Newton’s method and multiple roots.

Applying Newton’s method to \( f(x) = x^2 - 2xe^{-x} + e^{-2x} \) gives the iteration results:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x_i )</th>
<th>( \Delta x_i )</th>
<th>( f(x_i) )</th>
<th>( f'(x_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0000000E-01</td>
<td></td>
<td>1.1E-02</td>
<td>-3.4E-01</td>
</tr>
<tr>
<td>1</td>
<td>5.331555E-01</td>
<td>3.3E-02</td>
<td>2.9E-03</td>
<td>-1.7E-01</td>
</tr>
<tr>
<td>2</td>
<td>5.500438E-01</td>
<td>1.7E-02</td>
<td>7.2E-04</td>
<td>-8.5E-02</td>
</tr>
<tr>
<td>3</td>
<td>5.585670E-01</td>
<td>8.5E-03</td>
<td>1.8E-04</td>
<td>-4.2E-02</td>
</tr>
<tr>
<td>4</td>
<td>5.628485E-01</td>
<td>4.3E-03</td>
<td>4.5E-05</td>
<td>-2.1E-02</td>
</tr>
<tr>
<td>5</td>
<td>5.649942E-01</td>
<td>2.1E-03</td>
<td>1.1E-05</td>
<td>-1.1E-02</td>
</tr>
<tr>
<td>6</td>
<td>5.660683E-01</td>
<td>1.1E-03</td>
<td>2.8E-06</td>
<td>-5.3E-03</td>
</tr>
<tr>
<td>7</td>
<td>5.666057E-01</td>
<td>5.4E-04</td>
<td>7.1E-07</td>
<td>-2.6E-03</td>
</tr>
<tr>
<td>8</td>
<td>5.668745E-01</td>
<td>2.7E-04</td>
<td>1.8E-07</td>
<td>-1.3E-03</td>
</tr>
<tr>
<td>9</td>
<td>5.670089E-01</td>
<td>1.3E-04</td>
<td>4.4E-08</td>
<td>-6.6E-04</td>
</tr>
<tr>
<td>10</td>
<td>5.670761E-01</td>
<td>6.7E-05</td>
<td>1.1E-08</td>
<td>-3.3E-04</td>
</tr>
<tr>
<td>11</td>
<td>5.671097E-01</td>
<td>3.4E-05</td>
<td>2.8E-09</td>
<td>-1.7E-04</td>
</tr>
<tr>
<td>12</td>
<td>5.671265E-01</td>
<td>1.7E-05</td>
<td>6.9E-10</td>
<td>-8.3E-05</td>
</tr>
<tr>
<td>13</td>
<td>5.671349E-01</td>
<td>8.4E-06</td>
<td>1.7E-10</td>
<td>-4.1E-05</td>
</tr>
<tr>
<td>14</td>
<td>5.671391E-01</td>
<td>4.2E-06</td>
<td>4.3E-11</td>
<td>-2.1E-05</td>
</tr>
<tr>
<td>15</td>
<td>5.671412E-01</td>
<td>2.1E-06</td>
<td>1.1E-11</td>
<td>-1.0E-05</td>
</tr>
<tr>
<td>16</td>
<td>5.671422E-01</td>
<td>1.1E-06</td>
<td>2.7E-12</td>
<td>-5.2E-06</td>
</tr>
<tr>
<td>17</td>
<td>5.671430E-01</td>
<td>2.6E-07</td>
<td>1.7E-13</td>
<td>-1.3E-06</td>
</tr>
<tr>
<td>18</td>
<td>5.671430E-01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The slow convergence to the root.

Note: The ratio of \( f(x_i) \) to \( f'(x_i) \).

Applying Newton’s method to \( \mu(x) = f(x)/f'(x) \) gives the iteration results:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x_i )</th>
<th>( \Delta x_i )</th>
<th>( f(x_i) )</th>
<th>( f'(x_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0000000E-01</td>
<td></td>
<td>-1.1E-02</td>
<td>-3.4E-01</td>
</tr>
<tr>
<td>1</td>
<td>5.680137E-01</td>
<td>6.8E-02</td>
<td>1.9E-06</td>
<td>4.3E-03</td>
</tr>
<tr>
<td>2</td>
<td>5.671434E-01</td>
<td>8.7E-04</td>
<td>4.6E-14</td>
<td>6.7E-07</td>
</tr>
<tr>
<td>3</td>
<td>5.671433E-01</td>
<td>1.4E-07</td>
<td>0.0E+00</td>
<td>-3.0E-09</td>
</tr>
</tbody>
</table>

Note: We have recovered the fast convergence to the root.
Defn: A solution \( x^* \) of \( f(x) = 0 \) is called a root of multiplicity \( m \) of \( f \) if we can factor
\[
f(x) = (x - x^*)^m \cdot q(x)
\]
for some \( q(x) \) with \( q(x^*) \neq 0 \).

\( m = 1 \): a simple root.

Theorem: The function \( f \in C^m[a, b] \) has a root of multiplicity \( m \) at \( x^* \) iff
\[
f(x^*) = 0, \quad f'(x^*) = 0, \ldots, f^{(m-1)}(x^*) = 0,
\]
and \( f^{(m)}(x^*) \neq 0 \).

How to modify N. M. in this situation?

Suppose \( x^* \) is a multiplicity \( m \) root of \( f(x) = 0 \)

- can write \( f(x) = (x - x^*)^m \cdot q(x) \) for some unknown \( q(x) \) s.t. \( q(x^*) \neq 0 \).

Define \( M(x) = \frac{f(x)}{f'(x)} \)
\[
\begin{align*}
\frac{(x-x^*)^m}{m(x-x^*)^{m-1}q(x) + (x-x^*)^mq'(x)} &= (x-x^*) \left[ \frac{q(x)}{mq(x) + (x-x^*)q'(x)} \right] \\
\therefore M(x^*) &= 0 \quad \text{since } q(x^*) \neq 0 \\
\frac{q(x^*)}{mq(x^*) + (x-x^*)q'(x^*)} &= \frac{1}{m} \neq 0
\end{align*}
\]

so \( x^* \) is a simple root of \( M(x) \)

\[ M'(x) = \frac{f'(x)f''(x) - f(x)f'''(x)}{(f'(x))^2} \]

So N. m. becomes.

\[
\chi_{i+1} = \chi_i - \frac{M(x_i)}{M'(x_i)}
\]

\[ \vdots \]

\[ = \chi_i - \frac{f(x_i)f'(x_i)}{f''(x_i)} \quad \bigcirc \]
\[ f'(x_i) \cdot [ - f(x_i) f''(x_i) ] \]

- Why not always apply N.m. to \( N(x) \) instead of \( f(x) \)?
  - Need \( f''(x) \)
  - More calculations per iteration
  - So don't use universally.

**A Robust Algorithm**
- Seen many algorithms
- Which to implement

  - Combine a reliable but slow method like bisection with a fast but not always reliable (like N.m. or secant)

  
  
  
  
  How an iteration works.

  - Start with \( x = B, C \) with sign\( f(B) \)
    [1] = sign\( f(C) \)
and label \( B, C \) s.t. \( |f(B)| < |f(c)| \)

(Could use random sampling to find \( B, C \))

- Set \( D = B - \frac{f(B)(B-C)}{f(B)-f(c)} \) (Secant step)

- Set \( M = \frac{(B+C)}{2} \)

\[
\begin{array}{c}
B \quad M \quad D \quad C \\
\end{array}
\]

- If \( D \) is between \( M \) and \( C \):
  - Secant not working as expected
  - (Since \( |f(B)| < |f(c)| \), expect root closer to \( B \))
  - Set \( D = M \) and accept as next iterate

  Else
  - Accept \( D \) as next iterate

- Next step: starts with \( D \) and either \( B \) or \( C \)
  - If \( \text{sign}(f(D)) = \text{sign}(f(B)) \) keep \( C \)
  - Else:
    - Keep \( B \)
      - (to maintain bracket of root)
note: no guarantee curve is fast.

\[ f(x) = x \cdot \tan x \]

slow convergence

\[ B \quad B \quad B \quad B \quad D \quad M \quad C \]

- monitor rate of convergence of \([B, C]\)
- if after a few steps worse than bisection
  apply a few iterations of bisection
  in an attempt to move \(C\)
  then switch back to combined method
- once close to root, expect faster converge

this implemented in matlab fzero

\[
\text{root} = \text{fzero}(\text{f-name, [a,b]})
\]

finds a root in \([a,b]\) provided \(f(a)f(b)<0\)

or

\[
\text{root} = \text{fzero}(\text{f-name, x0})
\]

finds a root near \(x_0\).
higher dimensions, (systems of eq. n

\[ f(x): \mathbb{R}^n \rightarrow \mathbb{R}^n \]

- N.m. becomes \[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

\[ x_{i+1} = x_i - \Delta x_i \]

where \( \Delta x_i \) solves \( J(x_i) \Delta x_i = f(x_i) \)

Jacobian matrix of partial derivatives
- Each iteration requires solving a linear system.

- Also Broyden's method
  - Avoids derivatives.

end of root finding

Numerical Approximation (Ch 7 Heath)
Problem: Given a "complicated" function \( f(t) \) in the finite interval \([a,b]\), find a "simple" function \( p(t) \) s.t. the error \( |f(t) - p(t)| \) is < some \( \varepsilon > 0 \).

- complicated \( \rightarrow \) expensive to evaluate
- simple \( \rightarrow \) easy/fast to evaluate.

One solution: use a Taylor polynomial around \( t = a \)

\[
f(t) = f(a) + f'(a)(t-a) + \ldots + \frac{f^{(n-1)}(a)}{(n-1)!}(t-a)^{n-1}
\]

Choose the first \( n \) terms to approximate.

\[
p(t) = f(a) + f'(a)(t-a) + \ldots + \frac{f^{(n-1)}(a)}{(n-1)!}(t-a)^{n-1}
\]

The error is

\[
E_{n-1}(t) = \frac{f^{(n)}(\xi)}{n!}(t-a)^n
\]

for some \( \xi \) between \( a \) and \( t \).

good: have an approx. AND expr. for error
exact at $t = a$

**bad:** requires many derivatives of a "complicated" fun.

- may not exist.
- how does error behave away from $t = a$?

alg + analysis for this problem

Measuring approximation error.

Given construct $f(t) \Rightarrow p(t)$

$\Xi \Rightarrow |f(t) - p(t)| = ?$

**linear algebra.**

\[
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\]

\[
\|
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\|_1 = \sum_{i=1}^{n} |x_i| \\
\|
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \\
\|
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\|_{\infty} = \max_{1 \leq i \leq n} |x_i|
\]

$l_p$ norm: \[
\|
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}
\]
extend to $C[a,b]$, $l_p$ norms

$$||f||_p = \left( \int_a^b |f(t)|^p \, dt \right)^{1/p}$$

get

$$||f||_1 = \int_a^b |f(t)| \, dt,$$

$$||f||_2 = \sqrt{\int_a^b |f(t)|^2 \, dt},$$

$$||f||_\infty = \max_{a \leq t \leq b} |f(t)|$$

we use the $\infty$-norm in our work.

$p(t) \approx f(t)$

error $e(t) = f(t) - p(t)$

$$||e(t)||_\infty = ||f(t) - p(t)||_\infty$$

$$= \max_{a \leq t \leq b} |f(t) - p(t)|$$

= the largest deviation between $f$ and $p$

if largest deviation $< \varepsilon$ then

$$||e(t)||_\infty < \varepsilon$$
example

The choice of norm does matter.

+ effect only you determine

Consider $f(t) = 0$ on $[0, 1]$

and

$$P_k(t) = \begin{cases} 
  k(k^2t-1), & \frac{1}{k^2} \leq t \leq \frac{2}{k^2} \\
  -k(k^2t-3), & \frac{2}{k^2} \leq t \leq \frac{3}{k^2} \\
  0 & \text{otherwise}
\end{cases}$$

- Consider $e_k(t) = f(t) - P_k(t)$

- Can show:

  $$\| e_k(t) \|_1 = \frac{1}{k}$$
  $$\| e_k(t) \|_2 = \sqrt{\frac{2}{3}}$$
  $$\| e_k(t) \|_\infty = k$$

  as $k \to \infty$
\[ \| e_k(t) \|_1 \to 0 \quad (\text{suggests accurate approx}) \]
\[ \| e_k(t) \|_2 = \sqrt{\frac{1}{k}} \quad (\text{suggests inaccurate interp}) \]
\[ \| e_k(t) \|_\infty \to 0 \quad (\text{suggests accuracy}) \]

In software, don't wait large error at some \( t \) even if accurate for other values of \( t \).

\[ \lim_{t \to \infty} \| f(t) - p(t) \| < \varepsilon \]
\[ a \leq t \leq b \]

(called uniform approximation)

... on to finding approximately functions...

Suppose take \( f(t) \):

- evaluate it at \( m \) points
  \[ t_1 < t_2 < t_3 < \ldots < t_m \]
- gives dataset \( \{ (t_i, f(t_i)) \}_{i=1}^m \)
- construct a function \( g(t) \) that interpolates the dataset

\[ \forall t_i, \quad i = 1 : m \]
\[ g'(t_i) = f(t_i). \]

- Idea: if \( g(t) \) approx. \( f(t) \) exactly at \( t = t_i \) perhaps a good approx at other \( t \) values.

- Interpolant not unique

\[ \text{const} \]

\[ g_{x}(t) \]

\[ g_{y}(t) \]

\[ g_{z}(t) \]

- and not all interpolants are nice e.g. differentiable

Construct interpolants that:
- can be evaluated quickly
- give an accurate approx for \( t \neq t_i \)
- can be easily integrated/differentiated

Why? \( \int_{a}^{b} f(t) \, dt = \int_{a}^{b} g(t) \, dt \) (easy to)
\[ \frac{d}{dt} f(t) = \frac{d}{dt} \phi(t) \]

Start by finding polynomial interpolants which are easy to work with.

**Theorem:** Given distinct nodes \( t_1, t_2, \ldots, t_m \) and values \( f(t_i), \ i = 1:m \), there is a unique polynomial \( p_{m-1}(t) \) of degree at most \( m-1 \) that interpolates \( f(t) \) at \( t_1, t_2, \ldots, t_m \).