x - 3{,}11$
8	$3x^3 + 1{,}7x^2 - 15{,}42x + 6{,}89$	28	$-1{,}85x^3 - 4{,}75x^2 - 2{,}53x + 0{,}49$
9	$-1{,}8x^3 - 2{,}94x^2 + 10{,}37x + 5{,}38$	29	$-1{,}78x^3 - 5{,}05x^2 + 3{,}64x + 1{,}37$
10	$x^3 - 3{,}125x^2 - 3{,}5x + 2{,}458$	30	$-2{,}75x^3 - 4{,}53x^2 + 17{,}87x - 1{,}94$
11	$4{,}45x^3 + 7{,}81x^2 - 9{,}62x - 8{,}17$	31	$-3{,}64x^3 + 2{,}12x^2 + 10{,}73x + 1{,}49$
12	$x^3 - 4{,}5x^2 - 9{,}21x - 0{,}383$	32	$x^3 + 1{,}41x^2 - 5{,}472x - 7{,}38$
13	$x^3 + 4{,}81x^2 - 17{,}37x + 5{,}38$	33	$x^3 - 0{,}12x^2 - 1{,}475x + 0{,}192$
14	$2{,}3x^3 + 5{,}75x^2 - 7{,}41x - 10{,}6$	34	$x^3 - 0{,}77x^2 - 1{,}251x + 0{,}43$
15	$-2{,}4x^3 + 1{,}27x^2 + 8{,}63x + 2{,}31$	35	$x^3 - 0{,}78x^2 - 0{,}826x + 0{,}145$
16	$5{,}74x^3 - 2{,}95x^2 - 10{,}28x - 4{,}23$	36	$1{,}7x^3 - 3{,}45x^2 - 5{,}31x + 1{,}123$
17	$-0{,}38x^3 - 3{,}42x^2 + 2{,}51x + 8{,}75$	37	$x^3 - 3{,}75x^2 + 2{,}25x + 3{,}51$
18	$x^3 + 2{,}64x^2 - 5{,}41x - 11{,}76$	38	$-x^3 + 5{,}32x^2 - 6{,}12x + 0{,}34$
19	$2x^3 - 1{,}89x^2 - 5x + 2{,}34$	39	$x^3 - 2{,}95x^2 + 1{,}52x + 0{,}91$
20	$-2{,}8x^3 - 3{,}48x^2 + 10{,}23x + 9{,}35$	40	$0{,}5x^3 - 2{,}56x^2 - 1{,}35x + 4{,}39$

Выбор метода для вычислительной реализации задачи (табл. 1-5)

1 – Метод половинного деления
2 – Метод хорд
3 – Метод Ньютона
4 – Метод секущих
5 – Метод простой итерации

Таблица 7

Методы для вычислительной реализации

№ вари-анта	*Крайний правый корень*	*Крайний левый корень*	*Цен-траль-ный корень*	№ вари-анта	*Крайний правый корень*	*Крайний левый корень*	*Цен-траль-ный корень*
1	3	4	5	21	2	1	5
2	5	2	1	22	5	3	1
3	1	5	3	23	3	5	1
4	5	1	4	24	2	3	5
5	2	5	4	25	5	1	4
6	3	1	5	26	3	2	5
7	1	5	3	27	1	3	5
8	5	2	3	28	2	5	4
9	1	5	4	29	1	3	5
10	3	1	5	30	4	5	1
11	1	2	5	31	2	3	5
12	4	5	1	32	1	5	4
13	5	2	3	33	5	1	3
14	3	5	1	34	2	5	4
15	5	1	2	35	4	2	5
16	2	5	3	36	1	5	3
17	1	4	5	37	2	3	5
18	3	5	2	38	5	2	4
19	5	1	4	39	1	3	5
20	1	3	5	40	2	5	3

Выбор метода для программной реализации задачи

Решение нелинейных уравнений:

1 – Метод половинного деления
2 – Метод хорд
3 – Метод Ньютона
4 – Метод секущих
5 – Метод простой итерации

Решение систем нелинейных уравнений:

6 – Метод Ньютона
7 – Метод простой итерации

Таблица 8

Методы, реализуемые в программе

№ варианта	Методы в программе	№ варианта	Методы в программе
1	1, 3, 5, 6	21	1, 4, 5, 6
2	2, 3, 5, 7	22	2, 3, 5, 7
3	1, 4, 5, 6	23	1, 3, 5, 6
4	1, 3, 5, 7	24	2, 4, 5, 7
5	1, 3, 5, 7	25	2, 3, 5, 6
6	2, 4, 5, 6	26	1, 4, 3, 7
7	1, 4, 5, 6	27	2, 4, 5, 6
8	1, 3, 5, 7	28	1, 3, 5, 6
9	2, 3, 5, 7	29	2, 3, 5, 7
10	2, 3, 5, 6	30	1, 4, 5, 7
11	1, 4, 5, 7	31	2, 3, 5, 6
12	1, 3, 5, 6	32	1, 3, 5, 7
13	1, 4, 5, 6	33	1, 4, 5, 6
14	2, 4, 5, 7	34	2, 4, 5, 7
15	1, 4, 5, 6	35	2, 3, 5, 6
16	2, 3, 5, 6	36	1, 3, 5, 6
17	1, 4, 5, 7	37	1, 4, 5, 7
18	2, 3, 5, 6	38	2, 3, 5, 7
19	1, 4, 5, 6	39	1, 3, 5, 6
20	2, 4, 5, 7	40	2, 4, 5, 7

Контрольные вопросы к защите лабораторной работы:

1. Понятие точного и приближенного решений нелинейного уравнения.
2. Основная идея метода половинного деления?
3. Может ли метод половинного деления найти точное значение корня уравнения?
4. В чем суть метода Ньютона?
5. Как выбирается начальное приближение для метода Ньютона?
6. Идея метода хорд?
7. Как выбирается начальное приближение для метода хорд с фиксированным концом интервала изоляции корня?
8. По каким причинам методы хорд и касательных предпочтительнее метода простой итерации?
9. Какой из методов является трехшаговым методом? Как запустить этот метод?
10. В чем суть метода простой итерации?
11. Каковы условия применяемости метода простой итерации?
12. Как правильно преобразовать исходное нелинейное уравнение $y = f(x)$ к виду $x = \varphi(x)$?
13. Каковы основные критерии окончания итерационного процесса?
14. Как оценить необходимое количество итераций в методе биссекции при заданной точности?
15. Алгоритм решения системы нелинейных уравнений методом Ньютона?
16. Каковы преимущества и недостатки графического метода отделения решения для системы двух нелинейных уравнений?
17. В каких случаях можно применить метод простой итерации для решения системы нелинейных уравнений?
18. Когда можно считать итерационный процесс законченным при использовании метода простой итерации для решения системы нелинейных уравнений?
19. Что такое сходимость и скорость сходимости численных методов?
20. Дайте определение устойчивости итерационного метода?

Ответы в Readme на GitHub

2. Цель лабораторной работы.

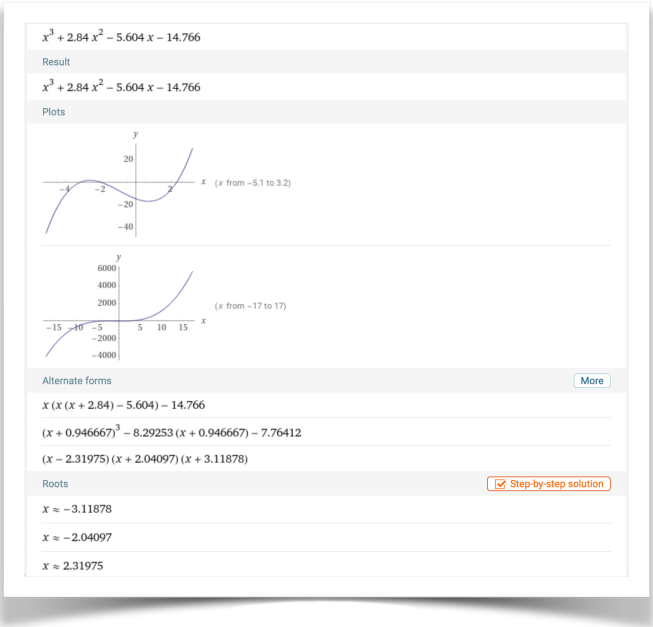
- изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

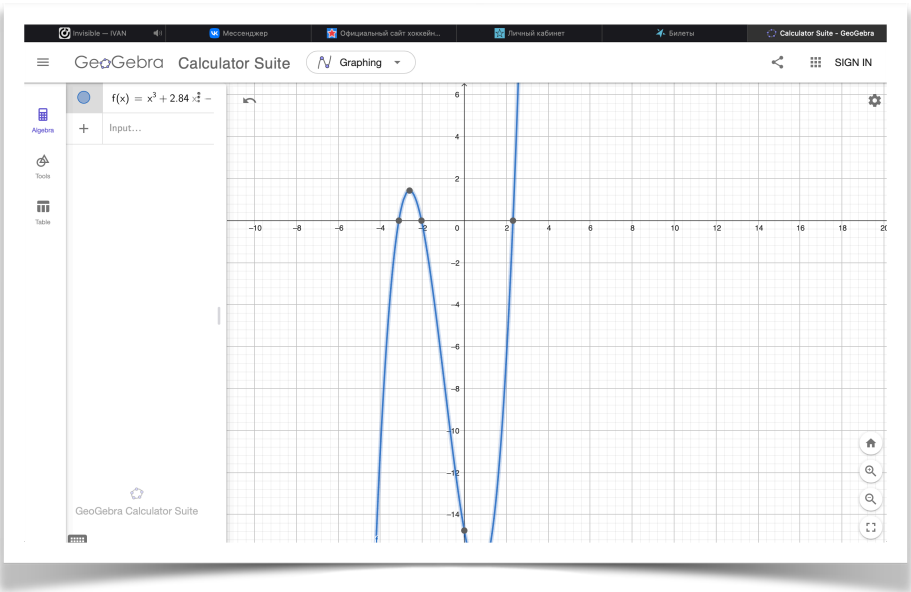
3. Мой вариант.

- Вариант номер 3;
- Уравнение: $x^3 + 2.84x^2 - 5.606x - 14.766$

No варианта	Крайний правый корень	Крайний левый корень	Центральный корень
3	1	5	3
3	Метод половинного деления	Метод простой итерации	Метод Ньютона

4. Решение.





1) Первым делом необходимо найти левый корень методом простых итераций.

Рабочая формула метода: $[a_0\ ;\ b_0]$ - исходный отрезок, $x_0 \in [a_0\ ;\ b_0]$

$x_0 = \frac{a_0 + b_0}{2}$, а критерий окончания: $|b_n - a_n| \le \varepsilon$ или $|f(x_n)| \le \varepsilon$

Буду искать решение с точностью $\varepsilon = 0.001$.
На графиках хорошо видно, что этот корень лежит между -4 и -3, поэтому и возьмем их для старта:

N_0 итерации	a	b	x	$F(a)$	$F(b)$	$F(x)$	$\lvert a-b \rvert$
0	-4,00000	-3,00000	-3,50000	-10,91000	0,60600	-3,23700	1,00000
1	-3,50000	-3,00000	-3,25000	-3,23700	0,60600	-0,88363	0,50000
2	-3,25000	-3,00000	-3,12500	-0,88363	0,60600	-0,03670	0,25000
3	-3,12500	-3,00000	-3,06250	-0,03670	0,60600	0,30944	0,12500
4	-3,12500	-3,06250	-3,09375	-0,03670	0,30944	0,14266	0,06250
5	-3,12500	-3,09375	-3,10938	-0,03670	0,14266	0,05456	0,03125
6	-3,12500	-3,10938	-3,11719	-0,03670	0,05456	0,00933	0,01563
7	-3,12500	-3,11719	-3,12109	-0,03670	0,00933	-0,01359	0,00781
8	-3,12109	-3,11719	-3,11914	-0,01359	0,00933	-0,00211	0,00391
9	-3,11914	-3,11719	-3,11816	-0,00211	0,00933	0,00362	0,00195
10	-3,11914	-3,11816	-3,11865	-0,00211	0,00362	0,00076	0,00098

$$x* = \frac{a_{10} + b_{10}}{2} = -3.118$$

2) Теперь методом Ньютона вынужден искать центральный корень.

Идея метода: функция $y = f(x)$ на отрезке [a, b] заменяется касательной и в качестве приближенного значения корня $x* = x_n$ принимается точка пересечения касательной с осью абсцисс. Рабочая формула метода:

$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$, а критерий окончания: $|f(x_n)| \le \varepsilon$

Буду искать ответ с точностью 0.00001.

N_0 итерации	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	$\lvert x_{n+1} - x_n \rvert$
0	-2,500000	1,369000	-1,054000	-1,201139	1,298861
1	-1,201139	-5,670379	-8,098266	-1,901335	0,700197

N_0 итерации	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	$\lvert x_{n+1} - x_n \rvert$
2	-1,901335	-0,717573	-5,558357	-2,030433	0,129098
3	-2,030433	-0,049884	-4,768883	-2,040894	0,010460
4	-2,040894	-0,000357	-4,700535	-2,040970	0,000076

$x* = -2.04097$

3) правый корень меня заставили искать методом простой итерации.
Уравнение $f\ x = 0$ приведем к эквивалентному виду: $x = \varphi(x)$, выразив x из исходного уравнения.

Зная начальное приближение: $x_0 \in a, b$, найдем очередные приближения: $x_1 = \varphi(x_0) \rightarrow x_2 = \varphi(x_1)$...

Рабочая формула метода: $x_{i+1} = \varphi(x)$

$x^3 + 2.84x^2 - 5.606x - 14.766$

Преобразую к виду : $x = \varphi(x)$.

$$\varphi(x) = \frac{x^3 + 2.84x^2 - 14.766}{5.606}$$

$\varphi'(x) = \frac{3x^2 + 2 \cdot 2.84x}{5.606}$ подставим $x = 2$. $\varphi'(2) = \frac{3 \cdot 2^2 + 2 \cdot 2.84 \cdot 2}{5.606}$>1. Способ не работает.

$$x = x + \lambda f(x), \quad \varphi(x) = x + \lambda f(x), \quad \varphi'(x) = 1 + \lambda f'(x).$$

$$\lambda = -\frac{1}{max_{[a,b]} |f'(x)|}, \qquad f'(2) = 3 \cdot 2^2 + 2 \cdot 2.84 \cdot 2 - 5.604 = 17.756$$

$f(3) = 38{,}436$

$$\lambda = -\frac{1}{38.436}. \quad x = x + \lambda x = -\frac{x^3}{38.436} - \frac{2.84x^2}{38.436} + \frac{44.04x}{38.436} + \frac{14.766}{38.436}$$

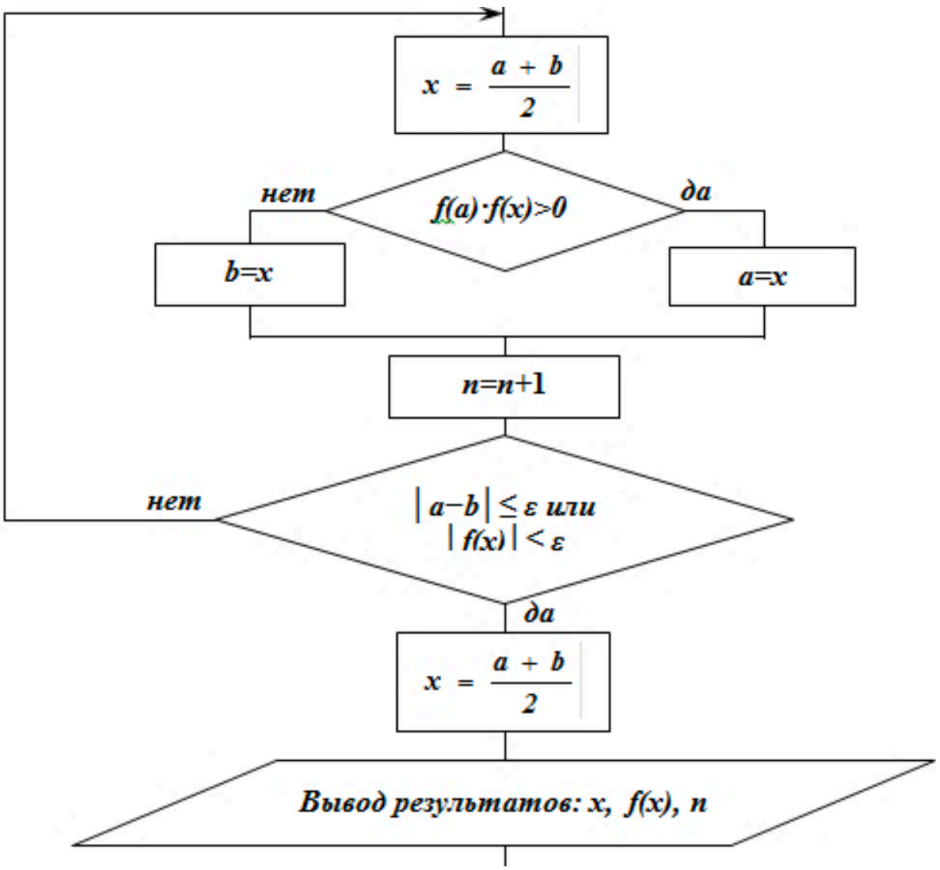
No итерации	x_i	x_{i+1}	$\varphi(x_{i+1})$	$f(x_{i+1})$	$\lvert x_{i+1} - x_i \rvert$
0	3,00000	2,45411	2,36654	3,36074	0,54589
1	2,45411	2,36654	2,33711	1,12647	0,08757
2	2,36654	2,33711	2,32632	0,40995	0,02943
3	2,33711	2,32632	2,32226	0,15161	0,01079
4	2,32632	2,32226	2,32071	0,05484	0,00407
5	2,32226	2,32071	2,32012	0,01809	0,00155
6	2,32071	2,32012	2,31989	0,00870	0,00059

$x* = 2.320$

No варианта	Методы			
3	1	4	5	6
3	Метод половинного деления	Метод секущих	Метод простой итерации	Метод Ньютона

Метод половинного деления.

Начальный интервал изоляции корня делим
пополам, получаем начальное приближение к корню: xo = (ao+bo)/2
И продолжим так делить до тех пор, пока |bn-an|>epsilon, конечно же, не забывая про теорему у существовании корня. (она поможет понять, какой границе интервала присвоить значение xi)



Примеры:

half
-〉f or in
f
-〉type the link
test1
-〉Enter interval as 2 numbers like this: 'a b'
-2 -1.5
-〉Enter epsilon
0.01
-〉-1.7890625

half
-〉f or in
f
-〉type the link
test2
-〉Enter interval as 2 numbers like this: 'a b'
2 3
-〉Enter epsilon
0.001
-〉2.3193359375

Метод секущих

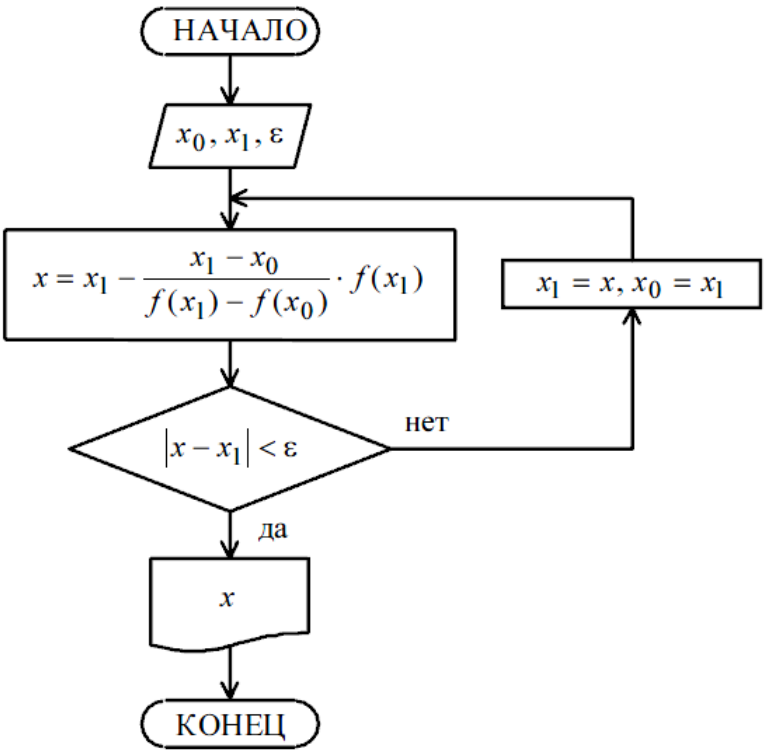
функция $y = f(x)$ на отрезке [a, b] заменяется касательной и в качестве приближенного значения корня
$x* = xn$ принимается точка пересечения касательной с осью абсцисс.

Упростим метод Ньютона, заменив $f'(x)$ разностным приближением:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Рабочая формула метода:

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i) \qquad i = 1,2 \ldots$$




```python
def secant_method(equation, interval):
    check_interval(equation, interval)
    epsilon = float(input("Enter epsilon\n").replace(",", "."))
    equation = list(equation)
    x_prev = interval[0]
    x_next = interval[1]

    while abs(x_next - x_prev) > epsilon:
        tmp = x_next
        x_next = x_next - (1 / iteration_derivative(equation, x_next, x_prev)) * function(equation, x_next)
        x_prev = tmp

    return x_next
```

Примеры работы:

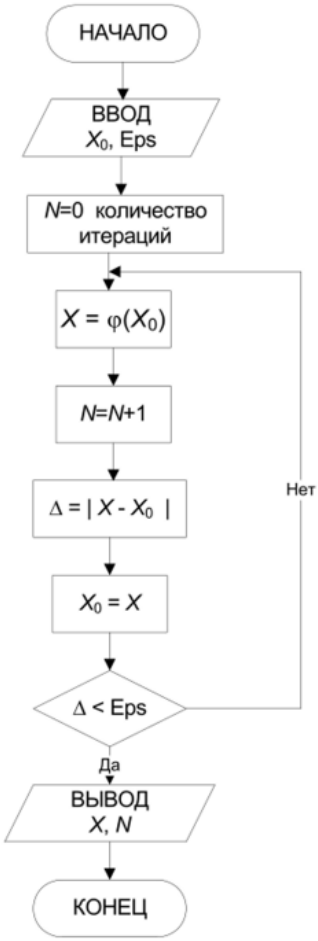
secant
->f or in
f
->type the link
test1
->Enter interval as 2 numbers like this: 'a b'
-1 -2
->Enter epsilon
0.01
->-1.7963084498052133

secant
-〉f or in
f
-〉type the link
test2
-〉Enter interval as 2 numbers like this: 'a b'
-3 -4
-〉Enter epsilon
0.001
-〉-3.1198434979214 3

Метод простой итерации

Уравнение $f\ x = 0$ приведем к эквивалентному виду: $x = \varphi(x)$, выразив x из исходного уравнения.
Зная начальное приближение: $x0 \in a, b$, найдем очередные приближения: $x1 = \varphi(x0) \rightarrow x2 = \varphi x1$ …
Важен выбор начального приближения к корню из малой окрестности для сходимости.
Достаточное условие сходимости метода:
$|\varphi'(x)| \leq q < 1$, где q – некоторая константа (коэффициент Липшица или коэффициент сжатия)
Чем меньше q, тем выше скорость сходимости.

```python
def simple_iteration(equation, interval):
    check_interval(equation, interval)
    epsilon = float(input("Enter epsilon\n").replace(",", "."))
    equation = list(equation)
    a = interval[0]
    b = interval[1]
    derivative_eq = derivative(equation)
    if (function(derivative_eq, a)) > function(derivative_eq, b):
        max_derivative = function(derivative_eq, a)
        xi = a
    else:
        max_derivative = function(derivative_eq, b)
        xi = b
    lambda_var = - 1 / max_derivative
    phi = equation[:]
    for i in range(len(equation)):
        phi[i] *= lambda_var
    if len(phi) < 2:
        phi.append(1)
    else:
        phi[1] += 1

    while abs(function(phi, xi) - xi) > epsilon:
        xi = function(phi, xi)

    return function(phi, xi)
```



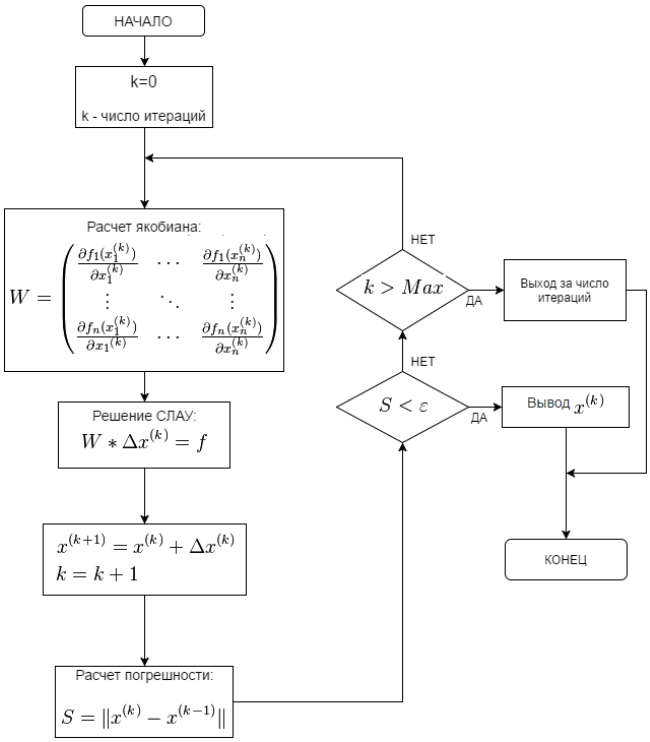
Примеры:

simple
->f or in
f
->type the link
test1
->Enter interval as 2 numbers like this: 'a b'
-2 -1,5
->Enter epsilon
0.01
->-1.797235072258232 4

simple
->f or in
f
->type the link
test2
->Enter interval as 2 numbers like this: 'a b'
-2.2 -1
->Enter epsilon
0.01
->-2.040943584702947 3

Метод Ньютона.

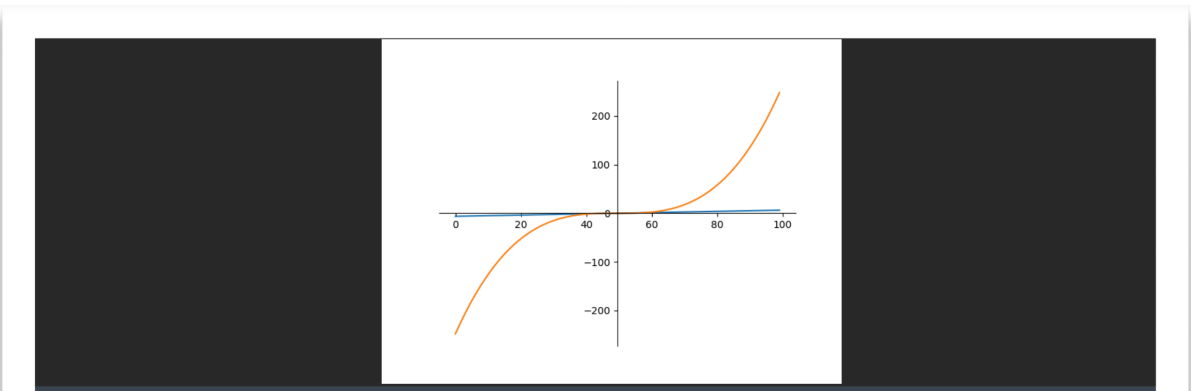
В основе метода лежит использование разложения функций $Fi\ x1, x2, \ldots, xn$ в ряд Тейлора в окрестности некоторой фиксированной точки, причем члены, содержащие вторые (и более высоких порядков) производные, отбрасываются.
Пусть начальные приближения неизвестных системы (1) получены и равны соответственно $a1, a2, \ldots, an$. Задача состоит в нахождении приращений (поправок) к этим значениям $\Delta x1$, $\Delta x2, \ldots, \Delta xn$, благодаря которым решение системы запишется в виде
$x1=a1+\Delta x1\ ,x2=a2+\Delta x2\ ,\ldots,xn=an+\Delta xn$



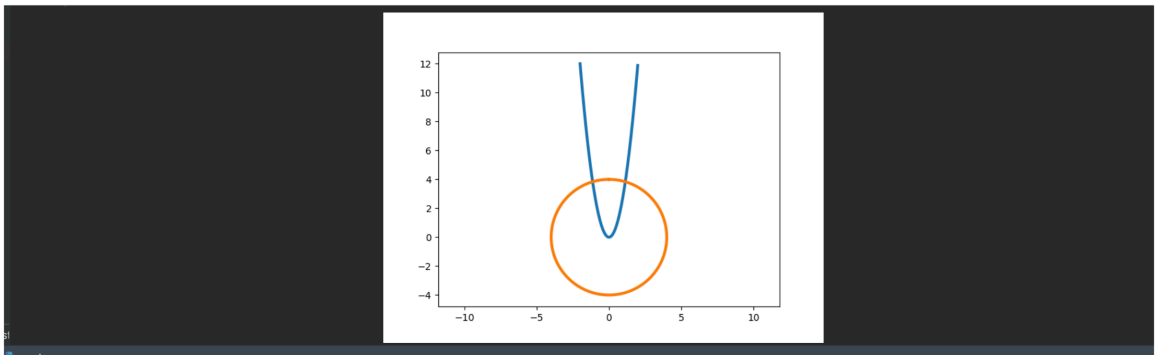

```python
from computational_math_lib import *


def system1():
    epsilon = float(input("Enter epsilon\n").replace(",", "."))
    print("x*x + y*y = 4")
    print("y = 3*x*x")
    a = np.arange(-2, 2, 0.01)
    t = np.arange(0, 2 * np.pi, 0.01)
    r = 4
    plt.plot(a, 3 * a * a, r * np.sin(t), r * np.cos(t), lw=3)
    plt.axis('equal')
    plt.show()

    """
       Jacobian:
       2x   2y
       -6x  1

       =>

       2xdx + 2ydy = 4 - x*x - y*y
       -6xdx + dy = 3*x*x - y
     """

    user_input = input("Enter x0, y0\n").replace(",", ".").split()
    x_first, y_first = parse_float_array(user_input)
    iteration_sys1(x_first, y_first, epsilon)


def generate_matrix_sys1(x, y):
    return [[2 * x, 2 * y, 4 - x * x - y * y], [-6 * x, 1, 3 * x * x - y]]


def iteration_sys1(x, y, epsilon):
    dx, dy = gauss(generate_matrix_sys1(x, y))
    xi = x + dx
    yi = y + dy

    while abs(x - xi) > epsilon or abs(y - yi) > epsilon:
        iteration_sys1(xi, yi, epsilon)
        return

    print("|x-xi|: " + str(abs(x - xi)) + " |y-yi|: " + str(abs(y - yi)))
    x = xi
    y = yi
    print("x: " + str(x) + " y: " + str(y))
    print("Check: x_i, y_i -> equations")
    print(x * x + y * y - 4)
    print(y - 3 * x * x)
    return
```

```python
from computational_math_lib import *


def system2():
    epsilon = float(input("Enter epsilon\n").replace(",", "."))
    print("2x = y/(1 + y*y)")
    # 2*x + 2*x*y*y - y = 0
    print("2y = x/(1+x*x)")
    # 2*y + 2*y*x*x - x = 0
    # 100 linearly spaced numbers
    x = np.linspace(-np.pi, np.pi, 100)
    t = np.linspace(-np.pi / 1000, np.pi / 1000, 100)

    # the functions, which are y = sin(x) and z = cos(x) here
    y = 2 * x + 2 * x * t * t - t
    z = 2 * x + 2 * x * y * y - y

    # setting the axes at the centre
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('center')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.yaxis.set_ticks_position('left')

    # plot the functions
    plt.plot(y)
    plt.plot(z)

    # show the plot
    plt.show()

    user_input = input("Enter x0, y0\n").replace(",", ".").split()
    x_first, y_first = parse_float_array(user_input)
    iteration_sys2(x_first, y_first, epsilon)


def iteration_sys2(x, y, epsilon):
    dx, dy = gauss(generate_matrix_sys2(x, y))
    xi = x + dx
    yi = y + dy

    while abs(x - xi) > epsilon or abs(y - yi) > epsilon:
        iteration_sys2(xi, yi, epsilon)
        return

    print("|x-xi|: " + str(abs(x - xi)) + " |y-yi|: " + str(abs(y - yi)))
    x = xi
    y = yi
    print("x: " + str(x) + " y: " + str(y))
    print("Check: x_i, y_i -> equations")
    print(-2 * x + y / (1 + y * y))
    print(-2 * y + x / (1 + x * x))
    return


def generate_matrix_sys2(x, y):
    return [[2 + 2 * y * y, 4 * x * y - 1, -2 * x - 2 * x * y * y + y],
            [4 * x * y - 1, 2 + 2 * x * x, -2 * y - 2 * y * x * x + x]]
```

Примеры:




```
main
/Users/demetrius/Desktop/Вычмат/Lab2/compLab2/venv/bin/python /Users/demetrius/Desktop/Вычмат/Lab2/compLab2/main.py
Enter Command or help
system2
Enter epsilon
0.01
2x = y/(1 + y*y)
2y = x/(1+x*x)
Enter x0, y0
-1 2
|x-xi|: 0.002028164134518169 |y-yi|: 0.0019475004061618416
x: -3.119489630932365e-08 y: -3.1619714794408844e-08
Check: x_i, y_i -> equations
3.077007782424649e-08
3.2044533279478064e-08
```




```
main
/Users/demetrius/Desktop/Вычмат/Lab2/compLab2/venv/bin/python /Users/demetrius/Desktop/Вычмат/Lab2/compLab2/main.py
Enter Command or help
system1
Enter epsilon
0.01
x*x + y*y = 4
y = 3*x*x
Enter x0, y0
-2 2
|x-xi|: 0.0020995966394228294 |y-yi|: 1.8478107932651255e-11
x: -0.7832153786475753 y: 1.8402657631320494
Check: x_i, y_i -> equations
4.4083060490862636e-06
-1.32249181448163e-05
```

Выводы:

В ходе лабораторной работы Я попробовал различные методы решения нелинейных уравнений. Узнал, что некогда использовавшийся способ из доказательств немалого количества теорем уже запрограммирован и используется в вычислительной математике (ПД), кроме того попробовал и метод простой итерации, который работает довольно просто, но необходимость выбора достаточно малого интервала не может радовать. Как любитель оптимизаций, не могу не отметить скорость сходимости метода ньютона, но, к сожалению, для него необходимо считать каждый раз производную… В общем случае это сильно бьет по производительности, но для многочленов этот не так сильно заметен.
Во второй части лабораторной работы я попробовал решить систему нелинейных уравнений методом Ньютона. Слабо понимаю смысл такой работы, тк уравнение в общем виде ввести нереально, а пересчитывать корни одной и той же системы никто не захочет… но в ознакомительных целях нормально, скорее всего. Кроме того, приятно знать, что все, чему на продмате СППО уделяли столько времени, пригодилось!