Numerical Project

Made By:
Aya Ashraf Saber (02)
Rita Samir (26)
Merit Vector(68)
Marina Zakaria (48)
Shereen Elkordi (33)

Content

Part 1:

Flowchart:

Bisection:

False Position:

Fixed Point Iteration:

Newton Raphson:

Secant :

Birge Vieta:

General Algorithm:

General Algorithm Description:

depends on 2 major root finding algorithms :

bisection method and Newton Raphson algorithm merged to get all roots of a function in a specified interval .
the main idea of the algorithm is to partition our problem into small sub problems to be solved one by one .
we scan the whole interval specified moving by a delta distance from the beginning to the end ; delta is a prespecified value that has to be very small
if the current sub interval has a root (single root or odd multiples of a root) the bisection method will detect the root and get its value, but the bisection will not determine how any multiples is this root so we use another method to determine that which is differentiation we differentiate the function and substitute with the value of the root till we get a value other than zero , that would tell us how many multiples of the root at a certain point.

one other problem left :

how to overcome the pitfall of bisection at double roots or roots with even multiples that bisection can't detect ,We simply use another method which is newton Raphson only when the value of the function at the beginning and end of each subinterval is close that usually means that there is a double root near this interval that we check multiplicity of the root found by differentiation.

Pitfalls:

- can take very long time if the interval is too long
- newton Raphson can't always detect the double roots or roots with even multiplicity

Functions And Data Structures:

Arrays : To hold lower bounds, upper bounds and their corresponding function values, the approximate root and its corresponding function value, Iteration Number and the error at each iteration.

feval : to evaluate the function’s value for the corresponding input.

Diff: to get the derivative of the function

Subs: returns a copy of x, replacing all occurrences of the default variable in x with new, and then evaluates function.
Eval: evaluates the MATLAB® code represented by expression. If you use eval within an

anonymous function, nested function, or function that contains a nested function, the evaluated expression cannot create a variable.

Full analysis and runs of equations:

Example 1:

x^3 - 0.165*x^2 + 3.993*10^-4 = 0

X lower = 0;

X upper = 0.11;

Solution Using Bisection:

Xr = (xl+xu)/2

If f(xl)*f(xr) < 0 then xu = xr

If f(xl)*f(xr) > 0 then xl = xr

If f(xl)*f(xr) = 0 then xr is the root; Terminate.

Iteration	xl	xu	Xr	error%	f(xr)
1	0.00000	0.11 0.11	0.055	--------	6.655×10^−5
2	0.055	0.11	0.0825	33.33	−1.622×10^−4
3	0.055	0.0825	0.06875	20.00	−5.563×10^−5
4	0.055	0.06875	0.06188	11.11	4.484×10^−6
5	0.06188	0.06875	0.06531	5.263	−2.593×10^−5
6	0.06188	0.06531	0.06359	2.702	−1.0804×10^−5
7	0.06188	0.06359	0.06273	1.370	−3.176×10^−6
8	0.06188	0.06273	0.0623	0.6897	6.497×10^−7
9	0.0623	0.06273	0.06252	0.3436	−1.265×10^−6
10	0.0623	0.06252	0.06241	0.1721	−3.0768×10^−7

Example 2:

f (x) = e^-x - x = 0.

Xi = 0.

Using Newton Raphson:

$X_{i+1} = X_i - (f(X_i) / f'(X_i))$

i	**xi**	**εt (%)**
0	0	100
1	0.500000000	11.8
2	0.566311003	0.147
3	0.567143165	0.0000220
4	0.567143290	< 10-8

<u>Example 3:</u>

$Y + x^2 - 0.5 - x = 0.$

$x^2 - 5xy - y = 0.$

Initial guess : $x = 1, y = 0.$

$$F = \begin{matrix} Y + x^2 - 0.5 - x \\ x^2 - 5xy - y \end{matrix} \qquad F' = \begin{matrix} 2x - 1 & 1 \\ 2x - 5y & -5x - 1 \end{matrix} \qquad X0 = \begin{matrix} 1 \\ 0 \end{matrix}$$

$$F = \begin{matrix} -0.5 \\ 1 \end{matrix} \qquad F' = \begin{matrix} 1 & 1 \\ 2 & -6 \end{matrix} \qquad X1 = \begin{matrix} 1.25 \\ 0.25 \end{matrix}$$

Iteration 2:

$$F = \begin{matrix} 0.0625 \\ -0.25 \end{matrix} \qquad F' = \begin{matrix} 1.5 & 1 \\ 1.25 & -7.25 \end{matrix} \qquad X2 = \begin{matrix} 1.2332 \\ 0.2126 \end{matrix}$$

Problematic functions:

Bisection:

If a function f(x) is such that it just touches the x-axis it will be unable to find the lower and upper guesses like x^2.

Function changes sign but root does not exist like $f(x) = 1/x$

False Position:

It converges slowly for some functions.

Suggestion: detect whether or not one of the bounds is stuck, in this case we can use bisection instead.

Fixed Point Iteration:

g(x) can converge or diverge therefore |g'(x)| must be less than 1.

Newton Raphson:

An inflection point (f"(x)=0) at the vicinity of a root causes divergence.

A local maximum or minimum causes oscillations.

It may jump from one location close to one root to a location that is several roots away.

A zero slope causes division by zero.

Sample Runs:

-Newton-Raphson

Lecture example:

Division by zero:

Function with no root:

Slow mode

Finish Slow Mode :

Read From File :

-Birge Vieta

-

Handling errors in Birge Vieta

- General Algorithm

- Secant Method

Bisection

- Read from file:

- Enter equation:

False-position:

Fixed-Point:

Part 2:

Flowchart:

Newton Interpolation:

Lagrange Interpolation:

Functions And Data Structures:

Array to store divided differences in Newton interpolation.

Syms : To create symbolic functions.

Simplify : To simplify the function obtained from Newton/Lagrange interpolation.

Feval : To get the function’s corresponding value for each query point.

Sortrows: to sort input points ascendingly according to the x coordinate; Needed to check whether or not query points lies in the input interval; Produces an error if they don't.

Any: to check for repeated x points, hence producing an error.

Full analysis and runs of equations:

Newton Interpolation:

Linear Interpolation:

Order = 1;

x = 2, y(x) = 7.2;

x= 4.25, y(x) = 7.1;

Query point = 4;

Sol:

B0 = y(x0) = 7.2;

B1 = (y(x1) - y(x2)) / (x1 - x0)

= (7.1 - 7.2) /(4.25 - 2)

= -0.044444

y(x) = b0 + b1 * (x - x0)

= 7.2 + 0.044444(x - 2.00)

At x = 4

y(4.00) = 7.2 - 0.044444*(4.00 - 2.00)

= 7.1111

Quadratic Interpolation:

Order = 2;

x = 2, y(x) = 7.2;

x= 4.25, y(x) = 7.1;

x = 5.25, y(x) = 6;

Query point = 4;

Sol:

B0 and b1 are the same as above;
B2 = (((y(x2) - y(x1))/ (x2 - x1)) - ((y(x1) - y(x0)) / (x1 - x0))) / (x2 - x0)
= - 0.32479

y(x) = b0 + b1 * (x - x0) + b2 *(x - x0)* (x - x1)
y(4.00) = 7.2 - 0.044444*(4.00 - 2.00) - 0.32479* (4.00 - 2.00)*(4.00 - 4.25)
= 7.2735

Lagrange Interpolation:

Linear Interpolation:

Order = 1;
x = 15, y(x) = 362.78;
x= 20, y(x) = 517.35;
Query point = 16;

y(x) = ((x - x1)/(x0 - x1))*y(x0) + ((x - x0)/(x1 - x0))*y(x1)
y(16) = 393.7

Quadratic Interpolation:

Order = 2;
x0 = 15, y(x) = 362.78;
x1= 20, y(x) = 517.35;
x2 = 10, y(x) = 227.04
Query point = 16;

y(x) = ((x - x1)/(x0 - x1))*((x - x2)/(x0 - x2))*y(x0) + ((x - x0)/(x1 - x0))*(x - x2)/(x1 - x2))
* y(x1) + ((x - x0)/(x2 - x0))*((x - x1)/(x2 - x1))*y(x2)
y(16) = 392.19

Problematic functions and suggestions:

There are cases where polynomials can lead to erroneous results because of round off error and overshoot.

Suggestions:

Using Spline Interpolation:

It is to apply lower-order polynomials to subsets of data points. Spline is a "sufficiently smooth" piecewise polynomial interpolation . The "sufficiently smooth" part comes from mandating how many orders of derivatives must be equal between successive splines, so that it is less defective than polynomial interpolation where the derivative at almost each interior point is different

Sample runs:

Lecture example (1): order of 5

Handling if the user enter two Xs with the same value

Handing if the query is out of the interval :

Sheet example :

Read From File