Chapter 8: Nonlinear Equations

Nonlinear Equations

• Definition

A value for parameter \( x \) that satisfies the equation

\[
f(x) = 0
\]

is called a root or a ("zero") of \( f(x) \).

• Exact Solutions

For some functions, we can calculate roots exactly;

e.g.,
- Polynomials up to degree 4
- Simple transcendental functions, such as
  \[ \sin x = 0 \]
  which has an infinite number of roots:
  \[ x = k\pi \quad (k = 0, \pm 1, \pm 2, \ldots) \]

• Numerical Methods

Used to estimate roots for nonlinear functions \( f(x) \).
- Bisection Method
- False Position Method
- Newton’s Method
- Secant Method

All are iterative techniques applied to continuous functions.

We will consider only methods for finding real roots, but methods discussed can be generalized for complex roots.
**Bisection Method**

A binary search procedure applied to an $x$ interval known to contain root of $f(x)$.

Example: Polynomial $f(x) = x^5 + x + 1$

- has exactly five roots, at least one real root.

**Step 1:** Determine an $x$ interval containing a real root.

We can do this by simple analysis of $f(x)$:

- compute (or estimate) $f(x)$ for convenient values of $x$, such as 0, ± 1.

- estimate position of local extrema with first derivative of $f(x)$.

- make rough sketch of $f(x)$.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>-1</td>
<td>-1.0</td>
</tr>
</tbody>
</table>

Have root in this interval.

**Step 2:** Bisect interval repeatedly until root is determined to desired accuracy.
If \( f(x_{\text{low}}) \) and \( f(x_{\text{mid}}) \) have opposite signs (\( f(x_{\text{low}}) \cdot f(x_{\text{mid}}) < 0 \)), root is in left half of interval.

If \( f(x_{\text{low}}) \) and \( f(x_{\text{mid}}) \) have same signs (\( f(x_{\text{low}}) \cdot f(x_{\text{mid}}) > 0 \)), root is in right half of interval.

Continue subdividing until interval width has been reduced to a size \( \leq \varepsilon \)

where

\[
\varepsilon = \text{selected } x \text{ tolerance.}
\]
Using a $y$ tolerance can result in poor estimate of root.

**Examples:**

**Oscillating function**

**Asymptotically converging function**
Pseudocode Algorithm: Bisection Method

Input xLower, xUpper, xTol
yLower = f(xLower) (* invokes fcn definition *)
xMid = (xLower + xUpper)/2.0
yMid = f(xMid)

iters = 0 (* count number of iterations *)

While ( (xUpper - xLower)/2.0 < xTol )
    iters = iters + 1
    if( yLower * yMid > 0.0) Then xLower = xMid
        Else xUpper = xMid
    Endofif
    xMid = (xLower + xUpper)/2.0
    yMid = f(xMid)
Endofwhile

Return xMid, yMid, iters (* xMid = approx to root *)

(Do not need to recalculate yLower in loop, since it can never change sign.)

For a given $x$ tolerance (epsilon), we can calculate the number of iterations directly. The number of divisions of the original interval is the smallest value of $n$ that satisfies:

$$\frac{x_{upper} - x_{lower}}{2^n} < \varepsilon \quad \text{or} \quad 2^n > \frac{x_{upper} - x_{lower}}{\varepsilon}$$

Thus $n > \log_2 \left( \frac{x_{upper} - x_{lower}}{\varepsilon} \right)$

In our previous example $x_{lower} = -1$, $x_{upper} = 0$

Choosing $\varepsilon = 10^{-4}$, we have $n > \log_2 10^{-4} = 13.29$

$\therefore n = 14$
**False-Position Method** (Regula Falsi)

Improvement on bisection search by interpolating next $x$ position, instead of having the interval.

![Graph of False-Position Method](image)

By similar triangles: \[ \frac{y_{up} - 0}{x_{up} - x} = \frac{0 - y_{low}}{x - x_{low}} \]

Then use this calculation in place of $x_{mid}$ in the Bisection Algorithm.

- False Position Method usually converges more rapidly than Bisection approach.
- Can improve false position method by adjusting interpolation line:

Each interpolation line (after first) is now drawn from $(x, f(x))$ to either $(x_{low}, y_{low}/2)$ or to $(x_{up}, y_{up}/2)$, depending on region containing root.

For continuous functions (single-valued), both Bisection and False Position are guaranteed to converge to root.

(Because interval is assumed to contain root.)
**Newton’s Method** (Newton-Raphson)

Attempts to locate root by repeatedly approximating \( f(x) \) with a linear function at each step:

\[
\text{Start with initial “guess”, } x_0, \text{ then calculate next approximation to root, } x_1.
\]

Slope of curve at \( x_0 \) is \( \frac{df}{dx} = \frac{f(x_0)}{f'(x_0)} \)

We repeat process to get next approximation, etc.

Thus, rapidly converges to root, with convergence accelerating as we approach \( f(x) = 0 \).
Newton-Raphson Algorithm

1. Start with an initial guess $x_0$ and an $x$-tolerance $\varepsilon$.
2. Calculate

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}, \quad k = 1, 2, 3, \ldots$$

until

$$\left| \frac{f(x_{k-1})}{f'(x_{k-1})} \right| < \varepsilon$$

Very fast root-finding method. Useful when $f'(x)$ is not too difficult to evaluate.

But Newton’s Method does not always converge; e.g., when $f'(x) \approx 0$ for some $x$.

Example: Newton-Raphson Solution of

$$f(x) = x^2 - 2, \quad x_0 = 1 \text{ (initial guess)}$$

derivative: $f'(x) = 2x$

Steps

(1) $f(x_0) = -1, \quad f'(x_0) = 2$
$$x_1 = 1 - (-1)/2 = 1.5000000$$

(2) $f(x_1) = 0.25, \quad f'(x_1) = 3.00$
$$x_2 = 1.5 - 0.25/3.00 = 1.4166667$$

(3) $f(x_2) \approx 0.0069444, \quad f'(x_2) = 2.8333333$
$$x_3 \approx 1.4166667 - 0.0069444/2.8333333 \approx 1.4142157$$

Continue until $|x_k - x_{k-1}| < \varepsilon$. 
**Pseudocode Algorithm - Newton’s Method**

Input $x_0$, $x_{Tol}$  

$\text{iters} = 1$  

$dx = -f(0)/f\text{Deriv}(x_0)$  

(* fcn s $f$ and $f\text{Deriv}$ *)  

$\text{root} = x_0 + dx$  

While (Abs($dx$) $\geq x_{Tol}$)  

$dx = -f(\text{root})/f\text{Deriv}(\text{root})$  

$\text{root} = \text{root} + dx$  

$\text{iters} = \text{iters} + 1$  

Endofwhile  

Return $\text{root}$, $\text{iters}$

**Newton’s Method**

- Converges much faster than Bisection Method.

- But convergence not guaranteed. Will diverge in some cases. For example:

To prevent such “run aways”, we can include the test: $\text{iters} > \text{maxiters}$ in above algorithm to halt the loop.

- For multiple roots (real and complex), boundaries between convergence regions are fractals.
Secant Method

Variation of Newton’s Method:

- Approximates derivative
- Requires two starting \( x \) values

Let \( x_k \) denote approximation to root of \( f(x) \) at step \( k \).

Using Newton’s Method, next approx. is

\[
x_{k+1} = x_k - f(x_k) / f'(x_k)
\]

Since the derivative

\[
f'(x) = \lim_{Ax \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x}
\]

can be approximated as

\[
f'(x) \approx \frac{f(x) - f(x_{k-1})}{x_k - x_{k-1}} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}
\]

the expression for \( x_{k+1} \) can be approximated as

\[
x_{k+1} = \frac{x_k - y_k / y_k - y_{k-1}}{y_k - y_{k-1}}
\]

Thus, Secant Method based on using a “finite-difference” approximation for derivative.
Example: Secant Method Solution of

\[ f(x) = x^2 - 2, \quad x_0 = 1, x_1 = 2 \] (starting values)

Steps

\[ y_0 = f(x_0) = -1, \quad y_1 = f(x_1) = 2 \]

(1) \[ x_2 = \frac{x_0y_1 - x_1y_0}{y_1 - y_0} \]

\[ = \frac{(1)(2) - (2)(-1)}{3} = \frac{4}{3} \approx 1.3333 \]

\[ y_2 = f(x_2) \approx -0.2222 \]

(2) \[ x_3 = \frac{x_1y_2 - x_2y_1}{y_2 - y_1} \]

\[ \approx \frac{(2) - (0.2222) - (1.3333)(2)}{(-0.2222) - (2)} \]

etc.
### Pseudocode Algorithm - Secant Method

Input $x_k$, $x_{k-1}$, $x_{tol}$, maxiters

$\text{iters} = 1$

$y_k = (x_k)$  
\[\text{(* invokes function } f \text{ *)}\]

$y_{k-1} = f(x_{k-1})$

$\text{root} = (x_{k-1}y_k - x_ky_{k-1})/(y_k - y_{k-1})$

$y_{k+1} = f(\text{root})$

\[\text{While}( \text{Abs}(\text{root} - x_k) \geq x_{tol}) \text{ and } (\text{iters } \leq \text{maxiters}) \text{ )}\]

\begin{align*}
  x_{k-1} & = x_k \\
  y_{k-1} & = y_k \\
  x_k & = \text{root} \\
  y_k & = y_{k+1} \\
  \text{root} & = (x_{k-1}y_k - x_ky_{k-1})/(y_k - y_{k-1}) \\
  y_{k+1} & = f(\text{root}) \\
  \text{iters} & = \text{iters} + 1
\end{align*}

Endofwhile

Return $\text{root}$, $y_{k+1}$, iters

Secant Method can be best choice if computation of $f'(x)$ is expensive

### Summary of Root-Finding Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Converge?</th>
<th>Converge Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bisection</td>
<td>$x_{low}$, $x_{up}$</td>
<td>yes</td>
<td>linear</td>
</tr>
<tr>
<td>False Pos.</td>
<td>$x_{low}$, $x_{up}$</td>
<td>yes</td>
<td>better</td>
</tr>
<tr>
<td>Secant</td>
<td>“any” 2 vals</td>
<td>maybe</td>
<td>better yet</td>
</tr>
<tr>
<td>Newton’s</td>
<td>“any” 1 val</td>
<td>maybe</td>
<td>usually best</td>
</tr>
</tbody>
</table>
**Root-Finding Methods in Mathematica**

Numerical solution of equations can be accomplished with either `Solve` or `Roots` (for polynomials) in combination with `N` function.

Numerical Root-Finding using Newton or Secant Method:

```mathematica
FindRoot[ f(x)==expr, {x, x0} ] - Newton’s Method using starting value x0.

FindRoot[ f(x)==expr, {x, x0, xmin, xmax}] - use starting value x0; stop if x goes outside range xmin to xmax.

FindRoot[ {eqn1, eqn2, ... }, {x, x0}, {y, y0}, ... ] - find numerical solution for a set of simultaneous equations; using starging values x0, x1, ...

FindRoot[ f(x)==expr, {x, {x0, x1}} ] - Secant Method with starting values x0, x1.
```

**Examples - Newton’s Method**

```mathematica
FindRoot[ x^2-2 == 0, {x, -1} ]
{x -> -1.41421}  {if starting value is neg., a neg. real root is sought. If pos., a pos. root is sought

FindRoot[ x^2+2 == 0, {x, I} ]
{x -> 1.41421 I}  {if starting value is complex, then Mathematica looks for a complex root.

FindRoot[ x^3 == -1, {x, I} ]
{x -> 0.5 + 0.866025 I}

FindRoot[ {x^2-2Sin[y]==0, y Sqrt[x]-1==0}, {x,1}, {x,1} ]  {simultaneous eqns.
{x -> 1.24905, y -> 0.894767}
```
Parameters for \texttt{FindRoot} include \texttt{MaxIterations} and \texttt{AccuracyGoal}.

E.g.,

\begin{verbatim}
FindRoot[x^2-2==0, {x,1}, MaxIterations->3]
\end{verbatim}

\texttt{FindRoot::cvnwt:}

Newton’s method failed to converge to converge
to the prescribed accuracy after 3 iterations.
\{x\to 1.41422\} \quad \text{(Default MaxIterations=15)}

(AccuracyGoal is number of digits required for 0.0 in solution check of \(f(x) = 0.0\);
default = 9(?); default Precision for calculations = 19.)

\textbf{Secant Method Examples:}

\begin{verbatim}
FindRoot[x^2-2 == 0, {x, {1, 2}} ]
\end{verbatim}

\(\{x\to 1.41421\} \quad \text{(starting values = 1,2)}

\begin{verbatim}
FindRoot[x^2-2Sin[y]==0, y Sqrt[x]-1==0),
\quad \{x,{1,2}\}, \{y, {1,2}\}]
\end{verbatim}

\(\{x \to 1.24905, y \to 0.894767\}

\text{(starting values for both x and y are 1,2)}

Note that \texttt{FindRoot} gives solutions as transformation rules (using ->).
Thus the solutions can be substituted into expression with /. operation. We can combine this with an assignment statement to store values in a variable name.

Example - Assign the positive square root of 9 to variable \texttt{xroot}:

\begin{verbatim}
xroot = x /.FindRoot[x^2 - 9 == 0, {x, 2}]
\end{verbatim}

3. \quad \text{(Initial value for Newton’s Method is 2)}
Function \texttt{Solve} also gives solutions to an equation, or set of equations, as transformation rules, which may then be substituted into expressions using the / operator.

\textbf{Examples:}

\texttt{Solve[x^2-9==0, x]}
\begin{verbatim}
{{x -> -3}, {x -> 3}}
\end{verbatim}

\texttt{NSolve[Cos[x]==0, x]}
\texttt{Solve::ifun:}

Warning: Inverse functions are being used by \texttt{Solve}, so some solutions may not be found.

\begin{verbatim}
{{x -> 1.5708}}
\end{verbatim}

Function \texttt{Roots}, on the other hand, gives roots of polynomials as logical expressions (i.e., with ==) that can be manipulated with Boolean operators.

\textbf{Examples:}

\texttt{Roots[x^2-9==0, x]}
\begin{verbatim}
x == 3 | | x == -3
\end{verbatim}

\texttt{NRoots[x^2-9==0, x]}
\begin{verbatim}
x == -3. | | x == 3.
\end{verbatim}
Summary

Mathematica Equation-Solving Functions

- **Roots** -
  
  Applied to polynomials.
  
  Solutions given as logical expressions, involving either variable names or numerical values.
  
  Can be used with `N` function.

- **Solve** -
  
  Applied to single equation or set of simultaneous equations.
  
  Solutions given as transformation rules, involving either variable names or numerical names.
  
  Can be used with `N` function.

- **FindRoot** -
  
  Applied to single equation or set of simultaneous equations.
  
  Uses either Newton’s Method or Secant Method to find numerical solutions.
  
  Solutions given as transformation rules.