```matlab
classdef set_2
    methods(Static)

        %% Problem 1: NR Method
        function[x_0] = Nr_method(a, x_0, err_acc)

            function[y] = fxn_prime(x)

                y = log(x / a) + 1;

            end


            function[y] = fxn_Dprime(x)
              y = 1 / x;


            end

            y = fxn_prime(x_0);

            while (abs(y) > err_acc)

                x_0 = x_0 - (fxn_prime(x_0))/(fxn_Dprime(x_0));
                y = fxn_prime(x_0);
            end


        end



        %% Problem 2: Eigenvalue Bisection
        function [B]= con_eig(C,N)

            % My Uncle told me the usefulness of this method relies in
            % manipulating the equation to equal zero -V- values. Low key
            % kind of elegant
            function y = fxn(b)

                y = 1 - b * cot(b) - C;
            end

            % 1 decimal more percise than the Universal gas constant's sig
            % figs according to google becuase fluids and gasses.
            err_acceptable= 1e-9;
            x_M = 0;

            x_L=0.00001:pi:N*pi;
            x_U=pi-0.00001:pi:N*pi;

            B=zeros(1,N); % and initialize space for B


            for n=1:N
```

```
        error = 100;
        % since code/alg is adjusted to find a zero for all given value
        % combinations, my error will approach zero the more percise it
        % becomes, thus normalization is not needed.
        while ( error > err_acceptable)
            x_M = (x_L(n) + x_U(n)) / 2;


            if (fxn(x_M) * fxn(x_L(n)) < 0)
                x_U(n) = x_M;
                error = fxn(x_U(n));

            else
                x_L(n) = x_M;
                error = fxn(x_L(n));

            end

        end

        B(n) = x_M;
    end


end

%% Problem 3: Bisection
function [B, it]= p3_bisect(x_l, x_u, C,  err_accept)
    % C = 22;
    error = 100;
    x_M = 0;
    it = 0;


    function y = fxn(b)
        it = it + 1;
        y = exp(b) * log(b) - C;
    end

     function METH()

       x_M = (x_u + x_l) / 2;
    end


    METH();
    % since code/alg is adjusted to find a zero for all given value
    % combinations, my error will approach zero the more percise it
    % becomes, thus normalization is not needed.
    while ( error > err_accept)
        x_old = x_M;

        if (fxn(x_M) * fxn(x_l) < 0)
            x_u = x_M;
```

```
        else
            x_l = x_M;


        end

        METH();
        error = abs(x_old - x_M) / x_M;

    end

        B= x_M;
end
%% Problem 3: False Posistion
function [B, it]= p3_false_pos(x_l, x_u, C, err_accept)
    error = 100;
    x_M = 0;
    it = 0;

    %
    x = [x_u x_l];
    f = [fxn(x_l) -fxn(x_u)]';

    function y = fxn(b)
        it = it + 1;
        y = exp(b) * log(b) - C;
    end

     % yes i did decide to do it this way and name the function
     % meth
     function METH()
       % anyone else see that this is defined as definition for
       % cosine. its an inner producted nomalized with a length -
       % i think
       x_M = (x*f) / (f(1) + f(2));
    end

    % Meth is so helpful here
    METH();

    while ( error > err_accept)

        x_old = x_M;



        if (fxn(x_M) * fxn(x(2)) < 0)
            x(1) = x_M;
            f(2) = - fxn(x_M);

        else
            x(2) = x_M;
            f(1) = fxn(x_M);
```

```
                end
                % the normal usage of meth does usually lead to an error
                % but this definiton of meth helps create a different kind
                % of error, surprisingly one that is useful
                METH();
                error = abs(x_old - x_M) / x_M;

            end

                B= x_M;
        end



    end
end
```