

# A Symbolic Newton-Raphson Method of Finding Roots

**Marc A. Murison**

*Astronomical Applications Department*

*U.S. Naval Observatory*

murison@aa.usno.navy.mil

<http://aa.usno.navy.mil/murison/>

December 1996

## 1. Introduction

Suppose we have a function  $f(x)$  for which we'd like to find a root. At some arbitrary point  $x = a$ , the equation of the line tangent to  $f(x)$  is

$$(1) \quad y = \left( \frac{\partial}{\partial x} f(a) \right) x + b$$

where  $b$  is the  $y$  intercept and we use the notational convention that  $\frac{\partial}{\partial x} f(a)$  is the derivative of  $f(x)$  evaluated at  $x = a$ . In the Newton-Raphson method of root-finding, we assume  $x = a$  is close to a root, which we will denote as  $x_0$ . Then the intersection of the line with the  $x$  axis, at the point  $x_1$ , will be closer to  $x_0$  than  $a$ . We then find the intersection of the line tangent to  $f(x_1)$  with the  $x$  axis, call it  $x_2$ . We iterate in this fashion until at some point the  $x$  axis intersection point, say  $x_n$ , is satisfactorily close to the actual root  $x_0$ .

The  $y$  intercept of the line described by eq. (1) is

$$(2) \quad b = f(a) - \left( \frac{\partial}{\partial x} f(a) \right) a$$

and the  $x$  intercept is found by setting  $y = 0$ , that is,  $x = \frac{-b}{\frac{\partial}{\partial x} f(a)}$ . Thus, the  $(k+1)$ th  $x$  axis

intersection point is

$$(3) \quad x_{k+1} = \frac{\left( \frac{\partial}{\partial x} f(x_k) \right) x_k - f(x_k)}{\frac{\partial}{\partial x} f(x_k)} \quad \text{or} \quad x_{k+1} = x_k - \frac{f(x_k)}{\frac{\partial}{\partial x} f(x_k)}$$

## 2. A Symbolic Newton-Raphson Procedure

We are all used to the Newton-Raphson method in a purely numerical context. What results if we perform the iterations *symbolically*? Here is a procedure that symbolically implements the Newton-Raphson method of finding roots as embodied in eq. (3). The arguments are the function  $f(x)$ , the starting point  $a$ , the number of iterations to perform  $n$ , and, if  $f(x)$  is not a procedure, a fourth argument: the independent variable  $x$ .

```
newt := proc(f::algebraic, a::algebraic, n::posint)
local k, d, dk, fk, x0, x;
  if type(f, procedure) then
    d := D(f);
    x0 := a;
    for k to n do
      dk := d(x0);
      fk := f(x0);
      if not type(fk, numeric) and  $1000 < \text{op}([2, 1], \text{cost}(fk / dk))$  then
        x0 := x0 - normal(fk / dk)
      else x0 := x0 - factor(fk / dk)
      fi
    od
  else
    if nargs  $\neq 4$  then ERROR(
      `: If f(x) is not a procedure, then the `4th argument must be the independent variable x.`)
    fi;
    if not type(args[4], name) then ERROR(`: Fourth argument x must be a name.`) fi;
    x := args[4];
    d := diff(f, x);
    x0 := a;
    for k to n do
      dk := subs(x = x0, d);
      fk := subs(x = x0, f);
      if not type(fk, numeric) and  $1000 < \text{op}([2, 1], \text{cost}(fk / dk))$  then
        x0 := x0 - normal(fk / dk)
      else x0 := x0 - factor(fk / dk)
      fi
    od
  fi
end
```

## 3. Examples

### 3.1. Roots of a Quadratic

First, let's consider a simple quadratic:

$f := \text{convert}([\text{seq}(a_k x^k, k = 0 .. 2)], +)$

$$f := a_0 + a_1 x + a_2 x^2$$

The second-order Newton-Raphson approximation to one of the roots of  $f$ , starting at a point  $x = b$ , is then given by

$\text{newt}(f, b, 2, x)$

$$b - \frac{a_0 + a_1 b + a_2 b^2}{a_1 + 2 a_2 b} - \frac{a_2 (a_0 + a_1 b + a_2 b^2)^2}{\left(a_1^2 + 2 a_1 a_2 b + 2 a_2^2 b^2 - 2 a_2 a_0\right) (a_1 + 2 a_2 b)}$$

We could put this in a simpler form, say ~~factor~~

$$- \frac{a_2^3 b^4 - 4 a_0 a_1 a_2 b - 6 a_0 a_2^2 b^2 - a_0 a_1^2 + a_2 a_0^2}{\left(-a_1^2 - 2 a_1 a_2 b - 2 a_2^2 b^2 + 2 a_2 a_0\right) (a_1 + 2 a_2 b)}$$

which involves significantly fewer operations, especially for larger iterations  $n$ . But the former version has the advantage of explicitly showing the correction terms, making them easy to isolate.

### 3.2. Roots of a Cubic

Let's consider now a simple cubic, for which we know the roots.

$f := (x - 3)(x + 2)(x - 1)$

$$f := (x - 3)(x + 2)(x - 1)$$

The second-order Newton-Raphson root approximation, again starting at a point  $x = b$ , is

$\text{newt}(f, b, 2, x)$

$$b - \frac{(b - 3)(b + 2)(b - 1)}{3 b^2 - 4 b - 5} - 2 \frac{(2 b - 1)(b - 1)^2 (2 b + 1)(b - 3)^2 (b - 2)(b + 2)^2}{(12 b^6 - 48 b^5 + 23 b^4 + 56 b^3 + 174 b^2 - 296 b - 137)(3 b^2 - 4 b - 5)}$$

We could also use a Maple procedure for the function  $f(x)$ :

$f := x \rightarrow (x - 3)(x + 2)(x - 1)$

$$f := x \rightarrow (x - 3)(x + 2)(x - 1)$$

$\text{newt}(f, b, 2)$

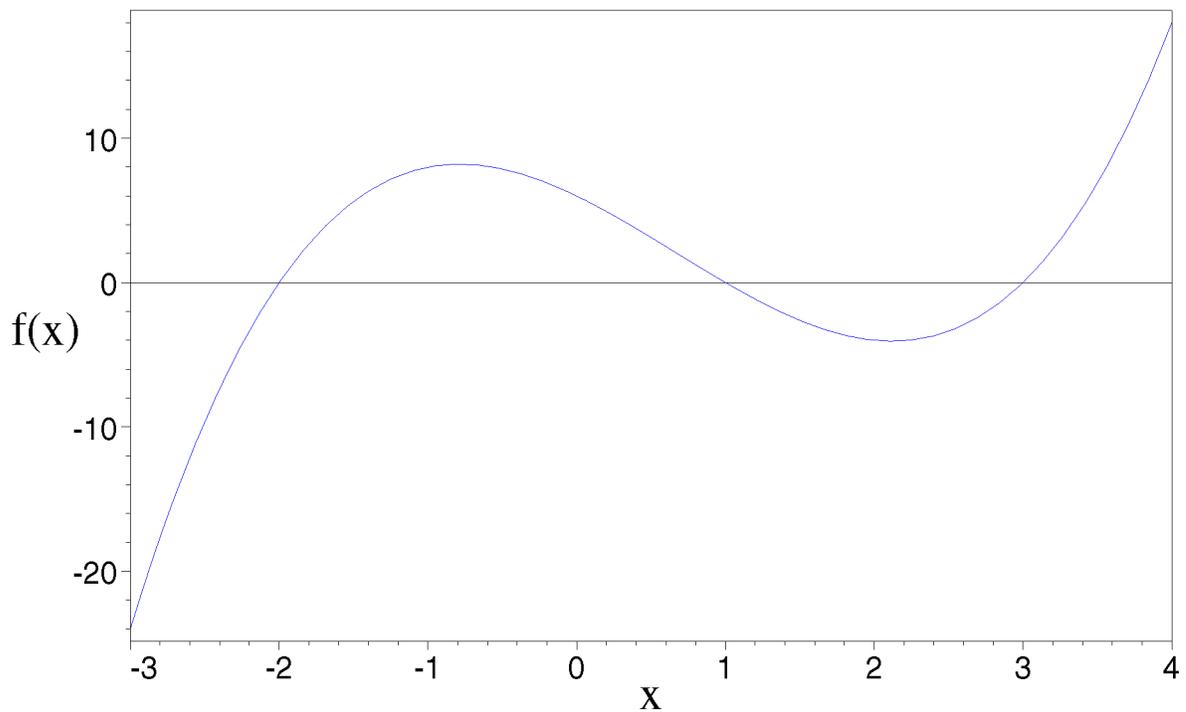
$$b - \frac{(b-3)(b+2)(b-1)}{3b^2 - 4b - 5}$$

$$- 2 \frac{(2b-1)(b-1)^2(2b+1)(b-3)^2(b-2)(b+2)^2}{(12b^6 - 48b^5 + 23b^4 + 56b^3 + 174b^2 - 296b - 137)(3b^2 - 4b - 5)}$$

## 4. An Illustrative Convergence Plot

Let's plot our simple test cubic:

```
plot([f(x), 0], x = -3 .. 4, color = [blue, black], labels = ["x", "f(x)"])
```



Now let's graphically illustrate the convergence to the roots, given several starting points. First define a few convenient procedures.

- A procedure to create a list of point pairs of the form [iteration number, x intercept]

```
p := proc(N, b) local n; [seq([n, newt(f, b, n)], n = 1 .. N)] end
```

- A procedure to plot the point pairs as circles and to connect the points with line segments:

```
P := proc(N, b, c)
  global p1, p2;
  p1 := plot(p(N, b), style = point, symbol = circle, color = black);
  p2 := plot(p(N, b), style = line, color = c);
  p1, p2
```

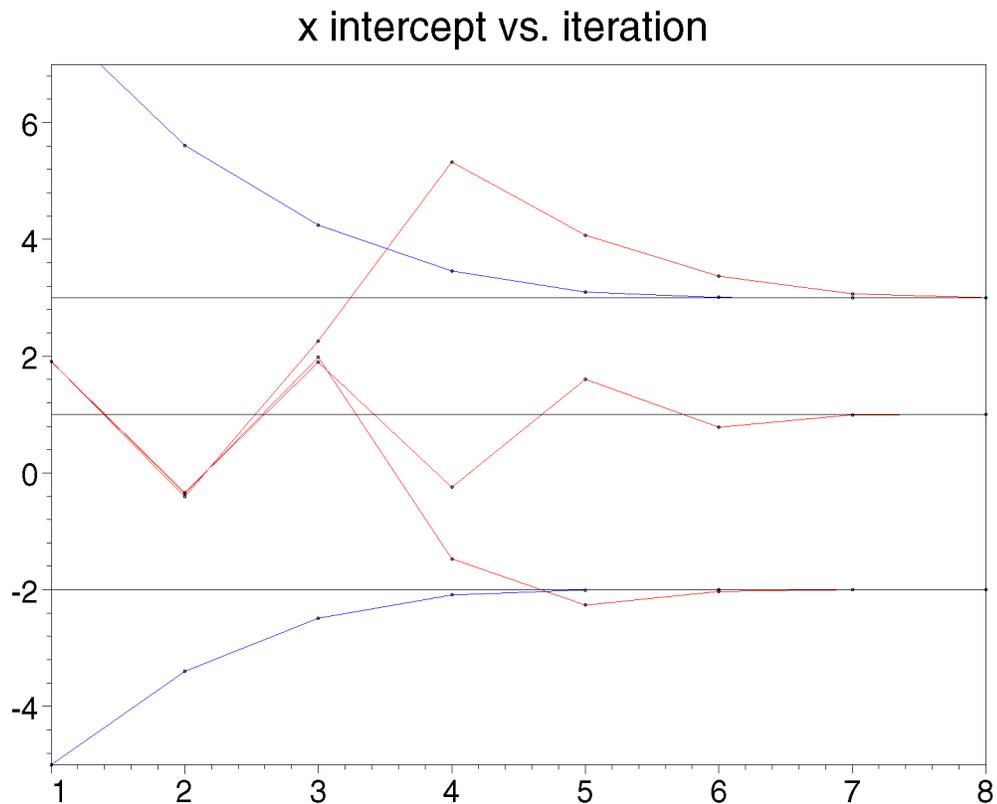
**end**

- A procedure to generate 5 illustrative cases and put them on one plot:

```
doit := (N, a1, a2, a3, a4, a5) → plots display([  
plot([3, -2, 1], color = black), P(N, a1, blue), P(N, a2, blue), P(N, a3, red), P(N, a4, red), P(N, a5, red)  
], view = [1 .. N, -5 .. 7], title = `x intercept vs. iteration`, titlefont = [HELVETICA, 16], axes = box)
```

Okay. Now create the plot.

```
doit(8, 2.2, -1, -.338, -.3375, -.3393)
```



The locations of the roots are shown by the black lines. The iteration starting points were purposely chosen to be perverse "guesses", leading to an artificial need for a higher number of iterations. In practice, one would be a little more careful to start close to a root. Notice (red curves) that the particular solution settled into can sensitively depend on the initial guess.

## 5. Equation Bloat

By 5th order or so, the *general* solutions (i.e., pure symbolics and no numerics) are getting somewhat ugly, even for this simple cubic test case. The "cost" of the 2nd order solution shown in the Examples section is

```
cost(newt(f, b, 2))
```

21 additions + 40 multiplications + 3 divisions

The "cost" of the 5th order solution is

`cost(newt(f, b, 5))`

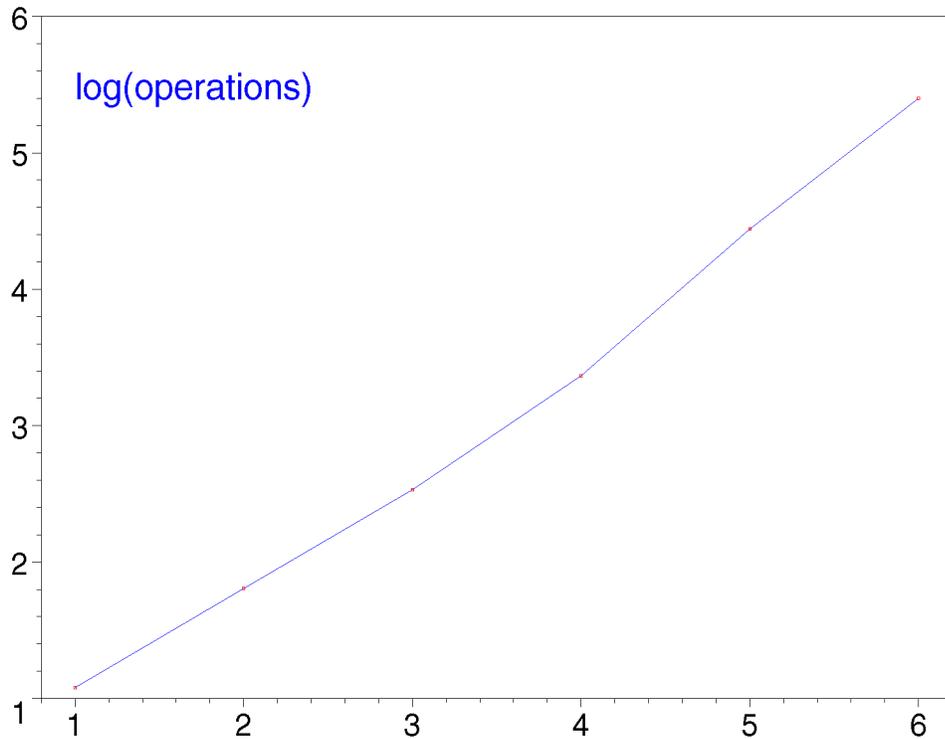
670 additions + 26979 multiplications + 15 divisions

Let's make a plot of the log of the total number of additions, multiplications, and divisions as a function of number of iterations. First we write a procedure to take care of it.

```
costplot := proc(N::posint)
local p, i, j, c, d, plt1, plt2, ylab, fsiz, ymin, ymax;
  p := [ ];
  for i to N do
    c := cost(newt(f, b, i));
    d := 0;
    for j to nops(c) do if nops(op(j, c)) = 1 then d := d + 1 else d := d + op([j, 1], c) fi od;
    p := [op(p), [i, log10(d)]];
  od;
  plt1 := plot(p, color = blue);
  plt2 := plot (color = , style = , symbol = );
  := 14;
  ymin := trunc(p[1][2] );
  ymax := trunc(p[nops(p)] [2]) + 1;
  ylab :=
    plots[textplot]( [ 1.0, ymax - .5, 'log(operations)`, font = [HELVETICA, fsiz ], align = RIGHT);
  plots[display]( [plt1, plt2, ylab ], view = [.8 .. N + .2, ymin .. ymax ],
    axesfont = [HELVETICA, fsiz - 2], axes = box)
end
```

Now create the equation bloat plot.

`costplot(6)`



We see that the equation bloat of the purely symbolic solution grows *exponentially* with the number of iterations. In fact, the slope of the curve is approximately (but not quite)  $m = 1$ , so that with each iteration the operation count mushrooms by a factor of almost 10.