Symbolic Manipulation and Computational Fluid Dynamics

Patrick J. Roache*
Ecodynamics Research Associates, Inc., Albuquerque, New Mexico
and
Stanly Steinberg†
University of New Mexico, Albuquerque, New Mexico

This paper is divided into three sections. The first section is a brief present presentation of the powerful capabilities of the symbolic manipulation code MACSYMA developed at MIT. The second section presents results recently achieved by the authors using symbolic manipulation on the problem of three-dimensional body-fitted coordinates. The third part of the paper surveys previous uses of symbolic manipulation in computational fluid dynamics and related areas, and potential future uses of the code.

Introduction

THE purpose of this paper is to provide a brief introduction to the uses of symbolic manipulation within the practice of disciplines such as computational fluid dynamics (CFD). The first section is a brief presentation of the powerful capabilities of the symbolic manipulation code MACSYMA. The next section presents results recently achieved by the present authors using symbolic manipulation on the problem of three-dimensional boundary-fitted coordinates. The last section consists of a short survey of previous uses of symbolic manipulation in CFD and related areas, and of what we consider to be promising areas for future use.

Symbolic manipulation performed by computer is an area that has just recently begun to be more widely appreciated and has tremendous potential. Symbolic manipulations are not floating point calculations, but symbolic operations (e.g., the chain rule differentiation) performed by computer logic. One of the earliest symbolic manipulation languages was FORMAC, which had the capability of doing simple differentiation symbolically. ALTRAN is a more powerful and later version of the symbolic manipulation language another version, "Micromath," is available on the better microcomputers. Perhaps the most powerful of the symbolic manipulation codes is MACSYMA, which has been developed over many years at the Massachusetts Institute of Technology.

MACSYMA Capabilities

The symbolic manipulation code MACSYMA has been developed through an extensive team effort. Many practitioners of computational fluid dynamics have some acquaintance, at least second-hand, with this code, but our impression is that few have an appreciation for its powerful capabilities or those of the general field of symbolic manipulation. These capabilities include variable-precision arithmetic and algebraic substitutions and simplifications, symbolic (rather than numeric) equation solving for scalars and matrices, truncated Taylor series expansions, power series, differentiation including the chain rule for single- and multivariate functions, very powerful definite and indefinite integration, taking limits of expressions, ordinary differential equation solving (closed-form solutions), and many other mathematical operations previously thought to be in the realm of human beings rather than computers. For an introduction to the subject of symbolic manipulation, see Refs. 1-3. Recently, a version of MACSYMA, called VAXIMA, has become available for the VAX family of minicomputers. Significantly, MACSYMA also has the capability of actually generating a Fortran code from symbolic mathematics in the usual scientific notation. MACSYMA is also a programming language and can concatenate alphanumeric variables.

The second author has developed a code called GEEWHIZ that demonstrates many of the impressive capabilities of MACSYMA. A sampling is included as an appendix in Ref. 21.

Three-Dimensional Boundary-Fitted Coordinates via Symbolic Manipulation

Background

Coding errors plague all computational work. The chance for coding error increases as the complexities of the problems increase. Computational fluid dynamics codes used in aerodynamics and ballistics calculations are commonly 5,000 or more lines of Fortran code in length. Codes used in nuclear reactor safety work have as many as 65,000 lines of Fortran. Although containing much complex physics, these codes are actually elementary in that they are written in very simple coordinate systems such as Cartesian, cylindrical, etc.

Recently, there has been a surge of activity in the application of coordinate transformation techniques. The proceedings of three recent conferences, two dedicated entirely to this problem[4,5] and one emphasizing it,[6] show a tremendous amount of work being done in this area. We have been impressed with the difficulty of code verification for transformed grid problems and with the complexity of three-dmensional equations written in general nonorthogonal grids. The most popular and most generally applicable coordinate transformation methods are boundary-fitted coordinate systems, which are generally nonconformal and therefore do not maintain the form of the original equations written in Cartesian or other elementary coordinates.

Overview

We have recently developed and validated symbolic manipulation codes for working a problem of considerable complexity—that of solving general second-order elliptic partial differential equations in a general (nonorthogonal) three-dimensional boundary-fitted coordinate system. We follow the approach of Thompson and his colleagues[7,8] and generate the coordinate system itself by solving elliptic partial differential equations in the transformed plane. We address

Presented as AIAA Paper 83-1952 at the AIAA 6th Computational Fluid Dynamics Conference, Danvers, Mass., July 13-15, 1983; submitted July 25, 1983; revision received Jan. 6, 1984. This paper is declared a work of the U.S Government and therefore is in the public domain.

*Member AIAA.

†Professor, Department of Mathematics and Statistics.

only the problems of coding complexity and validation of the problem formulation, rather than anything new in the generating equations or in the discrete solver.

We have used the VAX computer-based symbolic manipulation code VAXIMA to analytically perform the coordinate transformation of the hosted equations, to substitute the finite difference equations, to gather the coefficients together, and to write the Fortran code for the finite difference stencil. The matrices defining the stencil are then passed to a "canned" solver (the GEM spatial marching methods in two dimensions and hopscotch SOR or multigrid in three dimensions) and the solution is obtained without the user writing the Fortran code. The same procedure is followed for the generation of the three-dimensional grid using the elliptic grid generation techniques. Validation of the entire procedure is performed by systematic truncation error convergence testing. This use of symbolic manipulation has greatly speeded code development time and allows easy conversion of the equations solved and the problem dimensionality.

Symbolic Code

Many details of the symbolic code and the Fortran code that it produces are given in a more complete description of this work.[9] We note here the generality of the symbolic code. The following code segment defines a special two-dimensional "hosted equation" corresponding to $\nabla \cdot \sigma \nabla f = q$:

```
dep: [f, sigma]
ind: [x,y]
depends(dep,ind)
eqn:sum(diff(sigma*diff(f,ind[i]),ind[i]),i,l,nn)
```

These few lines define the dependencies, independent variables, and hosted equation to be transformed. Only this segment of the symbolic code would need to be changed in order to alter the hosted equation being transformed. For example, the corresponding segment for a more general three-dimensional hosted equation is:

```
ind: [x,y,z]
depends(f,ind)
eqn:sum(sum
  (concat(a,concat(i,j))*diff(f,ind[i],l,ind[j]),i,l,nn),j,l,nn)
  + sum(concat(b,i)*diff(f,ind[i]),i,l,nn) + c*f + d
```

The same localization occurs in the symbolic code for the grid generating equations, and we expect to be able to readily convert the code the more general elliptic generating systems for partial differential equations.

The finite difference expressions for the various derivatives are currently coded by hand (for the generic derivative types, not for each occurrence). It is possible to automate this procedure using symbolic manipulation, which is especially worthwhile for high-order accuracy derivative boundary conditions. However, this is not a trivial task.

The Fortran Codes

The Fortran subroutines for the finite difference stencils are given in Ref. 9. Two representative code segments are shown in Fig. 1.

One of the first things to notice is that there is considerable redundant arithmetic in the formulas that compute the stencil elements. We attempted to use The VAXIMA functions "optimize" and "factorsum" to improve the form of these expressions. Unfortunately, the expression for U1 shown in Fig. 1 is left unchanged by each of these VAXIMA functions. It is certainly possible to write programs to reorganize the formulas in Fortran subroutines for the particular case we studied here. It is also easily done with a text editor. We decided that a general solution of this problem is beyond the scope of the present work, but we hope to work on it in the near future. We note that it will be important to make the solution of this part of the problem compatible with the optimizers in various Fortran compilers and that a good Fortran optimizer will accomplish much of this task automatically.

Also, we made no attempt to make the formulas in the Fortran code readable. Instead, as a independent part of the VAXIMA code, we write all of the formulas in VAXIMA's planar format (which is designed for human reading, e.g., see Appendix of Ref. 21).

It is worth noting that in our applications the computations done by our subroutines represent a small percentage of the total computations; thus, optimizing the operation count in the subroutines is not critical. It is also easy to set up a test case for the subroutines, use an editor to modify the subroutines and then compare the results to the original subroutine to guard against errors. Also, in some applications, it may be desirable to have some part of the code compute the arrays of the derivatives of the coordinate functions and then pass these arrays to any other parts of the program. Again, these are easy changes to make in the subroutines.

If one looks at the mathematical formulation of the change of variables,[9] it seems natural to introduce arrays to represent terms such as T_i, U_i, and $S_{i,j}$. In our programs such arrays are not introduced and instead atomic variables of the form

T1, T2, T3, U1, U2, U3, S11, S12, ...,S33

are used. This represents a substantial savings in the run time of the program because of the elimination of the need to look up array elements. (This depends significantly on the machine architecture and the compiler.) A similar procedure is used to represent the derivatives of the coordinate functions.

We used an extremely modular approach in developing and validating these codes, and calculating and storing the 10 three-dimensional arrays defining the finite difference operator. (A first glance at the problem suggests a 27-point operator, but 8 of these coefficients are identically zero for second-order equations and 9 more are eliminated by symmetry and antisymmetry relations, provided that the expanded or nonconservation form of the operators is used.) This approach of "actualizing" the coefficient arrays is commonly followed in finite element codes, but less so in finite difference codes. Of course, the disadvantage is the storage penalty. The alternative is to recalculate the coefficients at each mesh point during the solution procedure. For linear problems, the "actualized" approach will be significantly faster, unless the memory demands cause significant paging costs on a virtual memory computer. The preferred approach is not clear and is certain to be machine dependent. In any case, the present subroutines could easily be converted to calculating and passing the nonsubscripted stencil values at a single (i,j,k) point. The fully modular approach aided the validation procedure significantly.

```
  ...
  X1 = (X(I+1,J,K)-X(I-1,J,K))/H1/2.0
  Y1 = (Y(I+1,J,K)-Y(I-1,J,K))/H1/2.0
  Z1 = (Z(I+1,J,K)-Z(I-1,J,K))/H1/2.0

  ...
  ...

  S23 = S0*(VK32*VK33+VK22*VK23+VK12*VK13)
  S33 = S0*(VK33**2+VK23**2+VK13**2)
  U1  = S33*VK31*Z33+2*S23*VK31*Z23+S22*VK31*Z22
                            +2*S13*VK31*Z13+2*S1
1   2*VK31*Z12+S11*VK31*Z11+S33*VK21*Y33+2*S23*
                            VK21*Y23+S22*VK21*Y2
2   2+2*S13*VK21*Y13+2*S12*VK21*Y12+S11*VK21*Y11
                             +S33*VK11*X33+2*S23
3   *VK11*X23+S22*VK11*X22+2*S13*VK11*X13+2*S12
                            *VK11*X12+S11*VK11*X
4   11

  ...
```

Fig. 1 Code segments of Fortran subroutines.

Code Validation

The potential for errors in either the problem formulation or the encoding procedure always exists in complex codes such as three-dimensional boundary-fitted coordinate codes. The need for validation was emphasized in the present work because symbolic manipulation was used; the resulting "psychological distance" from the work made it less likely to be satisfied with superficial plausibility exercises based on intuitive ideas of acceptable levels of absolute error.

We validated the codes by choosing a continuum solution for the class of problems treated by the code and validating the convergence to this continuum solution by systematic truncation error convergence testing over a sequence of grid sizes. Although well known in theory, this procedure is seldom followed in practice, especially for elliptic equations.

The selected form of the solution was chosen to insure that all of the derivative terms in the operator are exercised and that there is nonzero truncation error for finite mesh spacing even without the transformation of coordinates. (For example, a parabolic solution will show no truncation error using second-order accurate solutions with the identity transformation. Several published "validations" of the accuracy of upwind differencing are inadequate because the chosen solution structure does not exercise the meaningful terms in the truncation error, giving a false indication of accuracy.) Similar procedures were followed for both the hosted equation codes and the grid generation codes, the latter being somewhat more difficult to validate because of the need for a nonlinear solution. A range of stretching parameters and continuum equation coefficients were examined in a grid successively halved three times, from 5^3 to 33^3. Details are found in Ref. 9. The near constancy of maximum truncation error/h^2, where h is the grid spacing in the transformed space, indicated that the coding is correct and that the solution is second-order accurate.

Order of Solution Accuracy

The results obtained in this systematic convergence testing strongly confirm that strong and inappropriate coordinate stretching can indeed increase the size of the truncation error, as is well known, but that the asymptotic order of the accuracy, as indicated by the reduction in truncation error resulting from systematic reduction in grid size, remains $\mathcal{O}(h^2)$ if centered differences are used in the transformed equations.[9]

Stages in the Code Development

When we first started on this project we decided to implement the symbol manipulation code at as "high" a level as was possible, meaning that we would give the symbol code the differential equation in some natural coordinates and then let the chain rule do its work. This produces a surprisingly large equation in a general coordinate frame. The equation

$$\sum_{i=1}^{n} \frac{\partial}{\partial x_i} \sigma \frac{\partial}{\partial x_i} f = 0$$

in fully expanded form contains the number of terms indicated in Table 1. Because of the size of this expression, the remainder of the VAXIMA program became quite tricky. One of the earliest versions of the VAXIMA program for the three-dimensional variable σ case, ran about 60 h of CPU time on a VAX 11/780 operating under UNIX and produced a hosted equation subroutine of approximately 1800 lines of dense Fortran code. There were some minor problems in getting the subroutine to compile. However, as soon as the subroutine compiled, it ran and produced correct results. (Surprisingly, round-off error was not a problem, despite the arithmetical complexity.) This was accomplished in less than a month of full-time work. It is, we believe, essentially impossible to duplicate these results by hand in any amount of time.

Table 1 Equation implementing symbol manipulation code

σ	n	No. of terms
Constant	2	42
Constant	3	1611
Variable	2	50
Variable	3	1710

It seemed that the run time of the symbol program was out of line with the amount of work being done by the symbol code. It was found that during the replacement of derivatives by the atomic variables, the simplifier was rearranging the terms in the differential equations in an attempt to find simplification where none existed. At this point a code was written to break up all the large expressions into pieces that contain only a few terms. This prevented a major portion of the wasted algebra and reduced the run time of the VAXIMA code to under 3 CPU h.

The current approach teaches VAXIMA something about the structure of the intermediate expressions occurring in the chain rule, producing a code with a running time under 8 CPU min. This efficiency came from a formula for the derivative of the inverse of a matrix function. Thus, if $m(t)$ is a matrix whose entries depend on t, then

$$\frac{\mathrm{d}}{\mathrm{d}t} m^{-1}(t) = -m^{-1}(t) \frac{\mathrm{d}m(t)}{\mathrm{d}t} m^{-1}(t)$$

It is interesting to note that VAXIMA does not know this formula. On the other hand, VAXIMA does know the derivative of the square of a matrix function,

$$\frac{\mathrm{d}}{\mathrm{d}t} m^2(t) = m(t) \frac{\mathrm{d}m(t)}{\mathrm{d}t} + \frac{\mathrm{d}m(t)}{\mathrm{d}t} m(t)$$

The choice of the VAXIMA symbol manipulator was dictated by the experience of the second author. However, we believe that these programs could be implemented in most general-purpose symbol manipulators. It is important to note that this analytic formulation eliminates the need for the symbol manipulation program to know the multivariate form of the chain rule. It is also important to note that the completion of this project in a reasonable time while using reasonable effort was greatly facilitated by the large number of procedures available in VAXIMA and by the capability to run interactively.

Other Uses and Potential

Symbolic manipulation has already been used in computational fluid dynamics, although not as utilized in the present work. Warming and Hyett[10] used FORMAC to perform the algebra for their significant study of the stability and accuracy of algorithms for time-dependent equations. With the power of the symbolic manipulation, they were able to prove the equivalence of their modified equation analysis to the classical Von Neumann stability analysis for linear periodic problems. They also corrected the higher derivative terms in previously published heuristic stability analyses and clarified the interpretation of artificial damping and phase errors. Kok[11] has written an ALGOL rountine for computing the nine-point stencils for first- and second-order partial derivatives on a specified grid, including boundary conditions. Hyman[12] has similarly used symbolic manipulation to generate higher-order accuracy finite difference expressions. Chandler[13] has used it to generate new closed-form solutions for a common model, the Burgers equation. Anderson et al.[14] used the Fortran-writing ability of MACSYMA. Ng and Char[15] used MACSYMA to analytically calculate the gradients and Jacobians of multidimensional functions for use in steepest-descent and Newton-Raphson methods. The potential of symbolic manipulation for discretized partial differential equations was pointed out a decade ago by Moler.[16]

The previous work that is closest to the present work is that of Wirth,[17] who used MACSYMA to discretize and linearize equations on a rectangular grid. He has also used MACSYMA in a (local) Fourier stability analysis, although the lack of inequality rountines in MACSYMA (as well as some fundamental limitations of symbolic manipulation for doing algebraic simplification) require human intervention. Wirth's paper will be of interest to anyone involved in the analysis of methods for computational fluid dynamics.

There are other examples of the use of symbolic manipulation in the literature, but these are the only ones we have found with a clear computational inclination. For general work in the area of symbolic manipulation, See Refs. 18-20.

We consider this paper an interim report on our present work. We believe that what has been accomplished so far provides a useful tool for constructing the Fortran codes. However, there are many problems still left. One important problem is to extend our current VAXIMA code to handle general boundary conditions. Simple boundary conditions are now hand coded. It is also important to solve the problem of setting up intermediate expressions so that the evaluation of our formulas is more efficient.

We believe that it is now possible to extend the ideas of this paper to more general equations including systems and evolution problems. It should be possible to handle other methods of generating coordinates including hyperbolic generation and to use higher-order difference schemes. It is now possible to set up a package of symbol manipulation programs that would be useful to others working on problems similar to the problem studied here. Potential future uses include the following:

1) Multiple equations, including time-dependent terms.

2) High-order (continuum) equations.

3) Perturbation terms in the source term formulation so as to give deferred corrections for high-order accuracy.

4) Deferred corrections for nonlinear terms.

5) Validation procedures for all the above.

More difficult problems include the following:

6) Conservation forms (weak and strong).

7) Upwinding or other conditional differencing.

8) General boundary conditions.

9) Code optimization.

Finally, we consider some very difficult topics, which are very difficult indeed, but wherein symbolic manipulation holds great potential. At this time, the following appear to be the most interesting and promising areas of application:

10) Combination of perturbation methods and numerical methods. These "semianalytic" approaches have already been used with some success and are not difficult for regular perturbation problems. With insight, they can be used for singular perturbation problems and could be used in general grid problems to remove grid-introduced singularities.

11) The general area of coordinate transformations, including self-adaptive or solution-adaptive transformations.

12) Generation and analysis of new discrete forms via finite difference, finite element, least squares, etc., methodologies. The analysis would include analysis of truncation error, phase error, stability, etc., as in Refs. 10 and 17.

13) Constitutive equation testing, in areas such as turbulence modeling, non-Newtonian fluids, soil mechanics, multiphase flow in porous media, gravitational theory.

The prospect of virtually error-free testing of constitutive equations and difference forms is most attractive. Using the symbolic manipulation approach, human errors are still possible in this process, but they are at a different level than Fortran coding errors. Grand mistakes will occur, but not the petty ones of writing $s(i+1,j,k)$ when the term should have been $s(i-1,j,k)$, etc.

The improvement in capability of symbolic manipulation over languages such as Fortran is analogous to the advantage of Fortran over an assembly language. This capability *may* even apply to developing efficient solvers (e.g., to writing a multigrid solver in three dimensions, as opposed to just formulating the discrete problem as done herein), but we do not see how to attempt this at present.

As an alternative to producing the subroutine by way of symbolic manipulation, one could easily envision building up a library of subroutines for various systems of differential equations, as has been accomplished to some extent in finite element codes and in finite difference codes for one-dimensional time-dependent equations. The complexity and generality of the possible multidimensional high-order systems of partial differential equations makes this approach seem impractical. However, if it is to be done practically, symbolic manipulation codes of the type described herein would be the logical tool to use to develop such a library.

Summary

The symbolic manipulation procedures described in this paper already have been very successful in speeding code development time. These procedures hold great promise in the use of general three-dimensional coordinate transformations and in the testing of constitutive equations for turbulence, non-Newtonian fluids, soil mechanics, gravitational theory, etc. The prospect of virtually error-free testing of constitutive equations and difference forms is very attractive. We expect that the use of symbolic manipulation in these and other areas will shortly be recognized as the way of the future and that the practice of disciplines such as computational fluid dynamics will be revolutionized in the next decade as the power of symbolic manipulation becomes widely recognized.

Acknowledgments

This work was partially supported by the U.S. Army Research Office, the U.S. Air Force Office of Scientific Research, and the National Science Foundation. Our thanks to Dr. Frank Thames for bringing Ref. 17 to our attention.

References

[1]Pavelle, R., Rothstein, M., and Fitch, J., "Computer Algebra," *Scientific American,* Vol. 245, No. 6, Dec. 1981, p. 136.

[2]"MACSYMA Primer," Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, March 1978.

[3]"A Brief Overview of MACSYMA," Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, April 1982.

[4]*Numerical Grid Generation Techniques,* Proceedings of Workshop at NASA Langley Research Center, edited by R. E. Smith, NASA CP 2166, Oct. 1980.

[5]*Proceedings of Symposium on the Numerical Generation of Curvilinear Coordinate Systems and Use in the Numerical Solution of Partial Differential Equations,* Nashville, Tenn., edited by J. F. Thompson, North-Holland Publishing Co., Amsterdam, April 1982.

[6]*Proceedings of 1982 Army Numerical Analysis and Computers Conference,* The Army Mathematics Steering Committee, Vicksburg, Miss., Feb. 1982.

[7]Thompson, J. F., Thames, F. C., and Mastin, C,W., "Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing any Number of Arbitrary Two-Dimensional Bodies," *Journal of Computation Physics,* Vol. 15, 1974, pp. 299-319.

[8]Mastin, C. W., and Thompson, J. F., "Transformation of Three-Dimensional Regions onto Rectangular Regions by Elliptic Systems," *Numerishe Mathematik,* Vol. 29, 1978, pp. 397-407.

[9]Steinberg, S. and Roache, P. J., "Symbolic Manipulation and Computational Fluid Dynamics," *Journal of Computational Physics,* to be published.

[10]Warming, R. F. and Hyett, B. J., "The Modified Equation Approach to the Stability and Accuracy Analysis of Finite Difference Methods," *Journal of Computational Physics,* Vol. 14, 1974, pp. 159-179.

[11]Kok, J., "An Algol 68 Routine for the Approximation of Partial Derivatives on a Two-Dimensional Grid," Stichting Mathematisch Centrum, Amsterdam, Rept. NW 94/80, Nov. 1980.

[12]Hyman, M., Personal communication, 1983 (see also Ref. 6).

[13]Chandler, L., Personal communication, 1983.

[14]Anderson, J. D., Lau, E. L., and Hellings, R. W., "Use of MACSYMA as an Automatic Fortran Coder," *1979 MASYMA Users Conference,* edited by V. Ellen Lewis, Washington, D.C., 1979, pp. 583-595.

[15]Ng, E. and Char, B., "Gradient and Jacobian Computation for Numerical Applications," *1979 MACSYMA Users Conference,* edited by V. Ellen Lewis, Mathlab Group, Laboratory for Computer Science, MIT, Cambridge, Mass., 1979, pp. 604-621.

[16]Moler, C. B., "Semi-symbolic Methods in Partial Differential Equations," *Proceedings of Second ACM Symposium on Symbolic and Algebraic Manipulation,* edited by S. Petrik, Association of Computing Machinery, Philadelphia, 1971, pp. 349-352.

[17]Wirth, M.C., "Automatic Generation of Finite Difference Equations and Fourier Stability Analysis," in Ref. 18, pp. 73-78.

[18]*Proceedings of 1981 ACM Symposium on Algebraic and Symbolic Computation,* edited by Paul S. Wang, Association of Computing Machinery, Philadelphia, 1981.

[19]*Proceedings of 1977 MACSYMA Users Conference,* Berkeley, Calif., NASA CP-2012, 1977.

[20]*SIGSAM Bulletin,* Quarterly Publication of Special Interest Group on Symbolic and Algebraic Manipulation, Association of Computing Machinery, Philadelphia.

[21]Roache, P. J. and Steinberg, S., "Symbolic Manipulation and Computational Fluid Dynamics," *Proceedings of AIAA 6th Computational Fluid Dynamics Conference,* AIAA, New York, 1983, Paper 83-1952.