*J. Symbolic Computation* (1996) **11**, 1–17

# Two perturbation calculations in fluid mechanics using large-expression management

R. M. CORLESS[1], D. J. JEFFREY[1], M. B. MONAGAN[2], AND PRATIBHA[1]

[1] *Dept. Applied Math, University of Western Ontario, London,* CANADA
[2] *Dept. Math & Statistics, Simon Fraser University, Burnaby, B. C.,* CANADA

*(Received XXXX)*

Two fluid-flow problems are solved using perturbation expansions, with special emphasis on the reduction of intermediate expression swell. This is done by developing tools in Maple that contribute to the efficient representation and manipulation of large expressions. The tools share a common basis, which is the creation of a hierarchy of representation levels such that expressions located at higher levels are expressed using entries from lower levels. The evaluation of higher-level expressions by the algebra system does not proceed recursively to the lowest level, as would ordinarily be the case, but instead can be directly controlled by the user.

The first fluid-flow problem, arising in lubrication theory, is solved by implementing a technique of switch-controlled evaluation. The processes of simplification and evaluation are controlled at each level by user-manipulated switches. A perturbation solution is derived semi-interactively with the switch-controlled evaluation being used to reduce the size of intermediate expressions. The second fluid problem, in convection, is solved by extending a perturbation series in several variables to high order by implementing techniques for the automatic generation of hierarchical expression sequences.

## 1. Introduction

We discuss here the application of Maple to the solution of two problems from fluid mechanics. Both problems are solved as regular perturbations in a small parameter, and the main challenge is the handling of intermediate expression swell. For each problem, the leading terms were first computed 'by hand', i.e. without the help of a computer algebra system, by Cooley and O'Neill (1969), Jeffrey (1982) and Mack and Bishop (1968). Those older solutions had to be extended in order to obtain expressions accurate for higher values of the expansion parameters. The original derivations had stopped when expression swell prevented further progress by hand. The problem of expression swell is well known to mathematicians, and they have developed many strategies for carrying a perturbation calculation forward in spite of the daunting sight of equations growing ever longer as they feed on their predecessors. The literature contains many heroic perturbation solutions to high order (Van Dyke 1974, Delaunay 1867, Deprit *et al.* 1970) and their existence, and correctness, is a testimony not only to human endurance but also to the efficacy of the strategies that the calculators used to manage their intermediate expressions.

Since one of the attractions of computer algebra systems (CAS) is their ability to

0747–7171/90/000000 + 00 $03.00/0 © 1996 Academic Press Limited

manipulate long expressions, it is natural to turn to them to continue calculations that stopped because of expression swell. However, simply transferring perturbation calculations to a CAS does not automatically mean that they can be extended to the required order. In particular, the problems studied here lead to expressions that increase in size so rapidly that they exhaust the memory of any present machine. For this reason, we have considered how the successful strategies of the pre-computer age can be adapted for use in computer algebra systems, specifically Maple.

In addition to the straightforward problem of memory, another problem of importance to mathematicians is the comprehension of large expressions. A recent description of using Maple in the classroom (Boyce and Ecker 1992) commented that students laughed while an unexpectedly large expression scrolled across the screen. The reaction of the class is indicative of the discomfort many users feel with very large expressions. Current systems offer some assistance in this area: Mathematica will hide the expression from you†, and Maple will search for subexpressions that can be printed separately; we, however, would like a means for displaying the mathematical skeleton of an expression, in a way that the examples below will make clear.

This paper presents several tools that have reduced expression swell in the solution of problems arising in fluid mechanics. They are all based on one key idea, which here we call hierarchical representation. Related ideas have appeared in the literature and in software systems already, under a variety of names, such as computation sequences and straight-line programs.

Some of these related works, such as (Freeman *et al.* 1986) and (Díaz and Kaltofen 1995), concentrate on the manipulation of computation sequences or straight-line programs to compute (e.g.) greatest common divisors of polynomials defined by such sequences. Zippel (1993) shows how to use sparse interpolation to convert a computation sequence into a more standard representation of a polynomial. Other work, such as is embodied in Maple's 'optimize' command or the special purpose programs of Budgell and El Maraghy (1990), shows how to turn very large expressions, once generated, into more compact and useful computation sequences.

This paper concentrates instead on interactively generating an appropriate computation sequence from a natural hierarchy of the problem, which is discovered as the computation proceeds. This new approach has the advantage that the sequences are natural to the problem at hand, and their forms can be controlled by the user.

Also, because the simplifications are introduced early in a calculation, their benefits, particularly with regard to quicker processing and smaller memory requirements, can be felt throughout the rest of the calculation. Further comparisons with existing ideas are left to the final section of the paper.

**Definitions**. A *hierarchy* is an ordered list $[S_0, S_1, \ldots]$ of symbols, together with an associated list $[D_0, D_1, \ldots]$ of *definitions* of the symbols. For each $s \in S_i$ with $i \geq 1$, there is a definition $d \in D_i$ of the form $s = f(\sigma_1, \sigma_2, \ldots, \sigma_k)$ where $f$ is some well-understood function such as an elementary function and each $\sigma_j$ is a symbol in $[S_0, S_1, \ldots, S_{i-1}]$ and is thus lower in the hierarchy than $s$.

A *computation sequence* $c$ is recursively defined as an expression $c = g(s_1, s_2, \ldots, s_k)$ containing symbols $s_j$ from a known hierarchy, together with the computation sequences defined by the associated definitions $d_1, d_2, \ldots, d_k$ of the symbols appearing in $c$. Obvi-

† Maple's share library routine `sprint` will also do this.

ously a computation sequence defined in terms of symbols in $S_0$, the set of atoms of the system, is just an expression.

Intuitively, the hierarchy is the framework for constructing computation sequences, and a computation sequence is an expression defined in terms of simpler expressions. In mathematics, the hierarchy is rarely explicitly discussed (it is usually unique to the problem at hand), and the computation sequences are usually written in the recursive form mentioned above, separated by the word 'where'. For example, suppose our hierarchy is $[\{x\},\{s_1\},\{s_2\},\{s_{31},s_{32}\}]$ together with definitions $\{s_1 = 1/(1+x)\},\{s_2 = x^2 + s_1\},\{s_{31} = W(s_1/s_2), s_{32} = s_1 + \tan(s_2)]$. Then a mathematician would write $y = (1/(1+x) + \tan(x^2 + 1/(1+x))) + 1/(1+x)$ in the economical reprepresentation

$$s_{32} + s_1$$

where

$$s_{32} = s_1 + \tan(s_2)\,,$$

$$s_2 = x^2 + s_1\,,$$

and

$$s_1 = 1/(1+x)\,.$$

A computer programmer would of course reverse the order to get the following straight-line program or computation sequence.

```
s1 = 1/(1+x)
s2 = x*x + s1
s32 = s1 + tan(s2)
y = s32 + s1
```

We will refer to computation sequences and straight-line programs (in whatever order presented) as *hierarchical representations*, and indeed will often use the terms interchangeably.

## 2. A lubrication calculation using switch-controlled evaluation

The equations governing fluid flow in the region between two close rigid spheres were solved in Jeffrey (1982) using a perturbation scheme derived from lubrication theory. The solution printed in the paper is expressed in a compact form, with intermediate variables used judiciously, while in contrast a straightforward implementation of the same scheme in Maple gives a larger and less intelligible solution. Our aim here is to solve the equations with Maple in a way that incorporates the strategies of the pre-computer solution.

### 2.1. Solution before computer algebra

First we sketch the origin of the set of equations and then describe the method of solution, which in the present context is the object of interest. We calculate the fluid flow between two spherical surfaces, when one is at rest and the other approaches it at velocity $V$. The spheres have radii $a$ and $b$ and are separated by a gap $h$. The nondimensional parameter $\varepsilon = h/a$ is taken to be much smaller than 1, corresponding to a small relative gap between the spheres. The ratio of the sphere radii $\kappa = b/a$ can be any positive value,

but we assume that $\kappa = O(1)$ as $\varepsilon$ goes to zero. We use cylindrical coordinates $(r, \theta, z)$ to express the velocity field in terms of a stream function $\Psi$ as

$$u = U(r^{-1}\Psi_z, 0, -r^{-1}\Psi_r) \ . \tag{2.1}$$

The equation for $\Psi$ is

$$\left(\frac{\partial^2}{\partial z^2} + \frac{\partial^2}{\partial r^2} - \frac{1}{r}\frac{\partial}{\partial r}\right)^2 \Psi = 0 \ , \tag{2.2}$$

with boundary conditions $\Psi = r^2/2$ and $\Psi_z = 0$ on the moving sphere, and on the other sphere $\Psi = \Psi_z = 0$. The perturbation solution is based on the observation that when the nondimensional gap width $\varepsilon$ between the spheres is small, the equations and boundary conditions can be approximated as follows.

We stretch the coordinates $r, z$ locally in the gap using $(Z, R) = (z/(a\varepsilon), r/(a\varepsilon^{1/2}))$. This stretching reflects the physical observation that effects across the gap are more important that effects along the gap; see O'Neill and Stewartson (1967) for the first use of this stretching. The surface of the sphere with radius $a$ is described by

$$(\varepsilon Z - \varepsilon - 1)^2 + \varepsilon R^2 = 1 \ ,$$

and the solution of this can be expanded as

$$Z = H_1 + \tfrac{1}{8}\varepsilon R^4 + \tfrac{1}{16}\varepsilon^2 R^6 + O(\varepsilon^3) \ , \tag{2.3}$$

where $H_1 = 1 + \frac{1}{2}R^2$. Similarly the surface of the sphere with radius $b$ can be expanded as

$$Z = H_2 - \tfrac{1}{8}\varepsilon\kappa^3 R^4 - \tfrac{1}{16}\varepsilon^2\kappa^5 R^6 + O(\varepsilon^3) \ , \tag{2.4}$$

where $H_2 = -\frac{1}{2}\kappa R^2$, and we recall $\kappa = b/a$.

We give separate names $H_1$ and $H_2$ to the leading order approximations (which are paraboloidal approximations to the surfaces of the spheres) because it is on these surfaces that we shall apply all the boundary conditions later. The boundary conditions are applied here rather than at the exact surfaces because our approximations are simple power series in $\varepsilon$ and the exact equations for the surfaces contain square roots. Figure 1 illustrates some of the approximation scheme. Sections of the paraboloidal approximations $H_1$ and $H_2$ are shown as dashed lines.

We now look for a solution for $\Psi$ in the form of the expansion

$$\Psi = \Psi^{(0)} + \varepsilon\Psi^{(1)} + \varepsilon^2\Psi^{(2)} + O(\varepsilon^3) \ , \tag{2.5}$$

and derive equations for the $\Psi^{(i)}$. These quantities define a natural solution hierarchy, the members of which we shall calculate successively. The result will be a computation sequence for $\Psi$ with this hierarchical structure.

To express our equations in compact form, we introduce an operator

$$\Upsilon = \frac{\partial^2}{\partial R^2} - \frac{1}{R}\frac{\partial}{\partial R} \ ,$$

and then it can be shown (Jeffrey 1982) that

$$\frac{\partial^4}{\partial Z^4}\Psi^{(0)} = 0 \ , \tag{2.6}$$

$$\frac{\partial^4}{\partial Z^4}\Psi^{(1)} = -2\Upsilon\frac{\partial^2\Psi^{(0)}}{\partial Z^2} \ , \tag{2.7}$$

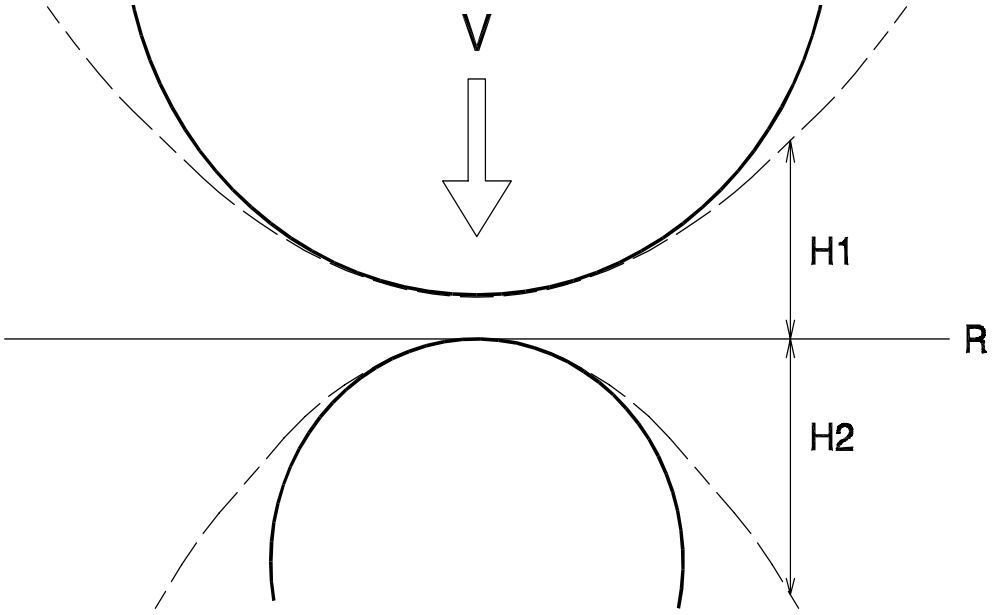


**Figure 1.** Cross-section of one sphere approaching another. The top sphere has radius $a$ and the bottom one has radius $b$. The dashed lines show paraboloidal approximations to the spheres. $H_1 - H_2$ is the total distance between the approximations.

$$\frac{\partial^4}{\partial Z^4}\Psi^{(2)} = -2\Upsilon\frac{\partial^2\Psi^{(1)}}{\partial Z^2} - \Upsilon^2\Psi^{(0)} , \tag{2.8}$$

and similarly for higher orders.

The solution of the first differential equation contains four arbitrary functions of $R$, which we label $A_0(R)$, $B_0(R)$, $C_0(R)$, and $D_0(R)$. These functions of $R$ will be determined by the boundary conditions. The solution for $\Psi^{(0)}$ is then

$$\Psi^{(0)} = A_0 Z^3 + B_0 Z^2 + C_0 Z + D_0 . \tag{2.9}$$

We find from the boundary conditions that

$$\begin{aligned} A_0 &= -R^2/H^3 , \\ B_0 &= \tfrac{3}{2}R^2(H_1 + H_2)/H^3 , \\ C_0 &= -3R^2H_1H_2/H^3 , \\ D_0 &= \tfrac{1}{2}R^2H_2^2(3H_1 - H_2)/H^3 . \end{aligned}$$

While solving the linear system to obtain $A_0$–$D_0$ we noticed that the matrix was a confluent Vandermonde matrix (see e.g. (Higham 1990)) with determinant $(H_1 - H_2)^4$. We therefore introduced the name $H = H_1 - H_2$. Notice that $H^3$ is the denominator of all the solutions. As well as reducing the size of the solution, $H$ has a geometrical significance, being the total distance between the parabolic approximations at any distance $R$ (recall that $H_2 < 0$, so $H_1 - H_2$ is the total distance).

At the next level in the solution hierarchy, we find

$$\Psi^{(1)} = -\tfrac{1}{10}Z^5\Upsilon A_0 - \tfrac{1}{6}Z^4\Upsilon B_0 + A_1 Z^3 + B_1 Z^2 + C_1 Z + D_1 , \tag{2.10}$$

where the arbitrary functions $A_1(R), \ldots, D_1(R)$ depend on $A_0, \ldots, D_0$, and involve $\Upsilon$. As

before these arbitrary functions are determined by the boundary conditions (and indeed as usual the matrix is the same as it was for the zeroth order). At this point an important aspect of simplification becomes evident. The expression for $A_1$, when it is first derived, contains several terms built from the coefficients $A_0$ and $B_0$. Specifically, the expression is

$$A_1 = (\tfrac{3}{10}H_1^2 + \tfrac{2}{5}H_1H_2 + \tfrac{3}{10}H_2^2)\Upsilon A_0 + (\tfrac{1}{3}H_1 + \tfrac{1}{3}H_2)\Upsilon B_0 \quad (2.11)$$

$$- \tfrac{1}{4}R^4 \frac{B_0 + 3A_0H_1 + \kappa^3 B_0 + 3\kappa^3 A_0 H_2}{H_1^2 - 2H_1H_2 + H_2^2} \ . \quad (2.12)$$

In the original paper, however, the second line of the above expression is simplified further and $A_1$ is published as

$$A_1 = \tfrac{1}{10}(3H_1^2 + 4H_1H_2 + 3H_2^2)\Upsilon A_0 + \tfrac{1}{3}(H_1 + H_2)\Upsilon B_0 + \tfrac{3}{8}(1-\kappa^3)R^6/H^4 \ . \quad (2.13)$$

In some parts of this expression, the coefficients $A_0$ and $B_0$ have been simplified away, while in other parts they have been left untouched. This kind of flexibility, instinctive to a mathematician, must be allowed in any CAS implementation. It would be difficult to anticipate this kind of simplification automatically, so it is important to include facilities in a CAS for allowing the user to evaluate subexpressions selectively.

Examining now the structure of the solution just derived, we see that the quantities $R$, $Z$, $\varepsilon$ and $\kappa$ are the independent variables. Some systems call them atoms, but we shall call them level 0 in a hierarchy. We can further divide this list into the primary independent variables $R$ and $Z$, and the parameters $\epsilon$ and $\kappa$. Thus we prefer to write $H_2(R)$ rather than the more explicit $H_2(R,\kappa)$. The quantities $H_1$ and $H_2$ are defined in terms of the level 0 variables, and hence form level 1 of the hierarchy. Then $H$ is defined at level 2, in terms of $H_1$ and $H_2$, and finally $A_0, \ldots, D_0$ form level 3 while $A_1, \ldots, D_1$ form level 4. The solutions $\Psi^{(i)}$ form a parallel hierarchy which is built up with the aid of the quantities in the first hierarchy. In addition there are assorted other quantities introduced for convenience, such as the operator $\Upsilon$.

## 2.2. The solution using Maple

In order first to derive and then to extend the solution above, we implemented a switch-controlled evaluation process in Maple, so that the quantities defined above can be reproduced in the system. This is done by replacing Maple expressions with Maple procedures that behave like the subexpressions above. As each procedure is defined, a Boolean-valued table entry is created whose value determines whether the procedure, when evaluated, returns simply its name, or the expression it represents. For example, we type

```
> let(H[1](R) = 1 + 1/2*R^2);
```

The procedure '`let`', automatically creates by side effect the new procedure '`H[1]`', which we present in Figure 2.

```
H[1] := proc()
    if assigned(allow_simplify[H[1]]) and allow_simplify[H[1]]  or
       assigned(allow_simplify['H[1]'(args)]) and
           allow_simplify['H[1]'(args)] then
        (R -> 1+1/2*R^2)(args)
    else
        'H[1]'(args)
    fi
end
```

**Figure 2.** The procedure created by `let` for $H_1(R) = 1 + R^2/2$.

Any reference to `H[1](R)` will be left unevaluated until we issue the `reveal` command to reveal its contents†.

```
> H[1](R)^2;
```

$$H_1(R)^2$$

```
> reveal(H[1]): H[1](R)^2;
```

$$(1 + \tfrac{1}{2}R^2)^2$$

We illustrate the use of the `let` command with the following short Maple session, in which $\Psi_0$ is calculated. We assume that $H_1$ and $H_2$ have been defined, as above, earlier in the session. The first line of the code below lets us use the single letter names $\Psi$ and $D$ without interference from Maple.

```
> alias(Psi=PSI,D=DD):
> Psi[0]:= A[0](R)*Z^3+B[0](R)*Z^2+C[0](R)*Z+D[0](R);
```

$$\Psi_0 := A_0(R)\,Z^3 + B_0(R)\,Z^2 + C_0(R)\,Z + D_0(R)$$

```
> bc1:= subs(Z=H[1](R),Psi[0])=R^2/2:
> bc2:= subs(Z=H[1](R),diff(Psi[0],Z))=0:
> bc3:= subs(Z=H[2](R),Psi[0])=0:
> bc4:= subs(Z=H[2](R),diff(Psi[0],Z))=0:
> sol:=solve({bc1,bc2,bc3,bc4},{A[0](R),B[0](R),C[0](R),D[0](R)}):
```

$$\left\{ A_0(R) = -2\,\frac{R^2}{\%1},\ B_0(R) = 3\,\frac{(H_1(R) + H_2(R))\,R^2}{\%1},\ C_0(R) = \right.$$
$$\left. -6\,\frac{R^2\,H_1(R)\,H_2(R)}{\%1},\ D_0(R) = \frac{R^2\,H_2(R)^2\,(-H_2(R) + 3\,H_1(R))}{\%1} \right\}$$

† The `reveal` command simply assigns the value `true` to the appropriate table entry, which allows Maple to see the definition of the symbol.

where

$$\%1 = H_1(R)^3 - 3\,H_1(R)^2\,H_2(R) - H_2(R)^3 + 3\,H_2(R)^2\,H_1(R)\,.$$

We suspect that the expression denoted by %1 could be simpler, so we try to factor it.

```
> factor( %1);
```

$$(H_1(R) - H_2(R))^3$$

This is why we decide to include a separate name for $H(R) = H_1(R) - H_2(R)$. We here omit details of how we replace $H_1(R) - H_2(R)$ with $H(R)$. We then define procedures for $A_0$, $B_0$, $C_0$, and $D_0$ by the following command.

```
> map(let, sol);
```

$$\{A_0(R),\ B_0(R),\ C_0(R),\ D_0(R)\}$$

We have given a more detailed discussion of the technicalities of the `let` command in a note submitted elsewhere (Corless *et al.* 1996). To see the details of the Maple solution extended to higher orders, see Pratibha (1995).

To illustrate the `reveal` command, we give a Maple session that verifies the boundary conditions $\Psi_0 = R^2/2$ on $Z = H_1$, and $\partial\Psi_0/\partial Z = 0$ on $Z = H_1$. We start with the second condition.

```
> bc2 := subs(Z=H[1](R),diff(Psi[0],Z));
```

$$3A_0(R)H_1^2(R) + 2B_0(R)H_1(R) + C_0(R)$$

The command `reveal` sets the value of the requisite Boolean-valued table entries so that the arguments of `reveal` are replaced by their defining expressions.

```
> reveal([A[0], B[0], C[0]]):
> simplify(bc2);
```

$$0$$

Notice that in this example the contents of $H$, $H_1$ and $H_2$ did not have to be revealed in order to verify the boundary condition. Thus if the system had worked with the longer expressions implied by the level 0 variables, it would have been doing unnecessary work. The verification of the boundary condition$\Psi_0 = R^2/2$ on $Z = H_1$, in contrast, requires one more simplification switch to be turned on.

```
> bc1 := subs(Z=H[1](R),Psi[0] ): reveal(D[0]): simplify(bc1);
```

$$\frac{R^2(H_1(R)^3 - 3H_1(R)^2H_2(R) + 3H_2(R)^2H_1(R) - H_2(R)^3)}{2H(R)^3}$$

```
> reveal(H): simplify("");
```

$$R^2/2$$

A comparison of the computing resources required to compute the solution to second order in $\varepsilon$ show that in spite of the extra overhead associated with procedure calls, both time and memory requirements were reduced by roughly 50% over the naive approach. Of course, the benefits multiply as the order increases. As well as the straight gain in resources, there is a gain in intelligibility of the output. This is important to a mathematician, but it is harder to measure. However, the solutions produced in this way are more compact, more readable, and offer more physical insight than the naive solutions.

## 3. A convection calculation using expression sequences

The problem of calculating the convective flow in the annular region between two cylinders is studied by Mack and Bishop (1968) and Corless and Naylor (1991). Similar work for porous flow, using a seminumerical approach, was done by Himasekhar and Bau (1988). The problem is to find the temperature field $T$ and the streamfunction $\psi$ (related to the velocity field) in terms of the polar coordinates $(r, \theta)$ and the non-dimensional parameters $R$, the radius ratio, $P$ the Prandtl number and $A$ the Rayleigh number.

Mack and Bishop (1968) derived equations valid in the semi-annular region $1 \leq r \leq R$, $0 \leq \theta \leq \pi$.

$$\nabla^4\psi = A\left[\sin\theta\frac{\partial T}{\partial r} + \frac{\cos\theta}{r}\frac{\partial T}{\partial \theta}\right] + \frac{1}{rP}\left[\frac{\partial \nabla^2\psi}{\partial r}\frac{\partial \psi}{\partial \theta} - \frac{\partial \nabla^2\psi}{\partial \theta}\frac{\partial \psi}{\partial r}\right] \tag{3.1}$$

$$\nabla^2 T = \frac{1}{r}\left[\frac{\partial T}{\partial r}\frac{\partial \psi}{\partial \theta} - \frac{\partial T}{\partial \theta}\frac{\partial \psi}{\partial r}\right] \tag{3.2}$$

The equations are subject to the boundary conditions:

$$T(1,\theta) = 1 \qquad T(R,\theta) = 0 \tag{3.3}$$

$$\psi(1,\theta) = \psi(R,\theta) = \frac{\partial \psi}{\partial r}(1,\theta) = \frac{\partial \psi}{\partial r}(R,\theta) = 0 \tag{3.4}$$

$$\frac{\partial T}{\partial \theta}(r,0) = \psi(r,0) = \frac{\partial^2 \psi}{\partial \theta^2}(r,0) = 0 \tag{3.5}$$

$$\frac{\partial T}{\partial \theta}(r,\pi) = \psi(r,\pi) = \frac{\partial^2 \psi}{\partial \theta^2}(r,\pi) = 0 \tag{3.6}$$

Mack and Bishop solve these equations by expanding all quantities in terms of the Rayleigh number $A$ first and then expanding those coefficients themselves in Fourier series. Thus the first expansions are

$$T = \sum_{k=0}^{\infty} A^k T_k(r,\theta)\ , \qquad \psi = \sum_{k=1}^{\infty} A^k \psi_k(r,\theta)\ . \tag{3.7}$$

Then the Fourier series, which are actually finite, are given by

$$T_k(r,\theta) = \sum_{m \equiv k \bmod 2}^{k} T_k^m(r)\cos(m\theta)\,, \tag{3.8}$$

and

$$\psi_k(r,\theta) = \sum_{m \equiv k \bmod 2}^{k} \psi_k^m(r)\sin(m\theta)\,. \tag{3.9}$$

At this point we note that the chosen method of solution has imposed a hierarchy on the problem already. The method of computing the solution that we describe below exploits this natural hierarchy.

Extending the work of Mack and Bishop (1968) by naive use of computer algebra systems runs into the problem of combinatorial growth in the solution. A straightforward use of Maple results in a nearly 3000-term expression for a single coefficient in the fourth-order term of the solution. Computation sequences must be used to permit reasonable extension of the hand calculations. We note that the hand calculation of Mack and Bishop (1968) also produced a computation sequence for an answer. This type of solution occurs often in applied mathematics.

Substituting equations (3.7–3.9) into the governing equations (3.1–3.2) gives a sequence of equations at each order in $A$. If we define $P_k$ and $Q_k$ by

$$Q_k(r,\theta) = \sin\theta\frac{\partial T_{k-1}}{\partial r} + \frac{\cos\theta}{r}\frac{\partial T_{k-1}}{\partial \theta} + \frac{1}{rP}\sum_{i+j=k}\left(\frac{\partial \nabla^2\psi_i}{\partial r}\frac{\partial \psi_j}{\partial \theta} - \frac{\partial \nabla^2\psi_i}{\partial \theta}\frac{\partial \psi_j}{\partial r}\right) \tag{3.10}$$

and

$$P_k(r,\theta) = \frac{1}{r}\sum_{i+j=k}\left(\frac{\partial T_i}{\partial r}\frac{\partial \psi_j}{\partial \theta} - \frac{\partial T_i}{\partial \theta}\frac{\partial \psi_j}{\partial r}\right)\,, \tag{3.11}$$

then the coefficients of $A^k$ in governing equations become (using $\left[A^k\right]\ f$ to denote the coefficient of $A^k$ in the expression $f$)

$$\left[A^k\right]\ (3.1) := \nabla^4\psi_k(r,\theta) = Q_k \tag{3.12}$$

and

$$\left[A^k\right]\ (3.2) := \nabla^2 T_k(r,\theta) = P_k\,. \tag{3.13}$$

Similarly using $[\cos(m\theta)]\ f$ to denote the coefficient of $\cos(m\theta)$ in the expression $f$, define

$$Q_k^m(r) = [\sin(m\theta)]\ Q_k(r,\theta) \tag{3.14}$$

and

$$P_k^m(r) = [\cos(m\theta)]\ P_k(r,\theta)\,. \tag{3.15}$$

Equating coefficients of $\cos(m\theta)$ in $\nabla^2 T_k(r,\theta)$ and $P_k(r,\theta)$ and coefficients of $\sin(m\theta)$ in $\nabla^4\psi_k(r,\theta)$ and $Q_k(r,\theta)$ gives, then, the following sequence of Euler differential equations to be solved for the unknown coefficients $T_k^m(r)$ and $\psi_k^m(r)$:

$$\frac{1}{r^2}\left(r\frac{d}{dr}+m\right)\left(r\frac{d}{dr}-m\right)T_k^m(r) = P_k^m(r) \tag{3.16}$$

$$\frac{1}{r^4}\left(r\frac{d}{dr}+m\right)\left(r\frac{d}{dr}-m\right)\left(r\frac{d}{dr}-m-2\right)\left(r\frac{d}{dr}+m-2\right)\psi_k^m(r) = Q_k^m(r)\,. \tag{3.17}$$

One can easily prove by induction that $P_k^m$ and $Q_k^m$ are always sums of terms of the form $C_i r^\alpha \ln^\mu r$ for some integers $\alpha$ and $\mu$. This uses the fact that analytical solutions to these inhomogeneous Euler equations are available, and the solutions are again sums of the same type of terms.

For efficiency, special-purpose solvers were written to take advantage of the factored form of these equations and the known form of the inhomogeneities. This improved the overall computation time, but further improvements are necessary, because it is the length of the explicit expressions for the $C_i$ which suffer from combinatorial growth. Generation of these explicit expressions, then, is to be avoided. Use of computation sequences for the $C_i$ is appropriate, as we shall see.

Unknown constants $K_i$ are introduced as each equation is solved: two for each temperature equation, and four for each stream-function equation. The constants $K_i$ are identified by using the boundary conditions. Since the boundary conditions (3.3–3.4) are linear, we must solve a (nonsingular if $R \neq 1$) linear system of equations, at each stage, exactly as was done for the lubrication calculation in the first half of the paper.

The linear system for each $T_k^m$ is 2 by 2, and is 4 by 4 for each $\psi_k^m$. If explicit expressions for the $K_i$, in terms of the $C_i$, are generated, then the size of the expressions determining the $K_i$ is approximately doubled, over the length of the expressions arising in the linear equations defining $K_i$. Leaving the $K_i$ defined as "the solutions of such-and-such a linear system", then, is a reasonable approach, given that effective means exist for solving linear systems numerically. This means that our computation sequence uses the solution of linear systems of equations as a basic operation of the sequence.

The calculation here is broken up into several stages, and it is possible to do some 'gardening' or organization at the completion of each stage. What is done is to collect the solution in powers of $r$ and $\ln r$, and each (moderately complicated) coefficient is then given an inert label $C_i$, where $i$ is chosen as the least unused integer so far. The solution is then represented as a sum of terms of this type, and the actual value of this coefficient is remembered in an array, called '`Computation_Sequence`' in our implementation. The Maple procedures used to do this relabeling are as follows:

```
flatten := proc(expr) collect(expr,[r,ln(r)],distributed,Weed) end:
```

This procedure 'flattens' an expression by first collecting terms of like powers of $r$ and $\ln r$, then calls `Weed` (given below) to replace the coefficients with unevaluated constants, and record the values of these constants in the expression sequence `Computation_Sequence`. This uses the fourth argument of collect, which applies the procedure named in the fourth argument to each coefficient after it has been collected. The procedure `Weed` (see Figure 3) simply replaces its argument with an unassigned constant from the array C, and remembers in the computation sequence what the actual argument was. Weed leaves products alone, so as not to introduce new constants for (say) $2C_1$ and $3C_1$, which would introduce unnecessary growth in the number of coefficients in the computation sequence. This procedure makes no attempt to identify already-seen subexpressions; experiments indicate that this is not helpful for the current application.

What follows is a brief overview of the algorithm used for solving this problem.

1 Set $T_0^0 = K_1 + K_2 \ln r$, and $\psi_0^0 = 0$.
2 for $k = 1, 2, \ldots, N$ do

(a) for all values of $m$ congruent to $k$ mod 2 in $0, 1, \ldots, k$ do

(i) Solve (3.17) for $\psi_k^m$ using the specialized Euler equation solver.

```
Weed := proc(term)
  local c,i,s:
  global Weed_Index, Computation_Sequence, C;
  c := normal(term);  # Recognize zero if you see it.
  if c=0 then RETURN(0) fi;
  i := icontent(c);
  s := sign(c);
  c := s*c/i;
  if hastype(c,`+`) then
    Weed_Index := Weed_Index + 1:
    Computation_Sequence[Weed_Index] := c:
    s*i*C[Weed_Index]
  else s*i*c
  fi
end:
```

**Figure 3.** Maple utility program for automatically generating a computation sequence when used in conjunction with `collect`.

(ii) 'flatten' the solution for $\psi_k^m$ (i.e. replace the coefficients of all terms with placeholder constants).

(b) Set up (but do not solve) the linear equations for the unknown $K$ constants introduced for $\psi_k$.

(c) for all values of $m$ congruent to $k$ mod 2 in $0, 1, \ldots, k$ do

(i) Solve (3.16) for $T_k^m$ using the specialized Euler equation solver.

(ii) 'flatten' the solution for $T_k^m$.

(d) Set up (but do not solve) the linear equations for the unknown $K$ constants introduced for $T_k$.

Notice that 'flattening' is done in the inner loops, keeping expressions as small as possible. Indeed, we have found that it is yet more efficient to do at least some 'flattening' inside the construction of each $\psi_k^m$ and $T_k^m$, though this is not stated in the above algorithm sketch. As an example of the output of this scheme, we include the computation-sequence representation for the first two terms of the temperature function and the stream function.

$$T_0^0 = K_1 + K_2 \ln r$$

$$\begin{aligned} \psi_1^1 &= -\tfrac{1}{32} C_1 r^3 + K_5 r + \tfrac{1}{r} K_3 + \tfrac{1}{16} K_2 r^3 \ln(r) + K_6 r \ln(r) \\ T_1^1 &= -\tfrac{1}{512} C_3\, r^3 + K_8\, r - \tfrac{1}{4r} C_5 + \tfrac{1}{128} {K_2}^2\, r^3 \ln(r) \\ &\quad - \tfrac{1}{2r} K_2\, K_3 \ln(r) - \tfrac{1}{4} C_4 \ln(r)\, r + \tfrac{1}{4} K_2\, K_6\, r \ln(r)^2 \end{aligned}$$

where the coefficients $C_i$ are given by the computation sequence

$$\begin{aligned} C_1 &= K_2 - 32\, K_4\,, \\ C_2 &= K_2\,(K_2 - 32\, K_4)\,, \\ C_3 &= 3\, {K_2}^2 + 2\, C_2\,, \\ C_4 &= K_2\, K_6 - 2\, K_2\, K_5\,, \text{and} \\ C_5 &= -4\, K_7 + K_2\, K_3\,. \end{aligned}$$

**Table 1.** Size of expressions without and with computation sequences

| Terms | Full Evaluation | computation sequence |
|---|---|---|
| $T_0^0$ | 2 | 2 |
| $\psi_1^1$ | 5 | 5 |
| $T_1^1$ | 11 | 7 |
| $\psi_2^2$ | 34 | 11 |
| $T_2^0, T_2^2$ | 130 | 33 |
| $\psi_3^1, \psi_3^3$ | 396 | 42 |
| $T_3^1, T_3^3$ | 1027 | 58 |
| $\psi_4^2, \psi_4^4$ | 1921 | 67 |
| $T_4^0$ | 2786 | 45 |

The constants $K_i$ are determined by the known linear systems arising from the boundary conditions. As previously discussed, this determination is left as part of the computation sequence. Note that the expression for $C_2$ contains the expression for $C_1$ as a subexpression. This was not noticed by the program because both the expressions $K_2 - 32K_4$ and $K_2(K_2 - 32K_4)$ were generated at the same time, in the expression for $\psi_1^1$, before the name $C_1$ was created. Coefficients identified within a single expression are not optimized with respect to each other. Further, retrospective optimization of the computation sequence is not performed. Note also that $C_2$ has in fact disappeared from the expression for $\psi_1^1$, having cancelled out, but was used in intermediate calculations.

Evaluation of these expressions is straightforward, once numerical values for $R$, the radius ratio, and $P$, the Prandtl number, are assigned. The evaluation proceeds in sequence, starting at the index 1. When the constants $K$ are encountered, the linear systems defining them are solved numerically. Thus $K_1$ and $K_2$ are defined before any $C_i$ which depends on them is computed.

The important gain with this system is seen in Table 1 and in Figure 4. In Table 1, the number of terms in the expanded solution is compared with the number in the compacted solution. The significant decrease in the number of terms can be seen both there and in Figure 4, which plots the number of terms in the solution against the order. The plot uses logarithmic scales and so it is easy to see that the number of terms grows on average like $n^2$, or at least not faster than $O(n^3)$.

### 3.1. Simplification of Computation Sequences

To verify that the solution as computed does in fact satisfy the differential equation and the boundary conditions, it is necessary to substitute our putative solution into the governing equations, and to attempt to simplify the resulting residual expressions to zero. There are several approaches to this simplification. First, we could simply assign all the elements of the computation sequence, and rely on the underlying CAS recursive evaluation of expressions to try to simplify the residual to zero. This works for small expressions, but actually results in the internal generation of the large expressions avoided by the process of construction of the computation sequence. Secondly, we could assign

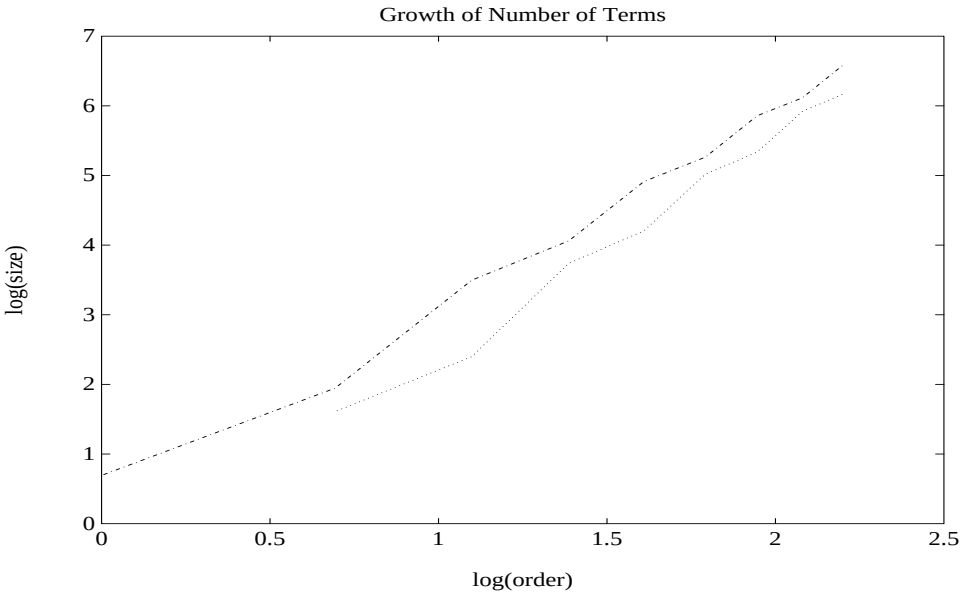


**Figure 4.** Polynomial growth of number of terms in the solution to the concentric cylinder problem. The index $k$ is plotted on the (logarithmically scaled) horizontal axis, while the number of terms in each coefficient of $A^k$ is plotted on the vertical axis. The curve is uniformly below $12k^3$ (not shown), and quite comparable to $12k^2$ (the growth appears to be *slightly* faster than quadratic, however).

several of the lowest-index members of the computation sequence (e.g. $C_1 = K_2 - 32\,K_4$, ..., $C_6 = -\frac{128}{P}\,K_2\,K_4 + \frac{1}{P}\,{K_2}^2 + {K_2}^2$, and so on) and try to simplify the resulting residuals, repeating the process with the next-lowest index members assigned, and so on. Finally, we could assign instead several of the highest-index members of the computation sequence, then simplify, collect terms, and assign the next-highest elements of the computation sequence. Clearly mixed approaches are also possible, using the `let` and `reveal` commands of the previous sections. It is not clear what strategy is optimal, but there are some heuristic reasons to prefer the 'top-down' strategy. It is entirely possible that the expressions may simplify to zero before assignment of the lower-index elements takes place, for a variety of reasons, while in the bottom-up strategy you must make all the assignments up to the index of the $C_i$ with the highest index. This is related to the fact that most of the $C_i$ are defined in terms of "recent" or "local" $C_i$. Experimentation with this process on this problem showed that when simplifying the residuals computed only to the third order, the first two techniques, simple assignment and bottom-up assignment with simplification, failed due to memory limitations, whereas the top-down approach was successful in verifying that the computed solutions did in fact satisfy the equations.

Zero recognition is especially important. Obviously we wish to avoid division by elements that are actually zero. Even in the absence of division, such as in this application, later evaluation of an undetected zero term is possibly subject to catastrophic cancellation error. In view of this, it might seem that an alternative strategy for the generation of the sequence in the first place would be the substitution of numerical values for some of the parameters so that the $C_i$ and the $K_i$ could be evaluated numerically as we proceed. This can be done, but it does not lead to significant improvement; quite the reverse, be-

cause we encounter a new phenomenon: *round-off induced expression swell*. Coefficients that should be exactly zero are not precisely so, and this leads to a greatly increased number of terms. For example, suppose that the term $10^{-8} r^3 \ln^3(r)$ arises in some part of $T_2$. Then in the solution for $\Psi_3$ at some point we solve a fourth-order Euler equation with this small term on the right hand side, producing

```
> streamsol( 10.^(-8)*r^3*ln(r)^3 , 3);
```

which yields the four spurious terms

$$-0.2242476852 \cdot 10^{-10}\, r^3 - 0.1736111111 \cdot 10^{-10}\, r^3\, \ln(r)^2$$
$$-0.7233796296 \cdot 10^{-10}\, r^3\, \ln(r) - 0.6944444444 \cdot 10^{-10}\, r^3\, \ln(r)^3$$

for inclusion in $\Psi_3$. These terms then lead to more terms, and so on. This effect combines the worst of both numeric and symbolic computation, and hence we recommend the purely symbolic approach to generate the sequence. Once the computation sequence has been generated, numerical values may be used for the atoms and the computation may proceed.

For verification of the solution, we did that, substituting numerical values of $R$ and $P$ into the computation sequence and seeing if the computed solution satisfied the differential equation to the correct order in the Rayleigh number $A$. It did, and the residual was zero to within roundoff—which decreased as we increased the precision of the calculation, as it should have.

An alternative approach that we implemented to test if elements of the computation sequence are zero is to use modular arithmetic (Monagan 1989). By choosing random integers for each of $R$, $1/P$, and $\ln(R)$, solving the resulting linear systems for the $K_i$ mod $p$, where $p$ was a suitably chosen large prime, we generated integer values for the computation sequence. When we did this, we found that the boundary conditions were satisfied exactly, mod $p$, and that the partial differential equations were satisfied exactly, mod $p$, up to the order of calculation. This provided an independent verification of the solution method. We were also interested in whether or not any of the entries in the computation sequence was zero, which might indicate that the term would be zero for all $R$ and $P$. However, no entry was zero, which proves that there are no unnecessary zeros in the computation sequence (to the computed order).

This raises the question of what the goals of these simplifications should be. Simplification of expressions has a rather vague goal, that of producing a more comprehensible expression; one that is usually shorter, but not necessarily so. Simplification of generalized representations of functions, such as computation sequences, has several possibly conflicting goals: we wish our representations to be compact, efficient to evaluate, and numerically stable. Production of such a representation would be likely to give insight into the nature of the problem, as well, but this may be regarded as a side-effect and would certainly be hard to quantify. In many cases these are compatible goals, but it would be very useful to have standardized tools for evaluating the comparative stability of representations, for example as in (Mutrie *et al.* 1989).

## 4. Concluding Remarks

There are three main points of interest in this paper, and a novel phenomenon observed that may be seen more widely if this type of technique is used more frequently. The main points are that

1 We have provided tools for interactive user control of evaluation of expressions.

2 We have demonstrated techniques for the automatic generation of computation sequences, once a hierarchy has been established interactively.

3 We have shown that apparently minor issues can have a significant effect on system performance.

The novel phenomenon is that of *roundoff-induced expression swell*, which may happen if numerical and symbolic computation are mixed injudiciously in a large problem. This mixes the worst of both types of computation, and should be avoided wherever possible.

Using two problems drawn from fluid mechanics, both of which suffer from expression swell, we have demonstrated techniques for reducing the sizes of the computed expressions by changing the way in which the solution is represented and manipulated. We have developed tools for constructing trees of expressions, ordered hierarchically, together with tools for simplifying intermediate expressions to whatever level of the hierarchy gives the best overall result. The tools at present are implemented on top of Maple rather than within it, and their efficiency and ease of use could be increased by integrating them more closely into the basic system. We have not considered the symbolic manipulation of computation sequences themselves, although this is also desirable. In (Freeman et. al. 1986) a package is described which implements several useful procedures for the *manipulation* of computation sequences, such as computing polynomial GCD's. Recent work (Díaz and Kaltofen 1995) shows that this area is still active.

Tools such as Maple's 'optimize' command, or even special-purpose post-processing software (for example that of (Budgell and El Maraghy 1990)) are not appropriate for the present application, because the intermediate quantities on which they operate cannot be obtained, because an out of memory error occurs at the third order. With computation sequences, we can go at least to eleventh order. It is better to avoid the generation of large expressions in the first place, if at all possible.

Of course it is clear that the idea of identifying common subexpressions is useful in several contexts. The Maple pretty-printer uses this idea for presentation purposes, and in this way identified the expanded form of the subexpression $(H_1-H_2)^3$ for us in section 2.2. Applied to the computation sequence generated for this example, it can find 257 common subexpressions even at the 4th order, and it may be possible to take advantage of this in some fashion to fine-tune this program. The Maple 'optimize' command uses the same idea as the pretty-printer to identify common subexpressions; it then constructs a computation sequence for the given expression, in order to speed up and/or stabilize numerical evaluation. We term this strategy a 'janitorial' optimization strategy, because the program is attempting to clean up an existing messy expression. More sophisticated strategies using automatic code generation as well as janitorial strategies have been described in (Wang *et al.* 1986) and applied to finite element calculations. As an aside, in contrast with the finite element code generation approach, our approach determines the *form* of the solution symbolically as the solution progresses, and code is only generated for the coefficients of the terms in that symbolic form.

Our approach contrasts with the general idea of a janitorial strategy in that the subexpressions are identified as they arise and used as necessary subsequently. We call this type of strategy a 'gardening' strategy, where the analogy is to 'weeding' the garden. If you do a little weeding every day, you never have a massive clean-up job to do.

## References


Boyce, W. E., Ecker, J. G. (1992). Revitalising calculus with a computer algebra system. *Siam News*, January:12.

Budgell, P. C., El Maraghy, W. H. (1990). Inverse dynamics of the Stanford arm developed with computer symbolic algebra. In *Proceedings CSME Mech. Eng. Forum*, volume III. Toronto, Canada.

Cooley, M. D., O'Neill, M. E. (1969). On the slow motion generated in a viscous fluid by the approach of a sphere to a plane wall or stationary sphere. *Mathematika*, 16:37–49.

Corless, R. M., Jeffrey, D. J., Monagan, M. B., Pratibha (1996). Substitution in Maple, or, what's inside a name? *submitted*.

Corless, R. M., Naylor, D. (1991). Low Rayleigh number convection between horizontal concentric cylinders. Technical Report AM-91-02, Dept. Applied Math, University of Western Ontario, London, CANADA.

Delaunay, C. E. (1867). *Théorièe du Mouvement de la Lune*, volume 1,2. Mallet-Bachelier, Paris.

Deprit, A., Henrard, J., Rom, A. (1970). Lunar ephemeris: Delaunay's theory revisited. *Science*, 168:1569–1570.

Díaz, A., Kaltofen, E. (1995). On computing greatest common divisors with polynomials given by black boxes for their evaluations. In Levelt, A., editor, *Proceedings ISSAC '95, Montréal*, pages 232–239.

Freeman, T., Imizian, G., Kaltofen, E. (1986). A system for manipulating polynomials given by straight-line programs. In *Proceedings ISSAC '86, Waterloo*.

Higham, N. J. (1990). Stability analysis of algorithms for solving confluent Vandermonde-like systems. *SIAM J. Matrix Anal. Appl.*, 11(1):23–41.

Himasekhar, K., Bau, H. H. (1988). Two-dimensional bifurcation phenomena in thermal convection in horizontal, concentric annuli containing saturated porous media. *J. Fluid Mech.*, 187:267–300.

Jeffrey, D. J. (1982). Low Reynolds-number flow between converging spheres. *Mathematika*, 29:58–66.

Mack, L. R., Bishop, E. H. (1968). Natural convection between horizontal concentric cylinders for low Rayleigh numbers. *Quart. J. Mech. Appl. Math*, 21:223–241.

Monagan, M. B. (1989). *Signatures + Abstract Data Types = Computer Algebra − Intermediate Expression Swell*. PhD thesis, Dept. Comp. Sci., University of Waterloo, Waterloo, Canada.

Mutrie, M. P. W., Char, B. W., Bartels, R. H. (1989). A survey of expression optimization in a symbolic-numeric interface. In Davenport, J. H., editor, *Proceedings Eurocal '87, Springer LNCS series nr. 378*, pages 64–70.

O'Neill, M. E., Stewartson, K. (1967). On the slow motion of a sphere parallel to a nearby plane wall. *J. Fluid Mech.*, 27:705–724.

Pratibha (1995). *Maple tools for hydrodynamic interaction problems*. PhD thesis, Department of Applied Mathematics, University of Western Ontario, London, CANADA.

Van Dyke, M. (1974). Computer extension of perturbation series in fluid mechanics. *SIAM Journal Appl. Maths.*, 28:720–734.

Wang, P. S., Tan, H., Saleeb, A. F., , Chang, T. P. (1986). Code generation for hybrid mixed mode formulation in finite element analysis. In *Proceedings ISSAC '86, Waterloo*.

Zippel, R. (1993). *Effective Polynomial Computation*. Kluwer Academic.