# Newton's Method

This Hypercard stack was prepared by:
Dennis L. Bricker,
Dept. of Industrial Engineering,
University of Iowa,
Iowa City, Iowa 52242
e-mail: dbricker@icaen.uiowa.edu

Reference: Section 3.5 of Linear & Nonlinear Programming, by A. Sofer and S. Nash

author

## Newton's Method

- Solving Nonlinear Equations
- Optimizing a Nonlinear Function

## Solving Nonlinear Equations

- Single equation with single variable
  $g(x) = 0$
- System of n equations in n variables
  $g_i(x_1, x_2, \cdots x_n) = 0 \quad \forall i=1,2,\cdots n$

## Solving a Nonlinear Equation $g(x) = 0$

...when an analytic solution doesn't exist.

For example, $2x - \sin x = 0.5$,
$x^2 + x \log x = 1$,
$x^4 - 2x + 5 = 0$, etc.

The Newton-Raphson method is an iterative, successive-approximation method for numerically solving such equations.

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

$$g(x_0 + \delta) \approx 0$$

Then $x_1 = x_0 + \delta$ becomes our improved approximation, and we repeat the procedure until $g(x_k)$ is "sufficiently close" to zero.

## Solving a Nonlinear Equation $g(x) = 0$

For some $z \in [x^*, x]$,

$$g(x) = g(x^*) + g'(x^*)(x - x^*) + \frac{g''(z)}{2}(x - x^*)^2$$

*Taylor's Formula*

Equivalently,

$\exists\, \alpha \in [0,1]$ such that

$$g(x + \delta) = g(x) + g'(x)\delta + \frac{g''(x+\alpha\delta)}{2}\delta^2$$

Suppose that $x_k$ is a "guess" or approximation to the solution of $g(x) = 0$ at iteration k.

$$g(x_k + \delta) = g(x_k) + g'(x_k)\delta + \underbrace{\frac{g''(x_k+\alpha\delta)}{2}\delta^2}_{\textit{assumed negligible}} \approx 0$$

Solve the equation

$$g(x_k + \delta) = g(x_k) + g'(x_k)\delta = 0$$

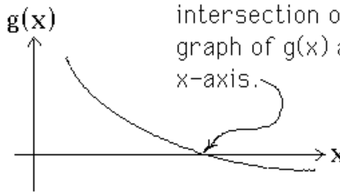
for the $\delta$ which will give (hopefully) an improved approximate solution.

$$g(x_k) + g'(x_k)\delta = 0 \;\Rightarrow\; \delta = -\frac{g(x_k)}{g'(x_k)}$$

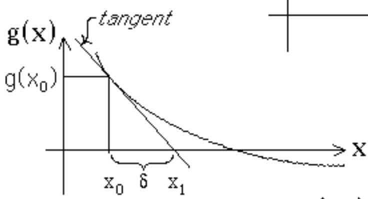
The new (improved?) approximation is then

$$x_{k+1} = x_k + \delta = x_k - g(x_k)/g'(x_k)$$

## Geometric Interpretation

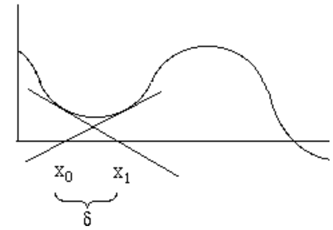



We are seeking the intersection of the graph of g(x) and the x-axis.




$x_1 = x_0 + \delta$ is the intersection with the x-axis of the tangent line to the graph at $(x_0, g(x_0))$

## Possible "Pathologies" of Newton's Method




## Rate of Convergence

Define the error at iteration k by $\varepsilon_k = x_* - x_k$

Taylor's formula implies that, for some $z \in [x_k, x_*]$,

$$0 = g(x_*) = g(x_k + \varepsilon_k) = g(x_k) + g'(x_k)\,\varepsilon_k + \frac{1}{2} g''(z)\varepsilon_k^2$$

$$-\varepsilon_k - g(x_k)/g'(x_k) = \frac{1}{2}\frac{g''(z)}{g'(x_k)}\varepsilon_k^2$$

$$-x_* + x_k - g(x_k)/g'(x_k) = \frac{1}{2}\frac{g''(z)}{g'(x_k)}(x_* - x_k)^2$$

$$\underbrace{-x_* + x_k - g(x_k)/g'(x_k)}_{x_{k+1}} = \frac{1}{2}\frac{g''(z)}{g'(x_k)}(x_* - x_k)^2$$

$$\Rightarrow \quad -x_* + x_{k+1} = \frac{1}{2}\frac{g''(z)}{g'(x_k)}(x_* - x_k)^2$$

$$\lim_{k\to\infty}\varepsilon_k = 0 \Rightarrow x_{k+1} - x_* \approx = \frac{1}{2}\frac{g''(z)}{g'(x_k)}(x_* - x_k)^2$$

That is, the error is approximately squared at each iteration

$\Rightarrow$ The convergence is ***quadratic*** with $C = \left|\frac{1}{2}\frac{g''(z)}{g'(x_k)}\right|$

```
     ∇Y←X ROOT C;I;⎕IO;MAX∆IT
[1]   ⍝
[2]   ⍝    Newton-Raphson method applied to finding a real
[3]   ⍝    root of a polynomial with coefficients C
[4]   ⍝    Appeared in APL Quote Quad, June '83, pp.19-20
[5]   ⍝    Author: Yuji Ijiri
[6]   ⍝
[7]   i←1  ⋄ ⎕IO←0   ⋄ MAX∆IT←50
[8]   Next:X←(Y←X)-X⊥C×X÷X⊥C×⌽⍳⍴C
[9]   →Next IF (Y≠X)∧(MAX∆IT≥i←i+1)
     ∇
```

$3X^3 + 2X^2 + X + 1 = 0$

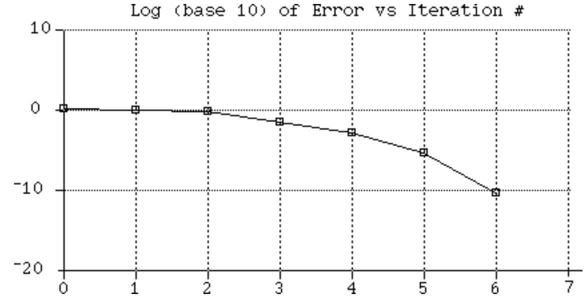
| t | $X_t$ | $G(X_t)$ |
|---|---|---|
| 0 | 1.000000000000000 | 7.000000000000000 |
| 1 | 0.500000000000000 | 2.375000000000000 |
| 2 | 0.047619047619048 | 1.052478134110787 |
| 3 | ¯0.821562332798288 | ¯0.135209637777137 |
| 4 | ¯0.785872207198749 | ¯0.006734489690139 |
| 5 | ¯0.783900098449740 | ¯0.000019706383372 |
| 6 | ¯0.783894293737122 | ¯0.000000000170329 |
| 7 | ¯0.783894293686949 | 0.000000000000000 |

| t | $X_t$ | $G(X_t)$ |
|---|---|---|
| 0 | ¯1.000000000000000 | ¯1.000000000000000 |
| 0 | ¯0.833333333333333 | ¯0.180555555555555 |
| 1 | ¯0.787234042553192 | ¯0.011394392379338 |
| 2 | ¯0.783910803476619 | ¯0.000056049392973 |
| 3 | ¯0.783894294092810 | ¯0.000000001377830 |
| 4 | ¯0.783894293686949 | 0.000000000000000 |

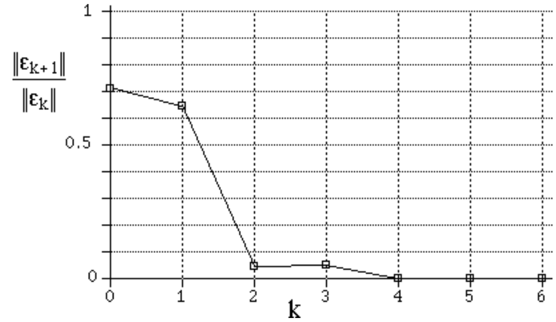
$G(X) = 3X^3 + 2X^2 + X + 1 = 0$

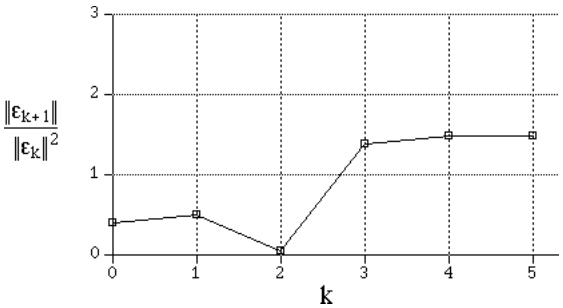
| t | $X_t$ | $G(X_t)$ | $\varepsilon_t$ |
|---|---|---|---|
| 0 | 1.000000000000000 | 7.000000000000000 | ¯1.783894293686949 |
| 1 | 0.500000000000000 | 2.375000000000000 | ¯1.283894293686949 |
| 2 | 0.047619047619048 | 1.052478134110787 | ¯0.831513341305997 |
| 3 | ¯0.821562332798288 | ¯0.135209637777137 | 0.037668039111338 |
| 4 | ¯0.785872207198749 | ¯0.006734489690139 | 0.001977913511799 |
| 5 | ¯0.783900098449740 | ¯0.000019706383372 | 0.000005804762791 |
| 6 | ¯0.783894293737122 | ¯0.000000000170329 | 0.000000000050173 |
| 7 | ¯0.783894293686949 | 0.000000000000000 | 0.000000000000000 |

Log (base 10) of Error vs Iteration #

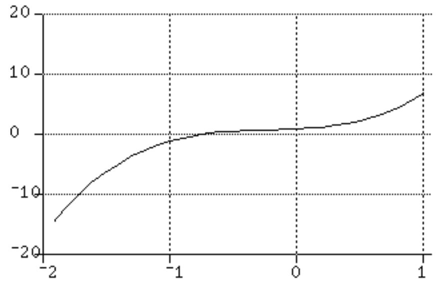


| k | $\frac{\lVert\varepsilon_{k+1}\rVert}{\lVert\varepsilon_k\rVert}$ | $\frac{\lVert\varepsilon_{k+1}\rVert}{\lVert\varepsilon_k\rVert^2}$ |
|---|---|---|
| 0 | 7.19714334100705E¯1 | 4.03451222781368E¯1 |
| 1 | 6.47649378453226E¯1 | 5.04441355988410E¯1 |
| 2 | 4.53005829734204E¯2 | 5.44796826738463E¯2 |
| 3 | 5.25090649383936E¯2 | 1.39399517939303E0 |
| 4 | 2.93479100899355E¯3 | 1.48378126317763E0 |
| 5 | 8.64345378706824E¯6 | 1.48902790661432E0 |
| 6 | 2.21278102318995E¯6 | 4.41028491391872E4 |



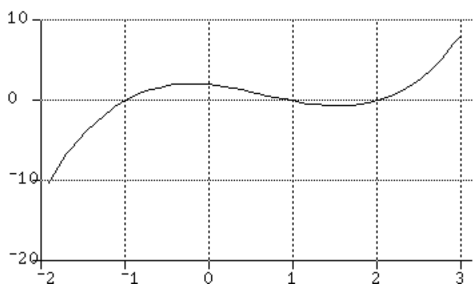

$G(X) = 3X^3 + 2X^2 + X + 1$



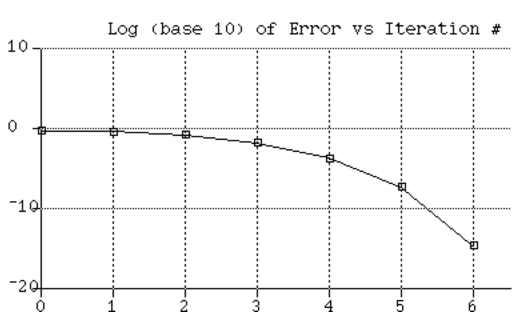

There appears to be only one real root!

$G(X) = X^3 - 2X^2 - X + 2$

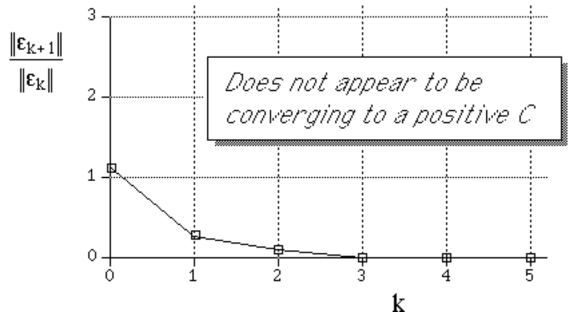
Roots are −1, 1, & 2

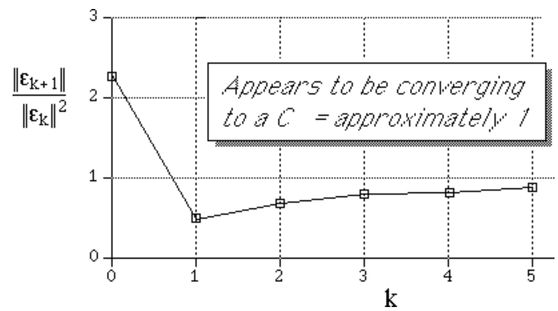



| t | $X_t$ | $G(X_t)$ | $\varepsilon_t$ |
|---|---|---|---|
| 0 | ¯0.500000000000000 | 1.875000000000000 | ¯0.500000000000000 |
| 1 | ¯1.571428571428571 | ¯5.247813411078717 | 0.571428571428571 |
| 2 | ¯1.158015617822692 | ¯1.076883866081184 | 0.158015617822692 |
| 3 | ¯1.017339588345219 | ¯0.105546050034808 | 0.017339588345219 |
| 4 | ¯1.000245166873965 | ¯0.001471301792509 | 0.000245166873965 |
| 5 | ¯1.000000050073447 | ¯0.000000300440693 | 0.000000050073447 |
| 6 | ¯1.000000000000002 | ¯0.000000000000013 | 0.000000000000002 |
| 7 | ¯1.000000000000000 | 0.000000000000000 | 0.000000000000000 |




| k | $\frac{\lVert\varepsilon_{k+1}\rVert}{\lVert\varepsilon_k\rVert}$ | $\frac{\lVert\varepsilon_{k+1}\rVert}{\lVert\varepsilon_k\rVert^2}$ |
|---|---|---|
| 0 | 1.14285714285714E0 | 2.28571428571429E0 |
| 1 | 2.76527331189711E¯1 | 4.83922829581994E¯1 |
| 2 | 1.09733383219595E¯1 | 6.94446439735635E¯1 |
| 3 | 1.413914039274E¯2 | 8.15425378690662E¯1 |
| 4 | 2.04242302038894E¯4 | 8.33074626826414E¯1 |
| 5 | 4.43437828748177E¯8 | 8.85574806157926E¯1 |

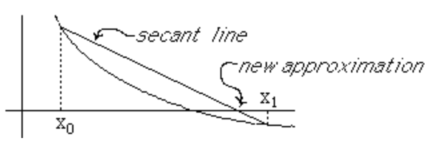



**Secant Method**

Often the derivative $g'(x)$ 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.*




**System of n equations in n variables**

Consider the system of nonlinear equations

$$\begin{cases} g_1(x_1,x_2, \ldots x_n) = 0 \\ g_2(x_1,x_2, \ldots x_n) = 0 \\ \vdots \\ g_n(x_1,x_2, \ldots x_n) = 0 \end{cases}$$

Dropping the quadratic terms from Taylor's Formula:

$$\mathbf{g_i (x^t+\Delta^t) \approx g_i (x^t) + \nabla g_i (x^t)\, \Delta^t \ \forall\ i=1,2,\cdots n}$$

If we choose the step $\Delta^t$ so that $\mathbf{g_i (x^t+\Delta^t)} \approx 0$ we get the system of *linear* equations

$$\boxed{\nabla \mathbf{g_i (x^t)\, \Delta^t = -\ g_i (x^t) \quad \forall\ i=1,2,\cdots n}}$$

which we must solve for $\Delta^t$

Let $J(x_1,x_2, \ldots x_n)$ be the *Jacobian* of this system, evaluated at x, i.e., the **n**x**n** 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 coefficient matrix of this linear system is the *Jacobian Matrix*

$$\mathbf{J(x^t)} \equiv \begin{bmatrix} \frac{\partial g_1(x^t)}{\partial x_1} & \frac{\partial g_1(x^t)}{\partial x_2} & \cdots & \frac{\partial g_1(x^t)}{\partial x_n} \\ \frac{\partial g_2(x^t)}{\partial x_1} & \frac{\partial g_2(x^t)}{\partial x_2} & \cdots & \frac{\partial g_2(x^t)}{\partial x_n} \\ \vdots & & \ddots & \\ \frac{\partial g_n(x^t)}{\partial x_1} & \frac{\partial g_n(x^t)}{\partial x_2} & \cdots & \frac{\partial g_n(x^t)}{\partial x_n} \end{bmatrix}$$

The *Newton-Raphson Method* for solving

$$\begin{cases} g_1(x_1,x_2, \ldots x_n) = 0 \\ g_2(x_1,x_2, \ldots x_n) = 0 \\ \vdots \\ g_n(x_1,x_2, \ldots x_n) = 0 \end{cases}$$

Let $x^t$ be the approximation to the solution at iteration #t.

Then the next approximation is $x^{t+1} = x^t + \Delta^t$

where $\Delta^t = -[J(x^t)]^{-1} g(x^t)$

*inverse of the Jacobian matrix*

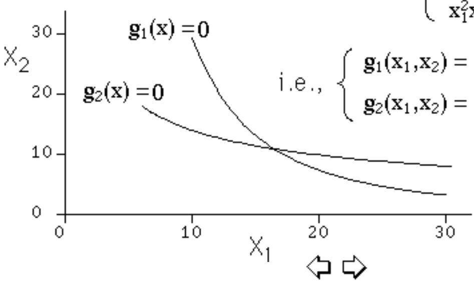
```
∇Y←NEWTON∆RAPHSON X;V;J;DY
  ⍝
  ⍝           SOLVE THE NONLINEAR SYSTEM OF EQUATIONS
  ⍝                          G(X) = 0
  ⍝           WHERE G IS A VECTOR-VALUED FUNCTION
  ⍝           (G AND JACOBIAN MUST BE USER-DEFINED,
  ⍝            AS WELL AS TOL AND MAX∆ITER)
  ⍝
  ⎕TCNL,'Tolerance (on ⌈/|G(X) )= ',⍕TOL
  start_timer ⋄ ITER←1 ⋄ Y←X ⋄ ⎕TCNL
  X∆PATH←((⍴X),1)⍴X
  NEXT:V←G Y
  →STOP IF TOL≥⌈/|V
  J←JACOBIAN Y
  DY←-V⌹J
  →bypass IF ~detail ⋄ NR∆OUTPUT
  bypass:X∆PATH←X∆PATH,Y←Y+DY
  →NEXT IF MAX∆ITER≥ITER←ITER+1
  'Warning... Newton-Raphson algorithm did not converge.'
  →0
  STOP:⎕TCNL,'*** Converged ***'
  '(Max. Abs. Value ≤ tolerance, ',(⍕TOL),')'
```

**APL code**

## EXAMPLE

Solve: $\begin{cases} \dfrac{3000}{x_1 x_2^2} = 1 \\ \dfrac{4000}{x_1^2 x_2} = 2 \end{cases}$

i.e., $\begin{cases} g_1(x_1,x_2) = 1 - 3000\, x_1^{-1}\, x_2^{-2} = 0 \\ g_2(x_1,x_2) = 2 - 4000\, x_1^{-2}\, x_2^{-1} = 0 \end{cases}$



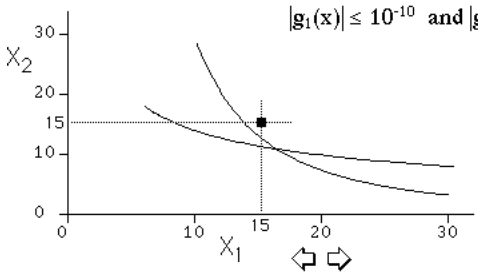

```
∇Z←G X
 ⍝
 ⍝   EXAMPLE EQUATIONS FOR NEWTON-RAPHSON ALGORITHM
 ⍝
 Z←(1-3000×X×.*¯2 ¯1)  ⍝ Left side of first equation
 Z←Z,2-4000×X×.*¯1 ¯2  ⍝ Left side of 2nd equation
                       ⍝      is appended
∇

∇A ← JACOBIAN X
 ⍝
 ⍝      JACOBIAN MATRIX FOR SAMPLE SYSTEM OF EQUATIONS
 ⍝      TO BE SOLVED BY NEWTON-RAPHSON ALGORITHM
 ⍝
 A←2 2⍴0
 A[1;1]←6000×X×.*¯3 ¯1
 A[2;2]←8000×X×.*¯1 ¯3
 A[1;2]←3000×X×.*¯2 ¯2
 A[2;1]←4000×X×.*¯2 ¯2
∇
```

**APL code**

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

and terminate when

$|g_1(x)| \le 10^{-10}$ and $|g_2(x)| \le 10^{-10}$




Tolerance (on ⌈/|G(X) )= 0.0000000001

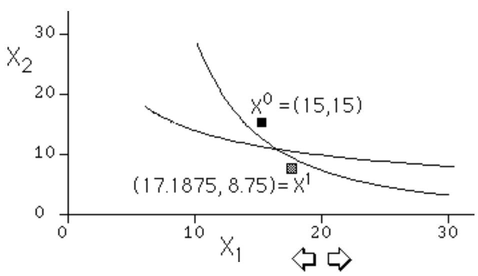
**Iteration 1**

X = 15 15
Function Values G(X) = 0.111111 0.814815
Jacobian Matrix:

$$\begin{bmatrix} 0.118519 & 0.0592593 \\ 0.0790123 & 0.158025 \end{bmatrix}$$

(Determinant = 0.0140466 )

Step is 2.1875 ¯6.25
with length 6.62176




**Iteration 2**

X = 17.1875 8.75
Function Values G(X) = ¯0.160614 ¯1.0397
Jacobian Matrix:

| | |
|---|---|
| 0.135053 | 0.132642 |
| 0.176855 | 0.694789 |

(Determinant = 0.0703752 )

Step is ¯0.373925 1.59161
with length 1.63494

**Iteration 3**

X = 16.8136 10.3416
Function Values G(X) = ¯0.026155 ¯0.224455
Jacobian Matrix:

| | |
|---|---|
| 0.122063 | 0.0992258 |
| 0.132301 | 0.430195 |

(Determinant = 0.0393831 )

Step is ¯0.279815 0.607805
with length 0.669121

**Iteration 4**

X = 16.5338 10.9494
Function Values G(X) = ¯0.00227541 ¯0.0179319
Jacobian Matrix:

| | |
|---|---|
| 0.12124 | 0.0915369 |
| 0.122049 | 0.368592 |

(Determinant = 0.033516 )

Step is ¯0.0239508 0.0565805
with length 0.061441

Iteration 5

```
X = 16.5098 11.006
Function Values G(X) = ¯0.000017987 ¯0.000134764
Jacobian Matrix:
                    0.121142   0.0908612
                    0.121148   0.363463
(Determinant = 0.033023 )
Step is ¯0.000172825 0.000428384
with length 0.000461932
```

Iteration 6

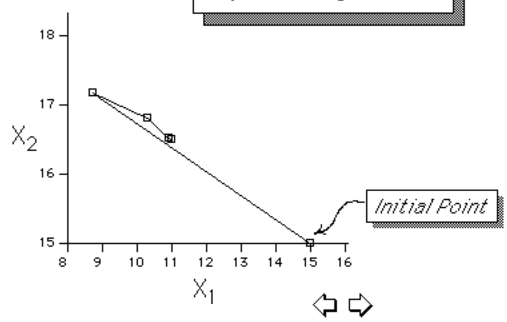
```
X = 16.5096 11.0064
Function Values G(X) = ¯1.02881E¯9 ¯7.67939E¯9
Jacobian Matrix:
                    0.121141   0.090856
                    0.121141   0.363424
(Determinant = 0.0330193 )
Step is ¯9.80709E¯9 2.43997E¯8
with length 2.62968E¯8
```

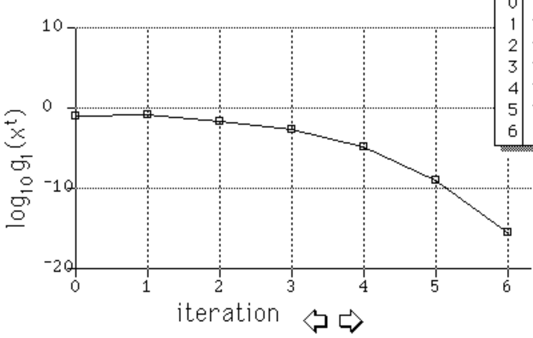
```
*** Converged ***
(Max. Abs. Value ≤ tolerance, 0.0000000001)
```

Final Solution

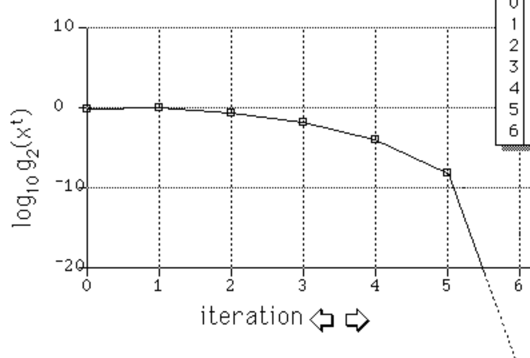
```
X = 16.5096 11.0064
G(X) = 3.33067E¯16 0
CPU time: 11.95 sec.
# of iterations = 7
```

Path Followed by the Algorithm



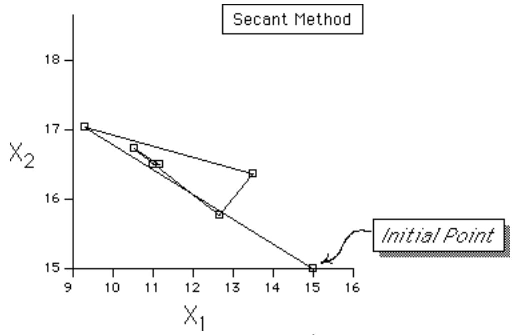




| t | Error |
|---|---|
| 0 | 1.11111E¯1 |
| 1 | ¯1.60614E¯1 |
| 2 | ¯2.6155E¯2 |
| 3 | ¯2.27541E¯3 |
| 4 | ¯1.7987E¯5 |
| 5 | ¯1.02881E¯9 |
| 6 | 3.33067E¯16 |




| t | Error |
|---|---|
| 0 | 8.14815E¯1 |
| 1 | ¯1.0397E0 |
| 2 | ¯2.24455E¯1 |
| 3 | ¯1.79319E¯2 |
| 4 | ¯1.34764E¯4 |
| 5 | ¯7.67939E¯9 |
| 6 | 0E0 |

Secant Method




## Optimizing a Nonlinear Function

- Function of a single variable

  **Minimize f (x)**

- Function of several variables

  **Minimize f($x_1$, $x_2$ , ⋯ $x_n$ )**

## Function of a single variable

**Minimize f (x)**

Consider the problem of minimizing a function of several variables:

Minimize $f(x_1, x_2, \dots x_n)$

Suppose that f is differentiable, i.e., the gradient

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \text{ is defined.}$$

**Necessary Condition for Optimality:**

If X* is optimal, then

$\nabla f(X^*) = 0$, i.e.,

$\frac{\partial f(X^*)}{\partial x_i} = 0$ for each $i=1,2,\dots n$

This necessary condition for optimality yields a system of (in general) n ***nonlinear*** equations in n variables.

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

**Necessary Condition for Optimality:**

If X* is optimal, then

$\nabla f(X^*) = 0$, i.e.,

$\frac{\partial f(X^*)}{\partial x_i} = 0$ for each $i=1,2,\dots n$

Applying the Newton-Raphson method to the solution of the equations $\nabla f(X^*) = 0$,

i.e.,
$$\begin{cases} g_1(x_1,x_2, \dots x_n) = \frac{\partial f}{\partial x_1} = 0 \\ g_2(x_1,x_2, \dots x_n) = \frac{\partial f}{\partial x_2} = 0 \\ \vdots \\ g_n(x_1,x_2, \dots x_n) = \frac{\partial f}{\partial x_n} = 0 \end{cases}$$

we find that the Jacobian matrix of this system is the **Hessian** matrix of the function f, i.e., $\nabla^2 f(x)$

### *Hessian Matrix*

$$\nabla^2 f(x^t) \equiv \begin{bmatrix} \frac{\partial^2 f(x^t)}{\partial^2 x_1} & \frac{\partial^2 f(x^t)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x^t)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x^t)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x^t)}{\partial^2 x_2} & \cdots & \frac{\partial^2 f(x^t)}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \\ \frac{\partial^2 f(x^t)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x^t)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x^t)}{\partial^2 x_n} \end{bmatrix}$$

### *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., $$\boxed{\Delta^t = -[\nabla^2 f(x^t)]^{-1} \nabla f(x^t)}$$

### *Comments*

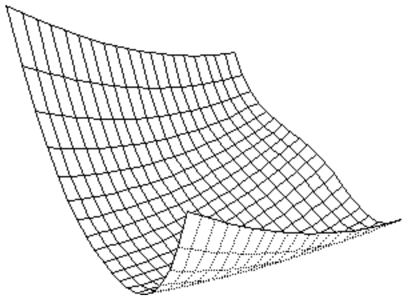
- If f(x) is a quadratic function, Newton's method converges to a stationary point in a single iteration.
- Newton's method does not discriminate among minimizers, maximizers, and saddle points
- If the Hessian matrix at an iteration is positive definite, then the direction $\delta^t$ is a direction of descent, i.e., $f(x^t + \varepsilon\Delta^t) < f(x^t)$ for some $\varepsilon > 0$, even though it might be that $f(x^t + \Delta^t) \geq f(x^t)$.
- 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.

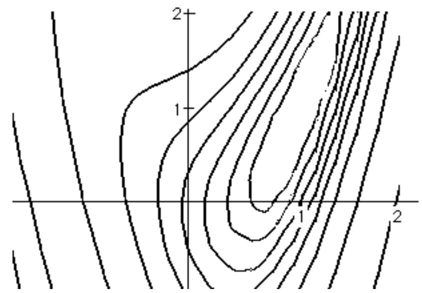
### Example

**Minimize** $f(x_1,x_2) = (x_2 - x_1^2)^2 + (1 - x_1)^2$

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

$$\nabla^2 f(x) = \begin{bmatrix} \partial^2 f / \partial x_1^2 & \partial^2 f / \partial x_1 \partial x_2 \\ \partial^2 f / \partial x_2 \partial x_1 & \partial^2 f / \partial x_2^2 \end{bmatrix} = \begin{bmatrix} 12x_1^2 - 4x_2 + 2 & -4x_1 \\ -4x_1 & 2 \end{bmatrix}$$






Let's begin at the point $x^0 = (2, 1)$

Iteration 1

```
x = 2 1
F(x) = 10
∇F(x) = 26 ¯6
Hessian Matrix =
        [ 46  ¯8 ]
        [ ¯8   2 ]
```

We need to solve the equations $\begin{cases} 46\delta_1 - 8\delta_2 = -26 \\ -8\delta_1 + 2\delta_2 = 6 \end{cases}$

$\Longrightarrow \begin{cases} \delta_1 = -0.1428 \\ \delta_2 = \ \ 2.4286 \end{cases}$

$\begin{cases} \delta_1 = -0.1428 \\ \delta_2 = \ \ 2.4286 \end{cases} \Longrightarrow \begin{array}{l} x_1^1 = x_1^0 + \delta_1 = 2 - 0.1428 \\ x_2^1 = x_2^0 + \delta_2 = 2 + 2.4286 \end{array}$

Iteration 2

```
x = 1.857142857 3.428571429
F(x) = 0.7351103707
∇F(x) = 1.865889213 ¯0.04081632653
Hessian Matrix =
        [ 29.67346939   ¯7.428571429 ]
        [ ¯7.428571429   2           ]

Improvement: 9.264889629
```

Iteration 3

```
x = 1.033613445 0.3901560624
F(x) = 0.4610860424
∇F(x) = 2.871216307 ¯1.356401384
Hessian Matrix =

        [ 13.2596568    ¯4.134453782 ]
        [ ¯4.134453782   2           ]

Improvement: 0.2740243283
```

Iteration 4

```
x = 1.019348709 1.038868307
F(x) = 0.0003744139315
∇F(x) = 0.03952709666 ¯0.000406965428
Hessian Matrix =
        [ 10.31338825   ¯4.077394835 ]
        [ ¯4.077394835   2           ]
Improvement: 0.4607116285
```

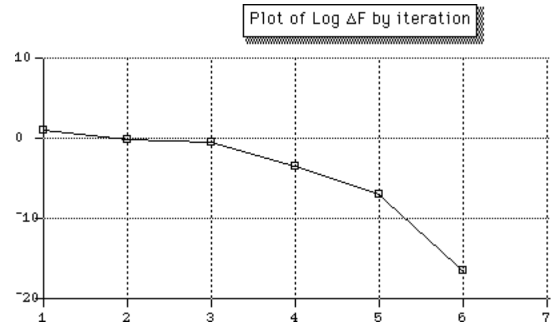
Iteration 5

```
x = 1.000007871 0.9996416742
F(x) = 1.399888212E¯7
∇F(x) = 0.001512025877 ¯0.0007481359977
Hessian Matrix =
        [ 10.00162221   ¯4.000031484 ]
        [ ¯4.000031484   2           ]

Improvement: 0.0003742739427
```

Iteration 6

```
x = 1.000000006 1.000000012
F(x) = 3.462781953E¯17
∇F(x) = 1.201587541E¯8 ¯1.237219216E¯10
Hessian Matrix =
        [ 10.00000009   ¯4.000000024 ]
        [ ¯4.000000024   2           ]

Improvement: 1.399888212E¯7
```

```
Convergence criterion satisfied:

Improvement in objective
Step size
Gradient
*** CONVERGED ***

Solution found is 1.000000006 1.000000012
where F is 3.462781953E¯17
and ∇F is 1.201587541E¯8 ¯1.237219216E¯10

# iterations = 6
Elapsed CPU time: 11.85 seconds
```




## *Disadvantages*

- Newton's method converges to a stationary point, which may or may not be a minimizer
- Newton's method requires the computation of $^1/_2\, n^2$ $2^{nd}$ partial derivatives (i.e., the Hessian matrix)
- Newton's method requires at each iteration the inversion of a matrix of order n (or 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.*