Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

**Лабораторная работа по вычислительной математике №2**

Численное решение нелинейных уравнений и систем

Преподаватель: Малышева Татьяна Алексеевна

Выполнила: Голованова Дарья Владимировна

Группа: Р3222

Санкт-Петербург,

2022г

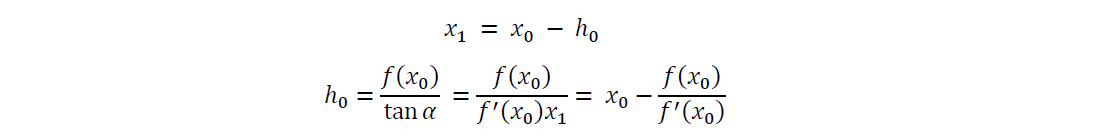
# Цель работы

Реализовать метод секущих и метод простой итераций для решения нелинейных уравнений и реализовать решение систем линейных уравнений методом простой итерации

# Описание использованного метода

Метод секущих:

Суть метода заключается в том, что функция 𝑦 = 𝑓(𝑥) на отрезке [𝑎, 𝑏] заменяется касательной, а в качестве приближенного значения корня 𝑥 ∗ = 𝑥𝑛 принимается точка пересечения касательной с осью абсцисс.



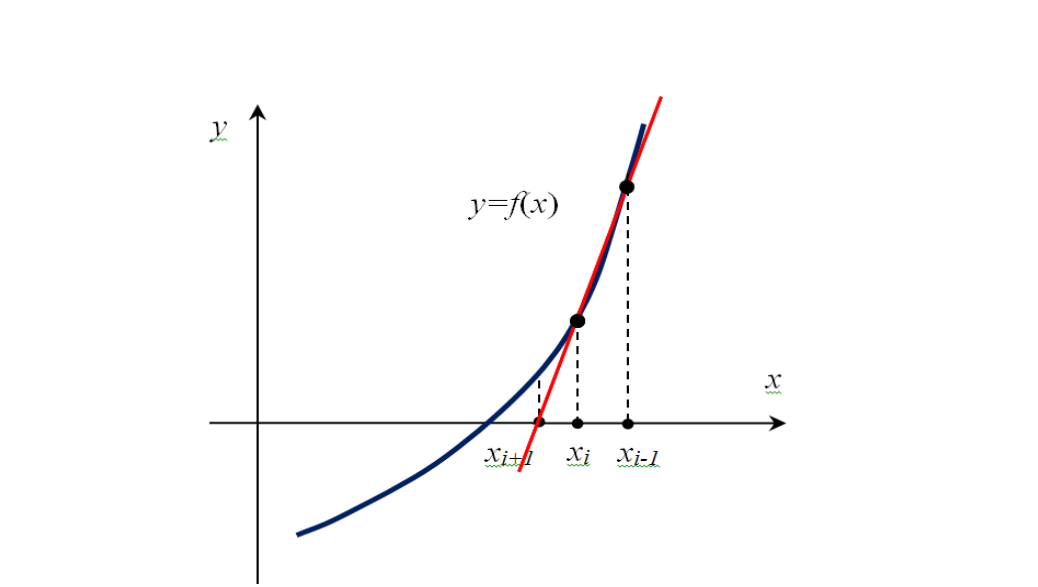
Однако, вышесказанное относится к методу Ньютона. Для того, чтобы получить метод секущих, нужно:

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

Рабочая формула метода:

Метод секущих является двухшаговым, т.е. новое приближение 𝑥 +1 определяется двумя предыдущими итерациями 𝑥𝑖 и 𝑥 𝑖−1. Выбор 𝑥0 определяется, как и в методе Ньютона, 𝑥1- выбирается рядом с начальным самостоятельно.

Критерий окончания итерационного процесса: |𝑥𝑛−𝑥𝑛−1 |≤ 𝜀 или|𝑓(𝑥𝑛)𝑓′(𝑥𝑛)|≤ 𝜀 или |𝑓(𝑥𝑛)| ≤ 𝜀

**

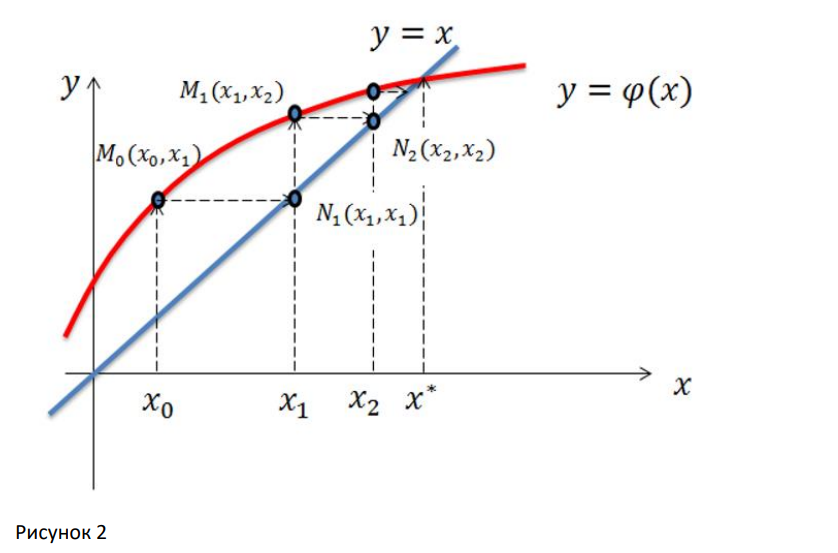
*Рисунок 1*

Метод простой итерации:

Суть метода заключается в том, что уравнение 𝑓(𝑥) = 0 с помощью некоторых преобразований необходимо переписать в виде 𝑥 = 𝜑(𝑥) (как показано на Рисунке 2).

Уравнение 𝑓(𝑥) = 0 эквивалентно уравнению 𝑥 = 𝑥 + 𝜆(𝑥)𝑓(𝑥) для любой функции 𝜆(𝑥) ≠ 0. Возьмем 𝜑(𝑥) = 𝑥 − 𝜆(𝑥)𝑓(𝑥) и выберем функцию (или переменную) 𝜆(𝑥) ≠ 0 так, чтобы функция 𝜑(𝑥) удовлетворяла необходимым условиям.

Для нахождения корня уравнения 𝑥 = 𝜑(𝑥) выберем некоторое начальное значение 𝑥0, которое должно находиться как можно ближе к корню уравнения. Дальше с помощью итерационной формулы 𝑥𝑛 + 1 = 𝜑(𝑥𝑛) будем находить каждое следующее приближение корня уравнения.



Рабочая формула метода:

𝑥𝑖+1 = 𝜑(𝑥𝑖)

Условия сходимости метода простой итерации определяются теоремой:

Если в некоторой 𝜎 - окрестности корня x\* уравнения 𝑓(𝑥) = 0 функция 𝑥 = 𝜑(𝑥) дифференцируема и удовлетворяет неравенству |𝜑′(𝑥)| < 𝑞, где 0 ≤ 𝑞 < 1 постоянная, то независимо от выбора начального приближения 𝑥0 из указанной 𝜎 окрестности итерационная последовательность 𝑥𝑛 не выходит из этой окрестности, метод сходится со скоростью геометрической прогрессии.

Достаточное условие сходимости метода:

|𝜑′(𝑥)| ≤ 𝑞 < 1, где 𝑞 – некоторая константа

Критерий окончания итерационного процесса:

|𝑥𝑛 – 𝑥𝑛−1| ≤ 𝜀

## Метод простой итерации для решения систем нелинейных уравнений:

Суть метода заключается в том, чтобы привести первоначальную систему уравнений к следующему виду:



Для этого нам необходимо задать начальное приближение 𝑥 (0) = (𝑥10, 𝑥20, …, 𝑥𝑛0) 𝑇 и малое положительное число 𝜀 (точность).

Затем вычислить 𝑥 (𝑘+1) по формуле 𝑥 (𝑘+1) = 𝜑1 (𝑥1 (𝑘), 𝑥2 (𝑘), … , 𝑥𝑛 (𝑘) ), так продолжать увеличивая k на единицу пока не будет достигнут критерий окончания итерационного процесса. Критерий завершения итерационного процесса: |𝑥𝑖 (𝑘+1) − 𝑥𝑖 (𝑘) | ≤ 𝜀, значит процесс завершен, 𝑥 ∗ = x (𝑘+1).

Выводы

В результате выполнения я изучила 5 методов решения нелинейных уравнений и пришла к следующим выводам относительно их преимуществ и недостатков:

1. Метод касательных:

Достоинства: Метод обладает квадратичной сходимостью.

Недостатки: Необходимость вычисления производной на каждой итерации.

1. Метод простой итерации:

Достоинства: Простота реализации

Недостатки: Сходимость метода в малой окрестности корня и вытекающая отсюда необходимость выбора начального приближения к корню из этой малой окрестности. В противном случае итерационный процесс расходится или сходится к другому корню этого уравнения. Также при |𝜑′(𝑥)| ≈ 1, то сходимость может быть очень медленной.

1. Метод секущих:

Достоинства: Меньший объем вычислений по сравнению с методом Ньютона, т.к. не требуется вычислять производную.

Недостатки: Порядок сходимости метода секущих ниже, чем у метода касательных и равен золотому сечению ≈1,618 (сверхлинейная).

1. Метод половинного деления:

Достоинства: Обладает абсолютной сходимостью (близость получаемого численного решения задачи к истинному решению.) Устойчив к ошибкам округления.

Недостатки: Если интервал содержит несколько корней, то неизвестно к какому относится вычислительный процесс. Медленный метод: имеет линейную сходимость. Имеет смысл применять в случаях, когда требуется высокая надежность счета, а скорость несущественна.

1. Метод хорд:

Достоинства: Простота реализации

Недостатки: Скорость сходимости – линейная. Порядок сходимости метода хорд выше, чем у метода половинного деления.

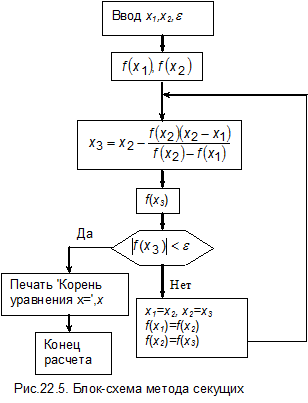
Что касается Методов решения систем нелинейных уравнений, то мною были изучены два метода решения и сделаны следующие выводы:

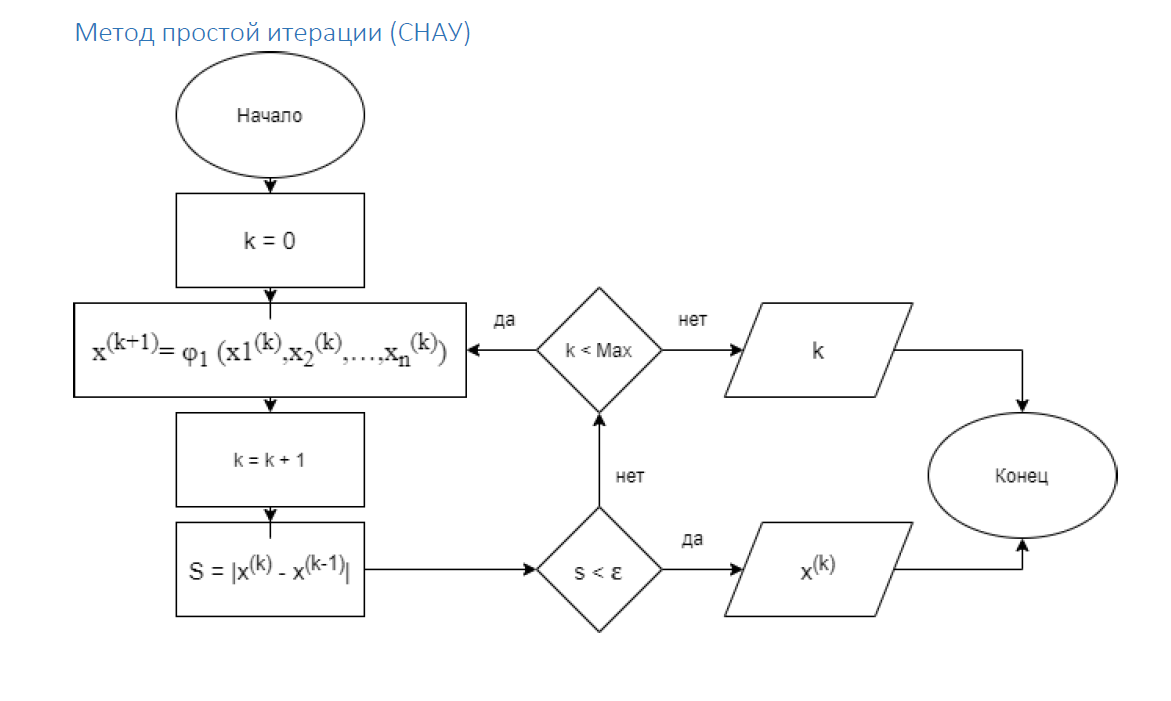
Метод Ньютона для решения СНАУ представляет собой обобщение метода Ньютона для решения НУ его сутью является попытка свести решение системы нелинейных уравнений к решению системы линейных уравнений. Основная сложность метода Ньютона заключается в обращении матрицы Якоби. Вводя обозначение 𝛥𝑥 (𝑘) = 𝑥 (𝑘+1) − 𝑥 (𝑘)𝛥𝑥 (𝑘) = 𝑥 (𝑘+1) − 𝑥 (𝑘) получаем СЛАУ для вычисления 𝛥𝑥 (𝑘). Решение этого СЛАУ создает основную вычислительную нагрузку алгоритма.

Метод простой итерации же в свою очередь позволяет, грубо говоря подобрать вектор решений системы уравнений путем выражения значения одной неизвестной через все остальные и постепенной подстановки значений неизвестных, вычисляемых на каждом шаге.

# Блок-схемы

## Метод секущих





# Листинг численных методов:

class NonLinearSystem:

    def iteration(fu1, fu2,  eps: float = 1e-7, kmax: int = 1e3) -> float:

        fig = plt.subplots()

        fuplot = lambda x: (-1) \* np.arcsin(x + 15) + 1.15

        xplt = np.linspace(-20, 10, 100)

        plt.plot(xplt, fuplot(xplt), label='y = arcsin(x + 15) - 1.5')

        plt.plot(xplt, fu1(xplt), label='y = - 0.5 - cos(x-2)')

        plt.minorticks\_on()

        plt.grid(which='major',

                 color='k')

        plt.grid(which='minor',

                 color='k',

                 linestyle=':')

        plt.legend()

        x0, y0, d1, d2, i = 0, 0, 1, 1, 0

        while 1:

            x, y = fu1(y0), fu2(x0)

            d1, d2 = fu1(x) - x, y - fu2(y)

            x0, y0, i = x, y, i + 1

            plt.scatter(y, fuplot(y), color='black')

            if not (abs(d1) > eps and abs(d2) > eps and i < kmax):

                break

        plt.scatter(y, fuplot(y), color='red')

        plt.show()

        return x, y, i

class NonLinearEquation:

    def secant(f: Callable[[float], float], x0: float, eps: float = 1e-7, kmax: int = 1e3, left: float = -5,

               right: float = 5) -> float:

        x, x\_prev, i = x0, x0 + 2 \* eps, 0

        while abs(x - x\_prev) >= eps and i < kmax:

            x, x\_prev, i = x - f(x) / (f(x) - f(x\_prev)) \* (x - x\_prev), x, i + 1

        fig = plt.subplots()

        xplt = np.linspace(left, right, 100)

        plt.plot(xplt, f(xplt), label="secant")

        plt.minorticks\_on()

        plt.grid(which='major',

                 color='k')

        plt.grid(which='minor',

                 color='k',

                 linestyle=':')

        plt.scatter(x, f(x), color='red')

        plt.legend()

        plt.show()

        return x, i

    def iteration(f: Callable[[float], float], x0: float, eps: float = 1e-7, kmax: int = 1e3) -> float:

        x, x\_prev, i = x0, 0, 0

        fig = plt.subplots()

        xplt = np.linspace(-5, 5, 100)

        plt.plot(xplt, f(xplt), label="iteration")

        plt.minorticks\_on()

        plt.grid(which='major',

                 color='k')

        plt.grid(which='minor',

                 color='k',

                 linestyle=':')

        while abs(x - x\_prev) >= eps and i < kmax:

            x\_prev, x, i = x, x - 0.003 \* f(x), i + 1

            i = i + 1

            plt.scatter(x, f(x), color='black')

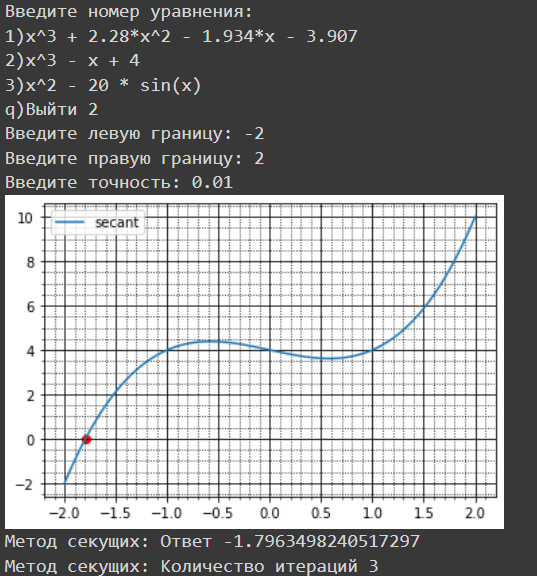
        plt.scatter(x, f(x), color='red')

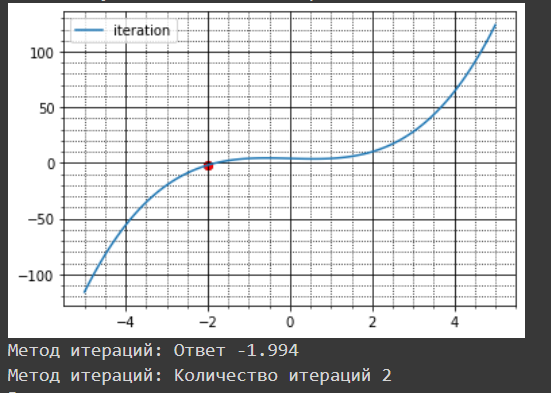
        plt.legend()

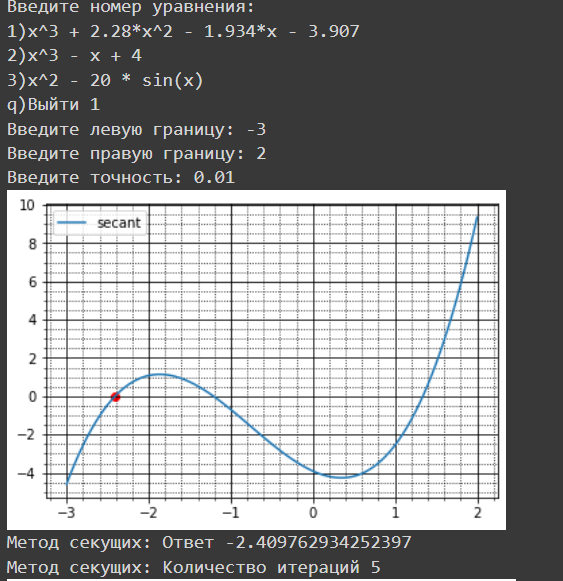
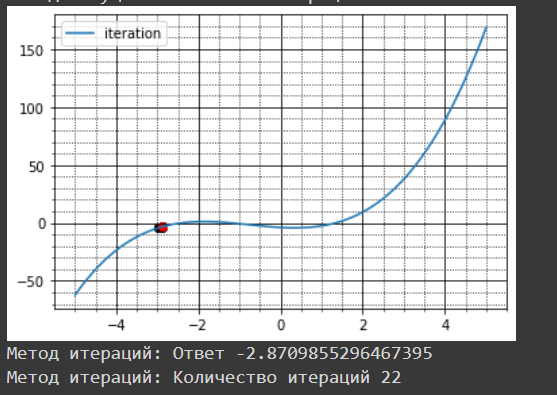
        plt.show()

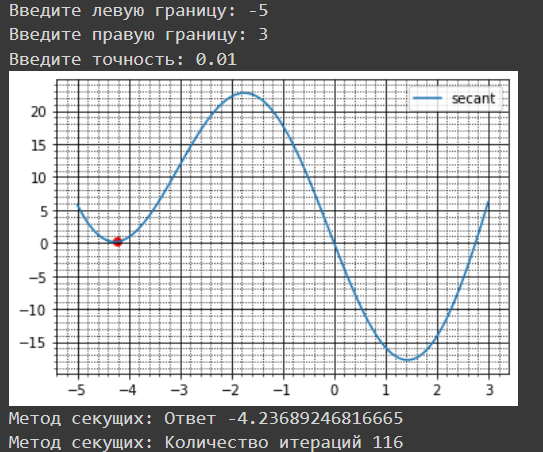
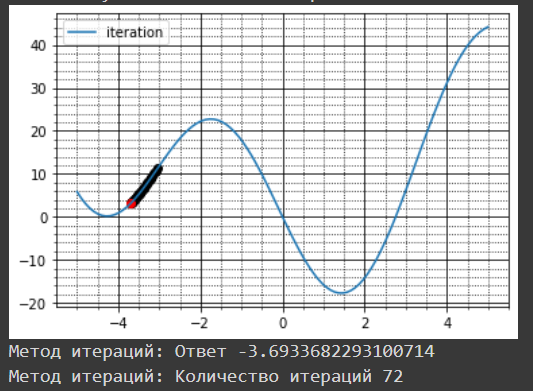
        return x, i

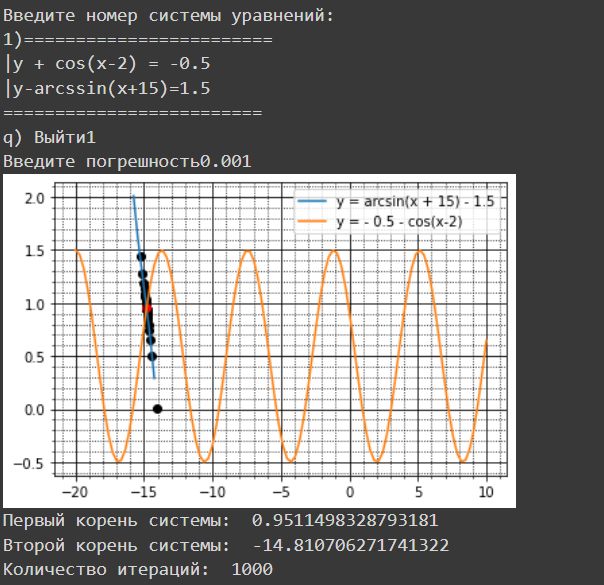
# Примеры







# Таблицы

## Уточнение корня уравнения методом половинного деления (хорд)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 1 | -2.5 | -2 | -2.25 | -0.4470 | 1.0809 | 0.5963 | 0.5 |
| 2 | -2.25 | -2 | -2.125 | 0.5963 | 1.0809 | 0.9026 | 0.25 |
| 3 | -2,125 | -2 | -2,0625 | 0.9026 | 1.0809 | 1.0070 | 0.125 |
| 4 | -2,125 | -2,0625 | -2.09375 | 0.9026 | 1.0070 | 0.9587 | 0.0625 |
| 5 | -2,125 | -2.09375 | -2.109375 | 0.9026 | 0.9587 | 0.9317 | 0.03125 |
| 6 | -2,125 | -2.109375 | -2.117187 | 0.9026 | 0.9317 | 0.9174 | 0.015625 |
| 7 | -2,125 | -2.117187 | -2.12109375 | 0.9026 | 0.9174 | 0.91012 | 0.0078125 |

## Уточнение корня уравнения методом Ньютона

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f*(*xk* ) | *f '*(*xk*) | *xk*+1 | │*xk* − *xk*+1│ |
| 1 | -1 | -0.6930 | -5.494 | -1.1261 | 0.12613 |
| 2 | -1.1261 | -0.2658 | -5.8009 | -1.1719 | 0.04583 |
| 3 | -1.1719 | -0.1187 | -5.9045 | -1.1920 | 0.02010 |
| 4 | -1.1920 | -0.0557 | -5.9486 | -1.2013 | 0.00937 |

## Уточнение корня уравнения методом простой итерации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f*(*xk* ) | *xk*+1 |  | │*xk* − *xk*+1│ |
| 1 | -3 | -4.429654 | -2.98624 | -2.972956 | 0.0279 |
| 2 | -2.986245 | -4.281972 | -2.972956 | -2.960110 | 0.0026 |
| 3 | -2.972956 | -4.14144 | -2.96011 | -2.947685 | 0.0127 |
| 4 | -2.96011 | -4.00759 | -2.947685 | -2.93566 | 0.0124 |
| 5 | -2.947685 | -3.8800 | -2.93566 | -2.92402 | 0.012 |