МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Решение нелинейных алгебраических уравнений**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧ»

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Сенокосова Владислава Владимировича

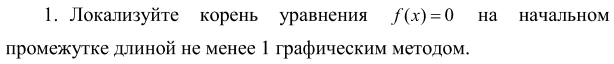
|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Преподаватель  Аспирант | \_\_\_\_\_\_\_\_\_\_ | В.M.Шкатов |
|  | подпись, дата |  |

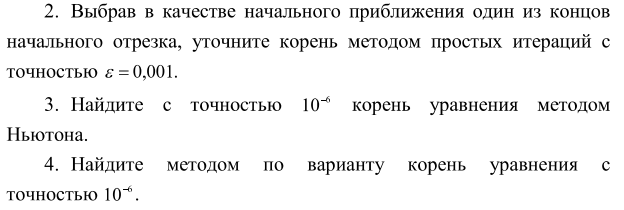
Саратов 2024

**Вариант 19**

****

**Задания**

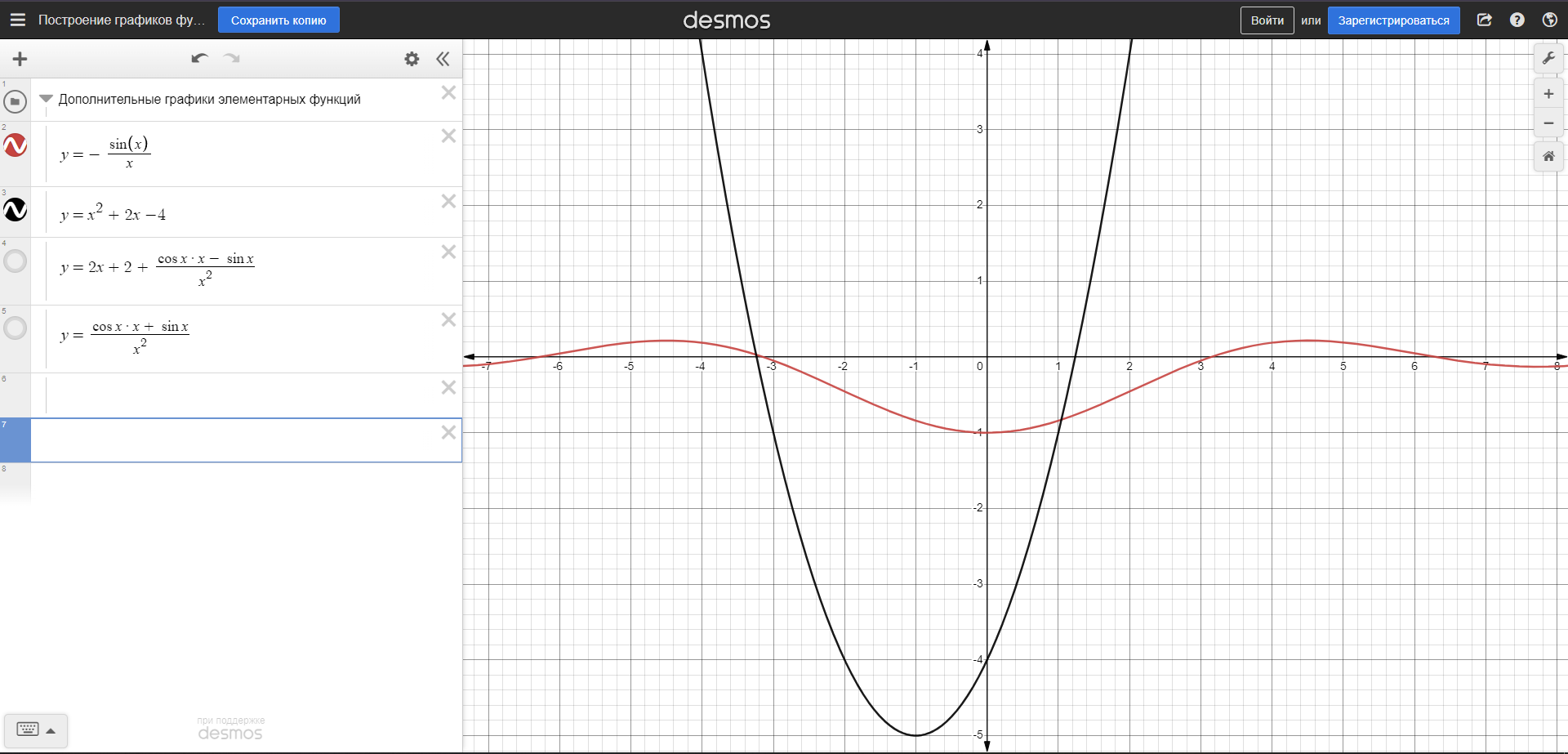


****Так как вариант 19 будет использоваться метод «лоцмана».

**Задание 1:**

Локализируем корень уравнения ­2 -1– на начальном промежутке длиной не менее 1 графическим методом. Построим график нашей функции, с использованием стороннего сервиса.

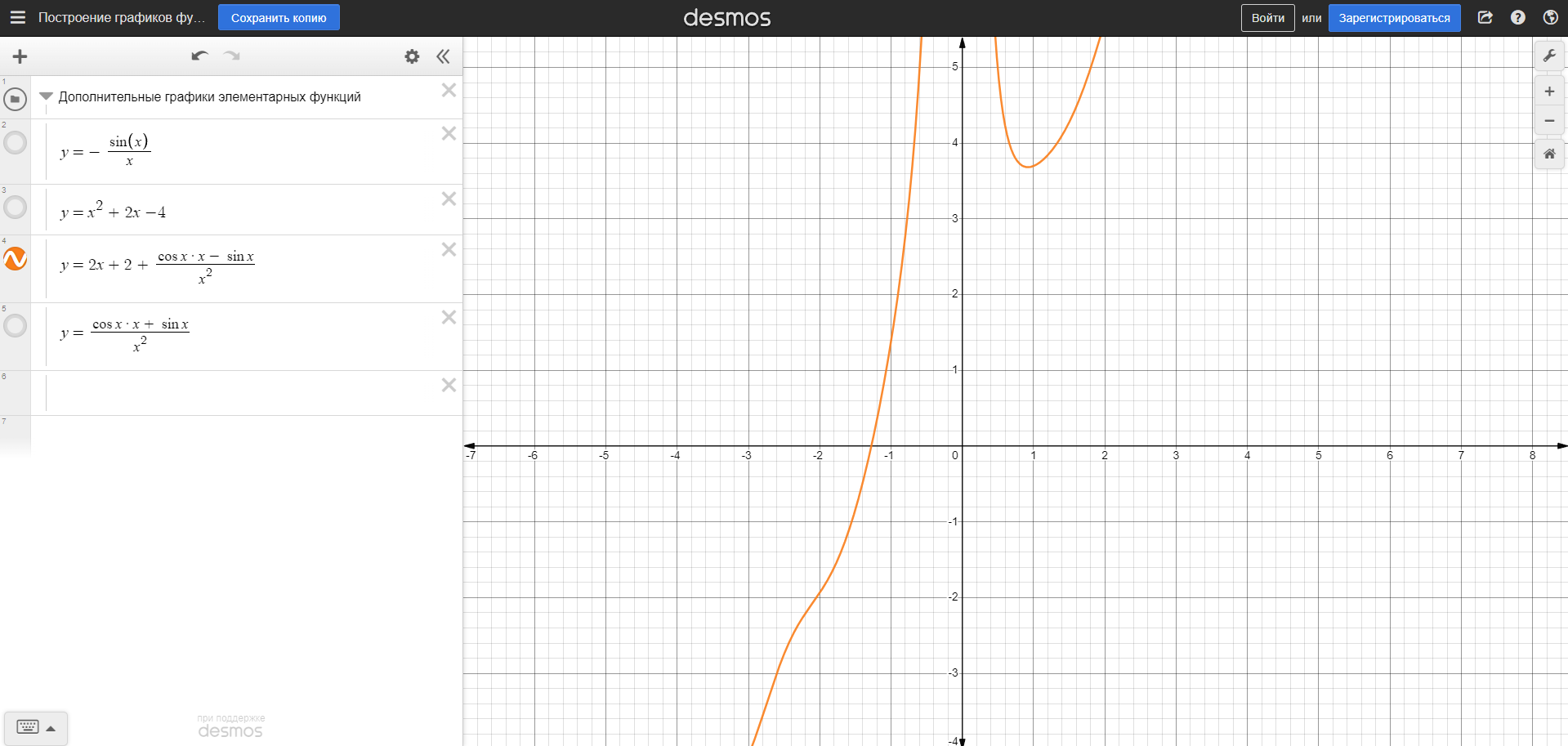
На первом скриншоте изображена функция после приведения ее к тождественному виду (тождественный вид представлен также на скриншоте):



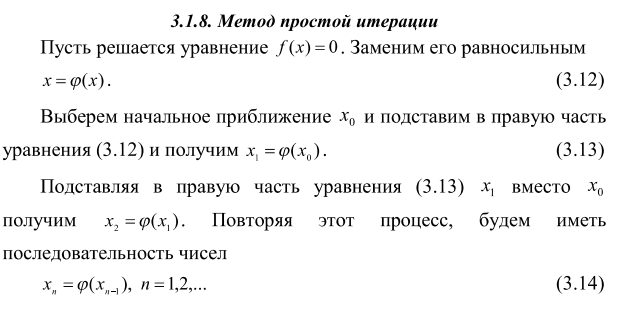
Нас будут интересовать значения в тех точках, в которых наши функции пересекаются, если мы сможем найти такой , то это и будет приближенным с некоторой точностью ответом.

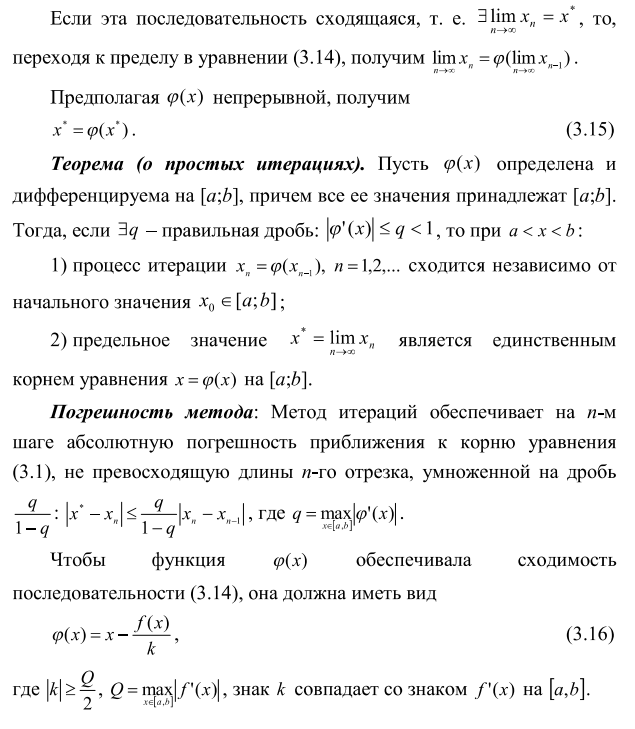
Таким образом локализировать корни можно путем перечисления области, в которой они находятся. В нашем случае это отрезки .

Вычислим также производную нашей функции:



**Задание 2:**

****

****

Применим метод простых итераций с приближением 0.001. Программная реализация была выполнена на языке Python, и функция, отвечающая за нахождения корней этим методом, возвращает корни как положительные, так и отрицательные. При решении лабораторной работы использовалась библиотека sympy, которая позволяет работать с функциями напрямую.

def method\_iteration(func, x, range\_roots):

    diff\_1 = sp.diff(func, x)

    lst = {diff\_1.subs({x: root}).evalf():root for root in range\_roots}

    Q = abs(list(lst.keys())[0])

    flag = False

    for i in list(lst.keys())[1:]:

        if abs(i) > Q:

            Q = abs(i)

        if i < 0:

            flag = True

    if flag: k = -int(Q / 2 + 0.99)

    else: k = int(Q / 2 + 0.99)

    phi\_x = x - func / k

    diff\_phi\_x = sp.diff(phi\_x, x)

    dict\_roots = {abs(diff\_phi\_x.subs({x: root}).evalf()):root for root in range\_roots}

    q = max(dict\_roots.keys())

    eps = round((q / (1 - q) \* 0.0000001).evalf(), 4)

    roots = []

    end = dict\_roots[q]

    start = (phi\_x.subs({x: end})).evalf()

    i = 0

    while abs(end - start) > eps:

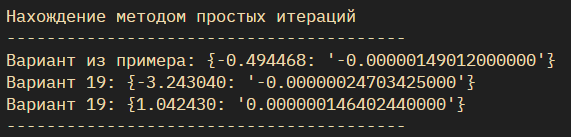
        start, end = (phi\_x.subs({x: start})).evalf(), start

        i += 1

    roots.append(round(start, 6))

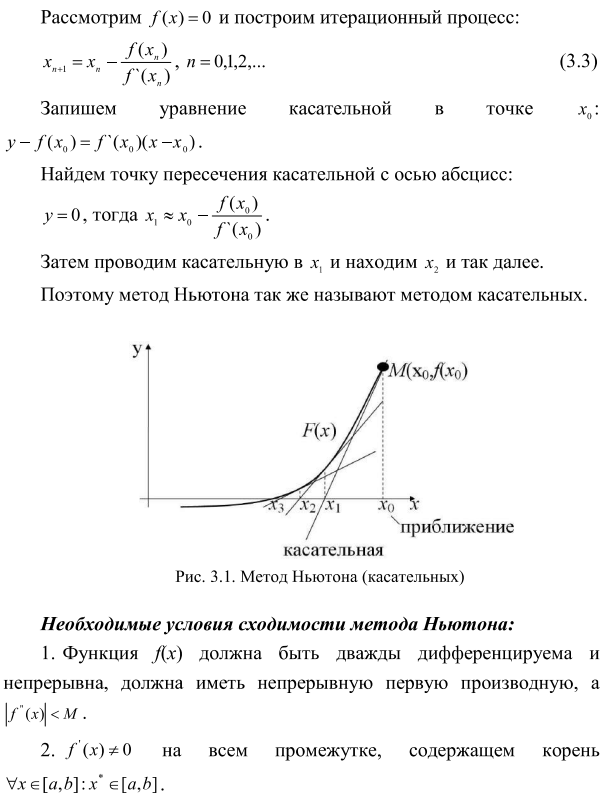
    return roots

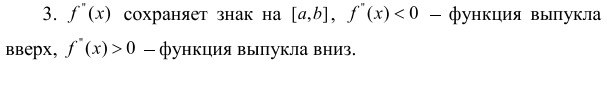
В качестве проверки корней с заданной точности при выводе идет подстановка в исходную функцию:

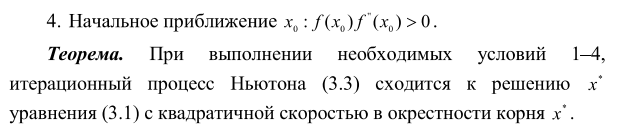


Первый элемент словаря — это найденный корень для заданного интервала, а значением ключа является подстановка найденного корня в исходную функцию. То есть можно заметить, что в заданной погрешности эти корни дают ноль при подстановке. Что говорит о том, что это корни нашей функции.

**Задание 3:**

****

****

****

Применим этот метод к нашей задаче. Так как в случае, если у нас несколько корней, то найден будет только один. Чтобы увеличить число возможный корней, я запускаю алгоритм на некотором участке положительном и отрицательном. Что позволяет найти по одному корню в двух частях.

Функция, отвечающая за метод Ньютона представлена следующим образом.

def method\_newton(func, x, range\_roots):

    eps = 0.000001

    diff\_1 = sp.diff(func, x)

    roots = []

    for border in range\_roots:

        start = border - 1

        end = border + 1

        i = 0

        while abs(end - start) > eps:

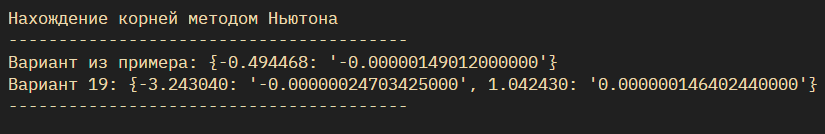
            start, end = start - (func.subs({x: start}) / diff\_1.subs({x: start})).evalf(), start

            i += 1

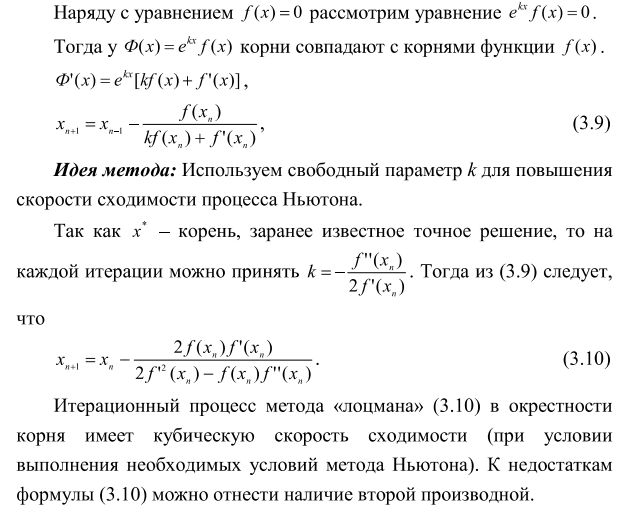
        roots.append(round(start, 6))

    return roots

Результаты вызова функции:



**Задание 4:**

****

Функция, реализующая данный метод:

def method\_lotsman(func, x, range\_roots):

    eps = 0.000001

    diff\_1 = sp.diff(func, x)

    diff\_2 = sp.diff(diff\_1, x)

    roots = []

    for border in range\_roots:

        start = border - 1

        end = border + 1

        i = 0

        while abs(end - start) > eps:

            up = 2 \* func.subs({x: start}) \* diff\_1.subs({x: start})

            down = 2 \* diff\_1.subs({x: start})\*\*2 - func.subs({x: start})\*diff\_2.subs({x: start})

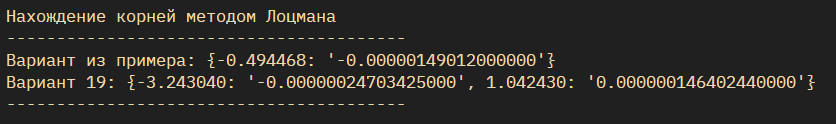
            start, end = (start - up / down).evalf(), start

            i += 1

        roots.append(round(start, 6))

    return roots

Результаты работы программы:



В результате проделанной работы были осуществлены различные методы нахождения корней нелинейных уравнений с заданной точностью.

**ПРИЛОЖЕНИЕ**

**Листинг программы**

import sympy as sp

def method\_iteration(func, x, range\_roots):

    diff\_1 = sp.diff(func, x)

    lst = {diff\_1.subs({x: root}).evalf():root for root in range\_roots}

    Q = abs(list(lst.keys())[0])

    flag = False

    for i in list(lst.keys())[1:]:

        if abs(i) > Q:

            Q = abs(i)

        if i < 0:

            flag = True

    if flag: k = -int(Q / 2 + 0.99)

    else: k = int(Q / 2 + 0.99)

    phi\_x = x - func / k

    diff\_phi\_x = sp.diff(phi\_x, x)

    dict\_roots = {abs(diff\_phi\_x.subs({x: root}).evalf()):root for root in range\_roots}

    q = max(dict\_roots.keys())

    eps = round((q / (1 - q) \* 0.0000001).evalf(), 4)

    roots = []

    end = dict\_roots[q]

    start = (phi\_x.subs({x: end})).evalf()

    i = 0

    while abs(end - start) > eps:

        start, end = (phi\_x.subs({x: start})).evalf(), start

        i += 1

    roots.append(round(start, 6))

    return roots

def method\_newton(func, x, range\_roots):

    eps = 0.000001

    diff\_1 = sp.diff(func, x)

    roots = []

    for border in range\_roots:

        start = border - 1

        end = border + 1

        i = 0

        while abs(end - start) > eps:

            start, end = start - (func.subs({x: start}) / diff\_1.subs({x: start})).evalf(), start

            i += 1

        roots.append(round(start, 6))

    return roots

def check\_roots(x, func, roots\_lst):

    dict\_roots = {}

    for root in roots\_lst:

        dict\_roots[root] = "{:0<20.15f}".format(func.subs({x: root}))

    return dict\_roots

def method\_lotsman(func, x, range\_roots):

    eps = 0.000001

    diff\_1 = sp.diff(func, x)

    diff\_2 = sp.diff(diff\_1, x)

    roots = []

    for border in range\_roots:

        start = border - 1

        end = border + 1

        i = 0

        while abs(end - start) > eps:

            up = 2 \* func.subs({x: start}) \* diff\_1.subs({x: start})

            down = 2 \* diff\_1.subs({x: start})\*\*2 - func.subs({x: start})\*diff\_2.subs({x: start})

            start, end = (start - up / down).evalf(), start

            i += 1

        roots.append(round(start, 6))

    return roots

if \_\_name\_\_ == "\_\_main\_\_":

*# Создание объектов функций*

    x = sp.Symbol("x")

    func = 2 - 0.5\*x\*\*2 - 0.5\*x\*\*-1\*sp.sin(x) - x

    func\_2 = 2\*x\*\*2 - x\*\*3 - sp.exp(x)

    range\_roots = [-100, 100]

*# В алгоритмах Ньютона и Лоцмана мы задаем некоторую*

*# область поиска корней в нашем примере от -100 до 100,*

*# если корень только один то он и будет выдан,*

*# а если есть два корня > 0 и < 0 то они будут найдены.*

    print()

    print("Нахождение корней методом Ньютона")

    print("----------------------------------------")

    roots = method\_newton(func\_2, x, range\_roots)

    print("Вариант из примера:", check\_roots(x, func\_2, roots))

    roots = method\_newton(func, x, range\_roots)

    print("Вариант 19:", check\_roots(x, func, roots))

    print("----------------------------------------\n")

    print("Нахождение корней методом Лоцмана")

    print("----------------------------------------")

    roots = method\_lotsman(func\_2, x, range\_roots)

    print("Вариант из примера:", check\_roots(x, func\_2, roots))

    roots = method\_lotsman(func, x, range\_roots)

    print("Вариант 19:", check\_roots(x, func, roots))

    print("----------------------------------------\n")

*# В алгоритме итераций необходимо первоначально знать*

*# область в которой необходимо осуществлять поиск*

*# Поэтому чтобы получить несколько корней*

*# надо знать диапозон значений. Их мы получаем*

*# путем граффического анализа функции.*

    print("Нахождение методом простых итераций")

    print("----------------------------------------")

    range\_roots = [-1, 2]

    roots = method\_iteration(func\_2, x, range\_roots)

    print("Вариант из примера:", check\_roots(x, func\_2, roots))

    range\_roots = [-4, -3]

    roots = method\_iteration(func, x, range\_roots)

    print("Вариант 19:", check\_roots(x, func, roots))

    range\_roots = [1, 2]

    roots = method\_iteration(func, x, range\_roots)

    print("Вариант 19:", check\_roots(x, func, roots))

    print("----------------------------------------")