РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И

ГАЗА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ) ИМЕНИ И.М. ГУБКИНА

Кафедра информатики

Расчетно-графическая работа по дисциплине:

«Основы алгоритмизации и программирования»

по теме: «Вычислительная математика»

Выполнил:

Вершинин Сергей Алексеевич,

студент группы АТ-24-01

Проверила:

ФИО,

Должность

Москва 2025

**Оглавление**

[Введение 3](#_Toc196077548)

[Глава 1. Решение нелинейного уравнения 4](#_Toc196077549)

[Глава 2. Численное интегрирование 11](#_Toc196077550)

[Глава 3. Решение ОДУ методом ломаных 16](#_Toc196077551)

[Заключение 20](#_Toc196077552)

[Список источников 21](#_Toc196077553)

[Приложение 1 22](#_Toc196077554)

[Приложение 2 24](#_Toc196077555)

[Приложение 3 25](#_Toc196077556)

# **Введение**

Вычислительная математика служит фундаментом для решения сложных задач в информатике, включая численные расчёты и оптимизацию алгоритмов. В рамках данной работы исследуются три ключевые проблемы вычислительной математики:

1. Решение нелинейного уравнения методом хорд и касательных.
2. Приближенное вычисление определенного интеграла по формуле «трех восьмых».
3. Решение задачи Коши для ОДУ первого порядка усовершенствованным методом ломаных.

Цель работы: изучение и практическое применение численных методов решения математических задач, таких как:

* нахождение корней нелинейных уравнений (метод хорд и касательных),
* приближенное вычисление определённых интегралов (метод «трёх восьмых»),
* численное решение задачи Коши для обыкновенного дифференциального уравнения (метод ломаных).

# **Глава 1. Решение нелинейного уравнения**

**Постановка задачи:**

Решить уравнение на заданном пользователем интервале с точностью до 0.001, используя комбинированный метод хорд и касательных. Найденные корни отобразить и построить график функции на указанном интервале.



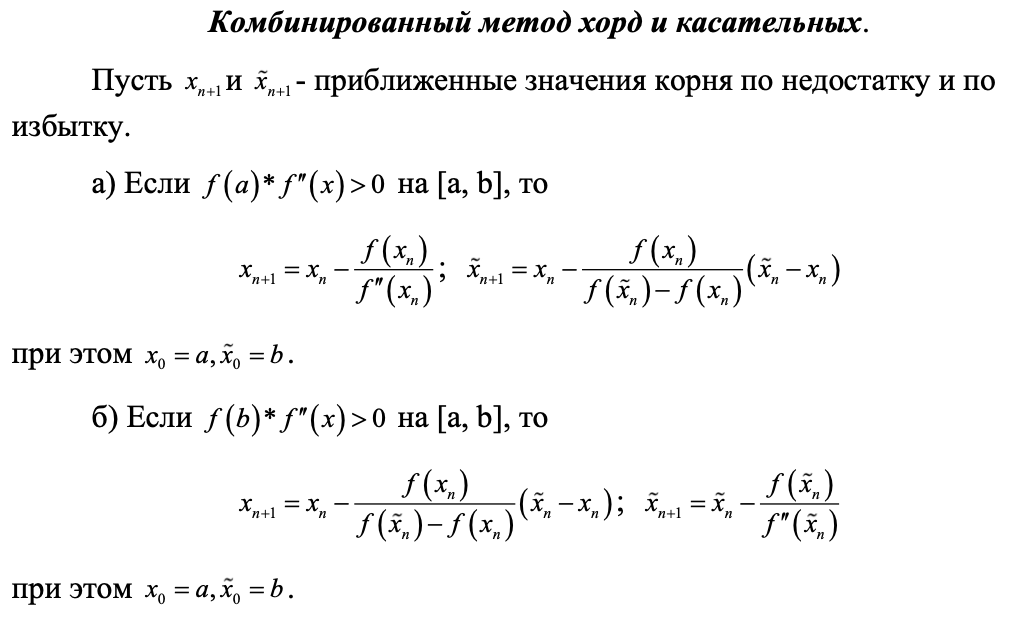
*Заданное уравнение*

**Алгоритм**

1. Инициализация:

* Определение функции f(x), её производных df(x) и ddf(x).
* Разбиение интервала [a, b] на точки и определение участков, где функция меняет знак.

1. Основной цикл:
   * Для каждого интервала со сменой знака определить точки x\_kasat и x\_hord в зависимости от знака второй производной.
   * Метод касательных: x\_kasat обновляется по формуле Ньютона: x\_new = x - f(x)/f'(x).
   * Метод хорд: x\_hord обновляется по формуле секущей: x\_new = x - f(x)(x - x\_other)/(f(x) - f(x\_other)).
   * Процесс повторяется до тех пор, пока разность между x\_kasat и x\_hord не станет меньше заданной точности (eps).
   * При возникновении деления на ноль алгоритм завершает итерации досрочно для устойчивости.



*Комбинированный метод хорд и касательных*

1. Формирование результата:
   * Среднее значение между последними x\_kasat и x\_hord считается корнем.
   * Добавление найденного корня в список, если он уникален. найденным, если он не совпадает с уже найденным (с учетом точности).

**Реализация кода на Python**

Реализовано на Python с использованием библиотек Tkinter для графического интерфейса и Matplotlib для построения графиков.   
class SolverNonlinearEquations:

def \_\_init\_\_(self, win\_frame):

self.win\_frame = win\_frame

self.fun = "x\*\*3 - 3\*x\*\*2 + 3"

self.name = 'First'

self.values = []

self.tk\_val = ["-1", "3"]

self.widgets = []

self.eps = 1e-3

self.max\_iter = 1000

self.left\_border = -1

self.right\_border = 3

self.names\_methods = ["Корень 1", "Корень 2", "Корень 3", "Scipy"]

self.names\_up = ["", "Значения"]

self.\_\_tkinter\_fun\_sne()

plot\_graph(self)

def \_\_tkinter\_fun\_sne(self):

Button(self.win\_frame, text="RESET", bg="red", command= lambda: rst(self)).place(x=5, y=5)

Label(self.win\_frame, text="Задание 1: Нелинейное уравнение", font="14", bg='bisque').place(x=50, y=70)

Label(self.win\_frame, text="x^3 - 3x^2 + 3 = 0", font="14", bg='bisque').place(x=120, y=100)

Label(self.win\_frame, text="Левая граница:", font= "14", bg='bisque').place(x=30, y=140)

entry\_a = Entry(self.win\_frame, textvariable=StringVar(value=-1), justify=CENTER)

entry\_a.place(x=230, y=145, width=100)

Label(self.win\_frame, text="Правая граница:", font="14", bg='bisque').place(x=30, y=170)

entry\_b = Entry(self.win\_frame, textvariable=StringVar(value=3), justify=CENTER)

entry\_b.place(x=230, y=175, width=100)

Button(self.win\_frame, text="Найти корни", font="14", bg='bisque2', command=lambda: disp\_info(self)).place(x=30, y=220)

Button(self.win\_frame, text="Построить график", font="10", bg='bisque2', command=lambda: plot\_graph(self)).place(x=180, y=220)

self.widgets.append(entry\_a)

self.widgets.extend([entry\_b])

def \_\_f(self, x):

return x\*\*3 - 3\*x\*\*2 + 3

def \_\_df(self, x):

return 3\*x\*\*2 - 6\*x

def \_\_ddf(self, x):

return 6\*x - 6

def hord\_kasat\_method(self):

roots = []

x = self.left\_border

b = self.right\_border

eps = self.eps

h = 0.1

while x < b:

x1 = x

x2 = min(x + h, b)

if self.\_\_f(x1) \* self.\_\_f(x2) < 0: # есть смена знака

mid = (x1 + x2) / 2

if self.\_\_f(x1) \* self.\_\_ddf(mid) > 0:

x\_n = x1

x\_v\_n = x2

for \_ in range(self.max\_iter):

de1 = self.\_\_df(x\_n)

de2 = self.\_\_f(x\_v\_n) - self.\_\_f(x\_n)

if abs(de1) < 1e-12 or abs(de2) < 1e-12:

break

x\_next = x\_n - self.\_\_f(x\_n) / de1

x\_v\_next = x\_n - self.\_\_f(x\_n) \* (x\_v\_n - x\_n) / de2

if abs(x\_v\_next - x\_next) < eps:

break

x\_n = x\_next

x\_v\_n = x\_v\_next

root = round((x\_n + x\_v\_n) / 2, 6)

if all(abs(root - r) > eps for r in roots):

roots.append(root)

elif self.\_\_f(x2) \* self.\_\_ddf(mid) > 0:

x\_n = x1

x\_v\_n = x2

for \_ in range(self.max\_iter):

de1 = self.\_\_df(x\_v\_n)

de2 = self.\_\_f(x\_v\_n) - self.\_\_f(x\_n)

if abs(de1) < 1e-12 or abs(de2) < 1e-12:

break

x\_next = x\_n - self.\_\_f(x\_v\_n) \* (x\_v\_n - x\_n) / de2

x\_v\_next = x\_v\_n - self.\_\_f(x\_v\_n) / de1

if abs(x\_v\_next - x\_next) < eps:

break

x\_n = x\_next

x\_v\_n = x\_v\_next

root = round((x\_n + x\_v\_n) / 2, 6)

if all(abs(root - r) > eps for r in roots):

roots.append(root)

else:

pass # ни одно из условий не выполняется

x += h

return roots

def scipy\_roots\_finder(self):

a, b = self.left\_border, self.right\_border

step = 0.1

roots = []

x\_values = np.arange(a, b + step, step)

for i in range(len(x\_values) - 1):

x1, x2 = x\_values[i], x\_values[i + 1]

if self.\_\_f(x1) \* self.\_\_f(x2) < 0:

result = root\_scalar(self.\_\_f, bracket=[x1, x2], method='brentq')

root = result.root

if not any(abs(root - r) < self.eps for r in roots):

roots.append(root)

return roots

def get\_roots(self):

**Реализация кода в Matlab**

clc

clear

e = 0.0001;

max\_iter = 100;

x = linspace(-3, 3, 1000);

figure;

plot(x, f(x), 'b-', 'LineWidth', 3);

grid on;

hold on;

plot(x, zeros(size(x)), 'k--');

title('x^3 - 3x^2 + 3 = 0');

xlabel('x');

ylabel('y');

a = input('Left border ');

b = input('Right border ');

if f(a)\*ddf(a) > 0

x = a;

x\_t = b;

elseif f(b)\*ddf(b) > 0

x = b;

x\_t = a;

else

disp('No roots')

end

for i = 1:max\_iter

x\_next = x - f(x)/df(x);

x\_t\_next = x - (f(x)\*(x\_t - x)) / (f(x\_t) - f(x));

if abs(x\_next - x\_t\_next) < e

root = (x\_next + x\_t\_next) / 2;

disp('Root')

disp(root)

break;

end

x = x\_next;

x\_t = x\_t\_next;

end

f.m

function y = f(x)

y = x.^3 - 3\*x.^2 + 3;

end

df.m

function y = df(x)

y = 3\*x.^2 - 6\*x;

end

ddf.m

function y = ddf(x)

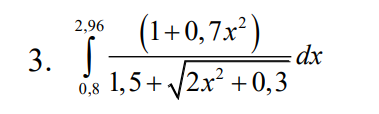
y = 6\*x - 6;

end   
Примеры работы программы даны в Приложении 1.

# **Глава 2. Численное интегрирование**

**Постановка задачи.**

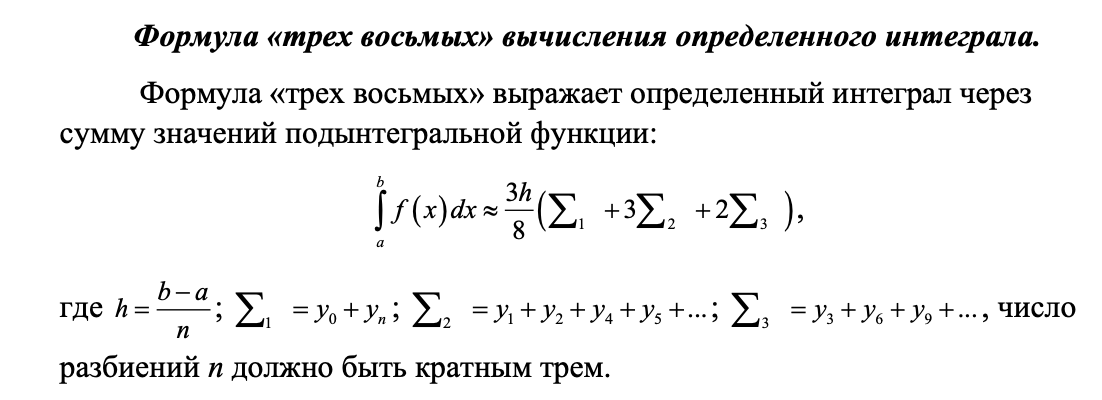
Вычислить приближенное значение интеграла с использованием формулы трех восьмых, для n1 = 9 и n2 = 12. Вычислить разницу между двумя значениями как оценку точности.



*Заданный интеграл*

**Алгоритм.**

1. Инициализация:
   * Определение функции под интегралом.
   * Проверка, что число разбиений n кратно 3.
   * Расчет шага h и значений функции на равномерной сетке.
2. Основной цикл:
   * Расчет значений функции на равномерной сетке от a до b.
   * Вычисление крайних значений: f(a) и f(b).
   * Суммирование значений функции на узлах кратных 3 (вес 2), кратных 1 и 2 между ними (вес 3), согласно формуле 3/8.
   * Контроль кратности 3 для корректности применения метода.



*Формула «трех восьмых»*

1. Формирование результата:
   * Применение формулы: I = (3 \* h / 8) \* (y0 + yn + 3 \* y\_кратные3 + 2 \* y\_остальные)
   * Возврат приближенного значения интеграла. I = 3h/8 \* (y0 + yn + 3 \* (y3 + y6 + ...) + 2 \* (y1 + y4 + ...))

**Реализация кода.**

Интервал интегрирования фиксирован. Для обоих значений n производится вычисление, результаты и погрешность отображаются в окне приложения.

**Реализация кода на Python**

class NumericalIntegration:

def \_\_init\_\_(self, win\_frame):

self.win\_frame = win\_frame

self.fun = "(1 + 0.7 \* x\*\*2) / (1.5 + (2\*x\*\*2 + 0.3)\*\*0.5)"

self.name = 'First'

self.values = []

self.tk\_val = ["0.8", "2.96"]

self.widgets = []

self.eps = 1e-3

self.max\_iter = 1000

self.left\_border = 0.8

self.right\_border = 2.96

self.names\_methods = ["Корень n=9", "Корень n=12", "Scipy"]

self.names\_up = ["", "Значения"]

self.\_\_tkinter\_fun\_ni()

plot\_graph(self)

def \_\_tkinter\_fun\_ni(self):

# Интеграл

fig = Figure(figsize=(4, 3), facecolor='bisque')

ax = fig.add\_subplot(111)

ax.axis('off')

integral\_text = r"$\int\_{0{,}8}^{2{,}96} \frac{1 + 0{,}7x^2}{1{,}5 + \sqrt{2x^2 + 0{,}3}} \, dx$"

ax.text(0.5, 0.5, integral\_text, fontsize=10,

ha='center', va='center', color='black')

canvas = FigureCanvasTkAgg(fig, master=self.win\_frame)

canvas.draw()

canvas.get\_tk\_widget().place(x=-10, y=-35)

Button(self.win\_frame, text="RESET", bg="red", command= lambda: rst(self)).place(x=5, y=5)

Label(self.win\_frame, text="Задание 2: Численное интегрирование", font="14", bg='bisque').place(x=50, y=70)

Label(self.win\_frame, text="Левая граница:", font= "14", bg='bisque').place(x=30, y=140)

entry\_a = Entry(self.win\_frame, textvariable=StringVar(value=0.8), justify=CENTER)

entry\_a.place(x=230, y=145, width=100)

Label(self.win\_frame, text="Правая граница:", font="14", bg='bisque').place(x=30, y=170)

entry\_b = Entry(self.win\_frame, textvariable=StringVar(value=2.96), justify=CENTER)

entry\_b.place(x=230, y=175, width=100)

Button(self.win\_frame, text="Найти корни", font="14", bg='bisque2', command=lambda: disp\_info(self)).place(x=30, y=220)

Button(self.win\_frame, text="Построить график", font="10", bg='bisque2', command=lambda: plot\_graph(self)).place(x=180, y=220)

self.widgets.append(entry\_a)

self.widgets.extend([entry\_b])

def \_\_integral(self, x):

return (1 + 0.7 \* x\*\*2) / (1.5 + (2\*x\*\*2 + 0.3)\*\*0.5)

def scipy\_roots\_finder(self):

result, error = quad(self.\_\_integral, self.left\_border, self.right\_border)

return result

def three\_eighths(self):

a = self.left\_border

b = self.right\_border

roots = []

for n in [9, 12]:

if n % 3 != 0:

raise ValueError("Число разбиений должно быть кратно 3")

h = (b - a) / n

x = np.linspace(a, b, n + 1)

y = self.\_\_integral(x)

sum3 = np.sum(y[3:-1:3])

other\_indices = [i for i in range(1, n) if i % 3 != 0]

sum2 = np.sum(y[other\_indices])

integral = (3 \* h / 8) \* (y[0] + y[-1] + 3 \* sum2 + 2 \* sum3)

roots.append(integral)

return roots

def get\_roots(self):

try:

self.left\_border = float(self.widgets[0].get())

self.right\_border = float(self.widgets[1].get())

except Exception as e:

return error(e)

self.values = [i for i in self.three\_eighths()]

self.values.append(self.scipy\_roots\_finder())

**Реализация кода в Matlab**

clc

clear

a = 0.8;

b = 2.96;

x = linspace(a, b, 100);

figure;

plot(x, f2(x), 'b-', 'LineWidth', 2);

title('График подынтегральной функции');

xlabel('x');

ylabel('y');

grid on;

% n1 = 9

n1 = 9;

h1 = (b - a)/n1;

sum1 = f2(a) + f2(b);

for i = 1:n1-1

x = a + i\*h1;

if mod(i,3) == 0

sum1 = sum1 + 2\*f2(x);

else

sum1 = sum1 + 3\*f2(x);

end

end

integral1 = 3\*h1/8 \* sum1;

% n2 = 12

n2 = 12;

h2 = (b - a)/n2;

sum2 = f2(a) + f2(b);

for i = 1:n2-1

x = a + i\*h2;

if mod(i,3) == 0

sum2 = sum2 + 2\*f2(x);

else

sum2 = sum2 + 3\*f2(x);

end

end

integral2 = 3\*h2/8 \* sum2;

% Tochnost

tochnost = abs(integral2 - integral1);

disp('Значение интеграла при n=9')

disp(integral1)

disp('Значение интеграла при n=12')

disp(integral2)

disp('Оценка погрешности:')

disp(tochnost)

f2.m

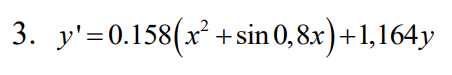
function y = f2(x)

y = (1 + 0.7\*x.^2) ./ (1.5 + sqrt(2\*x.^2 + 0.3));

end   
Пример работы программы даны в Приложении 2.

# **Глава 3. Решение ОДУ методом ломаных**

Решить задачу Коши для обыкновенного дифференциального уравнения первого порядка с использованием метода ломаных (модифицированного метода Эйлера) с шагом h = 0.1.



*Постановка задачи*

**Алгоритм**

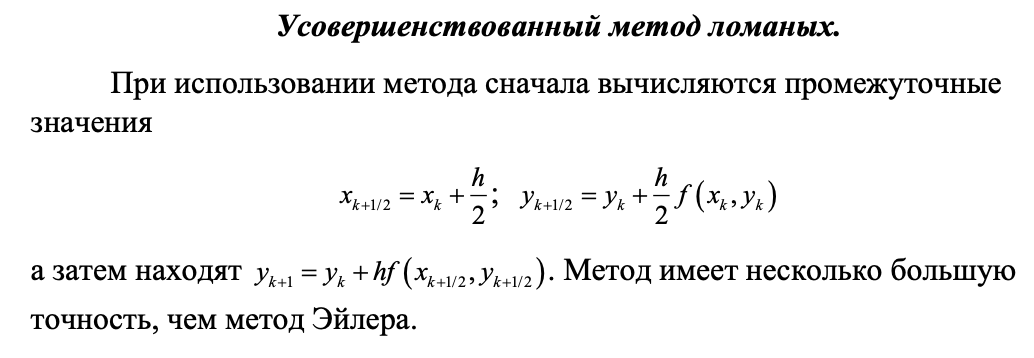
1. Инициализация:

Заданы начальные значения: x0 = 0.2, y0 = 0.25, шаг h = 0.1, конечное значение x\_end = 1.2. Создаются списки для хранения значений x и y, в которые записываются начальные данные.

1. Основной цикл:

Пока x не достигнет конца отрезка, выполняются следующие шаги:

* Вычисляется промежуточная точка: x\_half = x + h/2, y\_half = y + (h/2) \* f(x, y)
* Вычисляется новое значение y: y\_next = y + h \* f(x\_half, y\_half)
* Значения x и y обновляются и добавляются в соответствующие списки.
* Все значения округляются до 4 знаков после запятой для компактного и точного вывода.



*Усовершенствованный метод ломаных*

Таким образом, метод итеративно строит приближенное решение ОДУ.

1. Формирование результата

После завершения итераций возвращается таблица значений x и y.  
Полученные значения можно вывести в табличной форме или графически.

**Реализация кода на Python**

class SolverDifferentialEquations:

def \_\_init\_\_(self, win\_frame):

self.win\_frame = win\_frame

self.name = 'Third'

self.values = []

self.tk\_val = ["0.2", "1.2"]

self.widgets = []

self.names\_methods = ["X", "Y"]

self.names\_up = ["", "Значения"]

self.\_\_tkinter\_fun\_sde()

def \_\_tkinter\_fun\_sde(self):

Button(self.win\_frame, text="RESET", bg="red", command= lambda: rst(self)).place(x=5, y=5)

Label(self.win\_frame, text="Задание 3: Дифференциальное уравнение", font="14", bg='bisque').place(x=50, y=70)

Label(self.win\_frame, text="y' = 0.158 \* (x\*\*2 + sin(0.8 \* x)) + 1.164 \* y", font="14", bg='bisque').place(x=60, y=100)

Button(self.win\_frame, text="Найти корни", font="14", bg='bisque2', command=lambda: disp\_info(self)).place(x=140, y=220)

def \_\_ode\_func(self, x, y):

return 0.158 \* (x\*\*2 + np.sin(0.8 \* x)) + 1.164 \* y

def broken\_line\_method(self):

x0 = 0.2

y0 = 0.25

x\_end = 1.2

h = 0.1

x\_vals = [x0]

y\_vals = [y0]

while x0 < x\_end:

x\_half = x0 + h / 2

y\_half = y0 + (h / 2) \* self.\_\_ode\_func(x0, y0)

y\_next = y0 + h \* self.\_\_ode\_func(x\_half, y\_half)

x0 += h

y0 = y\_next

x\_vals.append(round(x0, 4))

y\_vals.append(round(y0, 4))

return x\_vals, y\_vals

def get\_roots(self):

self.values = [i for i in self.broken\_line\_method()]

plot\_graph(self)

**Реализация кода в Matlab**

clc

clear

a = 0.2;

b = 1.2;

h = 0.1;

y0 = 0.25;

x = a:h:b;

y = zeros(size(x));

y(1) = y0;

for i = 1:length(x)-1

y\_half = y(i) + (h/2) \* f3(x(i), y(i));

y(i+1) = y(i) + h \* f3(x(i) + h/2, y\_half);

end

fprintf(' x\t\ty(x)\n');

fprintf('-------------\n');

for i = 1:length(x)

fprintf('%.4f\t%.4f\n', x(i), y(i));

end

figure;

plot(x, y, '-\*', 'LineWidth', 3, 'MarkerSize', 6);

title('f = 0.158 \*(x^2 + sin(0.8 \* x)) + 1.164 \* y');

xlabel('x');

ylabel('y(x)');

grid on;

f3.m

function y = f3(x,y)

f = 0.158 \* (x.^2 + sin(0.8 \* x)) + 1.164 \* y;

end   
Примеры работы программы даны в Приложении 3.

# **Заключение**

В ходе данной работы были исследованы и реализованы три фундаментальных метода вычислительной математики, каждый из которых решает определенный класс математических задач:

1. Решение нелинейных уравнений комбинированным методом

* Был применен комбинированный метод хорд и касательных, особенностями которого являются:
* Автоматический выбор оптимального алгоритма на основе анализа производных
* Итерационный процесс с контролируемой точностью (ε=10⁻³)
* Визуализация функции с отметкой найденных корней
* Сравнение с эталонным решением через SciPy (метод Брента)

2. Численное интегрирование по формуле трех восьмых

* Классическая формула трех восьмых для равномерной сетки
* Вариации с разным числом разбиений (n=9, 12)
* Сравнение с результатами SciPy.integrate.quad
* Точное отображение интеграла в математической нотации

3. Решение задачи Коши модифицированным методом ломаных

Для дифференциального уравнения:

* Усовершенствованный метод ломаных (модификация Эйлера)
* Пошаговый расчет с контролем погрешности

Технологический стек:

* Язык программырования: Python
* Графический интерфейс: Tkinter (Python)
* Численные расчеты: NumPy, SciPy
* Визуализация: Matplotlib
* Сравнительные расчеты: MATLAB/Octave

Ключевые особенности реализации:

1. Объектно-ориентированный подход:

* Каждый метод инкапсулирован в отдельный класс
* Четкое разделение логики расчетов и интерфейса

1. Интерактивный интерфейс:

* Возможность задания параметров расчетов
* Кнопки управления (расчет, сброс, визуализация)
* Табличное представление результатов

1. Контроль точности:

* Встроенные проверки корректности ввода
* Сравнение с эталонными решениями
* Оценка погрешности для каждого метода

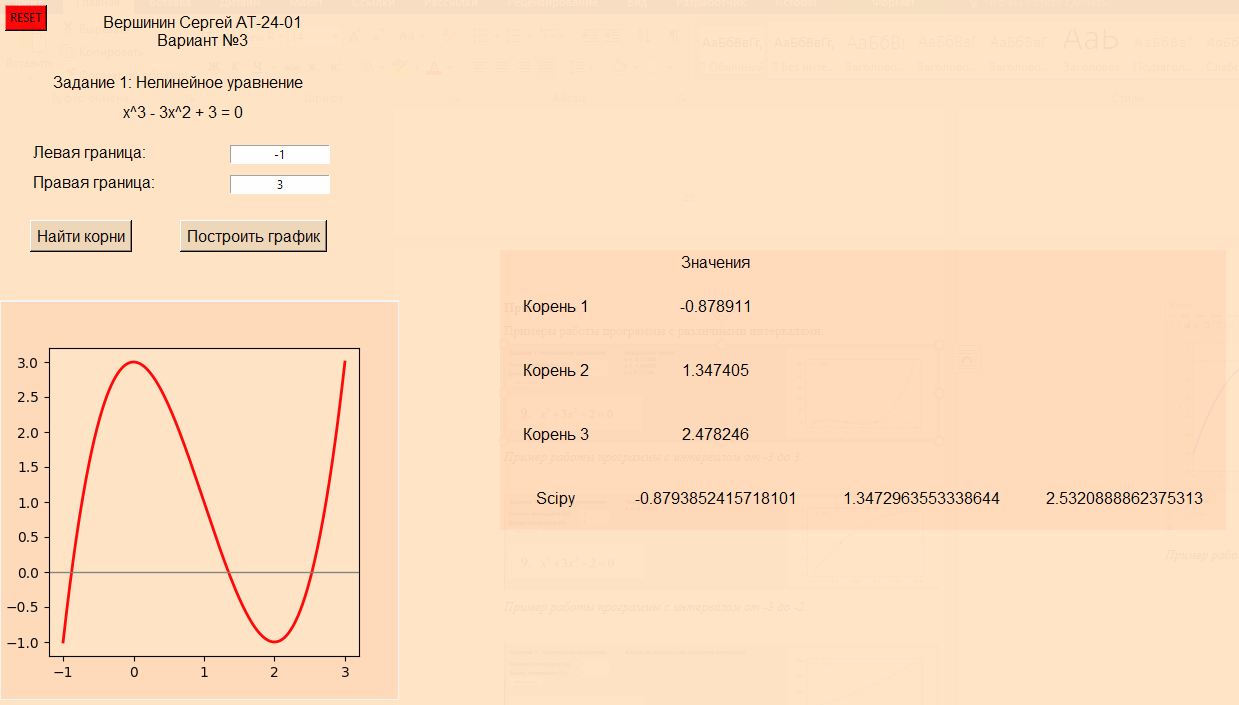
Работа демонстрирует эффективное применение численных методов для решения актуальных математических задач и может служить основой для создания более сложных вычислительных систем.

# **Список источников**

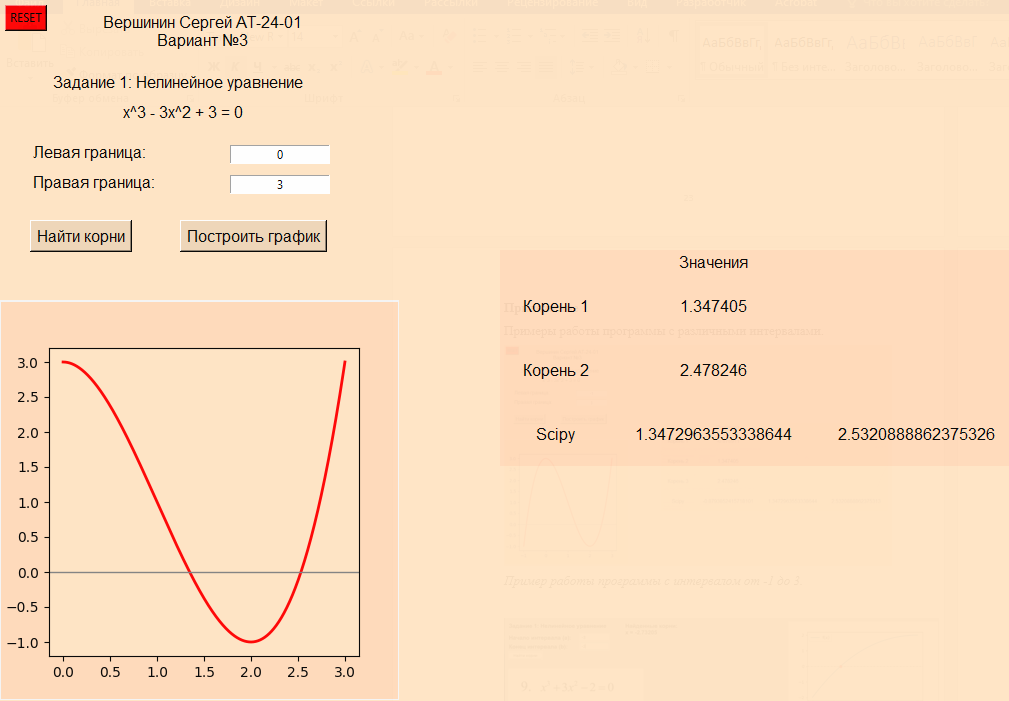
1. Метод Ньютона // [Электронный ресурс] //: https://ru.wikipedia.org/wiki/Метод\_Ньютона // Дата обращения: 20.04.2025
2. Метод хорд // [Электронный ресурс] // https://ru.wikipedia.org/wiki/Метод\_хорд  //Дата обращения: 20.04.2025
3. Официальная документация Python // [Электронный ресурс] // https://docs.python.org/3/ // Дата обращения: 20.04.2025
4. Численное интегрирование // [Электронный ресурс] //  https://ru.wikipedia.org/wiki/Численное\_интегрирование // Дата обращения: 20.04.2025
5. Tkinter: GUI Programming in Python // [Электронный ресурс] — //  https://docs.python.org/3/library/tkinter.html// Дата обращения: 20.04.2025

# **Приложение 1**

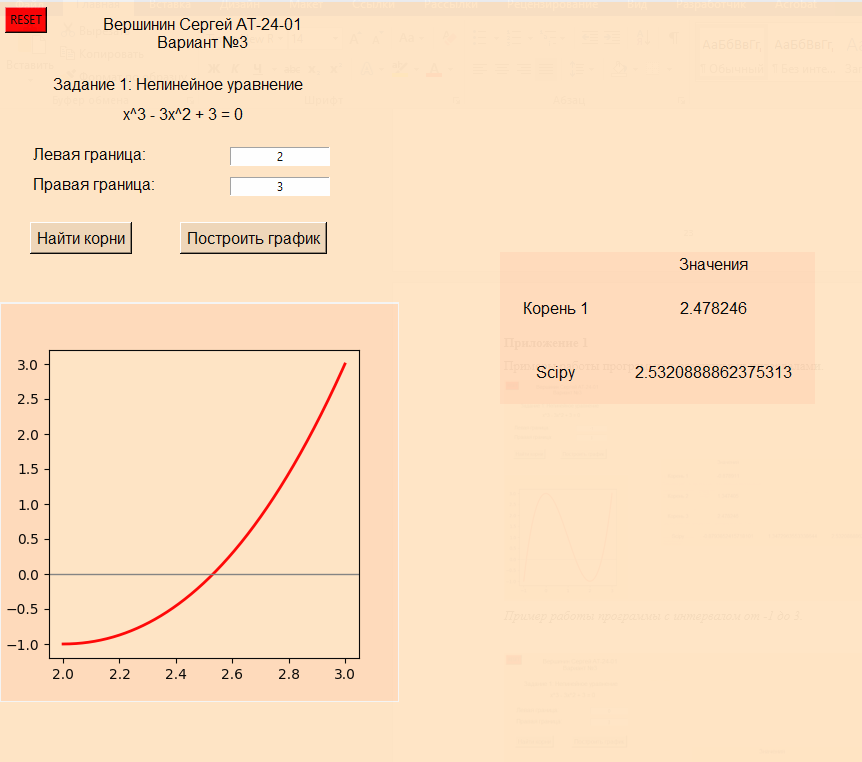
Примеры работы программы с различными интервалами.

**

*Пример работы программы с интервалом от -1 до 3.*

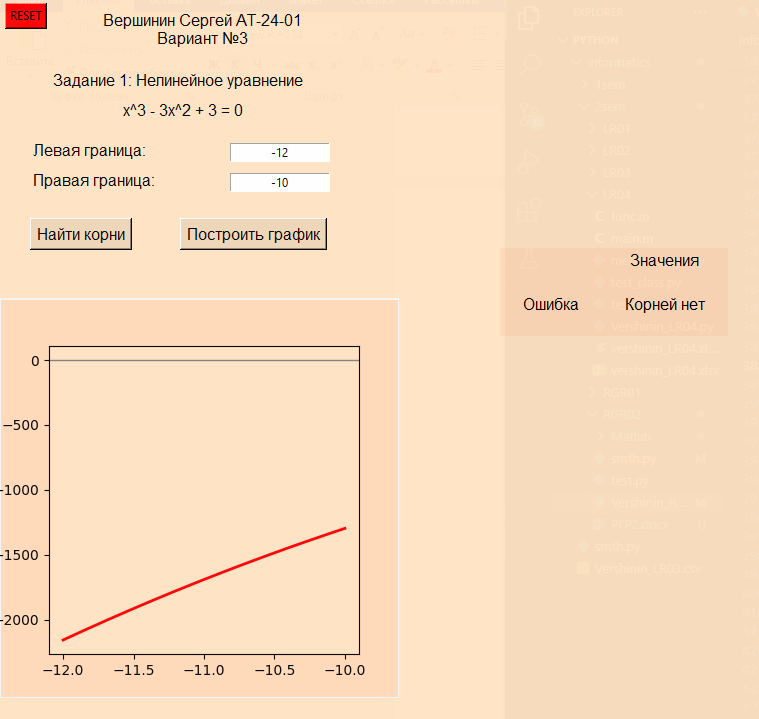


*Пример работы программы с интервалом от 0 до 3.*

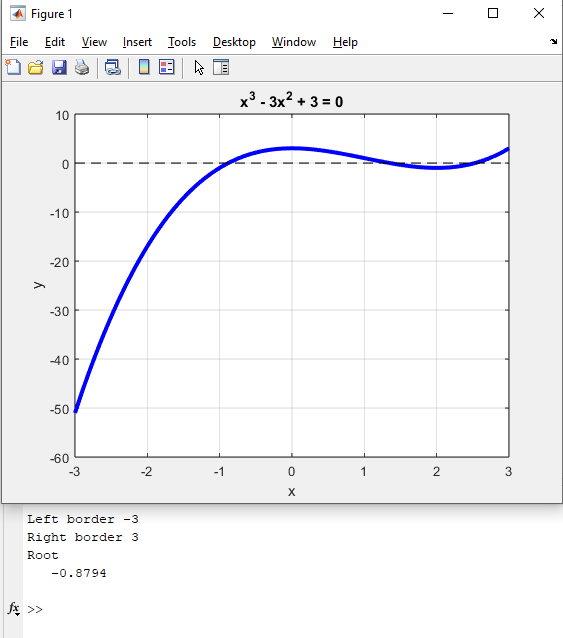


*Пример работы программы с интервалом от 2 до 3.*

Если ввести некорректный интервал, то программа выведет сообщение об ошибке.



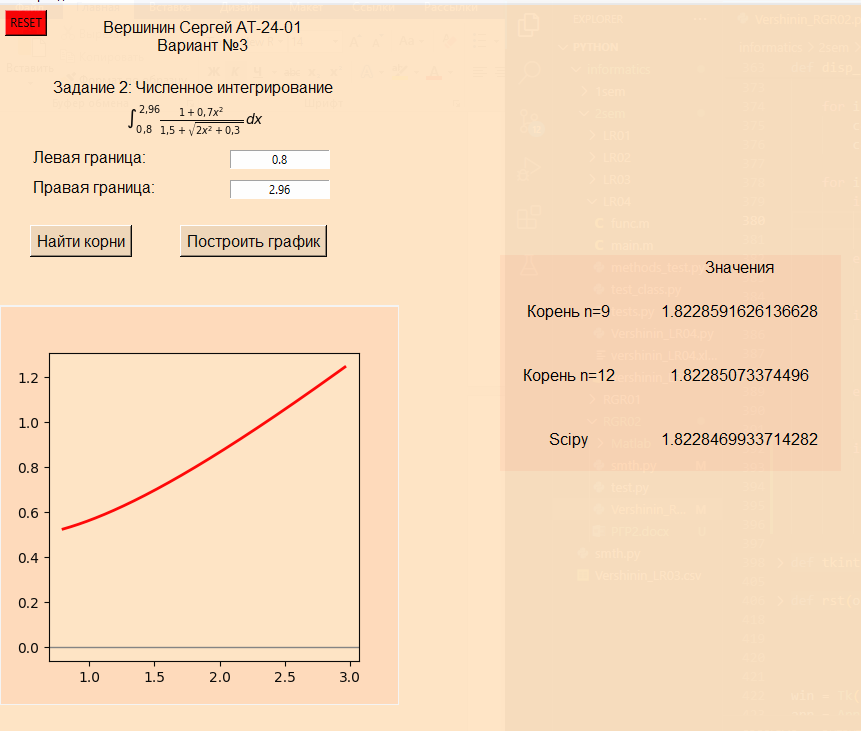
*Пример работы программы с интервалом [-12; -10].*

**

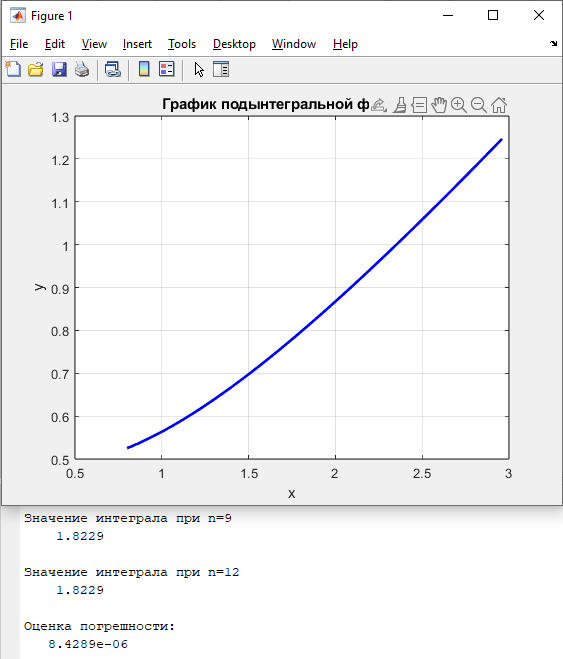
*Пример работы программы в Matlab.*

# **Приложение 2**

Примеры работы программы.



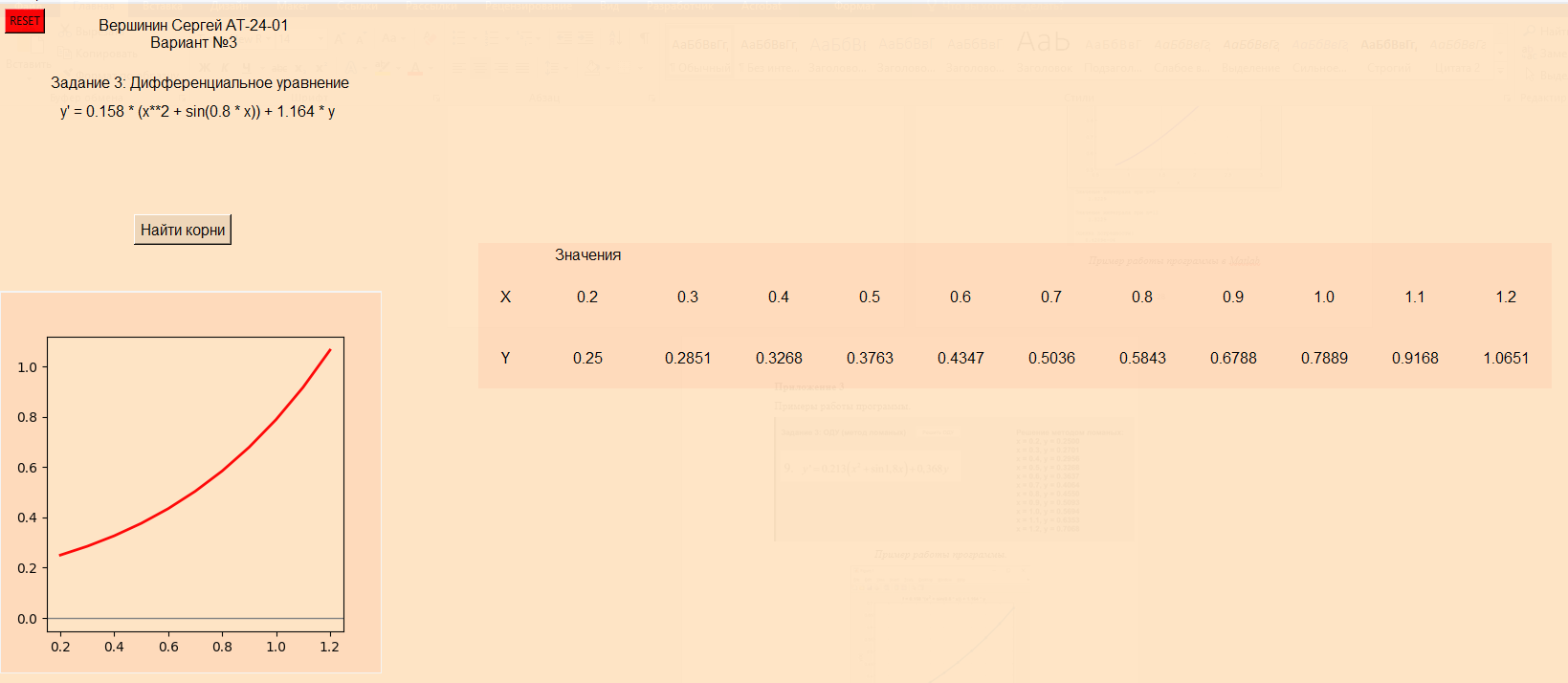
*Пример работы программы.*

**

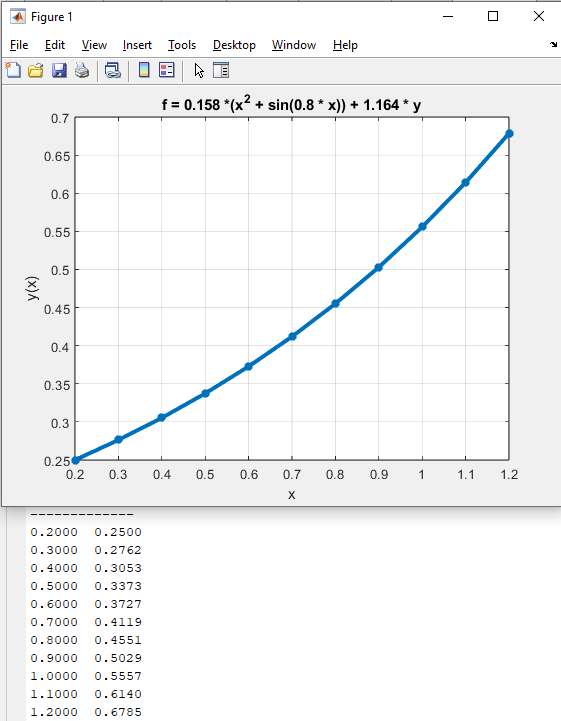
*Пример работы программы в Matlab*

# **Приложение 3**

Примеры работы программы.



*Пример работы программы.*

**

*Пример работы программы в Matlab*