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 $[\{x\}, \{$

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

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

The equation for Ψ 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 ε 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 (1 67) 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

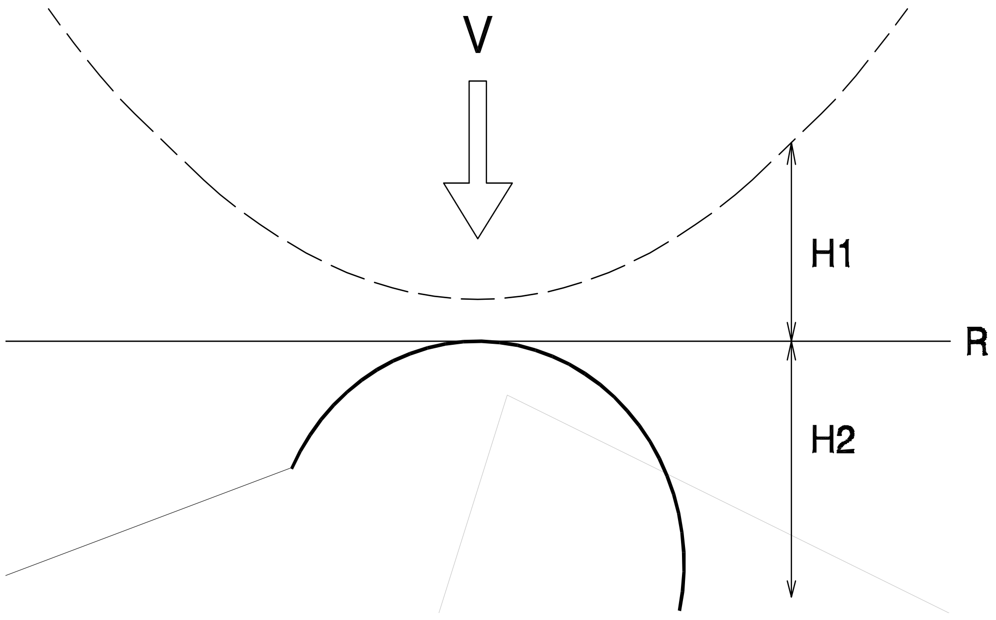
$$Z = H_1 + \tfrac{1}{8}\varepsilon R^4 + \tfrac{1}{16}\varepsilon^2 R^6 + O(\varepsilon^3) \ , \tag{2.3}$$

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

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

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

We give separate names H_1 and H_2 to the leading order approximations (which are paraboloidal approximations to [illegible]


V
H1
R
H2

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

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

$- \,^{1}$

$\frac{}{10}(3H$

```
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 Ψ_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 Ψ 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)}):
```

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

[†] 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.

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

$$R^2/2$$

A comparison of the computing resources required to compute the solution to second order in ε 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

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

$$T \quad i \;\&\quad i \;\&$$

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 α and μ. 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 whic at P

```
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 ψ_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 ψ_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 ψ_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

Table 1. Size of expressions without and with computation sequences

Terms	Full Evaluation	computation sequence
T_0^0	2	2
ψ_1^1	5	5
T_1^1	11	7
ψ_2^2	34	11
T_2^0, T_2^2	130	33
ψ_3^1, ψ_3^3	396	42
T_3^1, T_3^3	1027	58
ψ_4^2, ψ_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 ψ_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 ψ_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

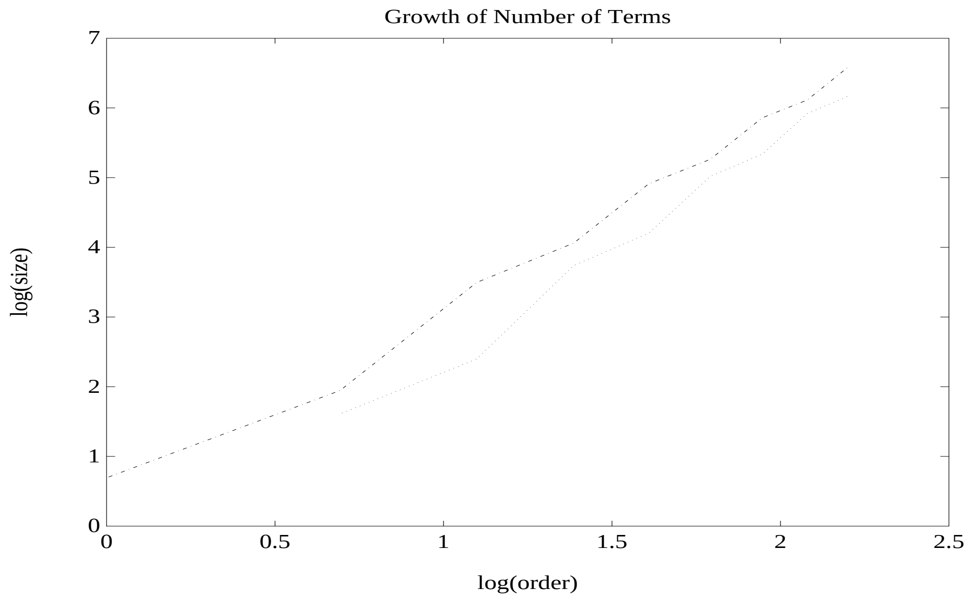


Figure 4. Polynomial growth of number of terms in the solution to the concentric cylinder problem. The index k is plotted on the (logarithmically scaled) horizontal axis, while the number of terms in each coefficient of A^k is plotted on the vertical axis. The curve is uniformly below $12k^3$ (not shown),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

several of the lowestcause 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 Ψ_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

$$\begin{aligned} &-0.2242476852\cdot 10^{-10}\,r^3 - 0.1736111111\cdot 10^{-10}\,r^3\,\ln(r)^2 \\ &\quad -0.72337\ \ 62\ \ 6\cdot 10^{-10}\,r^3\,\ln(r) - 0.6\ \ 44444444\cdot 10^{-10}\,r^3\,\ln(r)^3 \end{aligned}$$

for inclusion in Ψ_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 1 8). 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.* 1 8).

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 expres-

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 *Proceedin[illegible]s CSME Mech. En[illegible]. 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). *Théoriè 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 ev m [illegible] fen, [illegible] ondon, [illegible]