A Numerical Equation Solver in Prolog

Michael A. Covington
CSCI 6540
The University of Georgia

Fall 1998

1 Documentation

This paper presents a numerical equation solver that is written in Prolog.[1] A numerical equation solver is a computer program that solves an equation by doing arithmetic rather than by rearranging the equation. For example, to solve the equation

$$x - 1 = 1/x$$

a numerical equation solver will guess a value for x, compute whether it is too high or too low, and make a better guess until the result is sufficiently accurate.

The method of equation solving used here is called the *secant method* (Pollard 1977:21; Covington, Nute, and Vellino 1997:198–203; Covington 1989). The strategy is to guess *two* points, check which one is closer to the solution (and how much closer), and use the result to make a better guess. Note that solving

$$f(x) = g(x)$$

is equivalent to solving:

$$f(x) - g(x) = 0$$

We can therefore define

$$\mathit{Diff}(x) = f(x) - g(x)$$

[1] This is a sample of a paper for CSCI 6540. The program in it is taken from my earlier published work. Obviously, a real paper for this course must be your own original work.

and try to make $Diff(x) = 0$. Specifically, if g_1 and g_2 are two guesses, then:

$$g_3 = g_2 - \frac{Diff(g_2)}{s}$$

where s, the slope of the line connecting the two points, is:

$$s = \frac{Diff(g_2) - Diff(g_1)}{g_2 - g_1}$$

The program relies crucially on Prolog's ability to add code to the program as it runs. Specifically, given a query such as

```
?- solve(X=1+1/X).
```

the computer first finds the uninstantiated variable in the expression (here `X`), then creates a predicate called `dif` to compute *Diff* and asserts it into the knowledge base. The computation then starts with 1 and 2 as guesses, and continues iteratively until the results of two iterations are the same to within a factor of 1.000001. Finally, the program returns its result by instantiating the variable in the expression.

There are situations in which the secant method does not work. If, by chance, $Diff(g_1) = Diff(g_2)$, the guessing routine will crash, dividing by zero. Even if this does not happen, there are situations in which the secant method will not get a solution within a reasonable amount of time; for example, if two guesses are almost equally good, it may skip too far to the left or right on the graph of the function and end up getting lost. For this reason the program gives up after 100 iterations.

2 Program Listing

```
% File SOLVER.PL
% Numerical equation solver (Covington 1989)

% solve(+(Left=Right))
%  Left=Right is an arithmetic equation containing an uninstantiated
%  variable.  On exit, that variable is instantiated to a solution.

solve(Left=Right) :-
   free_in(Left=Right,X),
   !,                          % accept only one answer from free_in
```

```
   define_dif(X,Left=Right),
   solve_for(X).


% free_in(+Term,-Variable)
%  Variable occurs in Term and is uninstantiated.

free_in(X,X) :-% An atomic term
    var(X).

free_in(Term,X) :-% A complex term
    Term \== [[]],
    Term =.. [_,Arg|Args],
    (free_in(Arg,X) ; free_in(Args,X)).


% define_dif(-X,+(Left=Right))
%  Defines a predicate to compute Left-Right given X.
%  Here X is uninstantiated but occurs in Left=Right.

define_dif(X,Left=Right) :-
   abolish(dif,2),
   assert((dif(X,Dif) :- Dif is Left-Right)).


% solve_for(-Variable)
%  Sets up arguments and calls solve_aux (below).

solve_for(Variable) :-
   dif(1,Dif1),
   solve_aux(Variable,1,Dif1,2,1).


% solve_aux(-Variable,+Guess1,+Dif1,+Guess2,+Iteration)
%  Uses the secant method to solve for Variable (see text).
%  Other arguments:
%  Guess1    -- Previous estimated value.
%  Dif1    -- What 'dif' gave with Guess1.
%  Guess2    -- A better estimate.
%  Iteration -- Count of tries taken.

solve_aux(_,_,_,_,100) :-
   !,
   write('[Gave up at 100th iteration]'),nl,
   fail.
```

```
solve_aux(Guess2,Guess1,_,Guess2,_) :-
   close_enough(Guess1,Guess2),
   !,
   write('[Found a satisfactory solution]'),nl.

solve_aux(Variable,Guess1,Dif1,Guess2,Iteration) :-
   write([Guess2]),nl,
   dif(Guess2,Dif2),
   Slope is (Dif2-Dif1) / (Guess2-Guess1),
   Guess3 is Guess2 - (Dif2/Slope),
   NewIteration is Iteration + 1,
   solve_aux(Variable,Guess2,Dif2,Guess3,NewIteration).


% close_enough(+X,+Y)
%  True if X and Y are the same number to within a factor of 1.000001.

close_enough(X,Y) :-  Quot is X/Y, Quot > 0.999999, Quot < 1.000001.


% Demonstration predicate

test :- solve(X=1+1/X), write(X), nl.
```

3 References

Covington, Michael A. (1989) A numerical equation solver in Prolog. *Computer Language* 6.10 (October, 1989), 45–51.

Covington, Michael A.; Nute, Donald; and Vellino, André (1997) *Prolog Programming in Depth.* Upper Saddle River, New Jersey: Prentice-Hall.

Pollard, J. H. (1977) *Numerical and Statistical Techniques.* Cambridge, England: Cambridge University Press.