Checking for errors in calculations and software: Dimensional balance and conformance of units

Scott Ferson
Applied Biomathematics
100 North Country Road
Setauket, New York 11733
Telephone 631-751-4350
Facsimile 631-751-3435
Email scott@ramas.com

Running head / abbreviated title: Dimension and unit checking

In press, *Accountability in Research: Policies and Quality Assurance*

Checking for errors in calculations and software: Dimensional balance and conformance of units

ABSTRACT

Although there has always been a general awareness that mathematical expressions must make dimensional sense in terms of the units involved, it is very easy to make simple mistakes in quantitative work that result in profound and potentially dangerous errors. Such errors are ubiquitous in modern research, as can be seen by reviewing government publications where dimensional errors persist despite peer and public review. Software methods have recently become available for checking calculations, equations, algorithms and programs for dimensional soundness. Correctness depends on conformance at two levels: balance of dimensions and agreement among units. Error at either level can now be detected automatically by software. Disagreement among units can even be automatically corrected by software procedures. These software tools can be used to check for errors in calculations and software source code in a way that is similar to using a spelling or grammar checker for text.

KEYWORDS

dimensional analysis, agreement of units, quality assurance for calculations and algorithms

INTRODUCTION

While the widespread use of computers and calculators has surely reduced the incidence of numerical errors in calculations (see Wise, 1995), there are non-numerical errors that cannot be eliminated by traditional software. One fundamental consideration for almost any calculation or equation, numerical or not, is whether the dimensions and units balance appropriately. If they do not, the expression is nonsensical and could not possibly be correct no matter what numerical values it may contain (Hart, 1995; Edwards and Hamson, 1989; Wilson, 1952; inter alia). Although such errors are profound ones, there has heretofore been no way to check for them except by an ad hoc manual (and often painstaking) analysis. This paper describes recent software developments that can now automate this checking by performing dimensional analyses as well as a human analyst, but without the risk of careless mistakes.

DIMENSIONAL BALANCE AND UNIT CONFORMANCE

Virtually all textbooks on model building (e.g., Wilson, 1952; Bratley et al., 1983; Swartzman and Kaluzny, 1987; Edwards and Hamson, 1990; Adler, 1993) caution that mathematical expressions must make dimensional sense in terms of the units involved. The central rule is that terms to be added together or compared with one another must have balanced dimensions. In short, one may not add apples and oranges. In the following expressions, a and b must have balanced dimensions and conformable units.

$a + b$	$a < b$	$a \geq b$	$\max(a,b)$
$a - b$	$a > b$	$a \leq b$	$\min(a,b)$

Note that balance does not mean the units or even dimensions must be identical. For instance, ‘meter gram’ and ‘gram meter’ balance, as do ‘meter^2’, ‘square meter’ and ‘meter^3/meter’.

Moreover, the dimension [velocity] balances with [distance][time]$^{-1}$. Although easily appreciated by a human analyst, these balances can be difficult for a checking algorithm to recognize.

Having balanced dimensions among terms is, by itself, insufficient for an addition, subtraction or comparison to make sense. For instance, Russian rubles and U.S. dollars both measure monetary value, but until quite recently, they were not legally exchangeable and therefore one could not speak strictly about comparing quantities measured in the two currencies. Baud rate and bits per second are another example. Both measure the dimension [information flow], and yet there is no universal conversion between the two units. Thus, dimensional balance is weaker than unit conformance, which is what is actually required to make computational sense of additions, subtractions and comparisons.

There are other rules required for dimensional soundness. For instance, dimensionless numbers can be additively combined or compared only with other dimensionless numbers. Thus, dimensionlessness is itself essentially another dimension. Moreover, certain quantities, such as numbers to be used as powers, must be dimensionless. In the expressions below, for example, the argument a must be dimensionless.

b^a	$\ln(a)$	$\sin^{-1}(a)$	$a + 3$	$a^2 + a$
$\exp(a)$	$\log_{10}(a)$	$\arctan(a)$	$a < 4$	$a^3 < 8$

Again, all that is really required is that the dimensions of the argument a balance with a dimensionless quantity. Dimensionlessness does not necessarily mean unitlessness. For example, 4% and '3 meter / 2 meter' are dimensionless. So is '2 meter / 0.1 centimeter', although its magnitude is not 2/0.1=20, but 2000 because of the relationship between meter and centimeter.

Numbers that are ratios of similarly dimensioned quantities are generally regarded as dimensionless. For this reason, certain quantities, such as plane and solid angles (Wildi, 1980), probabilities (Finney, 1977), specific gravity and radioactivity (ICRU, 1980), are conventionally considered dimensionless, even though they may or must be expressed with non-degenerate units. To simplify the task of handling dimensions, some researchers (e.g., Lin and Segel, 1988; inter alia) have encouraged modelers and analysts to express equations in terms of dimensionless quantities. This practice, however, may be more mistake-prone than carrying units along through the calculations. For example, although it is possible to simplify a contaminant dose that has units "grams per kilogram" to a pure number, it may not be desirable to do so if its interpretation is thereby masked. Moreover, as Hart (1995) has argued, expressing equations with dimensionless variables does not escape the restrictions induced by dimensional algebra so much as it merely obscures them. For example, although international convention holds that both the radian and bequerel (a unit of radioactivity) are dimensionless, expressions such as the following are obviously nonsensical.

0.3 radians + 14	10 bequerels < 100
1 radian ≤ 200 bequerel	sin(36 bequerels)

Angles and radioactivity cannot be added to or compared with pure numbers. Neither are angles and radioactivity exchangeable or even conformable. Even if they are dimensionless, angles still have units (radians, degrees, minutes, seconds, grads, L-turns, etc.) which are certainly neither interchangeable nor ignorable. For the sake of checking the integrity of algorithms and calculations, it makes sense to retain all natural units (contra Lin and Segel, 1988) and propagate them through mathematical expressions.

CORRECTING CONFORMABLE UNITS

Computing the truth or numerical value of the following expressions involves implicit unit conversions.

14 centimeters + 10 inches

20 kilograms > 48 ounces

0.3 radians – 120 degrees

200 bequerel + 40 picocuries

100 watts × 15 seconds + 4.5 newtons × 2.2 meters + 9 joules

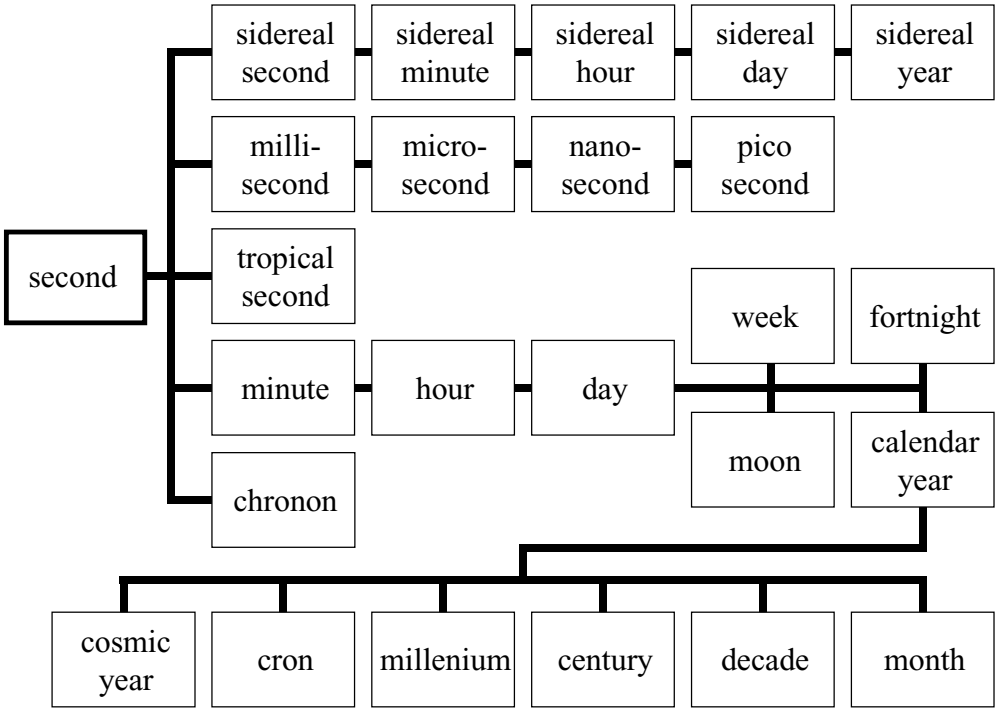
Although these expressions are obviously meaningful, they are not well handled by any of the calculation tools commonly in use today. Entering, for instance, "14 plus 10" on a calculator yields the result 24, but the sum of 14 centimeters and 10 inches is neither 24 centimeters nor 24 inches.

However, automatic unit conversions can be implemented using a graphical model of the relationships among units described by Wildi (1988). In Figure 1, each node of the graph represents a unit of time. Each edge of the graph connecting two units is associated with a conversion factor by which one unit can be converted into the other, through either multiplication or division. So long as the graph is a tree, there exists a unique path that defines the conversion factor between the units. It is a straightforward programming task to traverse such a tree and deduce the conversion factor between any pair of nodes on it. If the graph is not a tree, then any path connecting the two nodes will suffice to define a conversion factor between the units. In principle, multiple paths could be used to optimize precision of the conversion factors. Different dimensions are represented by different trees and a system of measurement by a forest of such trees.

Fluctuating or indeterminate unit definitions can be handled either by repeatedly updating the conversion factors for each edge of the tree or by using intervals to represent the factors. The former will be useful in currency conversions where monetary exchange rates vary daily. The latter will be useful in representing unresolvable uncertainty. For instance, the actual length of the ancient unit 'span' was variously defined in different localities and periods (May and Metzger, 1973; Dilke, 1987). Unless external calibration happens to be available, historical uses of the term are usually ambiguous. Interval representations of conversion factors will also be useful in high-precision calculations (Moore, 1966; Alefeld and Herzberger, 1983). Although many unit conversions are mathematically precise (e.g., there are exactly 12 inches in a foot, and exactly 100 centimeters in a meter by definition), some unit conversions are necessarily approximate. For instance, because 'kilogram' is still defined by reference to an artifact rather than a universal constant, an infinitely precise conversion between it and another unit of mass is impossible. Other units based on physical measurements, such as electron rest mass or solar year, cannot be measured with perfect precision.

Temperature conversions require a similar but slightly more complicated scheme that allows addition and subtractions as well as multiplications and divisions. Monetary currency conversions involving transaction or broker fees can be handled with the same approach.

Figure 1. Tree defining units of time in terms of the base unit second (cf. Wildi, 1988).



Some ambiguity persists in commonly used unit systems. For instance, 'minutes' measures both [time] and [angle]. The term 'rad' is the symbol for radians measuring [angle] and a unit of [radiologic dose]. Further ambiguities arise in colloquial usage (e.g., 'pound' measures both [force] and [mass]). These ambiguities can be resolved by context, conventions or explicit reference.

INADEQUACY OF COMMON CALCULATION TOOLS

All of the common tools used for making calculations, including hand-held calculators, microcomputer spreadsheets, and programming languages such as Java, Pascal, C, BASIC and FORTRAN, lack any provisions to check that the expressions they are computing are dimensionally correct. Interactive spreadsheets are especially vulnerable to unit and dimensional errors because the mathematical formulas are often cryptically expressed and because making unannotated changes is easy and virtually untraceable. Because the necessary checking can be very tedious by hand, errors are common. As Loehle (1991) has pointed out, it is startling to discover just how frequently even sophisticated modelers make dimensionally nonsensical models. Unless the work has been very carefully reviewed, complex models involving many equations are almost guaranteed to contain at least one fundamental error. It may simply be a matter of chance whether such an error turns out to be inconsequential or extremely serious. An example of the latter result is the recent crash of NASA's Mars Climate Orbiter (Isbell et al., 1999) due to a mismatch of English and metric units in the software controlling orbital insertion. It is interesting to note that even mandatory exclusive use of metric system units would have been insufficient to prevent the disaster if mismatches could still arise between the meter-kilogram-second (MKS) and the centimeter-gram-second (CGS) conventions.

Although comprehensive unit conversion tables are widely available (e.g., Beyer, 1978; Pennycuick, 1988; Wildi, 1988; ASTM, 1992), most programmers have limited themselves to the unfriendly option of forcing a user to make all inputs in specific units. This usually necessitates

the user's doing pencil-and-paper preprocessing of input, which is both inconvenient and prone to error. Some state-of-the-art scientific software allows users to specify units for the quantities entered, but the programs invariably treat these units as mere labeling and leave it to the user to ensure dimensional soundness. The problem may even become worse when inputs from users are permitted in arbitrary units as it requires the user to be very careful to express all measures in appropriate units or to make the cumbersome conversions whenever adding or comparing numbers. Burdening a user with this responsibility is unreasonable if the user is not privy to the inner workings of the software.

SOFTWARE STRATEGIES

Comprehensive software tools have recently become available for checking programs, equations and calculation streams for dimensional soundness. Correctness depends on conformance at two levels: balance of dimensions and agreement among units. Error at either level can be detected automatically by these software tools.

Loehle (1991) described a software tool called Dimensional Reasoner that allows modelers to check their programs and algorithms written in FORTRAN or BASIC for dimensional soundness. Units are specified for each variable in the program by adding comments to the source code. The source code is then pre-compiled by Dimensional Reasoner which symbolically evaluates the calculations embodied in the code to check for dimensional soundness. Figure 2 displays an example subroutine in FORTRAN. In this computer language, a `C` in the first column of a line marks it as a comment, which is ignored by the FORTRAN compiler. If the second character in a comment line is a `U`, Dimensional Reasoner will interpret the line as a unit assignment. FORTRAN uses "`**`" to denote exponentiation and `SQRT` for the square root function. Figure 3 displays part of the output from Dimensional Reasoner resulting from an analysis of the code in Figure 2. In this output, the symbol `N` represents a dimensionless quantity. Note that the subroutine is dimensionally correct, except for the last two equations. In the first case, the addition is not possible. In the second case, the assignment cannot be completed.

Figure 2. Sample input for Dimensional Reasoner.

```
      SUBROUTINE TESTDIM(U,V,Y,Z)
CU    U = meter
CU    V = dollar
CU    W = widget
CU    X = meter
CU    Y = meter**2
CU    Z = dollar/widget
      REAL U,V,W,X,Y,Z
      X = 5.0
      X = 6 * X + U + SQRT(X) * SQRT(U) + SQRT(Y) + (Y / U)
      W = 12
      V = Z * W
      Z = X**-1 * V * (X + X) / W
      Z = U + V
      Z = U * V
      STOP
      END
```

Figure 3. Part of the resulting output from Dimensional Reasoner.

```
X = 6 * X + U + SQRT(X) * SQRT(U) + SQRT(Y) + (Y / U)
METER=N**0*METER+METER+(METER)**0.5*(METER)**0.5+(METER**2)**0.5+(METER**2*METE
R**-1)
METER=N**0*METER
EQUATION OK

V = Z * W
DOLLAR=DOLLAR*WIDGET**-1*WIDGET
DOLLAR=DOLLAR*WIDGET**0
EQUATION OK

Z = X**-1 * V * (X + X) / W
DOLLAR*WIDGET**-1=METER**-1*DOLLAR*(METER+METER)*WIDGET**-1
DOLLAR*WIDGET**-1=METER**0*DOLLAR*WIDGET**-1
EQUATION OK

Z = U + V
DOLLAR*WIDGET**-1=METER+DOLLAR
ERROR: THE RIGHT SIDE OF THIS EQUATION DOES NOT BALANCE DIMENSIONALLY

Z = U * V
DOLLAR*WIDGET**-1=METER*DOLLAR
ERROR: THE TWO SIDES OF THIS EQUATION DO NOT BALANCE DIMENSIONALLY
```

Units Calculator (Ferson and Kuhn, 1993; Legendre 1994) also does dimensional and unit checking. It accepts input interactively from a user or reads mathematical expressions from ASCII files in which units are associated with numerical magnitudes and embedded directly in mathematical expressions. Units Calculator not only checks the dimensional soundness of calculations and comparisons, but also resolves unit conflicts when it is possible to do so. For instance, if *A* is 6 grams and *B* is 2 kilograms, the software knows that *A* is less than *B* even though 6 is greater than 2. It also knows that the sum of *A* and *B* is 2.006 kilograms. As a convention, it assumes the units of the last term mentioned are to be those of the result, so *B*+*A* would yield 2006 grams. If *C* is 15 centimeters, the program also knows that *C* cannot be added to *A* or *B*. The software detects attempts to (1) add, subtract or compare nonconforming units; (2) raise a number to a power that is not dimensionless; (3) take the logarithm of a number that is not dimensionless; or (4) take the sine, cosine or tangent of a quantity that is not an angle. The software knows over 600 units, symbols and abbreviations in 60 dimensions. The software fully supports automatic unit conversions, including units from five different temperature scales. It checks the dimensional soundness of expressions involving even units it does not recognize (such as 'widgets'). It understands rich-text-formatted exponents as well as pure ASCII files in which the exponents appear as trailing characters as in "meter3" or are enclosed in braces as in "meter{−3}". It also interprets words such as "squared", "cubic" and "per" that occur in unit strings. In addition to basic arithmetic operators (+, −, ×, ÷, ^, min, max), it supports standard transcendental functions (e.g., exp, ln), trigonometric functions, arbitrary powers, the absolute value function, and magnitude comparisons (<, ≤, >, ≥, =). The software automatically simplifies results by combining like units so that entering "3 m s × 1 m / s^2" yields the result "3 m^2 s^{-1}". However, different units are not combined unless it is necessary to do so to complete a calculation, or on specific request. Several further simplifications are possible, including

combining like dimensions, in which "0.6 cm meter yard day sec" is simplified to 620.001124 $yard^3\ sec^2$", and conversion to SI units, in which it is simplified to "474.02496 $meter^3\ second^2$". The software can also convert a quantity into specific units (if it is possible to do so). Figures 4 and 5 show sample input and output from Units Calculator. In these figures, ^ is used to denote exponentiation, @ marks a variable, and a comma instructs the software to convert a quantity to specified units.

Figure 4. Input for Units Calculator.

```
5 kg * 8 ft / 2 sec2 + 3.2 N - 0.1 newtons
@x = 6*5 meters + 12 m+sqrt(15 ft)*sqrt(1200cm)+(2 square meters)^0.5+(2 m2/m)
@x, meters
5 m ^ -1 * 40 dollar * (5 meter + 500 cm) / 12 widgets, dollar/widget
12 meters + 10 dollars
@z = 12 meters * 10 dollars
@z, dollar/widget
```

Figure 5. Resulting output from Units Calculator.

```
5 kg * 8 ft / 2 sec^2 + 3.2 N - 0.1 newtons
9.196 newtons
@x = 6*5 meters + 12 m+sqrt(15 ft)*sqrt(1200cm)+(2 square meters)^0.5+(2 m2/m)
Ok
@x, meters
52.82123726 meters
5 m ^ -1 * 40 dollar * (5 meter + 500 cm) / 12 widgets, dollar/widget
6.66666667 dollar·widget^-1
12 meters + 10 dollars
undefined
[length] and [currency] don't conform
@z = 12 meters * 10 dollars
Ok
@z, dollar/widget
undefined
[length]·[currency] and [currency]·widget^-1 don't conform
```

The public domain program DimCalc by G.W. Hart supports addition, subtraction, multiplication, division, integer powers and unit conversions of dimensioned quantities in arbitrary mathematical expressions. For instance, entering the expression "2000 cm per s + 0.1 mph" in DimCalc and requesting the result be expressed in the default units for velocity yields the answer "20.0447 m/s". DimCalc detects the mistake in the expression "2000 kg per s + 0.1 mph" and gives the explanation "Sum involves different types". The program was created as an interactive demonstration program and is not configured to read and check calculation streams or algorithms. The philosophy of DimCalc is that arguments must be dimensionless for transcendental functions like exp, ln, and log10, and (following the CIPM) for trigonometric functions like sine, cosine and tangent, and even for fractional powers (including square root) and the absolute value function. Consequently, it does not support any of these functions. Undefined units ('widgets') are not supported except by prior addition into a user-editable ASCII file that defines units. It does not support magnitude comparisons ($<$, $\leq$, $>$, $\geq$, $\approx$, $=$, min, max). Temperatures are interpreted as temperature differences; conversions are handled by a separate interface. However, DimCalc will detect and flag dimensional errors and automatically correct mismatched but conforming units to evaluate complex mathematical expressions. It knows 58 dimensions and hundreds of units in common use. It can be obtained by anonymous FTP from the ftp.ctr.columbia.edu/users/hart/ directory (see http://www.ctr.columbia.edu/~hart/hart.html).

The commercial scientific software Mathcad 2000 (MathSoft, 1999) also supports automatic unit conversions. It detects impossible unit conversions, use of dimensioned numbers as powers, inappropriate use of dimensioned numbers as arguments of transcendental and trigonometic functions. Mathcad knows 32 dimensions and over 300 common units, symbols and abbreviations. It supports arbitrary arithmetic expressions, transcendental and trigonometric functions, magnitude comparisons ($<$, $\leq$, $>$, $\geq$, min, max), absolute value and arbitrary (scalar, dimensionless) powers. Users can define their own units on the fly, but temperatures are not fully supported. Entering "2000 cm sec^{-1} + 0.1 mph =" yields the answer 20.045 m/s. It correctly evaluates an expression such as "(1 ft + 2 m) m" as 2.305 m^2. It detects the error in the expression "1 ft + (2m/ 2 cm)" and gives the hint "The units in this expression do not match". In Mathcad, units are treated as variables (with blanks denoting implicit multiplication). This means that defining a variable *m* means the predefined symbol for meter cannot be used, or will result in unexpected results.

Table 1 is a side-by-side comparison of the features of several commercially available software tools that check dimensional balance and conformity of units. The table reveals the various tradeoffs buyers must make in selecting a tool to use. Mathcad 2000, Unicalc, and Unisolver each allow users to evaluate trigonometric functions of angles with specified units as well as pure numbers, which are assumed to be angles in radians. Units Calculator considers the latter ambiguous and treats it as an error. Although most of the software packages include temperature conversions, only Units Calculator supports integrated calculations with temperatures so that it can compute, for instance, an average like (102 degrees Farenheit + 40 degrees Celsius)/2. Most of the packages uses infix notation, commonly called algebraic notation, but Unisolver employs reverse Polish notation. A software package can "automatically convert to SI" if a user can ask it for the SI equivalent of a quantity in, say, miles per hour without having to tell it the conversion should be to meters per second. Most of the software packages support interactive calculations; only Dimensional Reasoner performs checks on software source code.

Several of the packages support some form of dimensional reasoning, including calculations with undefined units ('widgets') and non-numerical checking of dimensional relationships. Only Unicalc allows users to solve numerical problems purely by dimensional analysis. For instance, suppose a farm combine cuts 30-foot swaths at 5 miles per hour through a corn field that produces 210 bushels per acre, and that corn sells for 3 dollars/bushel. Unicalc can compute the hourly monetary value of the harvest without being told what mathematical operations to use. When a user enters the expression "30 ft; 210 bu/acre; 5 mph; 3 $/bu ? $/hour", the software yields the answer "11454.5 $/hour". The question mark specifies the units of the desired output. The semicolons tell the software to automatically derive the appropriate equation relating the parameters entered, assuming a multiplicative relationship. The user need only enter the quantities known to affect the output. The usefulness of this feature is apparent, but caution should be exercised when employing it. Physicists have long used dimensional analysis of this kind, although there has been debate about whether the strength of dimensional checking extends to deduction (Wilson 1952). There is a danger that the software's flexibility could yield computational errors if used inattentively.

Of course, even with conscientious use these software tools cannot be a panacea for mathematical errors. In particular, there is no guarantee that all calculations that appear dimensionally sound are actually mathematically sensible and scientifically relevant. Nevertheless, these tools should help prevent a broad class of errors currently afflicting serious quantitative work.

AUDITING CALCULATIONS IN DOCUMENTS

An important use of the new software tools is to check the integrity of mathematical expressions and the calculations they yield that appear in documents. This checking can detect many kinds of errors of varying significance. For instance, in many circumstances where there are apparent dimension or unit errors, the problem may be the result of a simple *typographical error*. Such errors are roughly equivalent in gravity to a misspelling, whose consequence might be limited to temporary reader confusion and writer embarrassment. Another category of errors, which might be called c*omputational error*s, occurs when a mistake of dimensions or units leads to a numerically erroneous result. Such an error can occur when, for example, measurements in meters and feet are 'added' without first converting one of the quantities. In this case, the document is simply factually incorrect. In other circumstances, dimensional errors belie an even deeper problem, which might be called a *fundamental error*. Adding meters to seconds, raising a scalar to the 2 inch power, taking the sine of 6 meters, and similarly profound mistakes are examples of fundamental errors. Unlike computational errors, fundamental errors make the affected mathematical expressions utterly nonsensical and, as a consequence, may render the overall argument and perhaps the entire document meaningless.

In our experience, errors of all three kinds can persist even after review of the document. In the case of one government document we studied (EPA, 1993), dimensional errors were found after the periods for peer review and public comment for the document had ostensibly ended. Despite both peer and public review, there were dimensional errors in two out of the four central mathematical expressions in the document. Only after the errors were detected by our software was the document corrected in a revision.

IMPLICATIONS FOR SOFTWARE DEVELOPMENT

Over the last quarter century, concerted effort has been focused on refining the process by which computer software programs are created (Dahl et al., 1972). Although considerable sophistication has been introduced into the 'art' of programming, virtually no facilities have been available to programmers to help them ensure their code is dimensionally sound. There has likewise been very little attention to the issue in the development of new programming languages and software environments (see, e.g., Jorgensen, 1995). This situation should change now that convenient software tools are available to check the dimensional soundness of source code and run-time calculations. Such application would be a straightforward use of automation to remedy a large class of inadvertent errors by programmers and users (Juran and Gryna, 1988). Using the tools requires only that the programmer define the units of the variables and parameters used in the code. Associating units or dimensions with each variable and constant requires the same level of programmer effort as assigning types to them (i.e., specifying whether they are integers, real numbers or characters). In the same way that strict variable typing both improves the readability of code and facilitates much more comprehensive compile-time checking against errors, the extra effort required to associate units with variables should quickly pay for itself in allowing automatic review for dimensional concordance and in improving the self-documentation of the program. Indeed, a programmer's use of unitless variables or constant numbers should be discouraged for the same reasons that using typeless variables is frowned upon as poor programming practice, because it would prevent the compile-time detection of programmer mistakes.

DISCUSSION

The examples given at the beginning of the paper illustrate that checking for dimensional soundness should focus on units rather than dimensions. Dimensional balance does not require

that the units be identical, or even that dimensions be identical. In fact, even having identical dimensions is not by itself enough to guarantee conformance among the units. Dimensional balance is necessary but not sufficient to ensure soundness. Conformability of the units is the necessary and sufficient condition for additions, subtractions and comparisons to make sense. As a practical matter, the units of a quantity are more important than its dimensions per se. In order to detect errors of unit conformance, therefore, it is important to carry units of all quantities through calculations even when they are considered dimensionless.

All mathematical expressions, whether numerical or not, must balance dimensionally and must permit conformance among the units for the variables involved. Applying dimensional and unit analyses to complex models provides a measure of validation that ensures the absence of fundamental syntactical errors that would otherwise guarantee senselessness of the mathematical expression. Recently developed software tools for dimensional analysis of calculation streams, algorithms and programs provide convenient filters for finding these profound mathematical errors. That these checks can now be performed automatically by software is an important development because it means that comprehensive reviews of units and dimensions can now be routine.

Few authors today hesitate to use the sophisticated spell checking features available in modern document processing software (Salton, 1989, page 426ff, and numerous references therein). Some even routinely use grammar and style checkers to improve the composition itself. The use of such software allows a motivated user to detect and correct a variety of potentially embarrassing mistakes in a text ranging from typographical errors to awkward phrasing. Similar automated checking should be used for equations and calculations that constitute an important component of the non-textual content of scientific and technical documentation. Software that can check units provides a very convenient tool for reviewing calculations and programming. Applying such checks can in principle eliminate most instances of an entire class of errors. However, not all unit errors can be detected by such an approach. Just as a spell checker cannot detect that ‘witch’ is an incorrect spelling of ‘which’, an automatic units analysis will not be able to detect that ‘ng’ has been written where ‘mg’ was intended. In principle, however, such mistakes might be detectable by meta-level software—analogous to grammar checkers—that follow higher-level patterns in user input.

ACKNOWLEDGMENTS

I thank Rüdiger Kuhn (formerly of the Bielefeld Universität) and Craig Loehle (Argonne National Laboratory) for discussions about dimensional and unit concordance. I also thank Kim Thompson (Harvard School of Public Health) for sharing the EPA document, and Loren Henning (U.S. Environmental Protection Agency) for providing information about its evolution. David Myers (State University of New York) prepared the table comparing available software packages. Alan Jordan (Delta Engineering) and Ken Burgess (Calchemy Software) provided helpful descriptions of their software products. This publication is based on research supported in part by Small Business Innovation Research grants to Applied Biomathematics from the National Science Foundation (9160527) and the National Cancer Institute (9R44CA81741). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation or the National Cancer Institute.

REFERENCES

Adler, F.R. (1993) *Mathematics for Life Scientists I*. New York: Springer Verlag.

Alefeld, G. and Herzbergerm J. (1983) *Introduction to Interval Computations*. New York: Academic Press.

ASTM [American Society for Testing and Materials]. (1992) *Standard Practice for Use of the International System of Units (SI) (The Modernized Metric System)* (E380-92, PCN 03-543-092-34). Philadelphia: American Society for Testing and Materials.

Beyer, W.H. (1978) *CRC Standard Mathematical Tables* (25th edition). Boca Raton, Florida: CRC Press.

Bratley, P., Fox, B.L., and Schrage, L.E. (1983) *A Guide to Simulations*. New York: Springer-Verlag.

Bridgman, P.W. (1931) *Dimensional Analysis*. New Haven: Yale University Press.

Edwards, D. and Hamson, M. (1990) *Guide to Mathematical Modelling*. Boca Raton, Florida: CRC Press.

EPA [U.S. Environmental Protection Agency]. (1993) Draft soil screening level guidance. Quick reference fact sheet. September 1993. Office of Emergency and Remedial Response, Hazardous Site Control Division, Office of Solid Waste and Emergency Response. Washington, DC: National Technical Information Service.

Dahl, O.-J., Dijkstra, E.W., and Hoare, C.A.R. (1972) *Structured Programming*. London: Academic Press.

Dilke, O.A.W. (1987) *Mathematics and Measurement* (volume 2 in the Reading the Past series). Berkeley: The University of California Press/British Museum.

Ferson, S. and Kuhn, R. (1993) *Units Calculator*. Setauket, New York: Applied Biomathematics.

Ferson, S. (1995) Automated quality assurance checks on model structure in ecological risk assessments. *Human and Environmental Risk Assessment* 2:558-569.

Finney, D.J. (1977) Dimensions of statistics. *Applied Statistics* 26:285-289.

Hart, G.W. (1995) *Multidimensional Analysis: Algebras and Systems for Science and Engineering*. New York: Springer-Verlag.

Hart, G.W. (1995) *DimCalc*. http://www.ctr.columbia.edu/~hart/multanal.html.

ICRU [International Commission of Radiation Quantities and Units]. (1980) *Radiation Quanitities and Units, ICRU Report 33*. Washington: ICRU.

Isbell, D., M. Hardin and J. Underwood. (1999) Mars Climate Orbiter team finds likely cause of loss. http://mars.jpl.nasa.gov/msp98/news/mco990930.html.

Jerrard, H.G. and McNeill, D.B. (1992) *Dictionary of Scientific Units* (6th Edition). London: Chapman & Hall.

Jorgensen, P.C. (1995) *Software Testing: A Craftsman's Approach*. Boca Raton, Florida: CRC Press.

Juran, J.M. and Gryna, F.M. (1988) *Juran's Quality Control Handbook*. New York: McGraw-Hill.

Legendre, P. (1994) [Review of] Units Calculator. *The Quarterly Review of Biology* 69:315.

Lin, C.C. and Segel, L.A. (1988) *Mathematics Applied to Deterministic Problems in the Natural Sciences*. Philadelphia: SIAM.

Loehle, C. (1991) *Dimensional Reasoner*. Setauket, New York: Applied Biomathematics (distributor).

MathSoft. (1999) *Mathcad 2000 User's Guide*. Cambridge, Massachusetts: MathSoft. Inc.

May, H.G. and Metzger, B.M. (eds.) (1973*) The New Oxford Annotated Bible*. New York: Oxford University Press.

Moore, R.E. (1966) *Interval Analysis*. Englewood Cliffs, New Jersey: Prentice-Hall.

NBS [U.S. Department of Commerce, National Bureau of Standards]. (1977) The International System of Units (SI). National Bureau of Standards Special Publication 330, U.S. Department of Commerce, Washington, DC.

NBS [U.S. Department of Commerce, National Bureau of Standards]. (1979) Guidelines for use of the modernized metric system. Dimensions/NBS December 1979: 13-19.

NBS [U.S. Department of Commerce, National Bureau of Standards]. (1985) Units and Systems of Weights and Measures: Their Origin, Development, and Present Status. Letter Circular LC 1035.

Pennycuick, C.J. (1988) *Conversion Factors: SI Units and Many Others*. Chicago: The University of Chicago Press.

Taylor, B.N. (ed.) (1991) The International System of Units (SI). U.S. Department of Commerce, National Institute of Standards and Technology Special Publication 330. Washington: U.S. Government Printing Office,

Salton, G. (1989) *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Reading, MA: Addison-Wesley.

Swartzman, G.L. and S.P. Kaluzny (1987) *Ecological Simulation Primer*. New York: Macmillan.

Warwick, A. (1995) The laboratory of theory, or what's exact about the exact sciences? Pages 311-351 in *The Values of Precision*. M.N. Wise, (ed.). Princeton, New Jersey: Princeton University Press.

Wildi, T. (1988) *Units and Conversion Charts: A Handbook for Engineers and Scientists*. Sillery, Québec: Sperika Enterprises.

Wilson, E.B., Jr. (1952) *An Introduction to Scientific Research*. New York: McGraw-Hill.

Wise, M.N. (ed.) (1995) *The Values of Precision*. Princeton, New Jersey: Princeton University Press.

Table 1

	DimCalc	Dim.Reasoner	Mathcad 2000	Unicalc	Unisolver	Units Calculator
Errors detected						
Adding or comparing apples and oranges	Yes	Yes	Yes	Yes	Yes	Yes
Raising a number to a non-dimensionless power	Yes	No	Yes	Yes	Yes	Yes
Taking a logarithm of non-dimensionless number	No	No	Yes	Yes	Yes	Yes
Trigonometic function of non-angle value	No	No	Yes	Yes	Yes	Yes
Vocabulary						
Units, symbols and synonyms	300	0	300	300	200	600
Dimensions	59	0	32	35	18	60
Angle measures	7	0	3	5	4	8
Temperature scales	4	0	2	4	4	5
Currencies	5	0	0	1	1	70
User-defined units & dimensions	Yes	Yes	Yes	Yes	No	Yes
Arbitrary units (widgets)	No	Yes	Yes	Yes	No	Yes
Plurals recognized as synonyms	Yes	Yes	No	Yes	No	Yes
Modifiers "per", "square" & "cubic"	per	No	No	Yes	No	Yes
Operations						
Automatic conversion (e.g., 1 ft + 1 m)	Yes	No	Yes	Yes	Yes	Yes
Convert to specified units	Yes	No	Yes	Yes	Yes	Yes
Arithmetic operators +, -, x, /	Yes	Yes	Yes	Yes	Yes	Yes
Powers with ^ or **	Yes	Yes	Yes	Yes	Yes	Yes
min, max	No	No	Yes	No	No	Yes
<, ≤, >,≥	No	No	Yes	No	No	Yes
Trigonometric functions (e.g., cos(45 degrees))	No	No	Yes	Yes	Yes	Yes
Unitless angles (e.g., cos(0.6))	No	No	Yes	Yes	Yes	No
Transcendental functions (exp, ln, log10)	No	No	Yes	Yes	Yes	Yes
Square root, fractional roots	No	Yes	Yes	Yes	Yes	Yes
Absolute value, round & truncate functions	No	No	Yes	No	No	Yes
Integrated calculations with temperature	No	No	No	No	No	Yes
Assignment to variables	No	Yes	Yes	Yes	No	Yes
Notation for compound expressions	algebraic	algebraic	algebraic	Algebraic	RPN	algebraic
Uses						
Interactive calculations	Yes	No	Yes	Yes	Yes	Yes
Software code checking	No	FORTRAN	No	No	No	No
Dimensional reasoning ([force], widgets, mph+m/s)	No	Yes	Yes	Yes	No	Yes
Checking calculations in files	No	Yes	Yes	No	No	Yes
Other features						
Automatic conversion to SI	Yes	No	Yes	No	Yes	Yes
Automatic conversion to US units	No	No	Yes	No	Yes	No
Superscripts	No	No	Yes	No	No	Yes
Predefined constants	0	0	10	19	2	7
User extensible predefined constants	Yes	Yes	Yes	Yes	No	Yes
Documentation						
Manual	none	18 pages	355 pages	65 pages	11 pages	35 pages
Help file	63 K	none	1600 K	37 K	40 K	73 K
Error condition hints	Yes	Yes	Yes	Yes	No	Yes
Purchasing information						
Price	Free	$65	$100	$79/$29	$40	$60
Worldwide Web address	ctr.columbia.edu/~hart	ramas.com	mathsoft.com	calchemy.com	delta-eng.com	ramas.com