Computational Methods in Physics (PHY 365)

FA23

Dr. Muhammad Kamran

Department of Physics

COMSATS University Islamabad

Lab 4

Bisection method

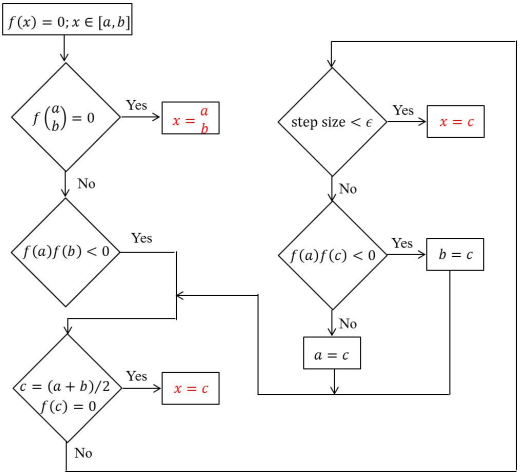

Figure: Flow chart for bisection method

Bisection method

Use the bisection algorithm to approximate the zero of the function $f(x) = 3x^2 - 5$, in the interval $[1, 2]$.

Bisection method

Use the bisection algorithm to approximate the zero of the function $f(x) = 3x^2 - 5$, in the interval $[1, 2]$.

- Function
  $f = 3 * x \;.\hat{}\; 2 - 5;$

Bisection method

Use the bisection algorithm to approximate the zero of the function $f(x) = 3x^2 - 5$, in the interval $[1, 2]$.

- Function

  $f = 3 * x\ \hat{.}\ 2 - 5;$ Error!!!

Bisection method

Use the bisection algorithm to approximate the zero of the function $f(x) = 3x^2 - 5$, in the interval $[1, 2]$.

- Function

  f = 3 * x .^ 2 - 5; Error!!!

  f = @ (x) 3 * x .^ 2 - 5;

Bisection method

Use the bisection algorithm to approximate the zero of the function $f(x) = 3x^2 - 5$, in the interval $[1, 2]$.

- Function

  $f = 3 * x \; .\hat{} \; 2 - 5;$ Error!!!

  $f = @ \; (x) \; 3 * x \; .\hat{} \; 2 - 5;$

  → @ creates a "function handle".

  → A function handle is a data type that stores an association to a function.

  → Function handle can be used to construct anonymous functions or specify call back functions.

Bisection method

- The interval

  a = 1;

  b = 2;

Bisection method

- The interval

  a = 1;

  b = 2;

- Minimum interval length

  min_step = 10 ^ – 4;

Bisection method

- The interval

```
a = 1;
b = 2;
```

- Minimum interval length

```
min_step = 10 ^ - 4;
```

- Checking for opposite signs at the endpoints

```
f_a = f (a);
f_b = f (b);
if f_a * f_b > 0
    error('Function has same sign at both endpoints.')
end
```

Bisection method

- Calling the function

```
[ my_root , iterations ] = bisect_fun (f , a , b , min_step)
```

Bisection method

- Calling the function

```
[ my_root , iterations ] = bisect_fun (f , a , b , min_step)
```

- Displaying the result

```
formatspec = 'The calculated root is %8.6f \n\n';
fprintf(formatspec, my_root)
```

Bisection method

- The function file (save with same name as the function)

```
function [my_root , iterations] = bisect_fun (f , a , b ,
min_step)
f_a = evalin ('base' , 'f_a');
iterations = 0;
step_size = abs(a – b) /2;
x = (a + b) /2;
while step_size > min_step
    f_x = f(x);
    if f_x == 0
        iterations = iterations + 1;
        break
    end
```

Bisection method

```
    if f_a * f_x < 0
        b = x;
    else
        a = x;
    end
    x = (a + b) /2;
    step_size = abs (a - b) /2;
    iterations = iterations + 1;
end
my_root = x;
```

Bisection method

```
% This code uses the bisection method to calculate roots of an equation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear variables

close all

clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function

f = @(x) 3 * x .^ 2 - 5;
```

Bisection method

```
% The interval

a = 1;

b = 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimum interval length

min_step = 10 ^ - 4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Checking for opposite signs at the endpoints

f_a = f(a);

f_b = f(b);

if f_a * f_b > 0

     error('Function has same sign at both endpoints.')

end
```

Bisection method

```
% Calling the function

[my_root,iterations] = bisect_fun(f,a,b,min_step);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Displaying the result

formatspec = 'The calculated root is %8.6f \n\n';

fprintf(formatspec,my_root)
```

Bisection method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is the function file for bisect_code

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [my_root,iterations] = bisect_fun(f,a,b,min_step)

f_a = evalin('base','f_a');

iterations = 0;

step_size = abs(a - b) / 2;

x = (a + b) / 2;
```

Bisection method

```
while step_size > min_step
    f_x = f(x);
    if f_x == 0
        iterations = iterations + 1;
        break
    end
    if f_a * f_x < 0
        b = x;
```

```
    else
        a = x;
    end
    x = (a + b) / 2;
    step_size = abs(a - b) / 2;
    iterations = iterations + 1;
end
my_root = x;
```

Bisection method

- Repeat the exercise for different functions, intervals and minimum step sizes.

Newton-Raphson method

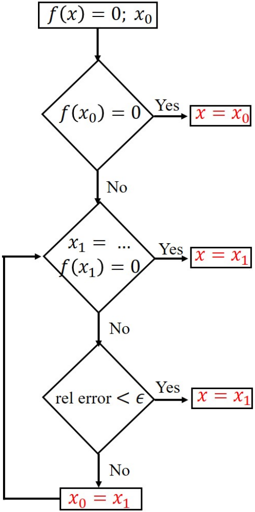

Figure: Flow chart for Newton-Raphson method

Newton-Raphson method

Use the Newton-Raphson method to approximate the zero of the function $f(x) = 3x^2 - 5$.

Newton-Raphson method

Use the Newton-Raphson method to approximate the zero of the function $f(x) = 3x^2 - 5$.

- Function

```
syms x
f = symfun (3 * x^2 - 5, x);
```

Newton-Raphson method

Use the Newton-Raphson method to approximate the zero of the function $f(x) = 3x^2 - 5$.

- Function

```
syms x
f = symfun (3 * x^2 - 5, x);
```

- Derivative of the function

```
f_der = diff(f, x);
```

Newton-Raphson method

Use the Newton-Raphson method to approximate the zero of the function $f(x) = 3x^2 - 5$.

- Function

```
syms x
f = symfun (3 * x^2 - 5, x);
```

- Derivative of the function

```
f_der = diff(f, x);
```

- The initial guess

```
x_0 = 1;
```

Newton-Raphson method

- Minimum error

```
min_err = 10^-4;
```

Newton-Raphson method

- Minimum error

  ```
  min_err = 10^-4;
  ```

- Calling the function

  ```
  [x_root , iterations] = new_raph_fun (f , f_der , x_0 ,
  min_err);
  ```

Newton-Raphson method

- Minimum error

  min_err = 10^−4;

- Calling the function

  [x_root , iterations] = new_raph_fun (f , f_der , x_0 , min_err);

- Displaying the result

  fprintf ('\n')

  disp (['Itertions = ' , num2str(iterations)])

  fprintf ('\n Root = %3.7f \n', x_root)

Newton-Raphson method

- The function file

```
function [x_root , iterations] = new_raph_fun (f , f_der ,
x_0 , min_err)

x_old = x_0;

rel_err = 1;

iterations = 0;

while rel_err > min_err

    f_x_old = double (subs (f , x_old));

    f_der_x_old = double (subs (f_der , x_old));

    x_new = x_old - f_x_old / f_der_x_old;
```

Newton-Raphson method

```
    rel_err = abs (x_new - x_old);
    x_old = x_new;
    iterations = iterations + 1;
end
x_root = x_new;
```

Newton-Raphson method

```
% The function

syms x

f = symfun(3 * x ^ 2 - 5, x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Derivative of the function

f_der = diff(f,x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The initial guess

x_0 =  1.5;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimum error

min_err = 10 ^ -4;
```

Newton-Raphson method

```
% Calling the function

[x_root,iterations] = new_raph_fun(f,f_der,x_0,min_err);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Displaying the result

fprintf('\n')

disp(['Itertions = ', num2str(iterations)])

fprintf('\n Root = %3.6f \n',x_root)
```

Newton-Raphson method

```
function [x_root,iterations] = new_raph_fun(f,f_der,x_0,min_err)

x_old = x_0;

rel_err = 1;

iterations = 0;

while rel_err > min_err

    f_x_old = double(subs(f,x_old));

    f_der_x_old = double(subs(f_der,x_old));
```

Newton-Raphson method

```
        x_new = x_old - f_x_old / f_der_x_old;

        rel_err = abs(x_new - x_old);

        x_old = x_new;

        iterations = iterations + 1;

end

x_root = x_new;
```

MATLAB fzero function

- fzero calculates root of nonlinear function.
- fzero cannot find a root of a function such as x^2.
- x = fzero (fun , x0) tries to find a point x where fun (x) = 0.
- The solution is where fun (x) changes sign.

MATLAB fzero function

- fzero calculates root of nonlinear function.
- fzero cannot find a root of a function such as x^2.
- x = fzero (fun , x0) tries to find a point x where fun (x) = 0.
- The solution is where fun (x) changes sign.
- Root from one point

```
fun = @ (x)   exp ( − 3 * x) − 5 * x .ˆ 3 + 20; % function
x0 = 3; % initial point
x = fzero (fun ,x0);
```

MATLAB fzero function

- Root within an interval

```
fun = @ (x)  x .^ 5 - 3 * x .^ 2 + 1;

x_int = [1,2]; % interval

x = fzero (fun, x_int)
```

MATLAB fsolve function

- x = fsolve (fun , x0) starts at x0 and tries to solve the equations fun (x) = 0, an array of zeros.
- x = fsolve (fun , x0 , options) solves the equations with the optimization options specified in options.
- [x , fval] = fsolve (__), for any syntax, returns the value of the objective function fun at the solution x.

MATLAB fsolve function

- x = fsolve (fun , x0) starts at x0 and tries to solve the equations fun (x) = 0, an array of zeros.

- x = fsolve (fun , x0 , options) solves the equations with the optimization options specified in options.

- [x , fval] = fsolve (__), for any syntax, returns the value of the objective function fun at the solution x.

- Example

  fun = @ (x) exp (– 3 * x) – 5 * x .^ 3 + 20; % function

  x0 = 3; % initial point

  [x , fval] = fsolve (fun , x0);

References

- https://www.codesansar.com/numerical-methods/bisection-method-using-matlab-output.html
- https://www.youtube.com/watch?v=fCKUOWiM-6s
- https://www.matrixlab-examples.com/bisection-method.html
- https://en.wikipedia.org/wiki/Bisection_method
- https://en.wikipedia.org/wiki/Newton%27s_method
- https://web.mit.edu/10.001/Web/Course_Notes/NLAE/node6.html
- https://://personal.math.ubc.ca/~anstee/math104/newtonmethod.pdf
- https://www.mathworks.com/help/matlab/ref/fzero.html
- https://www.mathworks.com/help/optim/ug/fsolve.html
- https://www.mathworks.com/help/matlab/ref/roots.html