Fixed-point method for Nonlinear equation and Non-linear System

Gautam Bansal (2020MCB1235)

Index

Problem Statement

We are given an equation and if we wish to find the root(s) of it, how would we do it?

Here comes root finding methods generally classified into direct and iterative ones. In this presentation we have discussed fixed point method which is an iterative method to find roots.

In iterative methods we use the data/output from one iteration in the next one and we continue this iterative process till we get our output within the accepted/desired tolerance.

Why use fixed point method?

Even though many other iterative root finding methods like Newton Raphson are available but these methods pose problems like some function not having a derivative. However, in fixed point method there is no such limitation.

Also, it is significantly easier to analyse the equation by fixed point method compared to other methods.

There are many beautiful functions like ‘x - cos(x)’ which can be found converging using fixed point method.

History

There have been many fixed point iterative methods and so it is very tough to say when or who invented this method.

One of the earliest known uses of fixed point method was ‘Picard’s iteration method’ for proving existence of solution of ODEs.

Algorithm

Overview

- Given a function g(x) we have to break it an iterative function
- Let the iterative function be $x_{n+1} = f(x_n)$
- Now we iterate till we get $x_{n+1} = x_n$

Example (single equation)

- Let $f(x) = 2x^3 - 2x - 5 = 0$
- This gives, $x = g(x) = \sqrt[3]{(2x + 5/2)}$
- Take $x_0 = 1.5$
- $x_1 = g(x_0) = 1.5874$
- $x_2 = g(x_1) = 1.5989$
- ..
- $x_6 = g(x_5) = 1.6006$

which is our approximate root

Algorithm (Non-Linear)

As given the basic idea of fixed point method for linear equation in previous slide, we will now extend that idea to the system of nonlinear equations. We can rearrange the given n equations of n variables such that

$$x_i = f_i(x_1, x_2, ..., x_{\square}) \text{ ; here } i \leqslant n$$

Now, just like we did in the linear equation we will iterate here as well. If X is the column vector containing x_i, X' be the vector containing x_{i+1} and F be the column vector of f_i, then

$$X' = F(X)$$

Algorithm (Non-Linear)

We will iterate till we get the relative error in the required/given tolerance limit which is given by

$$\text{norm}(X' - X)/\text{norm}(X') \leqslant E$$

where E is the tolerance.

However, given an equation g(x) it might be possible to break it into two or more equations of the form, $x = f(x)$. In such case how will we choose which one to take? I have discussed in later slides.

MATLAB Code

```
% Gautam Bansal - 2020MCB1235

%clearing memory and command line
clc;
clearvars;
clear all;

n=input("Number of equations = "); %number of equations/variables
tol=input("Input tolerence: "); % tolerance
x=input("Enter initial approximation as row matrix = "); % initial approximation, avoid 'zeros'
                                                         % as inital approximation to avoid
                                                         % division by zero at some point
itrMax=input("Enter maximum number of iterations = "); %maximum number of iterations to prevent looping

A=sym("x",[1,n]); %create symbol x1,x2,.....xn as row matrix


%equation store
E=cell(n,1); % stores functions only
F=cell(n,1); % stores functions as well as the variables

% define variables
disp("Variables are : ");
S="x1";
for i=2:n
    v=num2str(i); %converts number to charactor array
    y=strcat("x",v); %Horizontally adding 2 strings
    S=strcat(S, ",",y); % For printing [x1,x2,x3,x4,...xn]
end
disp(S);
```

```
% Assign values to E and F
for i=1:n
    s=input("Enter the equations = ","s" );  %returns the entered text,
                                             % without evaluating the input as an expression
    E{i,1}=str2sym(s); %Evaluate string representing symbolic expression.
    str=sprintf("@(%s)(%s)",S,s); % Write formatted data to string or character vector
    f=str2func(str);
    F{i}=f;
end

count_itr=0; %variable to store number of iterations
error = 1; %initializing variable that will store error

fprintf('\n # of iterations\t');

for i = 1:n
    fprintf('\n x%d\t', i);
end

while(abs(error) > tol && count_itr < itrMax) %main function
    count_itr = count_itr + 1;
    xold = x; % storing the old X, will be used in calculation relative error

    for i = 1:n % substituting X in F to find X'
        x(1,i) = subs(E{i,1}, A, xold); % X' = F(X)
    end

    error = norm(x - xold)/norm(x); %error = || X' - X || / || X' ||

    fprintf('%d\t', count_itr);

    for i = 1:n
```

```
    for i = 1:n
       fprintf('%f\n', x(1,i));
    end
end

if (count_itr >= itrMax) %does not converge in maximum iteration limit
    disp(' ')
    disp(['Failed to converge in ',num2str(itrMax),' iterations!'])
    disp(' ')
    disp('-------------------------------------------')
    disp(' ')
    answer = NaN;
else %converges within the given tolerance
    disp(' ')
    disp(['Solution has been found in ',num2str(count_itr),' iterations.'])
    for i = 1:n
        fprintf('\tx%f',i);
    end

    fprintf('\n');
    for i = 1:n
        fprintf('\t%f',x(i));
        fprintf('\n');
    end

    disp(' ')
    disp('-------------------------------------------')
    disp(' ')
    answer = x;
end
```

MATLAB Code Input

```
Number of equations = 2
Input tolerence: 0.00001
Enter initial approximation as row matrix = [1 1]
Enter maximum number of iterations = 100
Variables are :
x1,x2
Enter the equations = (1 - x2*x2)^0.5
Enter the equations = (9 - x1*x1)^0.5 - 2
```

MATLAB Code Output

# of iterations	x1	x2
1	0.000000	0.828427
2	0.560097	1.000000
3	0.000000	0.947252
4	0.320491	1.000000
5	0.000000	0.982832
6	0.184504	1.000000
7	0.000000	0.994321
8	0.106422	1.000000
9	0.000000	0.998112
10	0.061424	1.000000
11	0.000000	0.999371
12	0.035459	1.000000
13	0.000000	0.999790
14	0.020472	1.000000
15	0.000000	0.999930
16	0.011819	1.000000
17	0.000000	0.999977
18	0.006824	1.000000
19	0.000000	0.999992
20	0.003940	1.000000
21	0.000000	0.999997
22	0.002275	1.000000
23	0.000000	0.999999
24	0.001313	1.000000
25	0.000000	1.000000
26	0.000758	1.000000
27	0.000000	1.000000
28	0.000438	1.000000
29	0.000000	1.000000

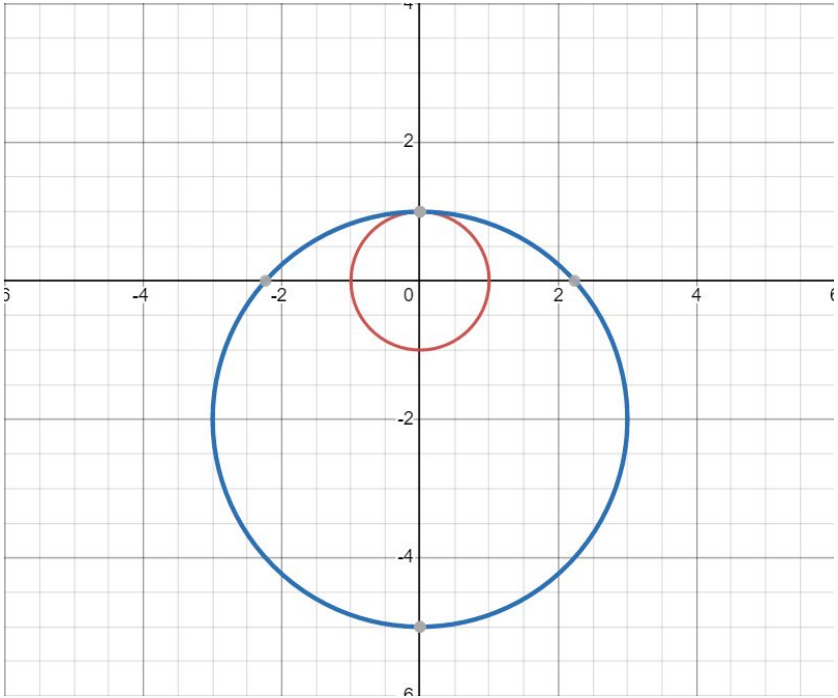
MATLAB Code Output

```
30      0.000253    1.000000
31      0.000000    1.000000
32      0.000146    1.000000
33      0.000000    1.000000
34      0.000084    1.000000
35      0.000000    1.000000
36      0.000049    1.000000
37      0.000000    1.000000
38      0.000028    1.000000
39      0.000000    1.000000
40      0.000016    1.000000
41      0.000000    1.000000
42      0.000009    1.000000
```

```
Solution has been found in 42 iterations.
    x1          x2
    0.000009    1.000000
-----------------------------------------
```

Verifying Output from MATLAB Code

Figure shows the equations of the previously shown example and from the figure it is very clear that the point (0,1) satisfies both the equations and hence is the root of our system.



Convergence Criteria

In the fixed point method for single equation for the output to converge we must select $x = f(x)$ such that

$$|f'(x)| \leq 1 \text{ (f' being derivative of f)}$$

Once we have found all such f we can continue with our algorithm as explained before till we get within the tolerance limit.

However, for non-linear system there is no general convergence criterion and depends on the function F.

Order of Error

Generally, in iterative we have a well defined order of error (depending on method). However, in fixed point method the order of error depends on the function matrix F (it's components indirectly).

Computational Costs

Time Complexity:

Like the order of error, time complexity of fixed point method is also not defined in general but depends on the matrix of functions F.

Space Complexity:

Space complexity of the code i have written is O(n).

Advantages

- Easy to implement
- Code is easy to implement/debug or edit for any future addition
- Low memory and time cost per iteration
- Output/Data is kept in same form as initial guess input which makes any further code implementation really easy
- Not many restrictions like some function not having a derivative

Disadvantages

- Error of order cannot be pre-determined
- Convergence criteria is not general but depends on the functions everytime
- Large number of iterations
- Functions need to be fed an input in the form of $x_i = f_i(x)$, and if we were to take input as $g_i(x)$ and then implement code in form of $x_i = f_i(x)$ then it may require use of derivatives which will cause the time complexity to be non-linear

Comparison with Regula Falsi

Fixed Point Method	Regula Falsi
• In fixed point given a function g(x) we break it in function given by $x_i = f_i(x_1, x_2, ..., x_{\square})$; here $i \leqslant n$ • Now the iterative function will be $X' = F'(X)$ • Do the iteration till we get the answer in the given tolerance	• In regular falsi method following is the iterative function X = (b1x2 - b2x1)/(b1 - b2) • In regula falsi we have to check for f(x).f(x1 or x2) < 0 • Division by (b1 - b2) may result in division by zero causing errors • Like fixed point method, looping may occur in regula falsi method as well

Real Life Applications of Fixed Point Method

Fixed-point method has many applications in communication engineering, genetics, testing of algorithms, solution of chemical equations etc.

Fixed-point method for nonlinear system is essential to solve nonlinear function analysis and general topology.

Some specific uses of fixed-point method are:

- Solving ODEs in banach spaces
- Abstract epilliptic problem
- Game Theory

References

- Who discovered the fixed point iteration method? - History of Science and Mathematics Stack Exchange
- Fixed-point iteration - Wikipedia
- Nonlinear Systems of Equations: Fixed-Point Iteration Method
- ECE 3040 Lecture 11: Numerical Solution of Nonlinear Equations I

Thank You!