Partial inverse heuristic for the approximate solution of non-linear equations

Gaston H. Gonnet and Allan Bonadio
Informatik E.T.H. Zurich, Switzerland
and
Waterloo Maple, San Francisco

Abstract

We show how to generate many fix-point iterators of the form $x_{i+1} = F(x_i)$ which could solve a given non-linear equation. In particular, these iterators tend to have good global convergence, and we show examples whereby obscure solutions can be discovered. Also, a systematic method for finding most or all solutions to nonlinear equations that have multiple solutions is described. The most successful iterators are constructed to have a small number of occurrences of x_i in F. We use grouping of polynomial terms and expressions in x, e^x and $\ln x$ using known inverse relations to obtain better iterators. Each iterator is tried in a limited way, in the expectation that at least one of them will succeed. This heuristic shows a very good behaviour in most cases, in particular when the answer involves extreme ranges.

1 Introduction

Problem definition. Solving non-linear equations is a classical topic in numerical analysis [6, 7, 9, 11, 1]. Many algorithms are well-known and well-studied. Few algorithms give any guarantee that they can find all the roots in a given interval, and the ones which do [10] depend on bounding higher order derivatives. It is fair to say that the behaviour of algorithms under convergence is well understood, and most of them are very efficient. In the absence of convergence, however, the algorithms usually perform a useless walk over the valid domain.

In our experience, the biggest challenge for a zero-finder is this focusing on a convergence area, and not on the performance during convergence. We will use the term *random walk* to describe the sequence of values before the convergence criteria are met, and *refinement* to describe the iterations performed under convergence. In this paper we suggest the use of fixed point iterators to find the first approximations to roots of non-linear equations. These approximate roots may be refined further with other methods.

Convergence, divergence, random walk, invalid domains and slow convergence. Most of the literature is concerned with the refinement phase, which in our experience is the least expensive one. The random walk phase is characterized by one or more of the following scenarios:

(a) *divergence.* The values grow steadily farther from any possible solution. E.g. xe^x when tried on an $x_0 \ll -1$.

(b) *iterating outside valid domains.* This may happen when the user specifies a domain for the solution, e.g. that the solutions be real, or in a given interval, and the computation returns a complex number or a number outside the given interval. Exponent overflow/underflow, division by zero, or other domain errors such as $\arcsin(x)$ where $|x| > 1$ are also symptoms of the same problem.

(c) *convergence to a non-root.* This is the case where the iterator mistakenly converges to a value which is not a root. A good example is the case of $1/x$ for bisection search starting with a positive and a negative value.

(d) *hopelessly slow convergence.* The convergence achieved by an iterator showing linear convergence with a constant very close to 1. Alternatively, showing just linear convergence in a range which is too large. A simple example is using Newton-Raphson on $x^{103} = 0$. For another example, the behaviour shown by $x^{1.001} - x \ln x = 0$ with starting values $x = 10$ and $x = 10^{5000}$ (the solution is $x = 0.7941... \times 10^{3960}$). This problem requires 16609 bisection steps to find the first significant digit of the answer.

(e) *pseudo-stable oscillation.* The iterator gives values belonging to two or more sets in a cyclic sequence. For example, $|x| + 10 - 20e^{-x^2}$ has two zeros at $x = \pm 0.785812...$, but if Newton's method is used, and the iteration is started on an $|x| > 1.911$, the iteration will eventually alternate steadily between $+10$ and -10.

(f) *incomplete multiple solution.* This is a situation where there are many roots, but the rootfinder finds only one of them, or only a few. Most algorithms involve a process where roots are eliminated by algebraic deflation. This leads to an N^2 algorithm as the expression to zero becomes more complicated. For example, $\sin(x) = x/100$ has 63 roots on the real axis.

Witness problems and new algorithms. Usually, advances in most disciplines happen whenever we are confronted with new problems which are not solved by the old theories. In this vein, we present four examples which present serious problems to all systems, in particular to the existing algorithms used in Maple's `fsolve`[2] function.

(1)

$$x^{1.001} - x \ln x = 0$$

for positive x, this equation has two solutions, one small, the other one huge in magnitude.

$$x = 2.721005..., \quad 0.794138... \times 10^{3960}$$

The problem here is to find the large positive root, which was the one wanted by the user who proposed the problem. The large magnitude of the root will make all methods based on classical iterations too slow to converge.

(2)

$$\frac{\sin^{-1} x - \tan x}{x^4} = 0$$

This problem and the next were proposed by Kahan[8]. For positive x, this equation has one solution, $x = 0.9999060124...$. This value is too close to 1, the limit where the function can be evaluated without going into the complex plane. Many algorithms are likely to overshoot (any algorithm based on the derivative or on estimating the derivative is a candidate to overshoot) and will try a value outside the valid range. In this case, the computation is likely to be aborted. To hit the area of convergence without going into values greater than 1 for Newton's method, we would have to start in the range $0.99964 \le x < 1$. This means that an equally spaced grid with at least 2800 values would have to be used.

(3)

$$x^2 + 5 + \ln(|x - \pi|) = 0$$

For positive x, this equation has two roots very close to π, $x = 3.1415923...$ and $x = 3.1415930....$ The difference between each root and π is about 0.348×10^{-6}. The convergence interval for such roots is extremely narrow, and most methods, even when started close to the roots, will diverge away from them. If we would use Newton's iteration, it would converge only if the initial x is within 0.2575×10^{-5} from π. If these initial points were sampled from a uniform grid between 0 and 10, we would need more than 3.8 million points. Graphing software will be often mislead and fail to point these roots.

(4)

$$sin(x) = x/100$$

This equation has 63 real roots in the interval $-96.1 < x < 96.1$, and close-call extrema near $x = \pm 102.4$. The roots come in pairs, each separated by 0.5 to 3.14, closer as you go away from zero. Root pairs occur approximately every 2π. In other words, the roots are systematically spaced; the problem is their multitide. Any good iterator will easily converge to any root given a suitable starting guess. The trick is to find all the roots in a systematic way.

Starting values. When the user does not supply a starting value but supplies a range (or a range is implicit), selecting the starting value is a problem in itself. Ideally we would like to try several starting points in the valid interval, because the iterator may fail for any of the reasons listed above. This suggests the following strategy for all iterators.

```
starting_points := {};
x[0] := most_isolated_valid_point( starting_points );
for i do
    x[i] := F(x[i-1]);
    starting_points := starting_points union {x[i-1]};
    if computation_failed or outside_domain(x[i]) then
         start_again
    else if convergence_achieved then
         successful_finish
    else if i>3 and diverging then
         start_again
    else if acceleration_possible then
         x[i] := acceleration(x[i], x[i-1], ...)
    end if
end do;
```

2 Algorithm of partial inversion

General fixed point iterations. An iterator of the form $x_{i+1} = F(x_i)$ is called a fixed point iterator. The solution of the equation is the fixed point of F. The conditions for convergence are quite straightforward, since a root α of the equation implies $f(\alpha) = 0$, and $\alpha = F(\alpha)$, then

$$x_{i+1} - \alpha = \varepsilon_{i+1} = F(x_i) - F(\alpha) = F'(\xi)\varepsilon_i$$

when ξ is in the interval (x_i, α). So for x where $|F'(x)| < 1$ we have convergence to a root of $f(x)$. Iterators with order of convergence γ satisfy the following asymptotic equation on their errors

$$\varepsilon_{i+1} = O(\varepsilon_i^\gamma)$$

and as a natural consequence, $F'(\alpha) = ... = F^{\gamma-1}(\alpha) = 0$ for integer γ.

Let $\#_x(f)$ be the number of occurrences of the variable x in the algebraic expression for $f(x)$. We will not take into account simplification for the time being, we will just consider the number of occurrences as they appear in the expression as it is written. E.g. $\#_x(x(\sin x - 1)) = 2$, whereas $\#_x(x \sin x - x) = 3$ Of course, any equation for which $\#_x(f) = 1$ is trivial to solve by algebraic isolation, if we know the inverses of all the functions used, even if there are multivalue issues. The cases where $\#_x(f) = 1$ are considered trivial and we will now study the case of $\#_x(f) > 1$ exclusively.

It is easy to see, that if $\#_x(f) = k$, then by algebraic isolation of each of the occurrences of x we can usually compute k iterators of the form $x = F(x)$ derived from $f(x) = 0$. Except for multivalued functions or principal values, all of these iterators will have the same solution set as the original[1]. At this point we will not perform any simplification or any additional manipulation to obtain the iterators. For each occurrence of x we simply isolate it. For example, from $x^{1.001} - x\ln x = 0$ we obtain 3 iterators, namely

$$\begin{aligned} x^{1.001} - x\ln x = 0 \quad &\Longrightarrow \quad x = (x\ln x)^{1/1.001} \\ &\Longrightarrow \quad x = \frac{x^{1.001}}{\ln x} \\ &\Longrightarrow \quad x = e^{x^{1.001}/x} \end{aligned}$$

and we intentionally did not simplify any of the resulting iterators. As we will see in the next section, it is, in general, possible to obtain more iterators from one equation.

We are now ready to describe the heuristic algorithm. For $f(x) = 0$ we compute all the iterators generated by isolation, $F_1(x)$, $F_2(x)$, ..., $F_k(x)$. For each initial value x_0 we try all the iterators a limited number of times. This is done according to the following procedure

```
for x[0] in set_of_initial_values do
    for m = 1 to k do
        for i = 1 to maximum_iterations do
            x[i] := F[m](x[i-1]);
            if computation_failed or outside_domain(x[i])
                 then next m
            else if convergence_achieved
                 then return x[i]
            else if i>3 and diverging
                 then break
```

[1]some inverse functions will have to be considered multivalued, and each choice of branch could give a solution. E.g. $\sin x = 1/2$ has infinitely many solutions. Alternatively, inverses of multivalued functions may give solutions which do not match their principal value. E.g. the iterator for $\sin^{-1} x = \pi$ would be $x = \sin\pi = 0$ which is not a solution. Section 4 describes how to solve these problems.

```
            else if acceleration_possible
                 then x[i] := acceleration(x[i],x[i-1],...)
            end if
        end for i
    end for m
end for set_of_initial_values;
```

The idea behind this heuristic is that while it is extremely improbable that all of the iterators will converge, it is also improbable that all of them will fail to converge. By keeping the cost of running each one low, we can try all of them, expecting at least one to succeed. In practice, this heuristic is highly successful even for contrived, ill-behaved examples.

Evaluation by simulation.

> "When you can measure what you are speaking about and express it in numbers you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind." *Lord Kelvin, 1883.*

In this vein, we have attempted to measure the behaviour of the iterators by simulation. We randomly generate equations to be solved. This is done top down, generating for each node of the expression tree a random operator. The number of occurrences of x is kept between 2 and 10, various functions are given different probabilities, the equation is forced to be a sum of various terms at the top level and polynomials are discarded. The following are 3 examples of such equations

$$x - 3 + \cosh^{-1}(-3 - 7x - 3x^2) = 0$$

$$\left((-8 + 3x - 3x^3 - 3x^4)^4 + \frac{e^{8-4x^4}}{83521} - 15\right)^4 - 7 + x = 0$$

$$x + 1 + 10x^2 + (-11)^{(11+5/2x)} + \frac{8}{7x(5 - \ln(x-7)/9)} = 0$$

Within the limits of the biases (unknown to us) that this random set of equations may impose, we evaluate the iterators by running and measuring them against these equations.

66644 random equations producing 162314 iterators			
method		failures	
converged to a root	69.50%	converged to a non-root	3.08%
failed	30.50%	diverged	31.85%
average time	2.639	outside domain	43.44%
time per root	3.797	too many iterations	9.59%
		iterator fails	.01%
iterators per equation	2.436	no iterators	12.03%

Table 1: Simple iterators generated by isolating an x to the lhs of the equation

For the simple iterators (not included in the groups in the next section) table 1 shows the results of extensive simulations. It is quite remarkable that about 70% of the equations are solved. This compares extremely favourably to the results for Newton's method shown in table 7.

3 Convergence of partial inverses

The question of convergence of the different iterators can be partially answered. For functions with two occurrences of the unknown, $\#_x(f) = 2$, in general either one or the other derived iterator converges. This is proven as follows.

Let $f(x_1, x_2) = 0$ be the equation to be solved, identifying each occurrence of x by x_1 and x_2. Let $F_i(x)$ be the iterator derived from inverting f on x_i. Formally this means that

$$f(F_1(x), x) = 0$$

and

$$f(x, F_2(x)) = 0$$

Let α be a root of f, $f(\alpha, \alpha) = 0$. The convergence of the iterators depends on $|F_i'(\alpha)| < 1$. By computing derivatives with respect to x of the above identities, we find

$$f_1'(\alpha, \alpha)F_1'(\alpha) + f_2'(\alpha, \alpha) = 0$$

$$f_1'(\alpha, \alpha) + f_2'(\alpha, \alpha)F_2'(\alpha) = 0$$

For this system of equations to have a non-null solution in $f'_i(\alpha,\alpha)$, its determinant must be zero, or

$$\begin{vmatrix} F'_1(\alpha) & 1 \\ 1 & F'_2(\alpha) \end{vmatrix} = F'_1(\alpha)F'_2(\alpha) - 1 = 0$$

and from this

$$|F'_1(\alpha)| = 1/|F'_2(\alpha)|$$

So unless both absolute values are equal to one, then one iterator converges while the other diverges. This is a very encouraging result, it guarantees that one of the iterators will succeed for *each* of the roots.

For $\#_x(f) = k > 2$ the results are much weaker. For general x, the condition becomes

$$\begin{vmatrix} F'_1(\alpha) & 1 & \cdots & 1 \\ 1 & F'_2(\alpha) & \cdots & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \cdots & F'_k(\alpha) \end{vmatrix} = \left(1 + \sum_{i=1}^{k} \frac{1}{F'_i(\alpha) - 1}\right) \prod_{i=1}^{k} (F'_i(\alpha) - 1) = 0$$

Then, if $k-1$ of the values $F'_i(\alpha) > 1$, the remaining one must be between 0 and 1 and hence the corresponding iterator converges. This does not guarantee convergence for every case, as all $F'_i(\alpha) = 1 - k$, i.e. $|F'_i(\alpha)| = k - 1 > 1$ is also a possible solution and in this case all iterators will diverge.

The above theorem for two occurrences, strongly suggests that grouping variables (if possible) while building the iterators, is a good strategy.

4 Building additional iterators

It is easy to see that we can create an infinite number of iterators. For instance, for any iterator $x = F(x)$, and arbitrary a,

$$x = \frac{F(x) + ax}{a + 1}$$

will also be fixed point iterator giving the same solution set for any $a \neq -1$.

A natural question to ask is, which iterator is better? This question is nontrivial to answer, and contradicts the spirit of the heuristic, which is to try all of them. We would like to try a large number of formulas and not conjecture which is the best one and use it alone. Yet an intuitive and very economic measure of the quality of the iterators is $\#_x(F(x))$, the number of occurrences of x on the right-hand side of the iterator. A smaller $\#_x(F)$ is considered better. The degenerate case is a good example. If $\#_x(F) = 0$, we have an explicit expression for the root. Using iterators which have the minimal $\#_x(F)$ guarantees, to a certain degree, that we will not be generating more complicated trivial iterators.

If at any point during our isolation or after trial simplification, the resulting equation has a smaller number of occurrences of x, we will derive all possible iterators arising from this shortened equation. For example,

$$\begin{aligned} x \ln x &= x^{1.001} \\ \ln x &= \frac{x^{1.001}}{x} \\ \ln x &= x^{0.001} \end{aligned}$$

The grouping of powers caused a simplification and from this equation we obtain two iterators

$$\begin{aligned} x &= e^{x^{0.001}} \\ x &= (\ln x)^{1000} \end{aligned}$$

The first one was obtained before but the second one is new and in this case it is useful to find the largest of the roots of the original equation.

The use of $\#_x(F)$ as a measure of iterator quality seems to work in practice, although we cannot prove this generally. Certainly there are examples of two iterators for the same equation, where $\#_x(F_1) < \#_x(F_2)$, and F_1 behaves worse than F_2.

Polynomial iterators. One of the ways in which we can create new, and hopefully better, iterators is by isolating subexpressions which are easy to invert, instead of isolating x. For example, the equation $6x^2-9x+6-\ln(8+x)$ generates the iterators

$$x = \frac{6x^2 + 6 - \ln(8 + x)}{9}$$

$$
\begin{aligned}
&= \pm\frac{\sqrt{-4+6x+2\ln(8+x)/3}}{2} \\
&= e^{6x^2-9x+6}-8
\end{aligned}
$$

For the initial value $x_0 = 1.23$, all these iterators fail. The first one and the third one grow without bound, the positive second iterator converges too slowly and the negative oscillates between one value and its conjugate. For this equation we could instead isolate the polynomial part

$$6x^2 - 9x + 6 = \ln(8+x)$$

solve it treating the $\ln(8+x)$ as a constant, and obtain two new iterators.

$$x = \frac{9 \pm \sqrt{24\ln(8+x) - 63}}{12}$$

Notice that these new iterators have only one occurrence of x on the rhs, compared to two for the previous ones. The positive choice for this iterator is successful and converges very quickly to $x = 0.7595... + 0.2753...i$. This is not a general rule, we could certainly find examples which show the contrary, but in general it is expected that such groupings will produce better iterators as they capture more information about the inverse.

So we will use the following rule, for each iterator $F(x)$ which is a sum of a polynomial part and a non polynomial part, i.e.

$$x = F(x) = G(x) + p(x)$$

we will produce an iterator which is $x - p(x) = G(x)$ and each iteration consists of evaluating $\beta = G(x)$ and then solving the polynomial $x - p(x) - \beta = 0$. Removing the polynomial part will decrease the number of occurrences of x on the rhs, and hence improve our measure of quality of the iterators. The iterator can be written using Maple's `RootOf`[2] notation as

$$x_{i+1} = \texttt{RootOf}(G(x_i) + p(z) - z = 0, z)$$

A polynomial iterator, with degree d on its lhs, will normally have d roots. This has a cardinality advantage over handling a polynomial iterator by its

[2]The notation $\texttt{RootOf}(F = 0, x)$ stands for the values of x which make $F(x) = 0$. This is Maple's mechanism to represent algebraic numbers, algebraic functions or any root of an equation.

individual terms. If the individual terms of the polynomial are isolated one by one, and for each monomial x^k, k roots have to be considered, up to $d(d+1)/2$ iterators could be generated. The drawback is obvious, instead of having a closed form solution for the iterator, we have to solve a polynomial of degree d, and this is typically an iterative process for $d > 2$.

Selecting which root of the polynomial to use is non trivial. When roots are given by a formula, e.g. for $d = 2$, then it is easy to write two iterators, one for each root. But when the roots are found numerically, it is not possible to have a consistent way of using the same root on successive iterations. Actually, the term "same root" does not even make sense for different polynomials, it only makes sense when the roots can be expressed as functions of the coefficients. On the other hand, if the polynomial iterator is converging, we want to select the root which is closest to the iteration value. I.e.

$$x_{i+1} = \texttt{RootOf}(p(z) - F(x_i), z)$$

and we select the root which is closest to x_i. Since this selection is necessary for convergence, we will use it in every step of the algorithm. Furthermore, many iterative methods for solving polynomials can profit significantly from starting their iterators at an approximation of the root; the correct root could come almost for free. For example, for the solution of the equation

$$x^5 - 6x^4 + 5x^3 - 7x^2 \ln x + 6x - 5 = 0$$

we define the iterator

$$x_{i+1} = \texttt{RootOf}(z^5 - 6z^4 + 5z^3 + 6z - 5 = 7x_i^2 \ln x_i, z)$$

If we start with $x_0 = 1$, then the rounded roots of the polynomial are

$$-0.526224 \pm 0.844348i, \quad 0.758845, \quad 1.34521, \quad 4.94839$$

The root which is closest to $x_0 = 1$ is 0.758845... Hence we set $x_1 =$ 0.758845.... For the second iteration the polynomial to solve, is $z^5 - 6z^4 + 5z^3 + 6z - 5 = -1.11236...$ The roots of this polynomial are

$$-0.489123 \pm 0.829261i, \quad 0.586566, \quad 1.44569, \quad 4.94599$$

and the root closest to x_1 is 0.586566... which becomes x_2. This process converges quite quickly, $x_3 = 0.561733...$, $x_4 = 0.563302...$, $x_5 = 0.563168...$, etc. This root selection can also be treated as a multiple value function problem, which is discussed in the next section. Table 2 shows the simulation results for polynomial lhs iterators. Almost 60% of the iterators yield a root, in time which is of the same order of magnitude as for the simple iterators.

66644 random equations producing 59428 iterators			
method		failures	
converged to a root	44.64%	converged to a non-root	1.67%
failed	55.36%	diverged	21.22%
average time	1.593	outside domain	18.06%
time per root	3.569	too many iterations	4.79%
		iterator fails	0.00%
iterators per equation	.892	no iterators	54.26%

Table 2: Iterators which have a non-linear polynomial in their lhs

Substitute all non-polynomials. Another possible way of constructing an iterator is to inspect the equation top-down and force the evaluation of any subexpression which is a special function, a power to a non-integer exponent, or anything else that is not algebraic. If after this is done, we are left with an expression containing x, this must be a rational polynomial expression. The solution of this is equivalent to the solution of a polynomial and we can proceed as with the previous iterators. For example

$$\frac{x-\sin x}{1-x^2/2}=\frac{\tan x - x}{1+x^2/2}$$

for $x_0 = 1.23$ would be handled by evaluating the $\sin x$ and $\tan x$, obtaining

$$\frac{x-0.94248...}{1-x^2/2}=\frac{2.8198...-x}{1+x^2/2}$$

which has the roots $-3.3331...$, $1.2024....$ From these possible values for x_1 we select, as before, the closest to x_0, $x_1 = 1.2024....$ More formally, using Maple's `RootOf` notation, the iterator becomes

$$x_{i+1} = \texttt{RootOf}\left(\frac{z-\sin x_i}{1-z^2/2}=\frac{\tan x_i - z}{1+z^2/2}, z\right)$$

Table 3 shows the simulation results for this heuristic.

Transcendental Iterators. The technique of explicitly finding roots of polynomials can be extended to cases involving transcendental functions with multivalued inverse functions.

66644 random equations producing 14398 iterators			
method		failures	
converged to a root	11.64%	converged to a non-root	.39%
failed	88.36%	diverged	6.22%
average time	.672	outside domain	3.51%
time per root	5.768	too many iterations	1.15%
		iterator fails	0.00%
iterators per equation	.216	no iterators	88.73%

Table 3: Rational polynomial iterators, obtained by solving all the transcendental subexpressions first

Consider the equation $\sin x = x/100$. Providing for all solutions of $\sin x$ yields three iterators:

$$\begin{aligned} x &= 100\sin x \\ x &= \sin^{-1}(x/100) + 2\pi n \\ x &= \pi - \sin^{-1}(x/100) + 2\pi n \end{aligned}$$

where n iterates over all of the integers and the arcsin returns the principal branch. The first iterator diverges chaotically. The other two, however, either converge to solutions or else fail due to the arcsin evaluating to a complex number ($|x/100| > 1$).

Furthermore, all solutions can be generated by simply using different values of n. For the first equation, values $-15 \leq n \leq 15$ will yield the even numbered roots, whereas values $-16 \leq n \leq 15$ will yield the remaining roots when used in the second equation. Other values of n diverge. (The asymmetry comes from the positive sign on the first π; the iterator $x = -\pi - \sin^{-1}(x/100) + 2\pi n$ would supply the same set of roots with $-15 \leq n \leq 16$). The solution obtained is unrelated to the initial guess used; almost any starting value acceptable to the arcsin converges to the same root for a given choice of iterator and n.

Other special cases. The technique used for polynomials can be extended to other cases, when we know how to solve some transcendental equations.

In Maple, the function $W(x)$[2, 4, 3] is the solution of $W(x)e^{W(x)} = x$. Hence many combinations of powers of x with e^x and with $\ln x$ are invertible. The resulting iterators have a lower number of occurrences of x on the rhs, and are considered better. Some of them are shown in table 4.

equation	iterator
$x^a e^{bx} = F(x)$	$x = \frac{aW\left(\frac{bF(x)^{1/a}}{a}\right)}{b}$
$a \ln x + bx = F(x)$	$\frac{aW(b\frac{e^{F(x)/a}}{a})}{b}$
$x^x = F(x)$	$e^{W(\ln F(x))}$
$x \ln x = F(x)$	$e^{W(F(x))}$

Table 4: Inverting expressions with the aid of the $W(x)$ function

66644 random equations producing 3174 iterators			
method		failures	
converged to a root	1.83%	converged to a non-root	.07%
failed	98.17%	diverged	.80%
average time	.128	outside domain	1.14%
time per root	6.985	too many iterations	.20%
		iterator fails	0.00%
iterators per equation	.048	no iterators	97.79%

Table 5: Iterators generated by solving a lhs involving $\ln x$, e^x and x in terms of $W(x)$

Table 5 shows the simulation results for lhs which can be resolved with the $W(x)$ function. The iterators generated in this way are also very successful in producing roots, just as the non-linear polynomials. In this case however, only 5% of the equations can produce such iterators, and their relative usefulness is decreased. The $W(x)$ function accepts a branch indicator, then as before we can compute an arbitrarily large number of iterators which will normally converge to different roots.

Remove linear term heuristic. At the beginning of this section we saw that an arbitrary iterator $x = \frac{F(x)+ax}{a+1}$ can be constructed for any $a \neq -1$.

For a general iterator, if $F'(\alpha) = 0$ the iterator has superlinear convergence. Hence if we have any knowledge about the value of $F'(\alpha)$, we could set $a = -F'(\alpha)$ and improve the convergence order. Even if we only know an approximation of $F'(\alpha)$ then using such an a will mean that the value of the derivative of the new iterator near the root will be small and linear convergence will be accelerated. Knowledge about $F'(\alpha)$ could be inferred from $F'(0)$ or from $F'(\pm\infty)$ when they exist. Iterators modified in this way are shown in table 12 with a †. Table 6 shows the results of simulations of the

66644 random equations producing 20656 iterators			
method		failures	
converged to a root	12.09%	converged to a non-root	.66%
failed	87.91%	diverged	9.24%
average time	.415	outside domain	.65%
time per root	3.429	too many iterations	7.74%
		iterator fails	0.00%
iterators per equation	.310	no iterators	81.70%

Table 6: Simple iterators, modified to remove a possible linear term $(-F'(0))$

simple iterators with $a = F'(0)$. Surprisingly, the results are only marginally better (about 3% better) than the ones for simple iterators, and the roots found are typically the same. This heuristic produced significantly fewer iterators than the simple ones. Hence we do not recommend this heuristic to generate extra iterators.

5 Inverting functions

To generate iterators we often need to invert functions or expressions. For example, if at one point of our isolation process we have the equation

$$x^2 = h(x)$$

we derive $x = \pm\sqrt{h(x)}$. The choice of signs presents a problem. The easiest way to resolve it is to derive two iterators, one for each sign as we have done before. Almost all functions except for lineal polynomials, present this or other problems for isolation.

An alternative to computing inverses which are multiple-valued is the following. We want to compute a value x_{i+1} from a previous value x_i. When there are multiple choices for x_{i+1} we want to select the one with minimal distance to x_i for convergence reasons. I.e. select the x_{i+1} which makes $|x_{i+1} - x_i|$ minimal. This observation is a crucial one for our algorithm. So from $x_{i+1}^2 = h(x_i)$ we will select $x_{i+1} = \pm\sqrt{h(x_i)}$, the sign depending on whether $|x_i - \sqrt{h(x_i)}|$ or $|x_i + \sqrt{h(x_i)}|$ is smaller. This resolves the first problem, the choice between multiple values. For isolating, we use a function `Inverse_square`. E.g. from $g(x_{i+1})^2 = h(x_i)$ we derive $g(x_{i+1}) = \texttt{Inverse\_square}(h(x_i), g(x_i))$, where

```
Inverse_square := proc( rhs, lhs )
    r := sqrt( rhs );
    if abs(r-lhs) <= abs(r+lhs) then r else -r fi
end;
```

In summary, when the result of an isolating step is a multivalued function, we can take two approaches

(a) Enumerate all iterators, either explicitly as in the case for $x^2 = h(x)$ or as a function of an arbitrary integer n as for $\sin x = x/100$.

(b) Leave the choice to be determined during the actual computation, by choosing the one closest to the previous value, and hence improving the chances of convergence.

A second problem that we have with inverting functions is a domain problem, and it can be illustrated with the equation

$$\sqrt{x_{i+1}} = h(x_i)$$

from which we isolate $x_{i+1} = h(x_i)^2$. In this case there is no choice of inverse, but if $h(x_i)$ is negative (or $|\arg h(x_i)| > \pi/2$) then the inverse cannot be computed, as we assume that $\sqrt{x}$ returns its principal value. E.g. for $\sqrt{x} = -1$, isolation gives $x = 1$ which is not a solution of the original equation. Such a situation should be treated as a domain error. In this case we use a new function to define this type of manipulation, $x_{i+1} = \texttt{Inverse\_sqrt}(h(x_i))$ and

```
Inverse_sqrt := proc( rhs )
    a := argument( rhs );
    if a <= -Pi/2 or a > Pi/2 then
         ERROR( `cannot invert sqrt` )
    else rhs^2 fi
end;
```

6 Conclusions

We have also run the random tests against several classical methods for finding zeros. Table 7 shows the results for Newton's method and table 8 shows the results for the Secant method. These are shown for comparison, and we can see that both are successful less often and require more time.

66641 random equations			
method		failures	
converged to a root	34.36%	converged to a non-root	2.95%
failed	65.64%	diverged	53.24%
average time	1.595	outside domain	19.94%
time per root	4.644	too many iterations	23.87%
		iterator fails	.00%

Table 7: Newton's method, $x - f(x)/f'(x)$

66641 random equations			
method		failures	
converged to a root	37.40%	converged to a non-root	4.46%
failed	62.60%	diverged	72.81%
average time	2.122	outside domain	7.19%
time per root	5.674	too many iterations	15.53%
		iterator fails	.01%

Table 8: Secant method (2-point method)

It is appropriate to run the witness examples that we proposed in the introduction with this method. We think the results speak for themselves. The results for each equation are shown in tables 9, 10 and 11, where the columns are for each iterator and the rows have the value of each iteration. In all cases, at least one of the iteration gives the values which were considered difficult to find. Table 12 shows other examples, the first two taken from [10] and the rest have been collected by the Maple development group. All these problems had an iterator which was successful.

$(x \ln x)^{1/1.001}$	$\frac{x^{1.001}}{\ln x}$	$e^{x^{0.001}}$	$(\ln x)^{1000}$
1.23	1.23	1.23	1.23
.254976...	5.94285...	2.71884...	$.100017... \times 10^{-683}$
$-.348812... + .00109473...i$	3.34053...	2.72100...	$.186958... \times 10^{3198}$
$.365255... - 1.09527...i$	2.77297...	2.72101...	$.989096... \times 10^{3867}$
$-1.31325... - .616913...i$	2.72159...	converged	$.388442... \times 10^{3950}$
$-2.14588... + 3.31938...i$	2.72101...		$.585638... \times 10^{3959}$
$-10.0449... - .0715578...i$	2.72101...		$.596622... \times 10^{3960}$
$-23.2443... + 31.2573...i$	converged		$.769617... \times 10^{3960}$
diverged			$.791411... \times 10^{3960}$
			$.793839... \times 10^{3960}$
			$.794105... \times 10^{3960}$
			$.794135... \times 10^{3960}$
			converged

Table 9: Partial inverse iterators for solving $x^{1.001} - x \ln x = 0$

Extension to systems of equations. The extension of this idea to systems of equations is quite desirable but non-trivial. If we have k equations with k unknowns, then for each equation we have many freedoms: we can choose the variable on which to construct the iterator, we can choose the iterator, and we can choose the order of the equations. And of course, an iterator for the system consists of k iterator equations, one for each of the individual variables, so we can choose each of these independently. This means that we have a huge number of choices to build an iterator for the entire system for small k, and the number of choices can quickly become astronomical for larger k. It would appear that this number is too high to justify its use. We

$\sin \tan x$	$\tan^{-1} \sin^{-1} x$
0.6	0.6
.632004...	.571793...
.668499...	.546785...
.710223...	.524475...
.757788...	.504453...
.811228...	.486381...
.868933...	.469981...
.925654...	.455027...
.970858...	.441327...
.994075...	.428723...
.999439...	.417083...
.999883...	.406296...
.999905...	.396264...
converged	.386907...
	.378155...
	.369947...
	.362231...
	.354961...
	.348096...
	.341601...
	.335445...
	too many iterations

Table 10: Partial inverse iterators for solving $\frac{\sin^{-1} x - \tan x}{x^4} = 0$

have some ideas which improve this situation, but these are developed in a separate paper[5], since they are out of the scope of this one.

References

[1] Åke Björck and Germund Dahlquist. *Numerical Methods*. Prentice-Hall, 2 edition, 1995.

[2] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong,

$\pi + e^{-5-x^2}$	$\pi - e^{-5-x^2}$	$\sqrt{-5-\ln\lvert x-\pi\rvert}$	$-\sqrt{-5-\ln\lvert x-\pi\rvert}$
0.6	0.6	0.6	0.6
3.14629356...	3.13689175...	2.43573211...i	$-2.43573211...i$
3.14159299...	3.14159230...	2.52588218...i	$-2.52588218...i$
3.14159300...	3.14159231...	2.52864327...i	$-2.52864327...i$
converged	converged	2.52872814...i	$-2.52872814...i$
		converged	converged

Table 11: Partial inverse iterators for solving $x^2 + 5 + \ln(|x - \pi|) = 0$

Michael B. Monagan, and Stephen M. Watt. *Maple V Language Reference Manual*. Springer-Verlag, 1991.

[3] Rob M. Corless, Gaston H. Gonnet, Dave E. G. Hare, and David J. Jefrey. Lambert's w function in maple. *Maple Technical Newsletter*, (9):12–22, 1993.

[4] Rob M. Corless, Gaston H. Gonnet, Dave E. G. Hare, David J. Jefrey, and Donald E. Knuth. On the lambert w function. *Advances in Computational Math*, to appear.

[5] Gaston H. Gonnet. Solution of non-linear systems of equations using partial inverses. In preparation, 1996.

[6] Peter Henrici. *Elements of Numerical Analysis*. John Wiley, New York, 1964.

[7] Peter Henrici. *Essentials of Numerical Analysis*. John Wiley, New York, 1982.

[8] W. Kahan. Personal communication, 1992. Two equations which Maple's fsolve could not solve.

[9] Anthony Ralston. *A First Course in Numerical Analysis*. McGraw-Hill, 1965.

[10] Kelly Roach. Symbolic-numeric nonlinear equation solving. In *Proceedings of the 1994 ISSAC*, pages 278–284. ACM Press, Jul 1994.

equation	
iterator	result
$e^x - 6x$	
$\ln(6x)$	converged to 2.83314...
$\frac{e^x}{6}$	converged to .20448...
$-W(-1/6)$	is 0.20448...
$\frac{e^x-1}{5}$ †	converged to .20448...
$\sin z^2 \ln(1+z) - \cos\sqrt{2}z$	
$\sqrt{\sin^{-1}(\frac{\cos\sqrt{2}z}{\ln(1+z)})}$	diverged
$-\sqrt{\sin^{-1}(\frac{\cos\sqrt{2}z}{\ln(1+z)})}$	too many iterations
$e^{\frac{\cos\sqrt{2}z}{\sin z^2}} - 1$	too many iterations
$\frac{\cos^{-1}(\sin z^2 \ln(1+z))}{\sqrt{2}}$	converged to 0.83102...
$\ln(x + \sqrt{\pi} + x^2) - \ln(x - \pi) = 4$	
$\pi + e^{-4+\ln(x+\sqrt{\pi}+x^2)}$	converged to 3.45613...
$-1/2 + 1/2\sqrt{1 - 4\sqrt{\pi} + 4(x-\pi)e^4}$	converged to 50.14201...
$-1/2 - 1/2\sqrt{1 - 4\sqrt{\pi} + 4(x-\pi)e^4}$	diverged
$\frac{\pi + e^{-4+\ln(x+\sqrt{\pi}+x^2)} - e^{-4}x}{1+e^{-4}}$ †	converged to 3.45613...
$e^{4+\ln(x-\pi)} - \sqrt{\pi} - x^2$	diverged
$\sqrt{e^{4+\ln(x-\pi)} - \sqrt{\pi} - x}$	converged to 50.14201...
$x^x(1 + \ln x) + 32 = 0$	
$\left(-\frac{32}{1+\ln x}\right)^{1/x}$	falls outside valid domain
$e^{-32x^{-x}-1}$	converged to 0.46588... $\times 10^{-14}$
$\frac{\ln\left(-\frac{32}{1+\ln(x)}\right)}{\ln x}$	too many iterations
$e^{W(\ln(-\frac{32}{1+\ln x}))}$	converged to 3.443... + 3.812...i

Table 12: Examples from various sources, all examples are started from $x_0 = 1.23$

[11] Joe F. Traub. *Iterative Methods for Solution of Equations.* Prentice-Hall, 1964.