An Approximate Solver for Symbolic Equations

Elisha Sacks
Department of Computer Science
Princeton University
Princeton, NJ 08544, USA

Abstract

This paper describes a program, called NEWTON, that finds approximate symbolic solutions to parameterized equations in one variable. NEWTON derives an initial approximation by solving for the dominant term in the equation, or if this fails, by bisection. It refines this approximation by a symbolic version of Newton's method. It tests whether the first Newton iterate lies closer to the solution than does the initial solution. If so, it returns this iterate; otherwise, it chooses a new initial solution and tries again.

1 Introduction

Research in symbolic equation solving has focused on exact solution methods. The resulting programs, such as MACSYMA [Mathlab Group, 1983] and PRESS [Bundy and Welham, 198l], either return an exact solution or fail. Yet, scientists and engineers routinely must solve problems that have no exact closed-form solution. They need an equation solver that finds adequate approximate solutions to such problems, rather than failing. In fact, they often prefer a simple approximate solution even when a complicated exact solution is available. For example, the equation $x^5 + x — k$ cannot be solved for x in closed form, but $x = k$ is an accurate approximation to the solution for k near 0. One might well prefer this approximation to the full solution of $x^4 + x = k$, which is very long and complicated. This paper describes an equation solver, called NEWTON, that finds approximate solutions to parameterized equations in one unknown.

Numeric analysis provides many algorithms for solving individual equations approximately, but says little about parameterized equations. One could solve numerically for specific parameter values and interpolate the results. This would provide little general understanding and require prohibitive amounts of computation, especially for equations containing multiple parameters. Instead, NEWTON constructs a single parameterized solution, such as $x = k$ above, for all legal parameter values. Users can instantiate a solution with multiple parameter values, rather than rederiving it numerically for each value. They can examine the influence of parameters on a solution analytically instead of by experimentation. For example, the approximate solution $x — k$ increases linearly in k.

2 The Algorithm

NEWTON takes as input a parameterized function /(x), an interval domain for x, and a set of constraints. Each constraint is a strict or nonstrict inequality between real-valued functions of the parameters, for example $k > 1$ and $k^2 < 2^m$. The constraints and $f(x)$ must be *extended elementary functions:* polynomials and compositions of exponentials, logarithms, trigonometric functions, inverse trigonometric functions, and absolute values. NEWTON assumes that $f(x)$ is differentiate and has a single, simple root[1] in the domain of x. I discuss multiple and nonsimple roots in the concluding section.

NEWTON narrows in on the root in stages. First, it brackets the root within an interval on which $f(x)$ is monotonic by the following algorithm:

1. let [a, 6] be the domain of x
2. if $f'(x)$ has a fixed sign on [a, 6] then return [a, 6]
3. let $m=(a + b)/2$; if $f(a)f(m) < 0$ then 6 <- m else $a \leftarrow m$
4. go to step 2.

If a = —oo or b = oo, it chooses a very small or very large finite number instead. It uses the BOUNDER inequality prover [Sacks, 1987a, Sacks, 1987b] to test whether the inequalities in steps 2 and 3 hold for all parameter values that satisfy the input constraints. The simple root assumption ensures that the algorithm terminates. For example, the function $f(x) = x^4+kx-l6$ with parameter k and constraints $— 1 < k < 1$ is monotone on the domain x E [1,10] because $/'(x) = 4x^3 + k > 4 - 1 = 3 > 0$.

Next, NEWTON calculates an initial approximate root, X_0 for $f(x)$. If $f(x)$ is a sum, it finds the addend of maximal absolute value among those containing x, deletes the remaining addends that contain x, and solves the resulting equation.[2] In our example, the addends x^4 and xk contain x with $|x^4| > \backslash kxl$ so NEWTON solves $x^4 - 16 = 0$ to obtain $x_0 = 2$. It rejects $x_0 = -2$ because that root lies outside the domain. NEWTON tries to calculate x_0 for general equations by reducing them

[1]A root r is simple if $f(r) = 0$ and $f'(r)$ ^ 0.
[2]McAllester [McAllester, 198l] uses a similar strategy.

to sums, as shown in Table 1. These reductions suffice for all rational functions and for all the examples in this paper. If NEWTON cannot reduce a function or cannot solve a reduced equation, it sets xo to be the middle of the monotone domain of x. One such case is f(x) = x* - 2.

function	reduced function
$g_1(x)\cdots g_n(x)$	$g_i(x)$ for $i = 1, \ldots, n$
$[g(x)]^p$	if $p > 0$ then $g(x)$ else fail
$h(g(x))$	if h invertible then $g(x) - h^{-1}(0)$ else fail

Table 1: Reduction rules for calculating x_o.

NEWTON tries to improve upon x_0 by applying Newton's method symbolically. It tests whether the first Newton iterate

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is closer than x_0 to the root, r, of $f(x)$, that is whether

$$|x_0 - r| > |x_1 - r|. \tag{1}$$

It must test this condition indirectly because r is unknown. If the test succeeds, NEWTON assumes that x_1 approximates r sufficiently well and returns x_1. This assumption normally works well. In our example,

$$x_1 = \frac{64}{32 + k}$$

has a relative error of at most 0.0015 for all k that satisfy the input constraint. A more sophisticated implementation could check x_1 against an accurate numeric solution for several parameter values and calculate further iterates when x_1 proved inadequate. The stopping condition would involve a tradeoff between the accuracy and the complexity of the approximate solution, both of which grow with each iteration.

To understand how NEWTON tests x_1, suppose first that /(x) increases monotonically and that $x_o > r$. Figure 1 depicts the geometric interpretation of Newton's method in this situation: x_1 is the intersection point between the tangent to /(x) at X_o and the x axis. As /'(xo) decreases from oo to 0, x_1 decreases from x_o to —oo via r. Any $x_1 > r$ satisfies equation (1), as Figure la illustrates. If $x_1 < r$ (Figure lb), equation (1) becomes

$$x_0 - r > r - x_1 = r - x_0 + \frac{f(x_0)}{f'(x_0)}. \tag{2}$$

By the mean value theorem, $f(x_0) = f'(\xi)(x_0 - r)$ for some ξ in (r, x_0). Substituting into equation (2) and rearranging yields $f'(\xi) < 2f'(x_0)$. A sufficient condition for this inequality is that $f'(x) < 2f'(x_0)$ for all x in $[a, x_0]$. This condition holds in our example:

$$\begin{aligned} f'(x) - 2f'(x_0) &= 4x^3 + k - 2(32 + k) \\ &= 4x^3 - k - 64 \leq 4(2^3) + 1 - 64 \\ &= -31 < 0 \end{aligned}$$

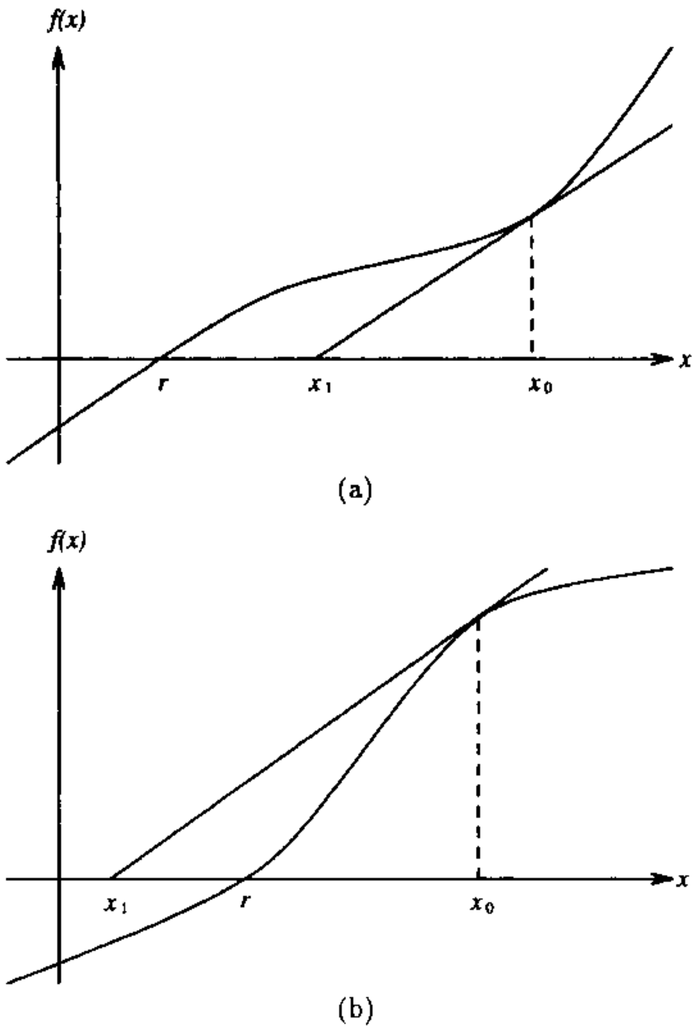

Figure 1: Geometric interpretation of Newton's method: (a) $r \leq x_1 < x_0$ (b) $x_1 < r < x_0$.

```
if $f(x_0)f(x_1) \geq 0$ {$x_1$ is between $x_0$ and $r$}
  then succeed
  else if $f'(x) > 0$ on $[a, b]$
         then if $f(x_0) > 0$
                then test $f'(x) < 2f'(x_0)$ for $x$ in $[a, x_0]$
                else test $f'(x) < 2f'(x_0)$ for $x$ in $[x_0, b]$
         else if $f(x_0) < 0$
                then test $f'(x) > 2f'(x_0)$ for $x$ in $[a, x_0]$
                else test $f'(x) > 2f'(x_0)$ for $x$ in $[x_0, b]$
```

Figure 2: A sufficient condition for $|x_0 - r| > |x_1 - r|$ on a monotone interval $[a, b]$.

for x in [1,2]. The cases of $x_0 < r$ and of /(x) monotone decreasing are analogous. The complete test for equation (1) appears in Figure 2. NEWTON resolves the inequalities with BOUNDER as before.

NEWTON tests progressively narrower intervals until it succeeds or the interval shrinks beneath a prespecified width. It starts with the monotone interval [a, 6] and repeatedly bisects the distance from X_0 to each current boundary. If the initial guess really is a good candidate for iteration, the shrinking should eventually make the test work, at which point NEWTON returns the first iterate. If not, NEWTON must choose a different value for x_0 by bisection and start over. If the test had failed in the example, the next test interval would have been [1.5,6], splitting the distance from the ends of [1,10] to $X_0 = 2$.

3 Examples

As another example, consider $f(x) = e^x + x - k$ with domain $x \in [0,\infty)$ and constraint $k \geq 1$. The function increases monotonically for all x because $f'(x) = e^x + 1 > 0$. It has a unique root because $f(0) = 1 - k \leq 0$ and $f(k) = e^k > 0$. Approximation is inevitable because the root cannot be expressed in closed form. NEWTON determines that $|e^x| > |x|$ and solves $e^x - k = 0$ to obtain $x_0 = \log k$. It verifies the convergence condition (Figure 2) and returns

$$x_1 = \frac{k \log k}{k+1}.$$

Table 2 demonstrates that the relative error of x_1 is less than 0.05 and drops rapidly as k increases.

k	x_1	root	% error
1	0.000	0.000	0.000
2	0.462	0.442	0.045
4	1.109	1.073	0.033
8	1.848	1.821	0.015
16	2.609	2.596	0.005
32	3.361	3.355	0.002
64	4.095	4.093	0.000

Table 2: Relative error between the root of $e^x + x - k$ and $x_1 = k \log k/(k+1)$.

Given the domain $x \in (-\infty, -1]$ and no constraints on k, NEWTON determines that $|x| > |e^x|$, obtains $x_0 = k$, verifies convergence, and returns

$$x_1 = k - \frac{e^k}{e^k + 1}.$$

For $f(x)$ to have a root, k must satisfy

$$k = e^x + x \leq e^{-1} - 1 \approx -0.632.$$

The relative error of x_1 is 0.2 for this value, 0.007 for $k = -1$, and less than 0.001 for $k < -2$.

The dominance argument only requires the sign of k: $k > 0$ implies $|e^x| > |x|$, whereas $k < 0$ implies $|x| > |e^x|$. NEWTON needs stronger constraints, such as those above, to derive dominance. Given only the sign of k, it must calculate x_0 by bisection. It can, however, prove convergence and obtain x_1 as before.

The previous examples are all *separable,* that is expressible in the form /(x,ik) = *g(x)* + *h(k).* The final example,

$$f(x) = x^2 + \log(x+k) - k^2$$

with $x \in (0,\infty)$ and $k \geq 0.5$ demonstrates that NEWTON handles nonseparable functions as well. The function increases monotonically and has a unique root. The convergence condition holds for all *k.* Given the modest constraint $k < e^{x^2} - x$, NEWTON can calculate $x_0 = k$ and

$$x_1 = k - \frac{2k \log 2k}{4k^2 + 1}.$$

The relative error of x_1 is 0 for *k* = 0.5, 0.048 for *k* — 1, and less than 0.002 for *k* > 4. One can obtain even better results with a more sophisticated choice of x_0, setting x = 0 in log(x+Ar), the dominated term of f (x), and solving the resulting equation to obtain $x_0' = \sqrt{k^2 - \log k}$. NEWTON does not implement this strategy. Newton's method appears to work for 0 < *k* < 0.5, even though the convergence condition fails. A more experimental implementation could validate x_1 empirically on a large number of sample parameter values, rather than give up.

Table 3 summarizes the examples in this paper.

4 Conclusions

This paper describes NEWTON, an equation-solver that finds approximate roots to parameterized functions by a symbolic version of Newton's method. If this fails, it turns to a symbolic bisection algorithm. NEWTON requires that the function have a single, simple root in its domain. One can handle a nonsimple root, r, of /(x) by applying NEWTON to /i(x) = /(x)//'(x), which has a simple root at r. One can find multiple roots by partitioning the domain into intervals on which /(x) is monotone, hence has at most one root, and applying NEWTON to each subinterval [a, 6] for which *f(a)f(b)* < 0. I describe a program that calculates the monotone intervals elsewhere [Sacks, 1985].

NEWTON can only find a root r of /(x) for which the sign of /'(r) is fixed for all legal parameter values. Otherwise, it can neither bracket the root within a monotone domain nor prove that Newton's method converges. For example, the function /(x) = (ax - \)e^x has a unique root of *l/a* for a not= 0, but /'(1/a) = *ae^l/a* can be positive or negative. A possible solution is to partition the space of legal parameter values into regions on which f'(x) has a fixed sign and solve for each subset separately. In our example, one can solve for *a* > 0 and for *a* < 0. Partition algorithms exist for polynomials [Arnon *et a*l., 1984]; partitioning other functions is a topic for future research.

The strategy of applying numeric techniques symbolically to parameterized problems has many other applications. Newton's method generalizes to vector equations. Additional numeric techniques, such as the secant method and fixed-point iteration [Press *et a*l., 1987], also extend directly to parameterized equations. Other domains for symbolic application of numeric algorithms include Markov theory (which I have worked on with

$f(x)$	constraints	x_1	% error
$x^4 + kx - 16$	$1 \leq x \leq 10; \lvert k \rvert \leq 1$	$\frac{64}{32+k}$	0.002
$e^x + x - k$	$x \geq 0; k \geq 1$	$\frac{k \log k}{k+1}$	0.045
$e^x + x - k$	$x \leq -1; k \leq -1$	$k - \frac{e^k}{e^k+1}$	0.007
$x^2 + \log(x+k) - k^2$	$x \geq 0; k \geq 0.5$	$k - \frac{2k \log 2k}{4k^2+1}$	0.048

Table 3: Summary of examples: x_1 is the approximate root of $f(x)$.

J. Doyle [Doyle and Sacks, 1989]), integration, and dynamic systems theory.

Acknowledgements

Harry Mynick of the Princeton Plasma Physics Laboratory motivated me to develop NEWTON. Forman Acton, Woody Bledsoe, Joel Friedman, Dick Lipton, Harry Mynick, and especially Mike Wellman provided valuable comments on this paper.

References

[Anion *et ai,* 1984] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: the basic algorithm. *SI AM Journal of Computtng,* 13(4):865-877, 1984.

[Bundy and Welham, 1981] Alan Bundy and Bob Welham. Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence,* 16(2):189-211, May 1981.

[Doyle and Sacks, 1989] Jon Doyle and Elisha P. Sacks. Stochastic analysis of qualitative dynamics. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* 1989.

[Mathlab Group, 1983] Mathlab Group. *Macsyma Reference Manual* MIT Laboratory for Computer Science, Cambridge, Mass., version 10 edition, January 1983.

[McAllester, 1981] David A. McAllester. Algebraic approximation. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* pages 1024-1026, 1981.

[Press *et ai,* 1987] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes.* Cambridge University Press, Cambridge, England, 1987.

[Sacks, 1985] Elisha P. Sacks. Qualitative mathematical reasoning. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* pages 137-139, 1985.

[Sacks, 1987a] Elisha P. Sacks. Hierarchical inequality reasoning. TM 312, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, 1987.

[Sacks, 1987b] Elisha P. Sacks. Hierarchical reasoning about inequalities. In *Proceedings of the National Conference on Artificial Intelligence,* pages 649-654. American Association for Artificial Intelligence, 1987.