Newton's Method

Solving Nonlinear Equations

1 Single equation with single variable 
\[ g(x) = 0 \]

2 System of \( n \) equations in \( n \) variables 
\[ g(x_1, x_2, \ldots, x_n) = 0 \quad \forall i = 1, 2, \ldots, n \]

Let \( x_0 \) be an initial "guess" at the solution of 
\[ g(x) = 0 \]
If \( g(x_0) \neq 0 \), then we wish to find a correction \( \delta \) so that 
\[ g(x_0 + \delta) = 0 \]
Then \( x_1 = x_0 + \delta \) becomes our improved approximation, and we repeat the procedure until \( g(x_n) \) is "sufficiently close" to zero.

Suppose that \( x_k \) is a "guess" or approximation to the solution of 
\[ g(x) = 0 \]
at iteration \( k \).
\[ g(x_k + \delta) = g(x_k) + g'(x_k) \delta + \frac{g''(x_k + \alpha \delta)}{2} \delta^2 = 0 \]
Assumed negligible.

Solve the equation 
\[ g(x_k + \delta) = g(x_k) + g'(x_k) \delta = 0 \]
for the \( \delta \) which will give (hopefully) an improved approximate solution.
We are seeking the intersection of the graph of \( g(x) \) and the \( x \)-axis.

\[ x_1 = x_0 + \delta \]

is the intersection with the \( x \)-axis of the tangent line to the graph at \((x_0, g(x_0))\)

### Rate of Convergence

Define the error at iteration \( k \) by

\[ e_k = x_k - x^* \]

Taylor's formula implies that, for some \( z \in [x_k, x^*] \),

\[
0 = g(x_k) = g(x^*) + e_k g'(x^*) + \frac{1}{2} e_k^2 g''(z)
\]

\[
- e_k = \frac{g(x_k)}{g'(x^*)} = \frac{1}{2} \frac{g''(z)}{g'(x^*)} e_k^2
\]

\[
- x_k + x^* = \frac{g(x_k)}{g'(x^*)} = \frac{1}{2} \frac{g''(z)}{g'(x^*)} (x_k - x^*)^2
\]

\[
\lim_{k \to \infty} e_k = 0 \implies x_{k+1} = \frac{1}{2} \frac{g''(z)}{g'(x^*)} (x_k - x^*)^2
\]

That is, the error is approximately squared at each iteration.

\[ \implies \text{The convergence is quadratic with} \quad C = \left| \frac{1}{2} \frac{g''(z)}{g'(x^*)} \right| \]

\[ 3x^3 + 2x^2 + x + 1 = 0 \]

\[ G(X) = 3X^3 + 2X^2 + X + 1 = 0 \]

<table>
<thead>
<tr>
<th>( t )</th>
<th>( X_t )</th>
<th>( G(X_t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000000000000</td>
<td>7.000000000000000</td>
</tr>
<tr>
<td>0.1</td>
<td>7.936495002665492</td>
<td>-0.000000000000000</td>
</tr>
<tr>
<td>0.2</td>
<td>7.936495002665492</td>
<td>1.000000000000000</td>
</tr>
<tr>
<td>0.3</td>
<td>7.936495002665492</td>
<td>-1.000000000000000</td>
</tr>
<tr>
<td>0.4</td>
<td>7.936495002665492</td>
<td>2.000000000000000</td>
</tr>
</tbody>
</table>

Log (base 10) of Error vs Iteration.
Newton 1/12/99  page 3

\[ |e_k|, |e_{k+1}|, |e_k|^2 \]

\begin{tabular}{c|c|c|c}
\hline
\( k \) & \( |e_{k-1}| \) & \( |e_{k-1}| \) & \( |e_k|^2 \) \\
\hline
0 & 7.257143340705E+0 & 4.034512227813668E+1 & 1.393351279390326E+1 \\
1 & 4.474978352853E+2 & 1.441206267516463E+3 & 1.441206267516463E+3 \\
2 & 2.934710892935E+3 & 1.441206267516463E+3 & 1.441206267516463E+3 \\
3 & 6.41457543333E+4 & 1.441206267516463E+3 & 1.441206267516463E+3 \\
4 & 1.934710892935E+5 & 1.441206267516463E+3 & 1.441206267516463E+3 \\
5 & 2.11231809995E+6 & 1.441206267516463E+3 & 1.441206267516463E+3 \\
\hline
\end{tabular}

\[ \frac{|e_{k+1}|}{|e_k|} \]

\begin{tabular}{c|c|c|c}
\hline
\( k \) & 0 & 1 & 2 \\
\hline
\hline
\end{tabular}

\[ G(X) = 3X^2 + 2X^2 + X + 1 \]

\[ G(X) = X^3 - 2X^2 - X + 2 \]

Roots are -1, 1, & 2

\begin{tabular}{c|c|c|c|c}
\hline
\( t \) & \( X_t \) & \( G(X_t) \) & \( r_t \) \\
\hline
0 & 0.560000000000000 & 1.875000000000000 & -0.500000000000000 \\
1 & -1.57342871436571 & 2.192314110737571 & 0.651085714285714 \\
2 & -1.650385176176176 & 1.076608680311804 & 0.192916666666667 \\
3 & -1.07351953465219 & -0.105490034807000 & 0.017395898459219 \\
4 & -1.96025556787395 & -0.0047110372309 & -0.0047110372309 \\
5 & -1.000000000000000 & 0.000000000000000 & 0.000000000000000 \\
6 & -1.000000000000000 & 0.000000000000000 & 0.000000000000000 \\
\hline
\end{tabular}

Log (base 10) of Error vs Iteration #

\begin{tabular}{c|c|c|c|c|c|c|c|c}
\hline
\( k \) & \( |e_{k-1}| \) & \( |e_{k-1}| \) & \( |e_k|^2 \) \\
\hline
0 & 1.4393571238571238571 & 2.28571428571428571428571428571428571 & 1.4393571238571238571 & 2.28571428571428571428571428571428571 \\
1 & 1.000000000000000 & 2.28571428571428571428571428571428571 & 1.000000000000000 & 2.28571428571428571428571428571428571 \\
2 & 0.500000000000000 & 2.28571428571428571428571428571428571 & 0.500000000000000 & 2.28571428571428571428571428571428571 \\
3 & 0.250000000000000 & 2.28571428571428571428571428571428571 & 0.250000000000000 & 2.28571428571428571428571428571428571 \\
4 & 0.125000000000000 & 2.28571428571428571428571428571428571 & 0.125000000000000 & 2.28571428571428571428571428571428571 \\
5 & 0.062500000000000 & 2.28571428571428571428571428571428571 & 0.062500000000000 & 2.28571428571428571428571428571428571 \\
\hline
\end{tabular}
Secant Method

Often the derivative \( g' \) is difficult to compute, making the Newton-Raphson method undesirable. The secant method avoids the use of derivatives by finding the intersection with the \( x \)-axis not of the tangent line, but a secant line.

Two initial "guesses" are required.

If we choose the step \( \Delta^i \) so that \( g_i(x^i + \Delta^i) = 0 \) we get the system of linear equations

\[
\nabla g_i(x^i) \Delta^i = -g_i(x^i) \quad \forall i=1,2,\ldots,n
\]

which we must solve for \( \Delta^i \).

Let \( J(x_1, x_2, \ldots, x_n) \) be the Jacobian of this system, evaluated at \( x \), i.e., the non-zero matrix with row \( i \) equal to \( \nabla g_i(x_1, x_2, \ldots, x_n) \), the gradient of the \( i \)th constraint function.

The Newton-Raphson Method for solving

\[
\begin{align*}
g_1(x_1, x_2, \ldots, x_n) &= 0 \\
g_2(x_1, x_2, \ldots, x_n) &= 0 \\
& \vdots \\
g_n(x_1, x_2, \ldots, x_n) &= 0
\end{align*}
\]

Let \( x^i \) be the approximation to the solution at iteration \( i \).

Then the next approximation is \( x^{i+1} = x^i + \Delta^i \) where \( \Delta^i = -[J(x^i)]^{-1}g(x^i) \)

**System of \( n \) equations in \( n \) variables**

Consider the system of nonlinear equations

\[
\begin{align*}
g_1(x_1, x_2, \ldots, x_n) &= 0 \\
g_2(x_1, x_2, \ldots, x_n) &= 0 \\
& \vdots \\
g_n(x_1, x_2, \ldots, x_n) &= 0
\end{align*}
\]

Dropping the quadratic terms from Taylor's Formula:

\[
g_i(x^i + \Delta^i) = g_i(x^i) + \nabla g_i(x^i) \Delta^i \quad \forall i=1,2,\ldots,n
\]

The coefficient matrix of this linear system is the Jacobian Matrix

\[
J(x) = \begin{bmatrix}
\frac{\partial g_1(x)}{\partial x_1} & \frac{\partial g_2(x)}{\partial x_1} & \cdots & \frac{\partial g_n(x)}{\partial x_1} \\
\frac{\partial g_1(x)}{\partial x_2} & \frac{\partial g_2(x)}{\partial x_2} & \cdots & \frac{\partial g_n(x)}{\partial x_2} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial g_1(x)}{\partial x_n} & \frac{\partial g_2(x)}{\partial x_n} & \cdots & \frac{\partial g_n(x)}{\partial x_n}
\end{bmatrix}
\]
**Example**

Solve:

\[
\begin{align*}
3000 &= x_1 x_2 \\
4000 &= 2
\end{align*}
\]

We will start at \( x = (x_1, x_2) = (15, 15) \)

and terminate when

\[ |g_1(x)| \leq 10^{-10} \text{ and } |g_2(x)| \leq 10^{-10} \]

**Iteration 1**

\[ X = 15 \quad 15 \]

Function Values \( G(X) = 0.11111 \quad 0.814815 \)

Jacobian Matrix:

\[
\begin{bmatrix}
0.014612 & 0.0692563 \\
0.0960123 & 0.154025 \\
\end{bmatrix}
\]

\(<\text{Determinant} = 0.0140466>\)

Step is 2.1875 \(*\) 0.25

with length 0.62716

**Iteration 2**

\[ X = 17.1875 \quad 8.75 \]

Function Values \( G(X) = -0.160614 \quad -1.0947 \)

Jacobian Matrix:

\[
\begin{bmatrix}
0.135963 & 0.132662 \\
0.176855 & 0.0947869 \\
\end{bmatrix}
\]

\(<\text{Determinant} = 0.0700752>\)

Step is 0.373925 \(*\) 1.59161

with length 1.65494

**Iteration 3**

\[ X = 16.8136 \quad 10.3416 \]

Function Values \( G(X) = -0.026155 \quad -0.224465 \)

Jacobian Matrix:

\[
\begin{bmatrix}
0.123069 & 0.0992258 \\
0.132501 & 0.410195 \\
\end{bmatrix}
\]

\(<\text{Determinant} = 0.0393631>\)

Step is 0.279815 \(*\) 0.607905

with length 0.689421

**Iteration 4**

\[ X = 16.5338 \quad 10.6494 \]

Function Values \( G(X) = -0.00227841 \quad -0.0176931 \)

Jacobian Matrix:

\[
\begin{bmatrix}
0.13124 & 0.0952569 \\
0.13249 & 0.368932 \\
\end{bmatrix}
\]

\(<\text{Determinant} = 0.033616>\)

Step is 0.0329609 \(*\) 0.9565805

with length 0.061441
Newton 1/12/99  page 6

**Iteration 5**

\[ X = 16.5094 \quad 11.0064 \]

Function Values \( g(x) = -0.000017087 \quad -0.000134764 \)

Jacobian Matrix:

\[
\begin{pmatrix}
0.131142 & 0.990861 \\
0.131138 & 0.990861 \\
\end{pmatrix}
\]

Determinant = 0.033023

Step is \(-0.0000172825 \quad 0.000428384 \)

with length 0.000461932

---

**Iteration 6**

\[ X = 16.5094 \quad 11.0064 \]

Function Values \( g(x) = -1.02381E-9 \quad -7.67939E-9 \)

Jacobian Matrix:

\[
\begin{pmatrix}
0.121141 & 0.990856 \\
0.121141 & 0.990856 \\
\end{pmatrix}
\]

Determinant = 0.033023

Step is \(-9.60709E-9 \quad 2.43997E-8 \)

with length 2.82960E-6

---

*** Converged ***

(Max. Abs. Value ≤ tolerance, 0.0000000001)

**Final Solution**

\[ X = 16.5094 \quad 11.0064 \]

\( g(x) = 3.33067E-16 \quad 0 \)

CPU time: 11.95 sec.

\# of iterations = 7

---

**Path Followed by the Algorithm**

---

**Optimizing a Nonlinear Function**

\[ \text{Function of a single variable} \]

Minimize \( f(x) \)

---

**Function of several variables**

Minimize \( f(x_1, x_2, \ldots, x_n) \)
Function of a single variable

Minimize \( f(x) \)

Consider the problem of minimizing a function of several variables:

Minimize \( f(x_1, x_2, \ldots, x_n) \)

Suppose that \( f \) is differentiable, i.e., the gradient:

\[
\nabla f(x) = \begin{bmatrix}
\frac{\partial f}{\partial x_1} \\
\frac{\partial f}{\partial x_2} \\
\vdots \\
\frac{\partial f}{\partial x_n}
\end{bmatrix}
\]

is defined.

Necessary Condition for Optimality:

If \( x^* \) is optimal, then

\[
\nabla f(x^*) = 0,
\]

i.e.,

\[
\frac{\partial f}{\partial x_1}(x^*) = 0, \quad \frac{\partial f}{\partial x_2}(x^*) = 0, \quad \ldots, \quad \frac{\partial f}{\partial x_n}(x^*) = 0.
\]

We find that the Jacobian matrix of this system is the Hessian matrix of the function \( f \), i.e.,

\[
\nabla^2 f(x^*)
\]

Newton's Method is simply the application of the Newton-Raphson method to solving this system of equations.

Hessian Matrix

\[
\nabla^2 f(x) = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix}
\]

Newton's Method

Let \( x^t \) be the approximation to the solution at iteration \( t \).

Then the approximation at iteration \( t+1 \) is given by

\[
x^{t+1} = x^t + \Delta^t,
\]

where \( \Delta^t \) is the solution of the (linear) equation

\[
\nabla^2 f(x^t) \Delta^t = -\nabla f(x^t)
\]

i.e.,

\[
\Delta^t = -\nabla^2 f(x^t)^{-1} \nabla f(x^t)
\]

Comments:

- If \( f(x) \) is a quadratic function, Newton's method converges to a stationary point in a single iteration.
- Newton's method does not discriminate among minimizers, maximizers, and saddle points.
- If the Hessian matrix at an iteration is positive definite, then the direction \( \Delta^t \) is a direction of descent, i.e.,

\[
K(x^t + \epsilon \Delta^t) < f(x^t)
\]

for some \( \epsilon > 0 \), even though it might be that \( f(x^t + \Delta^t) > f(x^t) \).
- It is more efficient and more numerically stable to solve the system of equations \( \nabla^2 f(x^t) \Delta^t = -\nabla f(x^t) \) by means other than inverting the Hessian matrix.

Example

Minimize

\[
f(x_1, x_2) = (x_2 - x_1^2)^3 + (1 - x_1)^2
\]

\[
\nabla f(x) = \begin{bmatrix}
\frac{\partial f}{\partial x_1} \\
\frac{\partial f}{\partial x_2}
\end{bmatrix} = \begin{bmatrix}
4x_1^3 - 4x_1x_2 + 2x_1 - 2 \\
2(x_2 - x_1^2)
\end{bmatrix}
\]

\[
\nabla^2 f(x) = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2}
\end{bmatrix} = \begin{bmatrix}
12x_2^2 - 4x_2 + 2 & -4x_1 \\
-4x_1 & 2
\end{bmatrix}
\]
Let's begin at the point \( x^0 = (2, 1) \)

\[ x = (2, 1) \]
\[ f(x) = 10 \]
\[ f'(x) = 2.73 \]
\[ Hessian Matrix = \begin{bmatrix} 4 & 9 \\ 9 & 2 \end{bmatrix} \]

We need to solve the equations:

\[ \begin{align*}
    48_1 - 88_2 &= -26 \\
    -88_1 + 28_2 &= 6
\end{align*} \]

\[ \begin{align*}
    \delta_1 &= -0.1426 \\
    \delta_2 &= 2.4286
\end{align*} \]

\( x^1 = x^0 + \delta \)

\[ x^1 = (2, 1) + (-0.1426, 2.4286) = (1.8574, 3.4258) \]

\[ f(x^1) = 1.8574 \]

\[ f'(x^1) = 2.73 \]

\[ Hessian Matrix = \begin{bmatrix} 4.00 & 9 \\ 9 & 2 \end{bmatrix} \]

Iterate again.

\[ x^2 = (1.8574, 3.4258) \]
\[ f(x^2) = 0.0186 \]
\[ f'(x^2) = 2.73 \]

\[ Hessian Matrix = \begin{bmatrix} 4.00 & 9 \\ 9 & 2 \end{bmatrix} \]

Iterate again.

\[ x^3 = (1.8063, 4.0966) \]
\[ f(x^3) = 0.0048 \]
\[ f'(x^3) = 2.73 \]

\[ Hessian Matrix = \begin{bmatrix} 4.00 & 9 \\ 9 & 2 \end{bmatrix} \]

Iterate again.

\[ x^4 = (1.8002, 4.0004) \]
\[ f(x^4) = 0.0000 \]
\[ f'(x^4) = 2.73 \]

\[ Hessian Matrix = \begin{bmatrix} 4.00 & 9 \\ 9 & 2 \end{bmatrix} \]
Disadvantages

- Newton's method converges to a stationary point, which may or may not be a minimizer.

- Newton's method requires the computation of $\frac{1}{2}n^2$ 2nd partial derivatives (i.e., the Hessian matrix).

- Newton's method requires at each iteration the inversion of a matrix of order $n$ (for the solution of a linear system of equations in $n$ variables).

For these reasons, Newton's method is not of practical significance for unconstrained optimization.