An alternative solve function for Maxima

Barton Willis

December 22, 2019

1 Aims

The `solve_alt` package aims to be a replacement for Maxima's solve function. The goals are to

a) support the needs for a variety users, including high school students, university students, engineers, statisticians, and scientists,
b) provide clear user documentation that is focused on user's questions,
c) focus on equations that are pertinent to science and engineering problems,
d) have Common Lisp source code that is easy to extend and to fix,
e) attempt to find all real and complex solutions to equations, but take care to not return spurious solutions.

Additionally, `solve_alt` aims to be compatible with Maxima's current solve function; by compatible, we require:

a) Solutions are expressed as Maxima lists.
b) The `solve_alt` code requires only modest modifications to the rest of the source code for Maxima.
c) Except for algebraically equivalent, but different results, `solve_alt` runs the testsuite and the share testsuite with no errors. This condition should hold for all combinations of the primary option variables that effect functioning of `solve_alt`.
d) The `solve_alt`code runs the testsuite and the share testsuite in about the same time as standard Maxima on all supported operating systems and Common Lisp versions.

We do *not* aim to

a) solve inequalities, diophantine equations, differential equations, or to use purely numerical methods (the Newton method, for example) solutions,
b) build a platform that could be used to display the steps done to find the solution,
c) write a new version of `algsys`.

Our development priorities are (in order from highest to least)

a) provide well written source code with ample informative comments,
b) have a modest code size,
c) be conservative with memory usage,
d) solve equations quickly.
e) source code should compile without few warnings on all supported versions of Common Lisp.

2 Representation of solutions

To be compatible with Maxima, solutions to equations with one variable have the form of a Maxima list; for example $[x = e_1, x = e_2, \cdots, x = e_n]$, where the expressions $e_1, e_2, \ldots, e_n$ are free of x. The order of solutions is *unspecified*.

When `solve_alt` is unable to find a solution and the option variable `solveexplicit` is false, return an equivalent, but possibly simplified equation; otherwise print a message that there is no method for solving the equation and return the empty list.

Atoms of the form %Zk, %Rk, and %Ck, where k is a nonnegative integer, represent arbitrary integers, real numbers, and complex numbers, respectively.

Solutions to equations with two or more variables are lists of lists of the form $[x_1 = e_1, x_2 = e_2, \cdots, x_n = e_n]$, where the expressions e_1 through e_n are free of the symbols x_1 through x_n. The order of variables in each solution is unspecified, and the order of the lists is unspecified.

3 Keep float mechanism

The top level solve function encloses all floats (Binary64) and big floats in labeled boxes. The floats are converted to exact rational form. After solve exits, numbers are converted back to floats (big floats are converted using the *current* value of `fpprec`.

It can, of course, happen that the simplification after conversion back to floats can involve subtractive cancellation. In such cases, purely numerical methods might be better.

4 Solve function hierarchy

The top level function solve function:

a) resets multiplicities to its default value,
b) expunges constant variables and duplicates from the list of variables,
c) checks for inequalities and signals an error when found,
d) creates a new super context–all assumptions made during solve are removed after exiting,
e) protects all floats and big floats in labeled boxes,
f) does gensym substitutions when solving for nonatoms,
g) dispatches the appropriate lower level solve function,
h) reverts gensym substitutions for nonatom solve,
i) unboxes the floats and big floats,
j) kills the super context.

The lower level solve functions are solve-single-equation (solve one equation in one unknown), redundant-equation-solve (solve two or more equations for one unknown), and solve-multiple-equations (solve two or more equations in two or more unknowns).

solve-single-equation If the input is a polynomial, dispatch the polynomial solve code.

First the equation is processed using The primary method for solving one equation in one unknown is to match the equation to the form $aF(X) - b = 0$, where X depends on the unknown, the function F has a known inverse, and a and b are free of the unknown x. When the match is successful, the new tasks is to solve $X = F^{-1}(b/a)$. In general, F^{-1} is multi-valued–in such cases, the task is to solve finitely many equations of the form $X = F^{-1}(b/a)$.

The multi inverses for functions are stored in a Common Lisp hashtable. To accommodate the needs of diverse users, there are two hashtables; one is named `multivalued_inverse` and the other `single_valued_inverse`. The option variable `solve_inverse_package` is the value of the current hashtable for the function multi inverses.

5 Polynomial zeros

The zeros of polynomials of degree four or less are expressed in terms of radicals. Also cyclotomic polynomials of degree six are solved in terms of radicals, as well as polynomial of degree five or greater that decompose into polynomials of degree four or less. *Well there option variable solvedecomp controls using polynomial decomposition, but I think this option could be removed.*

a) Polynomial coefficients that are not simplified to zero by the general simplifier are assumed to be nonzero. Even if an atom has been declared to be zero, the general simplifier does not simplify it to zero. Thus even if the atom a has been declared to be zero, the solution to $ax = b$ is $x = -b/a$.