**МАИ**

**Лабораторная работа №11**

 «Методы поиска приближённого значения трансцендентного уравнения и их программная реализация на основе итерационного и рекурсивного подходов»Вариант №14

Факультет робототехнических и интеллектуальных систем

Кафедра «Системы приводов летательных аппаратов»

**Выполнил:**

Студент группы М7О-114БВ-24

Фельдман Лев Борисович

**Проверил:**Доцент Кафедры 702 Козлова Н.М.

Ассистент Кафедры 702 Милославский Я.Г.

Москва 2025

**Цель работы: знакомство с методами поиска корня трансцендентных уравнений и закрепление навыков работы со структурами выбора, повтора и рекурсивными функциями**.

Методы поиска корней трансцендентных уравнений:

1. Метод половинного деления (дихотомии, бисекции).

2. Метод ложного положения.

3. Метод Ньютона.

4. Метод секущих.

Задания для каждого метода.

1. Написать скрипт-файл со структурой повторения while end для поиска корня и вывода результатов поиска в командное окно в виде таблицы и в графическое окно в виде графиков последовательных приближений (по оси абсцисс указать количество итераций, а по оси ординат - значение последовательных приближений и значение функции). На основе таблицы на листе бумаги изобразить геометрическую интерпретацию рассматриваемого метода.

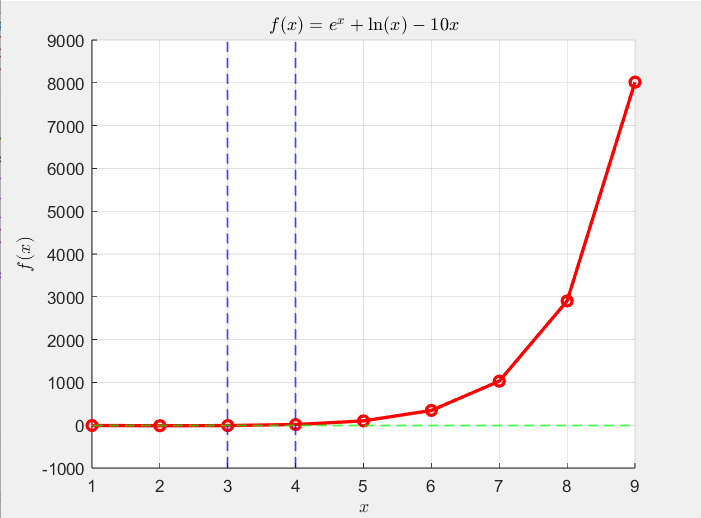
2. а) Написать функцию, принимающую четыре аргумента (дескриптор функции, диапазон или начальное приближение, точность и максимальное число повторов), последние, два из которых являются дополнительными. Функция должна возвращать два значения - значение корня и количество итераций. В функции необходимо реализовать проверку аргументов. При реализации алгоритма использовать структуру while end из предыдущего задания.

б) Написать скрипт-файл, в котором происходит обращение к созданной в пункте а функции для построения графика зависимости точности полученного корня от количества повторов. Ось ординат на графике должна иметь логарифмический масштаб.

3. Создать рекурсивную функцию для рассматриваемого метода.

4. Сделать вывод о целесообразности применения рассматриваемых методов.

|  |  |  |  |
| --- | --- | --- | --- |
| **Уравнение** | **Начальный**  **диапазон** | **Начальное**  **приближение** | **Приближённое**  **значение корня** |
|  | [3, 4] | 5.00 | 3.5265 |



**Метод половинного деления (дихотомии, бисекции).**

**1. Итерационный скрипт на основе цикла while**

f = @(x) exp(x) + log(x) - 10\*x;

a = 3;

b = 4;

tol = 1e-5; % критерий точности

max\_iter = 100; % максимальное число итераций

% Проверка наличия знакопеременности

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

error('На заданном интервале функция не меняет знак. Выберите другой интервал.');

end

% Инициализация переменных для хранения результатов итераций

iter = 0;

data = []; % матрица для хранения: [iter, mid, f(mid)]

fprintf(' Ит | x\_{mid} | f(x\_{mid})\n');

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

while iter < max\_iter

iter = iter + 1;

mid = (a + b)/2;

fmid = f(mid);

% Сохраняем данные для вывода

data = [data; iter, mid, fmid];

% Вывод текущей итерации в командное окно

fprintf('%4d | %14.6f | %14.6f\n', iter, mid, fmid);

if abs(fmid) < tol

break;

end

% Определение нового интервала

if f(a)\*fmid < 0

b = mid;

else

a = mid;

end

end

% Вывод результата

fprintf('\nПриблизительное значение корня: %.6f\n', mid);

fprintf('Количество итераций: %d\n', iter);

% Построение графиков:

figure;

subplot(2,1,1)

plot(data(:,1), data(:,2), 'b-o','LineWidth',1.5);

xlabel('Номер итерации');

ylabel('Приближение x');

title('График последовательных приближений');

subplot(2,1,2)

plot(data(:,1), data(:,3), 'r-o','LineWidth',1.5);

xlabel('Номер итерации');

ylabel('f(x)');

title('График значений функции');

Ит | x\_{mid} | f(x\_{mid})

-------------------------------------------

1 | 3.500000 | -0.631785

2 | 3.750000 | 6.342838

3 | 3.625000 | 2.562577

4 | 3.562500 | 0.896678

5 | 3.531250 | 0.115801

6 | 3.515625 | -0.262088

7 | 3.523438 | -0.074176

8 | 3.527344 | 0.020554

9 | 3.525391 | -0.026876

10 | 3.526367 | -0.003177

11 | 3.526855 | 0.008684

12 | 3.526611 | 0.002753

13 | 3.526489 | -0.000213

14 | 3.526550 | 0.001270

15 | 3.526520 | 0.000529

16 | 3.526505 | 0.000158

17 | 3.526497 | -0.000027

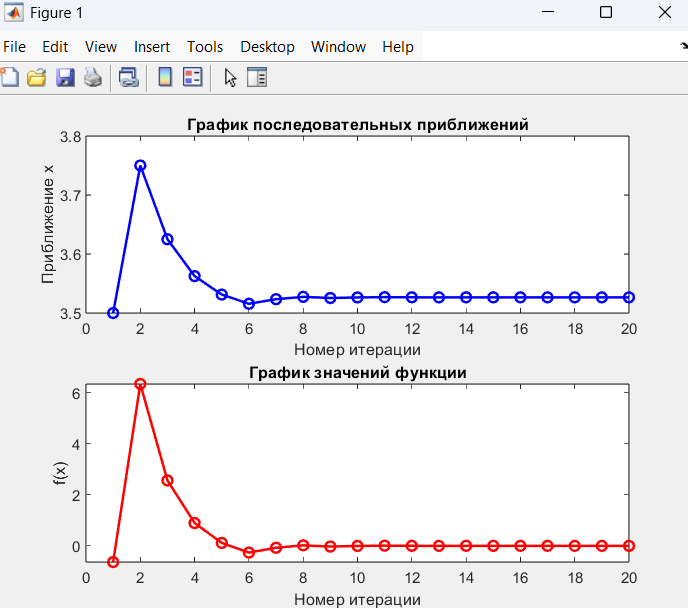
18 | 3.526501 | 0.000065

19 | 3.526499 | 0.000019

20 | 3.526498 | -0.000004

Приблизительное значение корня: 3.526498

Количество итераций: 20



**2a. Функция для вычисления корня методом бисекции**

function [root, iter] = bisection\_method(func, interval, tol, max\_iter)

% Входные параметры:

% func - дескриптор функции (анонимная функция или функция-файл)

% interval - вектор [a, b] с начальным диапазоном, где происходит поиск корня

% tol - требуемая точность (опционально, по умолчанию 1e-5)

% max\_iter - максимальное число итераций (опционально, по умолчанию 100)

%

% Выходные параметры:

% root - найденное приближённое значение корня

% iter - фактическое число итераций

% Проверка количества входных аргументов

if nargin < 2

error('Необходимо задать функцию и интервал.');

end

if nargin < 3 || isempty(tol)

tol = 1e-5;

end

if nargin < 4 || isempty(max\_iter)

max\_iter = 100;

end

if numel(interval) ~= 2

error('Диапазон должен быть в виде вектора из двух элементов: [a, b].');

end

a = interval(1);

b = interval(2);

% Проверка знакопеременности

if func(a)\*func(b) > 0

error('На заданном интервале функция не меняет знак.');

end

iter = 0;

while iter < max\_iter

iter = iter + 1;

mid = (a + b)/2;

fmid = func(mid);

if abs(fmid) < tol

root = mid;

return;

end

if func(a)\*fmid < 0

b = mid;

else

a = mid;

end

end

root = mid;

end

**>>** **Найденный корень: 3.526497 за 17 итераций**

**2b. Скрипт для построения графика зависимости точности от числа итераций**

f = @(x) exp(x) + log(x) - 10\*x;

true\_root = 3.5265; % известное приблизительное значение корня для сравнения

% Задаём диапазон

interval = [3, 4];

% Подготовка массива для хранения ошибок и числа итераций.

max\_iter\_array = 1:50; % варьируем число итераций от 1 до 50

errors = zeros(size(max\_iter\_array));

for i = 1:length(max\_iter\_array)

max\_iter = max\_iter\_array(i);

% Вычисление корня при заданном количестве итераций

[approx\_root, iter\_used] = bisection\_method(f, interval, 1e-5, max\_iter);

% Абсолютная ошибка относительно известного значения

errors(i) = abs(approx\_root - true\_root);

end

% Построение графика с логарифмической осью ординат

figure;

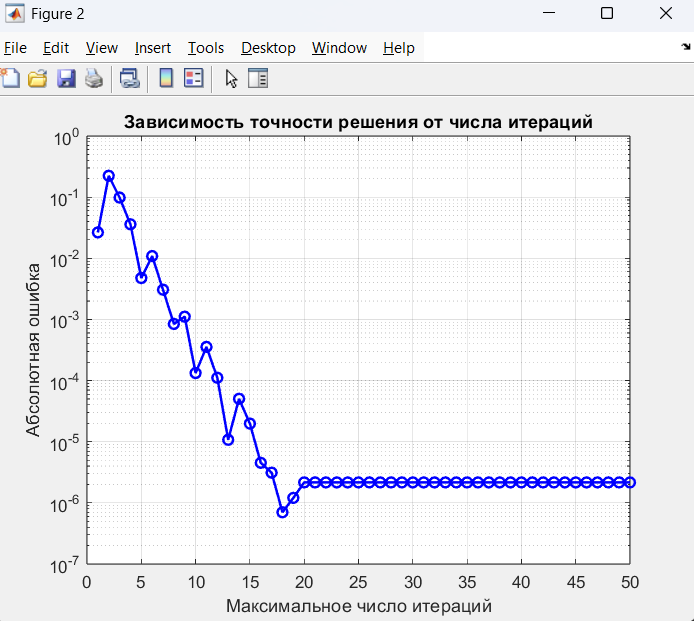
semilogy(max\_iter\_array, errors, 'b-o','LineWidth',1.5);

xlabel('Максимальное число итераций');

ylabel('Абсолютная ошибка');

title('Зависимость точности решения от числа итераций');

grid on;



**3. Рекурсивная функция для метода бисекции**

function [root, iter] = bisection\_recursive(func, a, b, tol, iter, max\_iter)

% Входные параметры:

% func - дескриптор функции

% a, b - границы интервала

% tol - требуемая точность (опционально, по умолчанию 1e-5)

% iter - текущий номер итерации (опционально, по умолчанию 0)

% max\_iter - максимальное число итераций (опционально, по умолчанию 100)

% Выходные параметры:

% root - приближённое значение корня

% iter - количество выполненных итераций

% Установка значений по умолчанию, если необходимо

if nargin < 4 || isempty(tol)

tol = 1e-5;

end

if nargin < 5 || isempty(iter)

iter = 0;

end

if nargin < 6 || isempty(max\_iter)

max\_iter = 100;

end

% Проверка знакопеременности при первом вызове (при iter==0)

if iter == 0 && func(a)\*func(b) > 0

error('На заданном интервале функция не меняет знак.');

end

mid = (a + b)/2;

fmid = func(mid);

iter = iter + 1;

if abs(fmid) < tol

root = mid;

return;

end

% Рекурсивный вызов для подходящего подотрезка

if func(a)\*fmid < 0

[root, iter] = bisection\_recursive(func, a, mid, tol, iter, max\_iter);

else

[root, iter] = bisection\_recursive(func, mid, b, tol, iter, max\_iter);

end

end

**>> Найденный корень: 3.526497, количество итераций: 117**

**Метод ложного положения.**

**1. Скрипт с использованием цикла while для поиска корня**

f = @(x) exp(x) + log(x) - 10\*x;

a = 3;

b = 4;

tol = 1e-4;

maxIter = 100;

% Проверяем, что функция меняет знак на концах интервала

if f(a)\*f(b) >= 0

error('Функция имеет одинаковые знаки на концах интервала. Метод ложного положения не применим!'); end

iter = 0;

approx = zeros(maxIter,1); % приближения корня

f\_values = zeros(maxIter,1); % значения функции в точке приближения

errors = zeros(maxIter,1); % разность между последовательными приближениями

% Заголовок таблицы в командном окне

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

fprintf('Iter|\ta\t\tb|\t\tc|\t\tf(c)|\t\terror|\n');

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

while iter < maxIter

% Вычисление точки по формуле метода ложного положения

c = (a\*f(b) - b\*f(a)) / (f(b) - f(a));

fc = f(c);

iter = iter + 1;

% Вычисляем ошибку как разность между текущим и предыдущим приближением

if iter > 1

errors(iter) = abs(c - approx(iter-1));

else

errors(iter) = 0; % для первой итерации

end

approx(iter) = c;

f\_values(iter) = fc;

% Вывод текущей итерации в командное окно

fprintf('%d\t%f\t%f\t%f\t%f\t%f\n', iter, a, b, c, fc, errors(iter));

% Если достигнута заданная точность по значению функции или по изменению приближения, то выходим

if abs(fc) < tol || (iter > 1 && errors(iter) < tol)

break;

end

% Обновление границ интервала

if f(a)\*fc < 0

b = c;

else

a = c;

end

end

% Обрезаем массивы до реально выполненного количества итераций

approx = approx(1:iter);

f\_values = f\_values(1:iter);

iter\_vector = 1:iter;

% Построение графиков

figure;

subplot(2,1,1)

plot(iter\_vector, approx, 'o-', 'LineWidth', 2);

xlabel('Количество итераций');

ylabel('Приближение c');

title('График последовательных приближений значения корня');

subplot(2,1,2)

plot(iter\_vector, f\_values, 's-', 'LineWidth', 2);

xlabel('Количество итераций');

ylabel('Значение f(c)');

title('График значений функции при последовательных приближениях');

-----------------------------------------------------

Iter| a b| c| f(c)| error|

-----------------------------------------------------

1 3.000000 4.000000 3.355474 -3.684969 0.000000

2 3.355474 4.000000 3.476222 -1.178942 0.120749

3 3.476222 4.000000 3.512200 -0.343812 0.035978

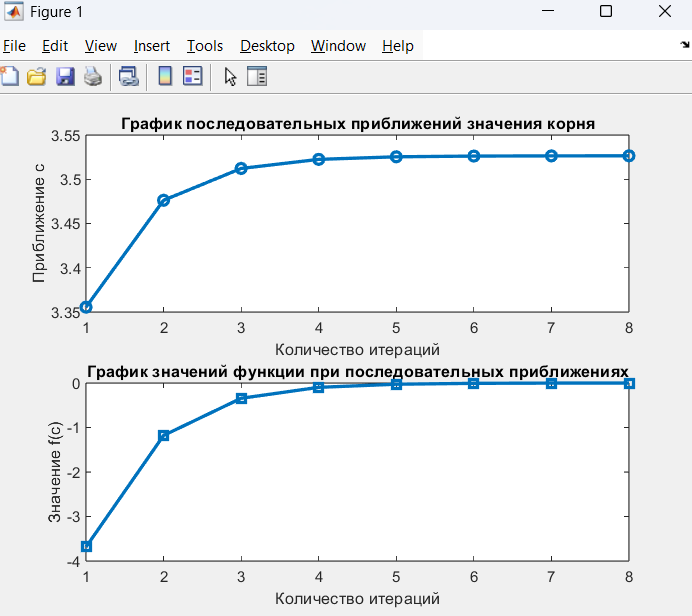
4 3.512200 4.000000 3.522472 -0.097518 0.010271

5 3.522472 4.000000 3.525367 -0.027441 0.002896

6 3.525367 4.000000 3.526181 -0.007704 0.000813

7 3.526181 4.000000 3.526409 -0.002162 0.000228

8 3.526409 4.000000 3.526473 -0.000606 0.000064

****

**2a. Функция с аргументами**

function [root, iter] = false\_position\_method(f, interval, tol, maxIter)

% Входные параметры:

% f - дескриптор функции (function handle)

% interval - интервал [a, b], в котором функция меняет знак

% tol - требуемая точность (опционально, по умолчанию 1e-4)

% maxIter - максимальное число итераций (опционально, по умолчанию 100)

%

% Выходные параметры:

% root - найденное приближение значения корня

% iter - количество выполненных итераций

if nargin < 3 || isempty(tol)

tol = 1e-4;

end

if nargin < 4 || isempty(maxIter)

maxIter = 100;

end

% Проверка корректности интервала

if numel(interval) ~= 2

error('Параметр interval должен быть в виде двухэлементного массива [a, b].');

end

a = interval(1);

b = interval(2);

if f(a) \* f(b) >= 0

error('Функция не меняет знак на концах интервала. Метод ложного положения не применим.');

end

iter = 0;

c\_old = a; % для вычисления разницы между итерациями

while iter < maxIter

c = (a \* f(b) - b \* f(a)) / (f(b) - f(a));

iter = iter + 1;

% Условия остановки: по значению функции или по изменению приближения

if abs(f(c)) < tol || abs(c - c\_old) < tol

root = c;

return

end

if f(a)\*f(c) < 0

b = c;

else

a = c;

end

c\_old = c;

end

% Если достигнуто максимальное число итераций, возвращается последнее приближение

root = c;

end

% Определяем функцию

f = @(x) exp(x) + log(x) - 10\*x;

interval = [3, 4];

tol = 1e-4;

maxIter = 100;

% Инициализируем массив для хранения конечной погрешности на каждой итерации

iterations = 1:maxIter;

final\_errors = zeros(size(iterations));

% Для каждого значения максимального числа итераций получаем корень и вычисляем абсолютную погрешность

for k = iterations

try

root = false\_position\_method(f, interval, tol, k);

final\_errors(k) = abs(f(root));

catch err

% В случае ошибки (например, неверный интервал) записываем NaN

final\_errors(k) = NaN;

end

end

% Построение графика с логарифмической осью ординат

figure;

semilogy(iterations, final\_errors, 'o-', 'LineWidth', 2);

xlabel('Количество итераций');

ylabel('Абсолютное значение f(root) (лог. шкала)');

title('Зависимость точности от количества итераций');

grid on;

>>Найденный корень: 3.526473 за 8 итераций

2b. Скрипт для построения графика зависимости точности от числа итераций

f = @(x) exp(x) + log(x) - 10\*x;

interval = [3, 4];

tol = 1e-4;

maxIter = 100;

% Инициализируем массив для хранения конечной погрешности на каждой итерации

iterations = 1:maxIter;

final\_errors = zeros(size(iterations));

% Для каждого значения максимального числа итераций получаем корень и вычисляем абсолютную погрешность

for k = iterations

try

root = false\_position\_method(f, interval, tol, k);

final\_errors(k) = abs(f(root));

catch err

% В случае ошибки (например, неверный интервал) записываем NaN

final\_errors(k) = NaN;

end

end

% Построение графика с логарифмической осью ординат

figure;

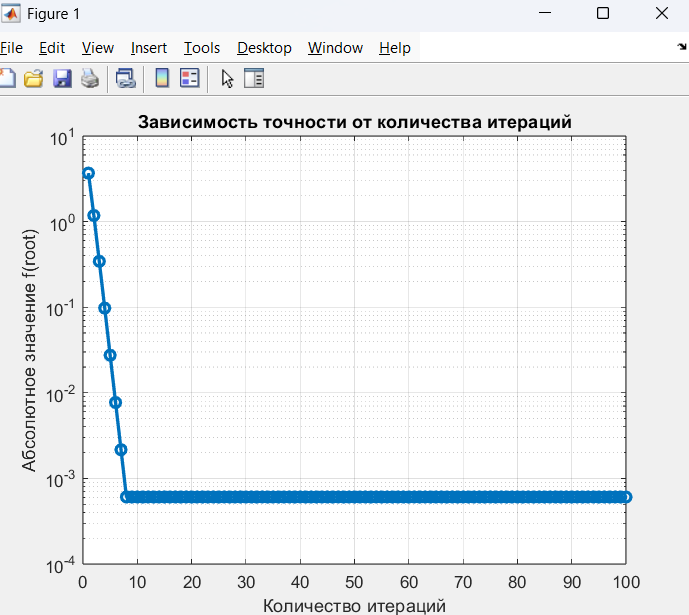
semilogy(iterations, final\_errors, 'o-', 'LineWidth', 2);

xlabel('Количество итераций');

ylabel('Абсолютное значение f(root)');

title('Зависимость точности от количества итераций');

grid on;



**3. Рекурсивная реализация метода ложного положения**

function [root, iter] = false\_position\_recursive(f, a, b, tol, maxIter, iter)

% Входные параметры:

% f - дескриптор функции

% a, b - начальный интервал [a, b]

% tol - требуемая точность

% maxIter - максимальное число итераций

% iter - текущее число итераций (необязательный параметр, по умолчанию 0)

%

% Выходные параметры:

% root - приближённое значение корня

% iter - общее число выполненных итераций

if nargin < 6 iter = 0; end

if iter >= maxIter

% Если превышено максимальное число итераций, возвращаем последнее приближение

root = (a \* f(b) - b \* f(a)) / (f(b) - f(a));

return;

end

% Вычисление приближения по методу ложного положения

c = (a \* f(b) - b \* f(a)) / (f(b) - f(a));

if abs(f(c)) < tol

root = c;

iter = iter + 1;

return;

end

iter = iter + 1;

if f(a)\*f(c) < 0 [root, iter] = false\_position\_recursive(f, a, c, tol, maxIter, iter);

else [root, iter] = false\_position\_recursive(f, c, b, tol, maxIter, iter); end

end

>> Найденный корень: 3.526496, количество итераций: 10

**Метод Ньютона.**

**1. Итерационный метод с циклом while**

f = @(x) exp(x) + log(x) - 10\*x;

df = @(x) exp(x) + 1./x - 10;

initial\_approximation = 5.00; % начальное приближение

tol = 1e-6; % требуемая точность

max\_iter = 100; % максимум итераций

% Массивы для хранения данных по итерациям

iter\_arr = [];

x\_arr = [];

f\_arr = [];

iter = 0;

x\_current = initial\_approximation;

while iter < max\_iter

f\_val = f(x\_current);

df\_val = df(x\_current);

% Сохраняем данные итерации

iter\_arr(end+1) = iter;

x\_arr(end+1) = x\_current;

f\_arr(end+1) = f\_val;

% Выход, если достигнута требуемая точность

if abs(f\_val) < tol

break;

end

% Формула Ньютона

x\_current = x\_current - f\_val / df\_val;

iter = iter + 1;

end

% Вывод результатов в командное окно в виде таблицы

T = table(iter\_arr', x\_arr', f\_arr', 'VariableNames', {'Iteration', 'Approximation', 'FunctionValue'});

disp(T);

% Построение графиков последовательных приближений и значений функции

figure;

subplot(2,1,1);

plot(iter\_arr, x\_arr, '-o', 'LineWidth',1.5);

xlabel('Итерация');

ylabel('Приближение x');

title('Приближения x vs Итерация');

grid on;

subplot(2,1,2);

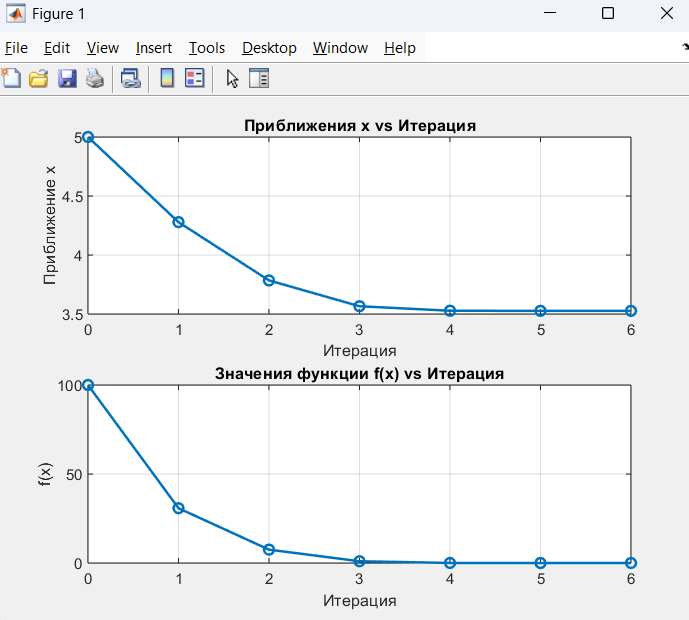
plot(iter\_arr, f\_arr, '-o', 'LineWidth',1.5);

xlabel('Итерация');

ylabel('f(x)');

title('Значения функции f(x) vs Итерация');

grid on;

****

Iteration Approximation FunctionValue  
 \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

0 5 100.02

1 4.2784 30.795

2 3.7846 7.5021

3 3.5657 0.97941

4 3.5275 0.025407

5 3.5265 1.8519e-05

6 3.5265 9.8623e-12

**2а. Функция newton\_method.m**

function [root, iter] = newton\_method(f, init, tol, max\_iter)

% Входные аргументы:

% f - дескриптор функции (функция от x)

% init - начальное приближение (числовое скалярное значение)

% tol - требуемая точность (необязательный, по умолчанию 1e-6)

% max\_iter - максимум итераций (необязательный, по умолчанию 100)

%

% Выходные аргументы:

% root - приближённое значение корня

% iter - число итераций, выполненных до сходимости

if nargin < 2

error('Требуется передать не менее двух аргументов: функцию и начальное приближение.');

end

if ~isa(f, 'function\_handle')

error('Первый аргумент должен быть дескриптором функции.');

end

if nargin < 3 || isempty(tol)

tol = 1e-6;

end

if nargin < 4 || isempty(max\_iter)

max\_iter = 100;

end

if ~isnumeric(init) || ~isscalar(init)

error('Начальное приближение должно быть числовым скалярным значением.');

end

% Параметры для численного дифференцирования

h = 1e-6;

% Инициализация переменной

x\_current = init;

iter = 0;

while iter < max\_iter

f\_val = f(x\_current);

% Проверка сходимости по значению функции

if abs(f\_val) < tol

break;

end

% Численное вычисление производной

df\_val = (f(x\_current + h) - f(x\_current)) / h;

if df\_val == 0

error('Производная равна нулю. Решение не найдено.');

end

% Итерационный шаг по методу Ньютона

x\_current = x\_current - f\_val / df\_val;

iter = iter + 1;

end

root = x\_current;

end

>> Найденный корень: 3.526499 за 5 итераций

**2б. Скрипт для построения графика newton\_accuracy\_graph.m**

f = @(x) exp(x) + log(x) - 10\*x;

initial\_approximation = 5.0;

tol = 1e-6;

actual\_root = 3.5265;

% Диапазон максимального числа итераций

max\_iter\_values = 1:100;

errors = zeros(size(max\_iter\_values));

for i = 1:length(max\_iter\_values)

max\_iter = max\_iter\_values(i);

try

[root, iter] = newton\_method(f, initial\_approximation, tol, max\_iter);

errors(i) = abs(root - actual\_root);

catch ME

% Если возникла ошибка (например, деление на ноль), записываем NaN

errors(i) = NaN;

end

end

figure;

semilogy(max\_iter\_values, errors, 'b-o', 'LineWidth',1.5);

xlabel('Максимальное число итераций');

ylabel('Ошибка |x - x\_{actual}|');

title('Зависимость точности найденного корня от числа итераций');

grid on;



**3. Рекурсивная функция для метода Ньютона**

function [root, iter] = newton\_recursive(f, x\_current, tol, max\_iter, iter)

% Входные аргументы:

% f - дескриптор функции

% x\_current - текущее приближение корня

% tol - требуемая точность (например, 1e-6)

% max\_iter - максимум допустимых итераций

% iter - текущий номер итерации (необязательный аргумент, по умолчанию 0)

%

% Выходные аргументы:

% root - найденное приближённое значение корня

% iter - общее число итераций, выполненных до сходимости

if nargin < 5

iter = 0;

end

if abs(f(x\_current)) < tol

root = x\_current;

return;

end

% Численное вычисление производной

h = 1e-6;

df\_val = (f(x\_current + h) - f(x\_current)) / h;

if df\_val == 0

error('Производная равна нулю. Решение не найдено.');

end

% Один итерационный шаг метода Ньютона

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

iter = iter + 1;

% Рекурсивный вызов

[root, iter] = newton\_recursive(f, x\_next, tol, max\_iter, iter);

end

>> Найденный корень: 3.584550, количество итераций: 102

**Метод Секущих**

**1. Скриптовый файл: Итерационный метод секущих с циклом while**

% Определяем функцию

f = @(x) exp(x) + log(x) - 10\*x;

% Начальные приближения

x0 = 3;

x1 = 4;

tol = 1e-4;

max\_iter = 100;

iter = 0;

data = []; % для хранения истории итераций

fprintf(' Итерация x\_n f(x\_n)\n');

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

while iter < max\_iter

f0 = f(x0);

f1 = f(x1);

if (f1 - f0) == 0

error('Обнаружено деление на 0. Метод не может продолжаться.');

end

% Формула метода секущих

x2 = x1 - f1 \* (x1 - x0) / (f1 - f0);

iter = iter + 1;

% Запись результатов итерации

data = [data; iter, x2, f(x2)];

fprintf(' %2d %.6f %.6e\n', iter, x2, f(x2));

if abs(x2 - x1) < tol

break;

end

% Обновление приближений

x0 = x1;

x1 = x2;

end

% Построение графиков:

% - График изменения приближённого значения корня с номером итерации

% - График изменения модуля значения функции

figure;

subplot(2,1,1);

plot(data(:,1), data(:,2), '-o');

xlabel('Номер итерации');

ylabel('Приближение x\_n');

title('График последовательных приближений (метод секущих)');

grid on;

subplot(2,1,2);

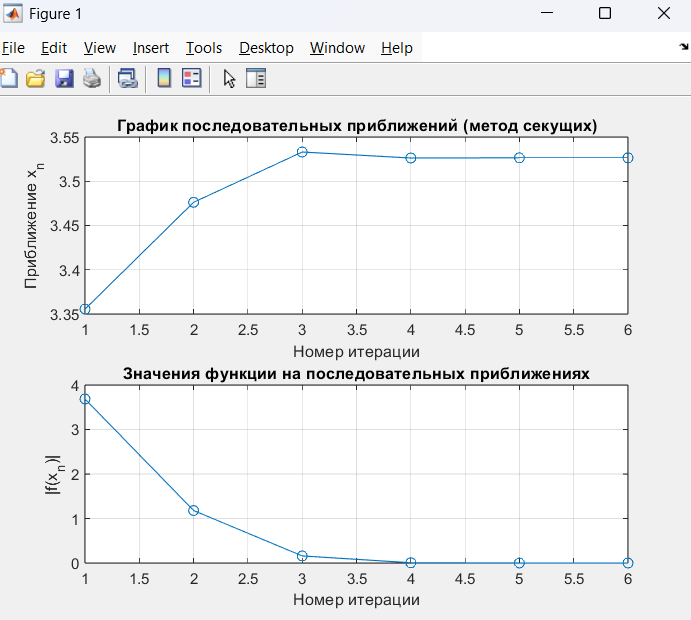
plot(data(:,1), abs(data(:,3)), '-o');

xlabel('Номер итерации');

ylabel('|f(x\_n)|');

title('Значения функции на последовательных приближениях');

grid on;



Итерация x\_n f(x\_n)

----------------------------------------

1 3.355474 -3.684969e+00

2 3.476222 -1.178942e+00

3 3.533028 1.593252e-01

4 3.526265 -5.657124e-03

5 3.526497 -2.574329e-05

6 3.526498 4.188472e-09

2a. Функция для вычисления корня методом секущих

function [root, iter] = secant\_method(func, range, tol, max\_iter)

% Входные аргументы:

% func - дескриптор функции (function handle)

% range - вектор из двух элементов с начальными приближениями [x0, x1]

% tol - требуемая точность (необязательный, по умолчанию 1e-4)

% max\_iter - максимальное число итераций (необязательный, по умолчанию 100)

%

% Выходные аргументы:

% root - найденное приближение корня

% iter - количество выполненных итераций

% Проверка аргументов

if nargin < 2

error('Необходимо задать функцию и диапазон начальных приближений.');

end

if ~isa(func, 'function\_handle')

error('Первый аргумент должен быть дескриптором функции.');

end

if numel(range) ~= 2

error('Диапазон начальных приближений должен содержать два элемента.');

end

if nargin < 3 || isempty(tol)

tol = 1e-4;

end

if nargin < 4 || isempty(max\_iter)

max\_iter = 100;

end

x0 = range(1); x1 = range(2); iter = 0;

while iter < max\_iter

f0 = func(x0);

f1 = func(x1);

if (f1 - f0) == 0

error('Деление на ноль, метод не может продолжаться.');

end

x2 = x1 - f1\*(x1 - x0)/(f1 - f0);

iter = iter + 1;

if abs(x2 - x1) < tol

root = x2;

return;

end

x0 = x1; x1 = x2;

end

root = x2;

warning('Максимальное число итераций достигнуто'); end

>> Найденный корень: 3.526498 за 6 итераций

**2b. Скрипт для построения графика зависимости точности от количества итераций**

f = @(x) exp(x) + log(x) - 10\*x;

exact\_root = 3.5265;

range = [3, 4];

tol = 1e-8;

max\_iter\_total = 20;

errors = zeros(max\_iter\_total, 1);

iters = zeros(max\_iter\_total, 1);

for k = 1:max\_iter\_total

[root, iter] = secant\_method(f, range, tol, k);

iters(k) = iter;

errors(k) = abs(root - exact\_root);

end

figure;

% Используем логарифмический масштаб по оси Y

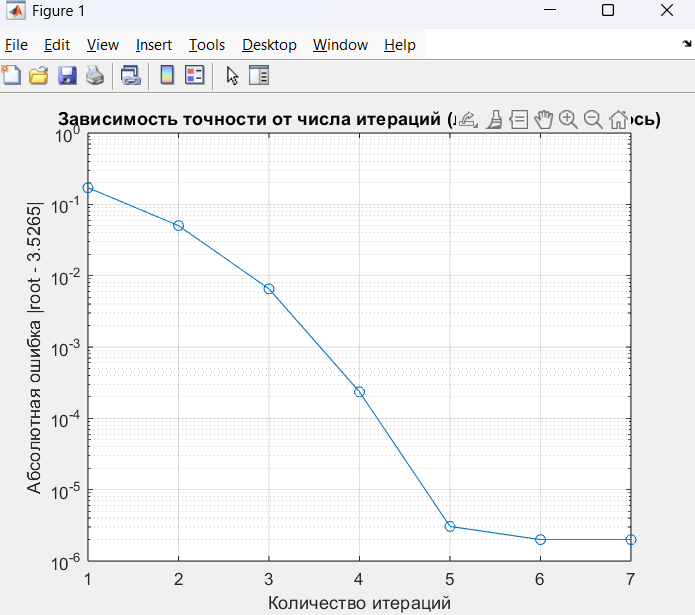
semilogy(iters, errors, '-o');

xlabel('Количество итераций');

ylabel('Абсолютная ошибка |root - 3.5265|');

title('Зависимость точности от числа итераций (логарифмическая ось)');

grid on;



3. Рекурсивная функция для метода секущих

function [root, iter] = secant\_method\_recursive(func, x0, x1, tol, max\_iter, iter)

% Входные аргументы:

% func - дескриптор функции (function handle)

% x0, x1 - два начальных приближения

% tol - требуемая точность (необязательный, по умолчанию 1e-4)

% max\_iter - максимальное число итераций (необязательный, по умолчанию 100)

%

% Выходные аргументы:

% root - найденное приближение корня

% iter - количество выполненных итераций (суммарно)

if nargin < 6

iter = 0;

end

if nargin < 4 || isempty(tol)

tol = 1e-4;

end

if nargin < 5 || isempty(max\_iter)

max\_iter = 100;

end

f0 = func(x0);

f1 = func(x1);

if (f1 - f0) == 0

error('Деление на ноль. Метод не может продолжаться.');

end

% Вычисление нового приближения

x2 = x1 - f1\*(x1 - x0)/(f1 - f0);

iter = iter + 1;

% Проверка условия остановки

if abs(x2 - x1) < tol || iter >= max\_iter

root = x2;

return;

else

% Рекурсивный вызов: передаем x1 и новое x2

[root, iter] = secant\_method\_recursive(func, x1, x2, tol, max\_iter, iter);

end

end

>>Найденный корень: 3.526498, количество итераций: 6

**Вызов функций для заданий 2а и 3**

f = @(x) exp(x) + log(x) - 10\*x;

interval = [3, 4];

tol = 1e-4;

maxIter = 100;

[root, iter] = secant\_method(f, interval, tol, maxIter);

% [root, iter] = newton\_method(f, 5, tol, maxIter);

fprintf('Найденный корень: %f за %d итераций\n', root, iter);

f = @(x) exp(x) + log(x) - 10\*x;

a = 3; b = 4;

tol = 1e-4;

maxIter = 100;

[root, iter] = false\_position\_recursive(f, a, b, tol, maxIter);

fprintf('Найденный корень: %f, количество итераций: %d\n', root, iter);