**МАИ**

**Лабораторная работа №12**

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

**Задания 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.**  
% newton\_method.m

% Решение уравнения: exp(x) + log(x) - 10 = 0 методом Ньютона

% Определяем функцию и её производную

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

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

% Начальное приближение (учтите, что ln(x) определена только при x > 0)

x0 = 2;

tol = 1e-6; % заданная точность

maxIter = 100; % максимальное число итераций

% Инициализация переменных для сохранения итераций

iter = 0;

X = x0; % вектор приближений

fX = f(x0); % вектор значений функции

% Итерационный процесс с использованием цикла while

while abs(fX(end)) > tol && iter < maxIter

iter = iter + 1;

x\_old = X(end);

% Метод Ньютона: x\_new = x\_old - f(x\_old)/f'(x\_old)

x\_new = x\_old - f(x\_old)/df(x\_old);

% Сохранение новых значений

X = [X; x\_new];

fX = [fX; f(x\_new)];

end

% Вывод результатов в виде таблицы

T = table((0:iter)', X, fX, 'VariableNames', {'Номер Итерации', 'Значение X', 'Значение f(x)'});

disp('Результаты поиска корня:')

disp(T)

%% Построение графиков последовательных приближений

figure;

% График приближений x по итерациям

subplot(2,1,1)

plot(0:iter, X, '-o', 'LineWidth', 1.5)

xlabel('Номер итерации')

ylabel('Приближение x')

title('Последовательные приближения корня')

grid on;

% График значений функции f(x) по итерациям

subplot(2,1,2)

plot(0:iter, fX, '-o', 'LineWidth', 1.5)

xlabel('Номер итерации')

ylabel('Значение f(x)')

title('Значения функции на последовательных приближениях')

grid on;

%% Геометрическая интерпретация метода Ньютона

figure;

% Зададим область для построения графика функции

x\_min = min(X) - 1;

x\_max = max(X) + 1;

x\_vals = linspace(x\_min, x\_max, 400);

plot(x\_vals, f(x\_vals), 'b-', 'LineWidth', 2)

hold on;

% Для каждой итерации строим касательную к графику функции

for i = 1:length(X)-1

xi = X(i);

fxi = f(xi);

dfxi = df(xi);

% Уравнение касательной: y = fxi + dfxi\*(x - xi)

xt = linspace(xi - 0.5, xi + 0.5, 50);

yt = fxi + dfxi\*(xt - xi);

plot(xt, yt, 'r--', 'LineWidth', 1)

% Отображаем точки приближений и проводим вертикальную линию до оси absciss

plot(xi, fxi, 'ko', 'MarkerFaceColor', 'k')

plot([xi, xi], [0, fxi], 'k:', 'LineWidth', 1)

end

% Отметим последнюю точку

plot(X(end), f(X(end)), 'ko', 'MarkerFaceColor', 'k')

xlabel('x')

ylabel('f(x)')

title('Геометрическая интерпретация метода Ньютона')

grid on;

hold off;

**Задание 2.**

**а) Разработать функцию, позволяющую вычислить сумму ряда на основе цикла while end. Функция должна принимать три аргумента - значение х, допустимая погрешность tol, максимальное число итераций k\_max - и возвращать два значения - вычисленная сумма и количество итераций. Второй и третий входные аргументы являются дополнительными. По умолчанию tol = 1e-4, k\_max = 50.**

**При выполнении задания найти коэффициент, связывающий ak член ряда с ak+1, и реализовать его вычисление в виде отдельной локальной функции, вызываемой внутри цикла while end;**

function [S, k] = sum\_series\_while(x, tol, k\_max)

% sum\_series\_while вычисляет сумму ряда a\_k = x^(2k)/4^(k+1)

% с использованием цикла while.

%

% Входные параметры:

% x - значение переменной (из области сходимости)

% tol - допустимая погрешность (по умолчанию 1e-4)

% k\_max- максимальное число итераций (по умолчанию 50)

%

% Выходные параметры:

% S - вычисленная сумма ряда

% k - число итераций, произведённых в цикле

if nargin < 2, tol = 1e-4; end

if nargin < 3, k\_max = 50; end

S = 0;

k = 0;

a = term(x, 0); % первый член

while (abs(a) > tol) && (k < k\_max)

S = S + a;

k = k + 1;

% вычисляем следующий член с помощью локальной функции, которая вычисляет коэффициент перехода

coeff = calc\_coeff(x, k-1);

a = a \* coeff;

end

% Локальная функция для вычисления коэффициента, связывающего a\_k и a\_{k+1}

function c = calc\_coeff(x\_local, k\_local)

% Для a\_k = x^(2k)/4^(k+1) получаем:

% a\_{k+1}/a\_k = x^2/4.

c = x\_local^2/4;

end

% Локальная функция для вычисления a\_k

function ak = term(x\_local, k\_val)

ak = x\_local^(2\*k\_val)/4^(k\_val+1);

end

end

**>> sum\_series\_while(0.5)**

**>> ans =**

**8.2173e+67**

**б) Написать скрипт-файл, в котором:**

**- выполняется вызов функции из пункта (а) с одним входным и одним выходным аргументами и осуществляется построение в командном окне таблицы, содержащей одиннадцать значений переменной x и соответствующих им значений сумм;**

**- выполняется вызов функции из пункта (а) с двумя или тремя входными аргументами и происходит построение в графическом окне графика изменения точности вычисления в зависимости от числа итераций для случайного значения x из построенной таблицы;**

%% task2\_script.m

clear; clc; close all;

% Часть 1. Построение таблицы для 11 значений x от 0 до 1.

x\_vals = linspace(0,1,11);

S\_vals = zeros(size(x\_vals));

fprintf(' x S(x)\n');

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

for i = 1:length(x\_vals)

% Вызываем функцию с одним входным (т.е. используются значения по умолчанию для tol и k\_max)

[S\_i, ~] = sum\_series\_while(x\_vals(i));

S\_vals(i) = S\_i;

fprintf('%6.3f %10.6f\n', x\_vals(i), S\_i);

end

% Часть 2. Построение графика зависимости погрешности от числа итераций.

% Выберем случайное x из таблицы (исключим 0, чтобы не было тривиального случая)

nonzero\_idx = find(x\_vals > 0);

rand\_idx = nonzero\_idx(randi(length(nonzero\_idx)));

x\_rand = x\_vals(rand\_idx);

% Для данного x сохраняем погрешность (разница между текущей суммой и точным значением)

tol = 1e-8; % можно задать малую точность для анализа

k\_max = 50;

S\_current = 0;

errors = [];

iters = [];

exact\_val = 1/(4 - x\_rand^2);

a = x\_rand^(0)/4^(1); % первый член, равен 1/4

k = 0;

while (k < k\_max)

S\_current = S\_current + a;

err = abs(S\_current - exact\_val);

errors(end+1) = err;

iters(end+1) = k;

% Используем локальную функцию для вычисления коэффициента (аналогично предыдущей функции)

coeff = local\_calc\_coeff(x\_rand);

a = a \* coeff;

k = k + 1;

end

% Построение графика зависимости погрешности от числа итераций

figure;

semilogy(iters, errors, 'b-o','LineWidth',1.5);

xlabel('Номер итерации');

ylabel('Абсолютная погрешность');

title(sprintf('Погрешность вычисления суммы ряда для x = %.3f', x\_rand));

grid on;

% Локальная функция для расчета коэффициента перехода

function c = local\_calc\_coeff(x\_val)

c = x\_val^2/4;

end

>> x S(x)

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

0.000 0.250000

0.100 0.250625

0.200 0.252500

0.300 0.255752

0.400 0.260400

0.500 0.266602

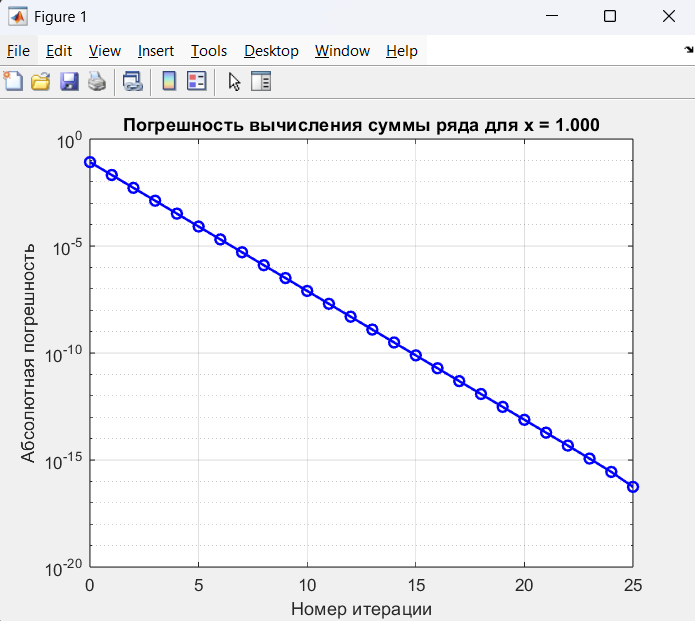
0.600 0.274707

0.700 0.284836

0.800 0.297588

0.900 0.313373

1.000 0.333252



**3. Создать рекурсивную функцию для вычисления суммы ряда. Рассмотреть два варианта с вложенной и локальной функциями. Отобразить рекурсивный процесс в виде рисунка в отчёте.**

**Вариант 1. С использованием вложенной функции**

function S = sum\_series\_recursive\_nested(x, tol)

% Рекурсивное вычисление суммы ряда с использованием вложенной функции.

% Входные параметры:

% x - значение переменной

% tol - допустимая погрешность (опционально, по умолчанию 1e-4)

if nargin < 2, tol = 1e-4; end

% Начинаем рекурсию с k = 0 и первым членом a0 = 1/4

[S, ~] = rec\_sum(0, x^(0)/4^(1));

function [S\_local, a\_local] = rec\_sum(k, a\_k)

% Точное значение вклада, если |a\_k| меньше tol, прекращаем рекурсию

if abs(a\_k) < tol

S\_local = a\_k;

else

% Вычисляем следующий член ряда через коэффициент перехода: a\_{k+1} = a\_k\*(x^2/4)

coeff = x^2/4;

a\_next = a\_k \* coeff;

[S\_next, ~] = rec\_sum(k+1, a\_next);

S\_local = a\_k + S\_next;

end

end

end

%% Пример вызова рекурсивной функции с вложенной функцией

x = 0.5; % значение x

tol = 1e-4; % допустимая погрешность

S\_nested = sum\_series\_recursive\_nested(x, tol);

fprintf('Рекурсивная сумма (вложенная функция) для x = %.2f: %.6f\n', x, S\_nested);

>> Рекурсивная сумма (вложенная функция) для x = 0.50: 0.266663

**Вариант 2. С использованием локальной функции (отдельного файла)**

function S = sum\_series\_recursive\_local(x, tol)

% Рекурсивное вычисление суммы ряда с использованием локальной функции

if nargin < 2, tol = 1e-4; end

S = rec\_sum(0, x^(0)/4^(1), x, tol);

end

function S = rec\_sum(k, a\_k, x, tol)

if abs(a\_k) < tol

S = a\_k;

else

coeff = calc\_coeff(x);

a\_next = a\_k \* coeff;

S = a\_k + rec\_sum(k+1, a\_next, x, tol);

end

end

function c = calc\_coeff(x)

c = x^2/4;

end

%% Пример вызова рекурсивной функции с вложенной функцией

x = 0.5; % значение x

tol = 1e-4; % допустимая погрешность

S\_nested = sum\_series\_recursive\_local(x, tol);

fprintf('Рекурсивная сумма (вложенная функция) для x = %.2f: %.6f\n', x, S\_nested);

>> Рекурсивная сумма (вложенная функция) для x = 0.50: 0.266663

**Задание 4**

**Сделать вывод о достоинствах и недостатках итеративного и рекурсивного подходов к вычислению суммы функционального ряда.**

**Итеративный подход:**

* **Плюсы:  
  – Более очевидное управление циклом и памятью;  
  – Обычно требует меньше накладных расходов (нет вызовов функций на каждом шаге);  
  – Легко реализуется и отлаживается.**
* **Минусы:  
  – При сложной логике может получаться громоздкий код;  
  – Реализация некоторых алгоритмов может быть менее естественной.**

**Рекурсивный подход:**

* **Плюсы:  
  – Выражает решение задачи в компактном виде, часто более математически «естественно»;  
  – Может быть проще для понимания некоторых алгоритмических решений.**
* **Минусы:  
  – Рекурсия может привести к значительному расходу памяти (глубокие вызовы функций) и возможному переполнению стека;  
  – Для задач с большим числом итераций итеративный подход обычно эффективнее.**