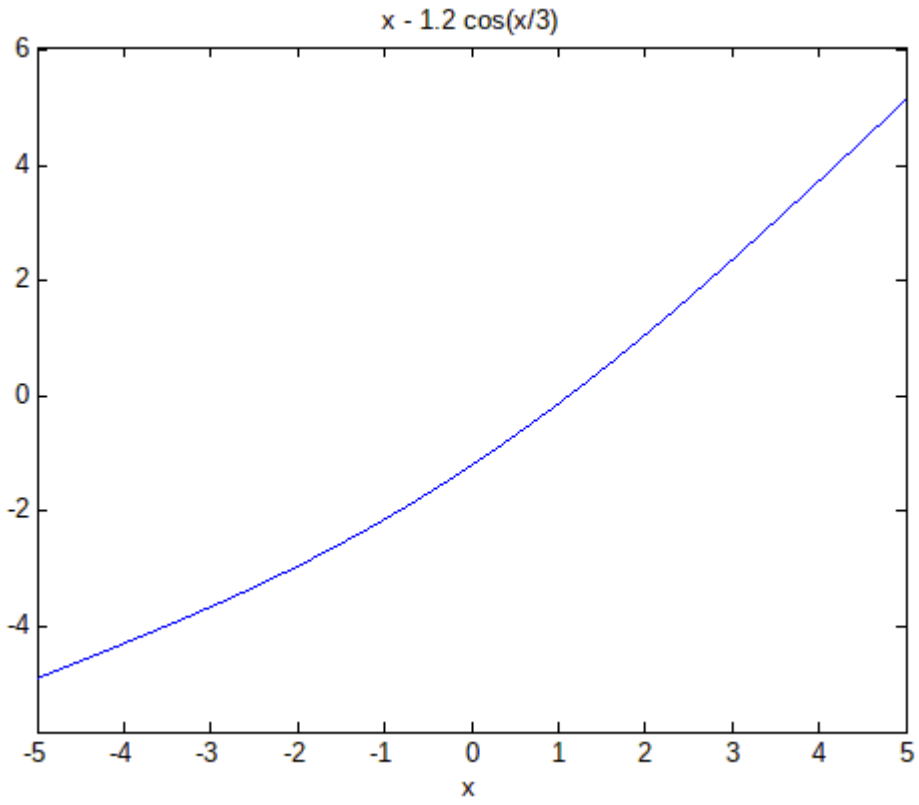
Отчет по лабораторной работе 4
Кобака Ф.А.
Вариант 4

Определения интервала на котором лежит корень

```
>> f = sym('x - 1.2*cos(x/3)')

f =

x - 1.2*cos(x/3)

>> ezplot(f , -5 , 5)

>> diff(f)

ans =

0.4*sin(x/3) + 1

>> solve(ans)

ans =

 4.700397710917233235992170587741 5*i - 4.712388980384689857693965074919 3
 14.137166941154069573081895224758 - 4.700397710917233235992170587741 5*i

>>
```

Производная не имеет дейсвительных корней, что говорит о том, что функция либо постоянно возрастает, либо постоянно убывает, значит имеет не более одного корня, графически видно, что он лежит на отрезке (0,2)



Код

Метод половинного деления:

```
% expr – символьное выражение описывающее уравнение
% a b – границы в которых требутеся искать корень
% info – надо ли отображать отладочную информацию
% epsil – точность решения

function res = halfDivisionMetod(expr , a , b , epsil, info)

    % до первой итерации проснициализируем предыдущую точку как а
    px = a;
    % тут будем считать число итераций
    i = 0;

    while (true)

        % рассчитаем длинну интервала, деленную на два
        len = (b - a) / 2;
        % от а шагнем на пол расстояния до б и получим новое приближение
        x = a + len;
        % найдем значение функции в новом приближении
        y = subs(expr , x);

        % проврим не выполнилься ли критерий сходисти, не попали
        % ли мы случайно в точное решение
        if(abs(px - x) < epsil || subs(expr , x) == 0)
            res = x; % полагаем решение найдено возвращаем результат
            return;
        end

        % отладочная инфа+++++++++++++++++++++++++++++++++++++++++++++
        if(info)
            disp(['++++++++++++++++++' , num2str(i)]);
            disp([num2str(x)]);
            disp(['++++++++++++++++++' , num2str(i)]);
        end
        % отладочная инфа+++++++++++++++++++++++++++++++++++++++++++++

        % доплюсовываем еще одну итерацию
        i = i + 1;
        % запоминаем предыдущую точку px
        px = x;

        % получаем значение функции в крайних точках
        ya = subs(expr , a);
        yb = subs(expr , b);

        % если знак у значения функции в полученном приближении и
        % а совпадает
        if(ya*y > 0)
            a = x;% перемещаемся в точку а
        else
            b = x;% если не совпадает то перемещаемся в точку б
        end


    end
```

```
end
```

Метод хорд:

```
% expr – символьное выражение описывающее уравнение
% a b – границы в которых требутеся искать корень
% info – надо ли отображать отладочную информацию
% epsil – точность решения


function [solution] = chordMetod(expr ,a , b ,epsil  , info)
    % отладочная информация++++++++++++++++++++++++++++++++++
    if(info)
        x = a:(b - a)/20:b;
        plot(x , subs(expr , x));
        hold on;
        line([a b] , [0 0] , 'LineStyle' , '--' , 'Color' , 'r');
    end
    % отладочная информация++++++++++++++++++++++++++++++++++

    % запомнаем предыдущий x
    px = a;
    % обнуляем считалльщик итераций
    i = 0;

    % для критерия сходимоти надо найти m – наименьшее значения модуля
    % производной на отрезке a b

    % ищем производную
    dexpr = diff(expr);
    % точку в котрой модуль производной принимает минимальное значение
    minX = fminbnd(matlabFunction(abs(dexpr)) , a , b);
    % теперь находим значение модуля производной в этой точке
    m = subs(abs(dexpr) , minX);



    while(true)
        % вычисляем значение функции на конце б
        yb = subs(expr , b);
        % вычисляем значение функции на конце a
        ya = subs(expr , a);
        % по формуле считаем новый x
        x = (-ya / (yb - ya))*(b - a) + a;
        % вычисляем занчение функции в этом новом x
        y = subs(expr , x);
        % приращаем показатель числа итераций на 1
        i = i+1;

        % проверяем сходимоть
        if(y == 0 || abs(subs(expr, x))/m <= epsil)
            solution = x;% критерий выполнен => возвращаем результат
            return;
        end
```

```
        % запоминаес x для следующей итерации
        px = x;

        % отладочная информация++++++++++++++++++++++++++++++++++
        if(info)

            disp('++++++++++++++++++++++++++++++');
            disp(['iteration' , num2str(i)]);
            disp(['x = ', num2str(x)]);
            disp(['y = ', num2str(y)]);
            disp(['f(a) = ' , num2str(ya)]);
            disp(['f(b) = ' , num2str(yb)])
            disp('++++++++++++++++++++++++++++++');

            line([a b] , [ya yb]);
            plot(x , 0 , '.' , 'MarkerSize' , 20);

        end
        % отладочная информация++++++++++++++++++++++++++++++++++


        % проверяем какую точку надо сместить
        if(y * ya > 0)
            a = x; % если оказалось, что новый y и значение функции в предыдущем
                   значении a имеют один знак, то надо сметить именно a к x
        else
            b = x; % в этом случае надо b сместить к x
        end

    end

    hold off;
end
```

Метод Простой итерации:

```
% expr – символьное выражение описывающее уравнение
% a b – границы в которых требутеся искать корень
% info – надо ли отображать отладочную информацию
% epsil – точность решения
function res = simpIterMetod(expr , a , b , epsil , info)
    %++++++++++получение функции ФИ +++++++++++++++
    % получаем производную с минусом
    mdexpr = -diff(expr);
    % по сути минимум функции с минусом это максимум функции без него
    maxX = fminbnd(matlabFunction(mdexpr) , a , b);

    % получем точку из отрезка [a b] в которой производная максимльна
    M = subs(expr , maxX);

    % формируем функцию fi
    fi = sym('x') – (1/M)*expr;
    %++++++++++получение функции ФИ +++++++++++++++

    % запоминаем передыдущее приближение как a, но это только для первой итерации
    px = a;
    % обнуляем счетчик итераций
    i = 0 ;
```

```
    % опрределим q для критерия сходимоти – по сути за него можно взять
    % максимум модуля производной фи на отрезке a b

    % найдем производную фи
    dfi = diff(fi);
    % ищем функцию – подуль фи с минусом. Зачем минус? см. далее
    antiABSdfi = -abs(dfi);
    % тут мы найдем максимум,используя фунцию для минимума, когда поставили минус
    % перед функцией максимум стал минимумом => получаем точку где |fi| максимален
    maxX = fminbnd(matlabFunction(antiABSdfi) , a , b);
    % находим заначение |fi| в этой точке
    q = subs(abs(dfi) , maxX);

    while(true)
        % получаем значение фи в пердыдущем приближении
        x = subs(fi , px);

        % проверяем не сошелся ли метод
        if(y == 0 || abs(subs(expr, x))/m <= epsil)
            res = x; % если сошелся, возвращаем значение
            return;
        end

        % запоминаем предыдущее приближение
        px = x;

        % отладочная инфа++++++++++++++++++++++++++++++++
        if (info)
            disp(['+++++++++++++++++' , num2str(i)]);
            disp(num2str(x));
            disp(['+++++++++++++++++' , num2str(i)]);
        end
        % отладочная инфа++++++++++++++++++++++++++++++++

        i = i + 1;
    end

end
```

Метод Ньютона:

```
% expr – символьное выражение описывающее уравнение
% a b – границы в которых требутеся искать корень
% info – надо ли отображать отладочную информацию
function res = eqSolveNewton(expr , a , b , epsil , info)

    % вычисляем производную
    dexpr = diff(expr);
    % вычисляем вторую производную
    d2expr = diff(expr);

    % узнаем имя производной используемой
    varname = symvar(expr);

    % находим функцию фи
    fi = varname - (expr / dexpr);

    % определим оптимальное начальное прилижение
```

```
    if(subs(d2expr , a)*subs(expr , a) > 0)
        px = a;
    else
        px = b;
    end

    % ищем минимум производной, на отрезке a b
    m = subs(dexpr , fminbnd(matlabFunction(dexpr) , a , b));
    % обнуляем счетчик итераций
    i = 0;

    while(true)

        % находим следующее приблмжение
        x = subs(fi , px);
        % находим значение функции в новом приближении
        y = subs(expr , x);

        % определяем сходимость метода
        if(abs(y/m) <= epsil)
            res = x;
            return;% если метод сошелся, возвращаем значение
        end

        px = x; % запоминаем предыдущее приближение

        % отладочная информация++++++++++++++++++++++++++
        if(info)
            disp(['+++++++++++++++++' , num2str(i)]);
            disp(num2str(x));
            disp(['+++++++++++++++++' , num2str(i)]);
        end
        % отладочная информация++++++++++++++++++++++++++
        % доплюсовываем чило итераций
        i = i + 1;
    end

end
```

Результирующий файл, для моего варианта:

```
f = sym('x-1.2*cos(x/3)');
a = 0; b =2 ; epsil = 0.01; % задаем начальные значения

% метод половинного деления
disp('half division metod result+++++++++++++++');
halfDivisionMetod(f , a , b , epsil , false)
% метод хорд
disp('chords metod result+++++++++++++++++');
chordMetod(f , a, b, epsil , false)
% метод простой иетрации
disp('simple iterarions metod resutl++++++++++++++++')
simpIterMetod(f , a , b,  epsil , false)
% метод Ньютона
disp('Newton metod resutl++++++++++++++++')
eqSolveNewton(f , a , b,  epsil , false)
```

Результат выполнения

half division metod result+++++++++++++++

ans =

    1.1172

chords metod result++++++++++++++++++

ans =

    1.1153

simple iterarions metod resutl+++++++++++++++++

ans =

    1.1162

Newton metod resutl+++++++++++++++++

ans =

    1.1177