Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Рязанский государственный радиотехнический университет имени В. Ф. Уткина»

Кафедра вычислительной и прикладной математики

Отчет
о лабораторной работе № 2
по дисциплине
«Вычислительные алгоритмы»
на тему
“Решение уравнений с одной переменной”

Выполнила: ст. гр. 343 Гаджиева А.В
Проверила:
доц. Проказникова Е.Н.
ас. Щенева Ю.Б.

Рязань 2025

Дата выполнения лабораторной работы: 09.03.2025

Задание

Найти решение уравнения с точностью $\varepsilon = 0.0001$ следующими методами:

– дихотомии;

– пропорциональных частей (хорд);

– касательных (Ньютона);

– модифицированным методом Ньютона;

– комбинированным методом;

– итерационным.

Найти наименьший положительный корень уравнения tg(x) = x

Анализ задания

1 этап: Отделение корня, удовлетворяющего заданию

Для этого протабулируем функцию с небольшим шагом и найдём самое первое значение x.

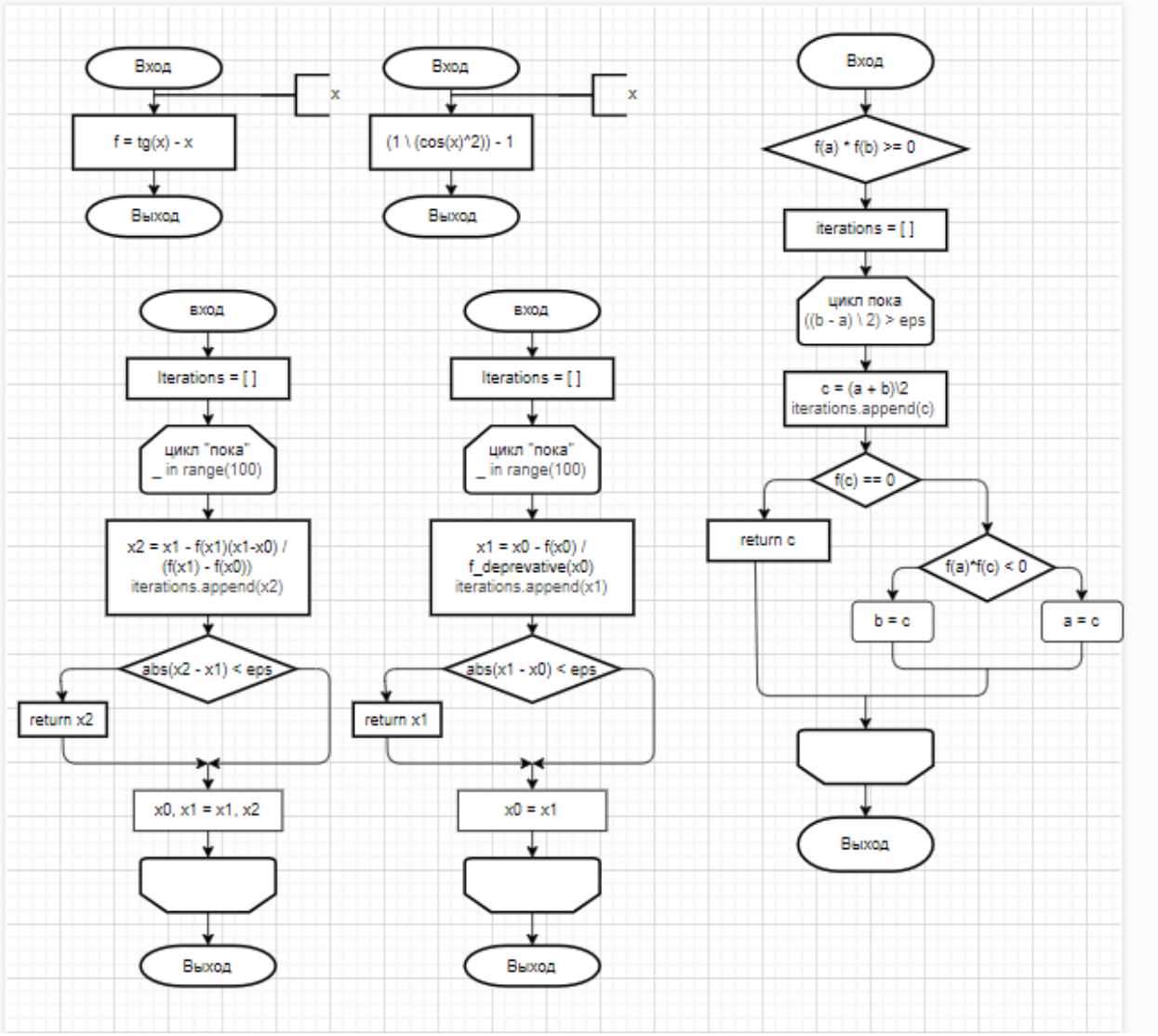
2 этап: Уточнение корня

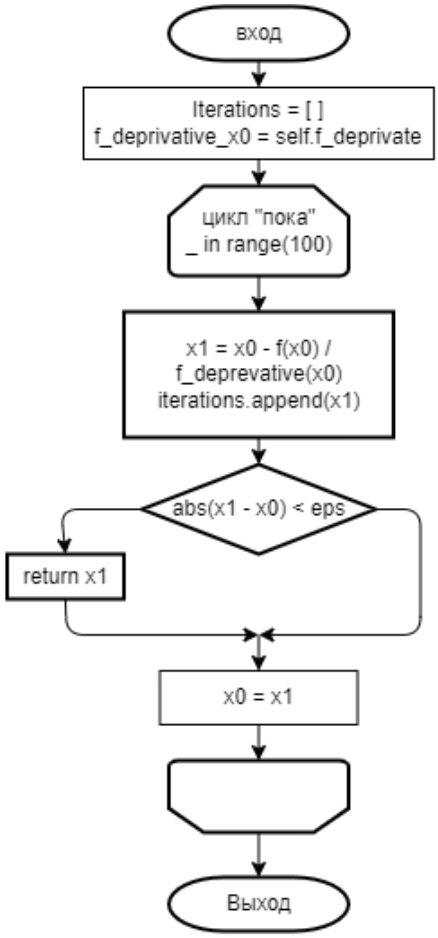
Зная отрезок, содержащий корень, можно запустить любой итерационный метод для уточнения корня

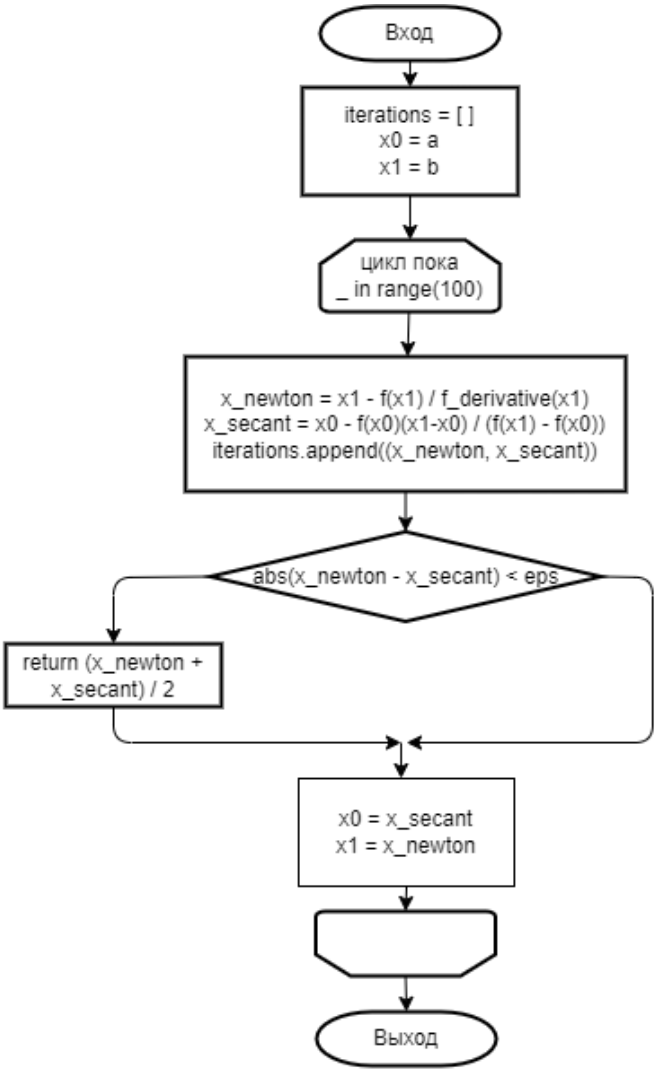
3 этап: Проверка

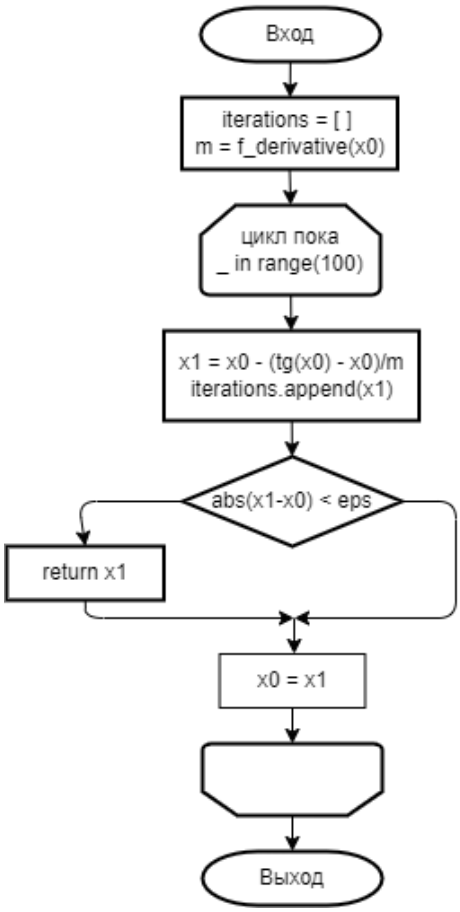
Для проверки правильности работы алгоритмов нахождения корня использовалась система GeoGebra

Схемы алгоритмов программы




вход
Iterations = [ ]
f_deprivative_x0 = self.f_deprivate
цикл "пока"
_ in range(100)
x1 = x0 - f(x0) /
f_deprevative(x0)
iterations.append(x1)
abs(x1 - x0) < eps
return x1
x0 = x1
Выход


Вход
iterations = [ ]
x0 = a
x1 = b
цикл пока
_ in range(100)
x_newton = x1 - f(x1) / f_derivative(x1)
x_secant = x0 - f(x0)(x1-x0) / (f(x1) - f(x0))
iterations.append((x_newton, x_secant))
abs(x_newton - x_secant) < eps
return (x_newton +
x_secant) / 2
x0 = x_secant
x1 = x_newton
Выход

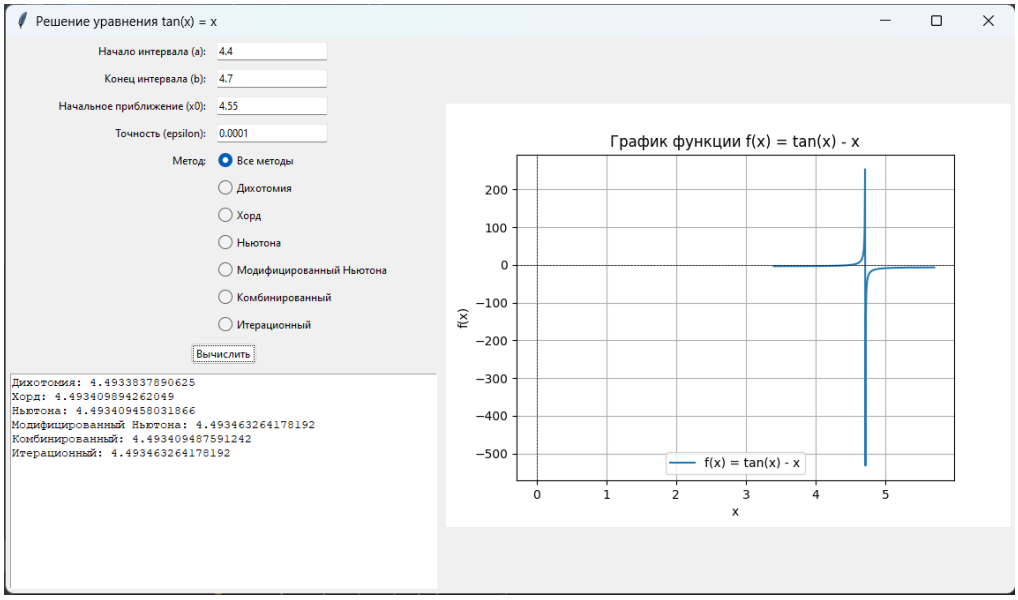

Вход
iterations = [ ]
m = f_derivative(x0)
цикл пока
_ in range(100)
x1 = x0 - (tg(x0) - x0)/m
iterations.append(x1)
abs(x1-x0) < eps
return x1
x0 = x1
Выход

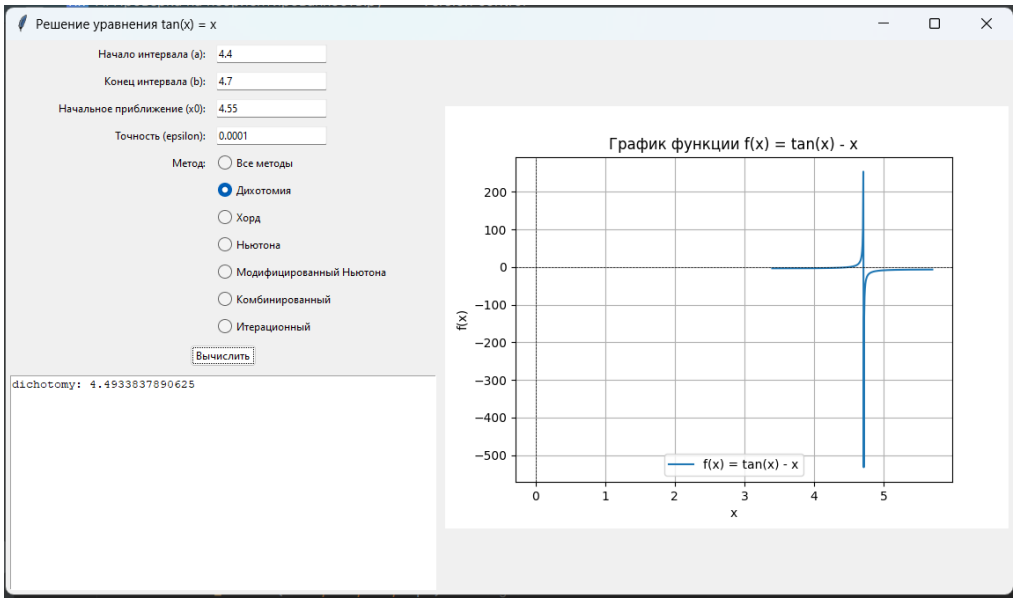
Листинг кода основных методов

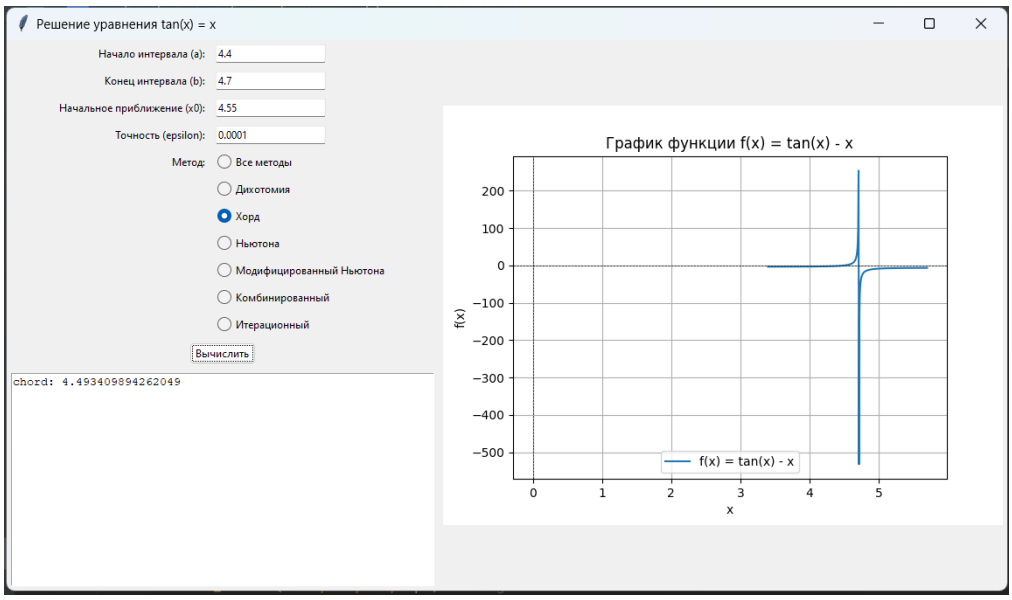
```
def f(self, x):
    return math.tan(x) - x

def f_derivative(self, x):
    return (1 / math.cos(x)**2) - 1

def dichotomy_method(self, a, b, eps):
    if self.f(a) * self.f(b) >= 0:
        return None

    iterations = []
    while (b - a) / 2 > eps:
        c = (a + b) / 2
        iterations.append(c)
        if self.f(c) == 0:
            return c
        elif self.f(a) * self.f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

def chord_method(self, x0, x1, eps):
    iterations = []
    for _ in range(100):
        try:
            x2 = x1 - self.f(x1) * (x1 - x0) / (self.f(x1) - self.f(x0))
            iterations.append(x2)
        except ZeroDivisionError:
            return None
        if abs(x2 - x1) < eps:
            return x2
        x0, x1 = x1, x2
    return None

def newton_method(self, x0, eps):
    iterations = []
    for _ in range(100):
        try:
            x1 = x0 - self.f(x0) / self.f_derivative(x0)
            iterations.append(x1)
        except ZeroDivisionError:
            return None
        if abs(x1 - x0) < eps:
            return x1
        x0 = x1
    return None

def modified_newton_method(self, x0, eps):
    iterations = []
    f_derivative_x0 = self.f_derivative(x0)
    for _ in range(100):
        try:
            x1 = x0 - self.f(x0) / f_derivative_x0
            iterations.append(x1)
        except ZeroDivisionError:
            return None
        if abs(x1 - x0) < eps:
            return x1
        x0 = x1
    return None
```

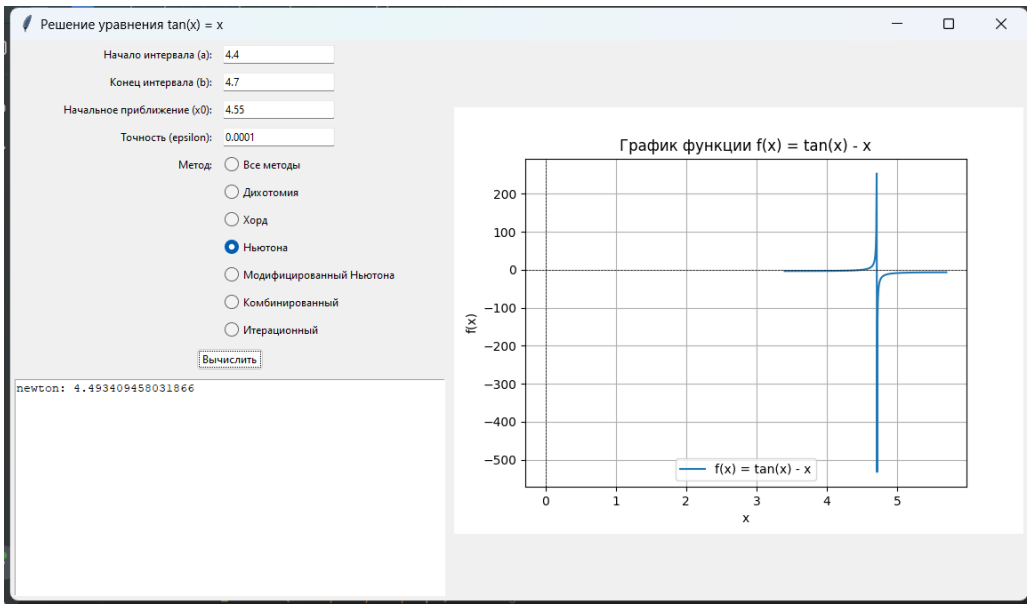
```
def combined_method(self, a, b, eps):
    iterations = []
    x0 = a
    x1 = b
    for _ in range(100):
        try:
            x_newton = x1 - self.f(x1) / self.f_derivative(x1)
            x_secant = x0 - self.f(x0) * (x1 - x0) / (self.f(x1) -
self.f(x0))
            iterations.append((x_newton, x_secant))
        except ZeroDivisionError:
            return None

        if abs(x_newton - x_secant) < eps:
            return (x_newton + x_secant) / 2

        x0 = x_secant
        x1 = x_newton
    return None

def iterative_method(self, x0, eps):
    iterations = []
    m = self.f_derivative(x0)

    for _ in range(100):
        try:
            x1 = x0 - (math.tan(x0) - x0) / m
            iterations.append(x1)
        except ValueError:
            return None
        if abs(x1 - x0) < eps:
            return x1
        x0 = x1
    return None
```

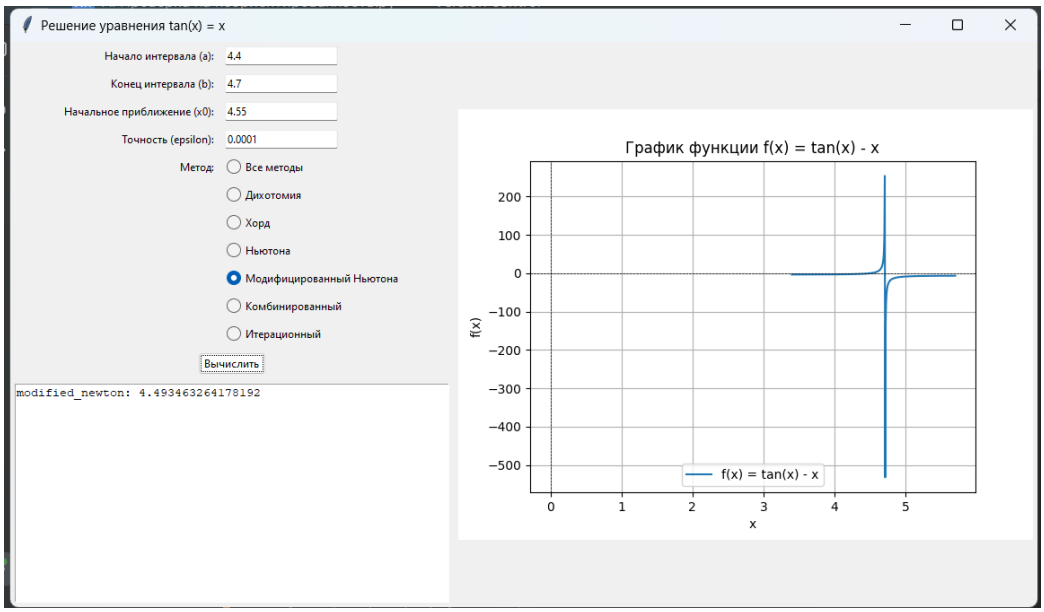
Графический интерфейс программы

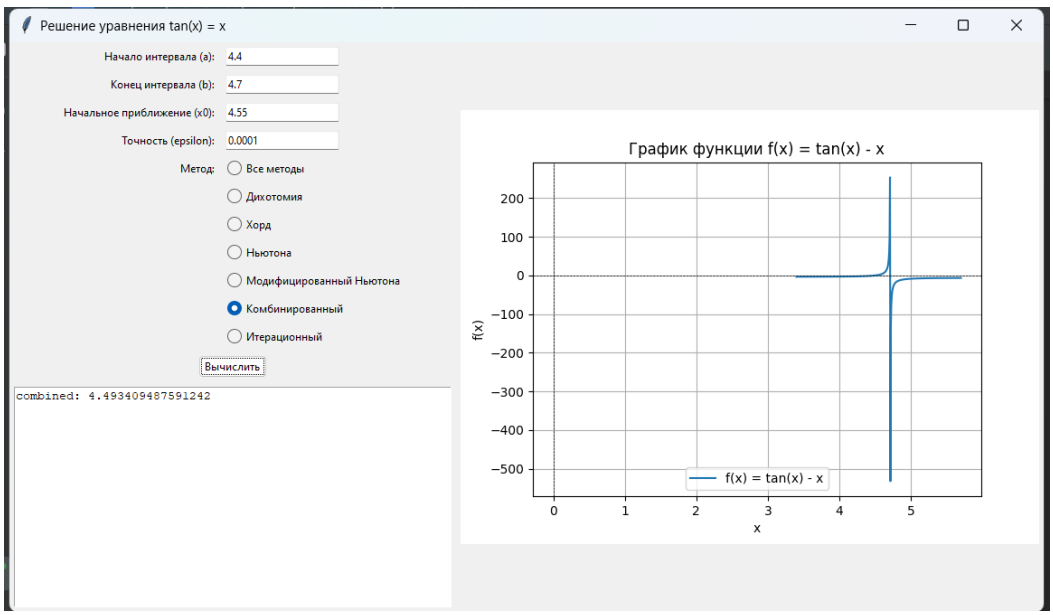


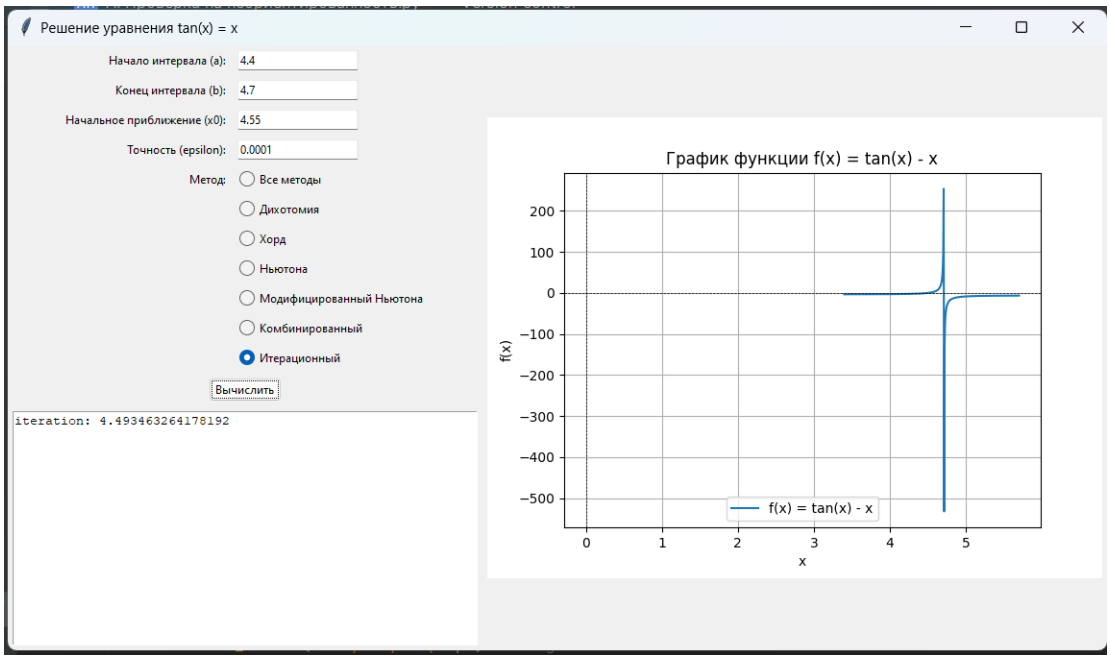





Решение уравнения tan(x) = x
Начало интервала (a): 4.4
Конец интервала (b): 4.7
Начальное приближение (x0): 4.55
Точность (epsilon): 0.0001
Метод:
Все методы
Дихотомия
Хорд
Ньютона
Модифицированный Ньютона
Комбинированный
Итерационный
Вычислить
newton: 4.493409458031866
График функции f(x) = tan(x) - x
f(x) = tan(x) - x
f(x)
x


Решение уравнения tan(x) = x
Начало интервала (a): 4.4
Конец интервала (b): 4.7
Начальное приближение (x0): 4.55
Точность (epsilon): 0.0001
Метод:
Все методы
Дихотомия
Хорд
Ньютона
Модифицированный Ньютона
Комбинированный
Итерационный
Вычислить
modified_newton: 4.493463264178192
График функции f(x) = tan(x) - x
f(x) = tan(x) - x
f(x)
x


Решение уравнения tan(x) = x
Начало интервала (a): 4.4
Конец интервала (b): 4.7
Начальное приближение (x0): 4.55
Точность (epsilon): 0.0001
Метод:
Все методы
Дихотомия
Хорд
Ньютона
Модифицированный Ньютона
Комбинированный
Итерационный
Вычислить
combined: 4.493409487591242
График функции f(x) = tan(x) - x
f(x) = tan(x) - x
f(x)
x



Проверка правильности работы программы

Проверка выполнена с помощью сервиса GeoGebra