FLAP: A Matlab Package for Adjustable Precision Floating-Point Arithmetic*

G. W. Stewart†

Jan 30 2009

ABSTRACT

Flap is a package to implement floating-point arithmetic with adjustable precision, in which arithmetic operations are performed on Matlab doubles but are rounded to a specified number of decimal digits after each operation. It is intended to make it easy to generate examples of the effects of rounding error for classroom use. The rounding level is controlled by a global variable that can be changed in the course of a computation. Modes for single and double precision binary arithmetic are also provided. The operations are performed on objects of class flap, which consists of an integer indicating the current rounding level for the object and a double containing its data. Most of Matlab's commonly used operations and functions have been overloaded to work with flap objects. In addition, Flap provides functions to compute the LU, QR, and Cholesky decompositions of a flap, as well as routines for manipulating Householder transformations and plane rotations.

1. Introduction

The original Matlab [1], written by Cleve Moler to support a student laboratory for linear algebra, had an option that allowed one to specify the precision to which computations were performed. Because the effects of rounding error are especially visible when the numbers involved are not too long, the option was useful for constructing classroom examples of the how rounding error plays out in numerical algorithms. Unfortunately, this option has vanished in the version of Matlab produced by Mathworks.

The purpose of this paper is to describe a Matlab package Flap (**Fl**oating-point arithmetic with **a**djustable **p**recision) that implements adjustable-precision floating-point arithmetic. This arithmetic is not the same as that of the typical variable precision package that allows computations to arbitrarily high precision. Instead, Flap uses standard IEEE double-precision floating-point arithmetic to perform its operations, after which the results are rounded to a prespecified number of decimal digits. This has three consequences. First, the maximum precision is limited by the precision of the underlying arithmetic — approximately 16 decimal digits for IEEE double precision. Second,

*FLAP is available via the author's home page `http://www.cs.umd.edu/~stewart/`

†Department of Computer Science, University of Maryland, College Park MD 20742.
`stewart@cs.umd.edu`

because each arithmetic operation involves an expensive rounding process, Flap programs are not really suitable for production runs. Finally, since the rounding is in decimal arithmetic, printouts reveal the precision by trailing zeros — e.g., `3.1400e+00` for π rounded to three digits. This is useful in following the ongoing effects of rounding error in a computation — especially cancellation.

In addition to this decimal mode, Flap provides a binary mode in which rounding takes place in IEEE double or single precision. This mode is designed for people who develop algorithms in Matlab that are to run in single precision on specialized hardware. The move from double precision to the less forgiving single precision can reveal unanticipated glitches, and the single-precision option provides an opportunity to test the algorithms in single precision without having to implement them on the target hardware.

Flap is a simple package that does not require a long paper to explain itself. In the next section we will describe the Matlab class `flap` that supports adjustable precision arithmetic.[1] As we will see, rounding is done, for the most part, by the `flap` constructor, and we describe the rounding protocol is some detail. Section 3 gives more details on the capabilities of Flab. A concluding section gives an illustrative example.

To fix our nomenclature, Flap will denote the package; `flap`, the class; and a `flap` object will be called a flap.

2. Flap objects and operations

Flap consists of three parts.

1. The class `flap` and its methods.

2. The global variables `FlapRlevel` and `FlapLineLength`.

3. Public functions and scripts.

The class `flap` is the heart of the package. A `flap` object `f` is a structure containing the following two fields.

```
f.Rlevel
f.d
```

[1]The reader is assumed to have at least a passing familiarity with Matlab classes. But to keep our terminology straight, here is a summary. A Matlab class consists of a constructor to create class objects and class methods to manipulate the objects. Both the constructor and the methods are Matlab functions residing in a directory named `@<classname>`. A class object is represented by a structure whose fields are defined by the class. The fields must appear in the same order in all objects. The fields of a class object are private — they can only be accessed by class methods. This restriction, however, can be overridden by class methods, as is done in Flap.

The `d` (for **d**ouble) field is an array of doubles, which may be complex. The field `Rlevel` (for **R**ounding **level**) contains an integer between 1 and 16 (inclusive). The values 15 and 16 specify single and double precision in the binary mode. In the decimal mode `Rlevel` contains the number of decimal digits required to represent the data in `f.d`.

Rounding in Flap has two aspects: How Flap rounds individual numbers in the three rounding modes and how Flap rounds dynamically in an ongoing computation. We begin with the individual numbers.

1. **Single precision binary:** `Rlevel = 15.` The result of rounding a number `a` to single precision is a double precision number whose bit pattern is the same as that of `a` rounded to single precision and extended to double precision by appending zero bits. The Matlab statement

   ```
   double(single(a))
   ```

   will perform this rounding.[2]

2. **Double precision binary:** `Rlevel = 16.` Since all Flap arrays are double, rounding in this mode is essentially the default rounding mode of the machine on which a Flap program is running.

3. **Decimal:** `0 < Rlevel < 15.` The rounded value of `a` is the double precision representation of `a` rounded to `Rlevel` decimal digits.[3]

The global variable `FlapRlevel` contains the rounding level for flap operations. It can be changed freely, but we recommend that you use the Flap function

```
SetFlapRlevel(Rlevel),
```

which produces an error return if `Rlevel` is not an integer in the range `0:16`.

In a Flap program, most rounding is done by calls to the flap constructor. It has three forms. (Note: You do not have to work through the details in the list below to use Flap. Flap does pretty much what you would do if you were performing a hand calculation, rounding as you go.)

1. `f = flap`: With no input arguments `flap` returns returns a flap `f` with `f.d` an empty array and `f.Rlevel` set to 16.

2. `f = flap(a)`. Here `a` is either a flap or an array of doubles.

[2] If `a` overflows on being converted to single precision the final result will be `Inf`.

[3] The rounding is accomplished by a variant of Matlab's `chop` function. It is expensive, requiring the evaluation of a logarithm and a subsequent exponentiation to round a single number. For details see the appendix.

1. `a` is a flap: `flap` returns a flap `f`. The field `f.d` is the `a.d` rounded as specified by `FlapRlevel`. The details depend on the rounding mode.
   1. In decimal mode if `a.Rlevel` nonzero and less than or equal `FlapRlevel`, no rounding is done and `f.Rlevel` is set to `a.Rlevel`. Otherwise rounding occurs and `f.Rlevel` is set to `FlapRlevel`.
   2. In single mode rounding occurs if `a.level ~= 15`. `f.Rlevel` is set to 15.
   3. In double mode no rounding occurs, and `f.Rlevel` is set to 16.
2. `a` is a double array: the field `f.d` of `f` is the array `a` rounded as specified by `FlapRlevel`. The field `f.Rlevel` is set to `FlapRlevel`.

3. `f = flap(a, Rlevel)`. This is the same as `flap(a)`, except that `FlapRlevel` is replaced by `Rlevel`.

Note that in item 2.1.1, if `a.Rlevel` is less than or equal to `FlapRlevel`, `a.d` is already rounded to `FlapRelevel`, and no rounding is performed. The reason for assigning `a.Rlevel`, instead of `FlapRlevel` to `f.Rlevel` is to avoid unnecessary roundings later. Thus in the decimal mode, the `Rlevel` field of a flap represents its actual rounding status, not the global rounding represented by `FlapRlevel`.

An important consequence of this rounding algorithm is that the statement

```
f = flap(f)
```
(2.1)

rounds `f.d` to the precision specified by `FlapRlevel`, adjusting `f.Rlevel` appropriately. This fact is used by Flap to implement its adjustable precision operations.

Operations are performed on flaps by overloaded operators and functions. For example, if `f` and `g` are flaps, the statement

```
h = f*g;
```

causes Matlab to call the `flap` method `times(f,g)`, which creates a new flap containing the properly rounded product of `f.d` and `g.d`. It is instructive to look at the code for `times`.

```
function h = times(f, g)
f = flap(f);
g = flap(g);
h = flap(f.d*g.d);
```

The constructor `flap` is used to round `f` and `g` to the precision specified by `FlapRlevel`. Then the sum is computed and rounded to the same level.

Paradoxically, the statement `h = f*g` does not cause either `f` or `g` to be rounded to the current `FlapRlevel`. The reason is that the rounding occurs inside the function

`times`. Since Matlab passes arguments by value (i.e., it passes copies of them), the changes made in `f` and `g` in `times` do not propagate back to the flaps in the argument list of the call to `times`. This has the consequence that if we change `FlapRlevel`, then a flap `f` appearing in multiple operations may be rounded each time in the implementing functions. One can avoid this by converting `f` to the current rounding level via (2.1).

Flap's public functions and scripts reside in the directory `FlapBase`, which contains the directory `@flap`. For example, the public script `FlapStartup` declares the global variable `FlapRound` and queries the user for a value. The function `SetFlapRlevel` mentioned earlier is also a public function, as is `PrintAry`, which prints a double array, and `FlapRound`, which rounds a double array. Other public functions are described later.

Further information about Flap methods can be obtained by using Matlab's `help` command. For example

```
help PrintAry
```

gives the following output

```
PrintAry    Prints a double array.
     PrintAry(ARY, RLEVEL) prints the ARY to RLEVEL decimal digits.
.   .   .   .   .  .
```

and

```
help flap/qrd
```

gives

```
QRD Orthogonal-triangular decomposition.
    [Q,R] = QRD(A) produces an upper triangular matrix R of the same
    dimension as A and a unitary matrix Q so that A = Q*R.
    In this and the cases below, Q and R are normalized so that
    the diagonal elements of R are nonnegative.
.   .   .   .   .   .
```

The commands `help FlapBase` and `help flap` will list Flap's public functions and Flap's methods respectively.

3. Flap's capabilities

In this section we will give further details on Flap.

• **Subscripting and fields.** Let `<se>` denote a subscript expression — e.g., `:,2`. Then the expressions `f(<se>)`, `f.d`, `f.d(<se>)`, and `f.Rlevel` are allowed in any expression. Thus you can interrogate the value of `Rlevel` for a flap `f` by writing

```
f.Rlevel
```

Only `f.d` and `f(<se>)` are allowed on the left hand side of an assignment statement. Thus you can write

```
f.d = [1, 2, 3];
```

You can also write

```
f(:,1) = g(:,2);
```

But you cannot write

```
f.d(3,2) = 5;
```

In all these assignments, Flap insures that numbers are rounded according to the protocol described it the previous section.

Flap does not allow one to change the values of the Rlevel field of a flap. That privilege is reserved for flap methods.

• **Operators.** The operators `+`, `-` (unary and binary), `.*`, `.^`, `.\`, `./`, `kron`, and `*`, `\`, and `/` have been implemented. All but the binary forms of first operators two and last three operators are equivalent to something like

```
f = flap(f); g = flap(g);
h = flap(f.d <op> g.d);
```

or in the case of unary operators

```
f = flap(f);
g = flap(<op> f.d);
```

The operators of addition and subtraction can cause the cancellation of leading significant digits. When this happens, it is necessary to round at a level strictly less than `FlapRlevel`. The algorithm for determining the rounding level is tricky and involves additional work. For more details see the appendix in §5.

The last three operators in the above list are different from the others in requiring support functions that round with each operation. For example, `*` requires a matrix multiply function. The `\` and `/` operators must compute an LU-decomposition and solve solve triangular systems (see `lud` below).

In the multiplication and division of complex numbers, the real and imaginary components must by computed by formulas involving more than one operation. For example,

$$(v + iw)(x + iy) = (vx - wy) + i(wx + vy). \tag{3.1}$$

Flap rounds the real and imaginary parts computed by Matlab to `FlapRlevel`. This is clearly not the same as evaluating the formulas in (3.1) in precision `FlapRlevel`. In particular, the smaller of the real or imaginary parts of the results of these two procedures may differ considerably. However, if you regard the complex number as a single entity, then in decimal mode the two procedures give results differing by a relative error of about $10^{-\texttt{FlapRlevel}}$. A similar statement holds for the binary mode.

Flap overloads Matlab's relational operations: `==`, `~=`, `<=`, etc. The result is equivalent to performing the same operation on the `d` fields of the operators, and the array returned is of class `double`, not `flap`. In particular, two flaps with identical `d` fields will be equal even if their `Rlevel` fields are different.

• **Extraction and combination.** Some operations extract information from flaps or combine flaps without rounding. As we have seen, Flap permits such concatenation of flaps as

```
[f, g; h, k]
```

In addition, Flap overloads, `size`, `disp`, `isnumeric`, `diag`, `tril`, `triu`, `fliprl`, and `flipud`.

• **Elementary and special functions.** Flap overloads Matlab's elementary functions — `sin`, `cos`, `exp`, etc. It does not overload the special functions like `airy`, `erf`, etc. They can be referenced by writing `flap(fun(f.d))`.

• **Constructors.** Most of Matlab's constructors, such as `zeros`, `ones`, `eye` do not contain a double in their argument, and hence cannot be overloaded. Although it would be easy to write flap methods with a different name — e.g., `feye` — it is almost as simple to code, say,

```
flap(eye(10));
```

and that is the solution taken here.

Note, however, that in expressions where a flap is involved, it may be unnecessary to use this dodge. For example, in the frequently occurring expression

```
A - lambda*eye(n)
```
(3.2)

where `A` is a flap, `lambda*eye(n)` will be computed as a double, but Matlab will call the overloaded `minus` to compute the difference. The function `minus` will round the double `lambda*eye(n)` to the working precision.

• **Matrix functions, decompositions, and transformations.** The majority of Matlab's matrix functions are either decompositions, such as the singular value decomposition; scalar functions, such as the determinant; or mathematical functions of matrices that produce a matrix value, such as the exponential, logarithm, or square root. These functions all have in common that their results can be very sensitive to the precision to which they are computed. Consequently, the only way that they can be included in Flap is to implement them in Flap, rounding at each step — a difficult, though not impossible, undertaking.

Flap provides three widely used decompositions: the pivoted LU decomposition (`lud`), the QR decomposition (`qrd`), and the Cholesky decomposition (`chold`). The LU decomposition is used to implement the matrix divide operators \ and /. The algorithms for computing these decompositions round each operation as they progress. For more details us the Matlab `help` command.

For those who want to compute other decompositions, Flap provides functions to construct and apply Householder transformations and plane rotations. To find out more, query `housegen`, `houseappl`, `houseappr`, `rotgen`, and `rotapp`. These functions act on flaps. But if you decide to code a new decomposition for incorporation into Flap itself, you will want to work directly with the double arrays. Accordingly these functions are mirrored in `FlapBase`, by `FlapHousegen`, `FlapHouseappl`, etc., which act on double arrays.

As noted above, owing to the possibility of cancellation, the functions `plus` and `minus` that implement the sum and difference operators are not trivial. The public functions `FlapPlus` and `FlapMinus` in `FlapBase` perform the same operations on doubles. For more on all these functions, use the Matlab `help` command

• **Printing** Flap provides a method `PrintFlap` to print the `d` field of a flap. In its simplest form

```
PrintFlap(f)
```

It prints `f.d` to `f.Rlevel` digits on the screen. The global variable `FlapLineLength` specifies the length of the output lines. You will generally set it at the beginning of your session when you execute `FlapStartup`. Optional arguments specify a label to be printed along with the `f` or override the default line length.

`PrintFlap` uses a public function `PrintAry` to actually print `f.d`. This function can be useful when one wants to print a flap array to a different number of digits than `f.Rlevel`. For more details use Matlab's `help` command.

4. An example

The well-known method of iterative refinement for solving a linear system

$$Ax = b$$

proceeds as follows.

```
1. Solve the system Ax = b
2. While (not converged)
3.     Compute r = b − Ax
4.     Solve the system Ad = r
5.     Set x = x + d
6. End while
```
(4.1)

In exact arithmetic this is just a way of computing the solution all over again. In fact, for any x, approximate solution or not, we have

$$x + d = x + A^{-1}r = x + A^{-1}(b - Ax) = A^{-1}b.$$

so that we get the solution in one step.

On the other hand, when rounding error is involved, the method can be used to improve the initial solution [line 1 in (4.1)]. The key is to compute r to higher precision than is used to solve the linear systems involving A. Specifically, let

1. ρ be the precision to which the systems are solved (our `FlapRLevel`),
2. σ be the common logarithm of the condition number of A,
3. τ be the precision to which the residual is computed.

Then is is known that [2, Section 3.4.5]

1. $\rho - \sigma$ is the number of accurate digits in the initial solution.
2. $\rho - \sigma$ is the number of accurate digits added by each refinement step.
3. $\min\{\rho, \tau - \sigma\}$ is the maximal attainable accuracy.

These statements should be taken with a grain of salt, especially when the computation is done in decimal arithmetic with its coarse rounding. But as we shall see, they characterize the typical behavior of iterative refinement.

Figure 4.1 contains a Matlab script to probe the properties of iterative refinement. It begins by setting `FlapRlevel` (to 7 in the examples that follow).

```
 1. global FlapRlevel
 2.
 3. % Set the basic rounding level.
 4.
 5. SetFlapRlevel(rho);
 6.
 7. % Generate a system with log(cond(A)) = sig.
 8.
 9. n = 100;
10. s = logspace(0,-sig, n);
11. [U, trash] = qr(randn(n));
12. [V, trash] = qr(randn(n));
13. A = U*diag(s)*V';
14. A = flap(A);
15. b = A*ones(n,1);
16. xt = A.d\b.d;
17.
18. % Solve the system with iterative refinement.
19. % The residuals are evaluated to tau digits.
20.
21. fprintf('tau = %2d\n', fix(tau));
22.
23. x = A\b;
24. fprintf('%8.1e\n', norm(x.d - xt)/norm(xt));
25.
26. for i=1:4
27.    SetFlapRlevel(tau);
28.       r = b - A*x;
29.    SetFlapRlevel(rho);
30.    x = x + A\r;
31.    fprintf('%8.1e\n', norm(x.d - xt)/norm(xt));
32. end
```

Figure 4.1: Iterative refinement example.

In the system generation, lines 10–13 are standard code for generating a test matrix with known singular values. The `logspace` function generates a sequence of `n` singular values decreasing geometrically from 1 to $10^{-\texttt{sig}}$. `U` and `V` are random orthogonal matrices, which makes `A` a matrix with the required condition number.

The right-hand side of the system is generated in line 15. Note that this expression mixes a flap and a double. This is another example of a mixed expression [see (3.2)].

In line 16, the solution is computed in standard double precision. It will be compared with the solutions computed by the iterative refinement process. Because A has the degree of ill-conditioning specified by `sig`, `xt` will have only about `16-sig` correct digits. However, if `sig` is less than seven, as will be the case here, this represents at least nine digits of accuracy, which is quite enough for the seven digit computations we will be doing.

Lines 23–32 contain a straightforward implementation of the iterative refinement method. At line 27, Flap shifts into a higher precision to compute the residual and then back to normal precision at line 29. The loop performs four iterations of the refinement procedure prints out the relative error in the current `x`.

Here are runs with a very ill-condition matrix (ill-condition with respect to `rho=7`, that is).

```
rho = 7, sig = 6.5

tau =  7   tau =  9   tau = 11   tau = 13
 7.6e-02    7.7e-02    6.7e-02    7.4e-02
 7.5e-02    5.5e-03    2.8e-03    3.3e-03
 1.1e-01    1.1e-03    1.9e-04    1.6e-04
 1.0e-01    1.2e-03    1.8e-05    1.2e-05
 9.1e-02    1.1e-03    1.1e-05    6.1e-07
```

The results generally agree with the analysis given above. For `tau=7` there is no improvement in the solution. As `tau` increases, the refined solution improves; but the convergence, which is controlled by the condition of `A` is slow.

Here are runs with with less ill-conditioned matrix.

```
rho = 7, sig = 4

tau =  7   tau =  9   tau = 11   tau = 13
 3.1e-04    4.6e-04    5.2e-04    3.4e-04
 4.3e-04    3.6e-06    3.0e-07    2.2e-07
 4.1e-04    4.2e-06    2.2e-07    1.9e-07
 4.0e-04    3.4e-06    2.3e-07    1.9e-07
 6.3e-04    5.2e-06    2.2e-07    1.9e-07
```

The convergence is faster, and we can get away with a smaller precision in evaluating the residual and still get full seven digit accuracy.

The example also illustrates the slowness of Flap. Matlab's `tic/toc` timer shows that the time to evaluate the expression `A\r` on my workstation is about 0.7 seconds. Since the \ operator requires $O(n^3)$ operations, we can expect a time of about 5.6 seconds for $n = 200$ and about 45 seconds for $n = 400$.

But Flap was never intended to be a number cruncher. The above example shows that for its stated purpose — probing the effects of rounding error on problems of moderate size — Flap brings adjustable rounding error to Matlab in an easy-to-use form.

5. Appendix: Rounding and cancellation

In this appendix we describe the treatment of rounding and cancellation in Flap. In what follows, the term binary number will mean an IEEE double-precision floating-point number.

The rounding problem may be stated as follows. We are given binary number $x \neq 0$ and an integer rounding level R. We wish to determine a binary number $\hat{x}$ that is the binary representation of x rounded to R decimal digits. Note that $\hat{x}$ is not simply value of x rounded to R decimal digits, for that value may have to be further rounded to obtain its binary representation. For example, the value of 0.16 rounded to one digit, is 0.2 which has a nonterminating representation in binary.

The idea behind the rounding algorithm is simple. Find a power s of 10 such that sx in decimal representation has exactly R decimal digits to the left of the decimal point. Then use the Matlab function `round` to round sx to the nearest integer and divide by s. For example, if $x = 0.16$ and $R = 1$, we take $s = 10$. Then $sx = 1.6$, round$(sx) = 2$ and round$(sx)/10 = 0.2$. The formula for s used in Flap is

$$s = 10^{R-\lceil \log_{10}(x) \rceil},$$

where $\lceil x \rceil$ is the smallest integer greater than or equal to x (i.e., the matlab `ceil` function).

As we have mentioned, the binary representation of a number rounded in decimal need not be the same as the rounded number. The difference is small — at most a relative error on the order of 10^{-16}. But cancellation in the addition or subtraction of two numbers can escalate this error to the point where it causes trouble. Consider the numbers

$$a = .1234567891 \quad \text{and} \quad b = .1234567890$$

which differ only in their tenth digits. Flap, operating with a `FlapRlevel` of 10, should produce a value of `1.000000000e-10` for $a - b$. Instead it produces `1.000000083e-10`. The reason becomes apparent when we write out $b - a$ to more places:

```
1.000000082740371e-10.
```

The errors in the binary representation of a and b, which were initially in the 16th digit, have been promoted by the cancellation so that their combined effects appear starting in the 8th digit. Our rounding algorithm has not failed — it has correctly rounded the number it was given.[4] Note that this problem does not occur when the rounding level is less than eight.

A cure is suggested by the fact that if we round `1.000000082740371e-10` at any of the first seven digits we get the correct answer. There is a buffer of zeros between the true value and the garbage from the 16th place (in other examples it may be a buffer of nines). The problem is then one of determining a rounding level that lies in the buffer. The formula used by Flap is

$$S = \min\left\{R,\ 1 + R + \left\lceil \log_{10}\left(\frac{|a-b|}{\max\{|a|,|b|\}}\right)\right\rceil\right\}. \tag{5.1}$$

To see how this formula was derived, consider the following the following diagram.

$$\begin{array}{lcl} & & p \quad q \quad\quad s \\ a & : & \texttt{AAAAAAAAAA00000E} \\ b & : & -\texttt{AAAAAABBBB00000F} \\ c = a - b & : & \quad\texttt{CCCC00000GGGGGGG} \end{array}$$

It illustrates symbolically the cancellations of six digits in the subtraction of two ten digit flaps a and b. The leading digits of a and b (which are the same) are assumed to be of order 10^p; that is, they can range from $1{\cdot}10^p$ to $9{\cdot}10^p$. The leading digit of $c = a - b$ is or order $10^q = 10^{p-6}$. The digits under s, which are of order $10^s = 10^{p-15}$ are the errors in the sixteenth place introduced by binary rounding. Note that the cancellation has has promoted the sum of `F` and `E` to the tenth digit. If we were to round to ten digits we would get a spurious nonzero tenth digit. If, on the other hand, we were to round to four, five, six, seven, or eigth digits, we would get the correct result. In the general case, we would like to use the smallest of these numbers, since the size of the buffer is reduced as the rounding level R increases.

The smallest permissible rounding is $R - (p - q)$. Thus our problem is to estimate $p - q$. As (5.1) suggests, we will estimate $p - q$ by $\log_{10}[|a - b|/\max\{|a|, |b|\}]$, which is an approximation of the log relative error between a and b, which in turn is loosely related to the number of significant digits to which a and b agree. To derive precise error bounds, we assume that a and b are positive and $b < a$. Then we must have

$$10^p \le a < 10^{p+1}, \tag{5.2}$$

[4]Nor was there any error in the subtraction. Its a common misconception that cancellation creates error. What it actually does is increase the influence of errors made earlier.

the first bound being obtained for $a = 1.00\ldots\times10^p$ and the last being most nearly attained for $a = 9.99\ldots\times10^p$. Similarly,

$$10^q \le a - b < 10^{q+1}.$$

Hence

$$10^{-(p-q)-1} < \frac{a-b}{a} < 10^{-(p-q)+1} \tag{5.3}$$

It follows that $\lceil \log_{10}(a-b)/a \rceil$ is either $-(p-q)$ or $-(p-q)+1$. Thus S in (5.1) is either the smallest possible rounding value, $R-(p-q)$, or is one greater, both of which are in the buffer for $R \le 14$. For this reason Flap restricts `FlapRlevel` in the decimal mode to be not greater than 14.

Unfortunately, rounding error can cause the first inequality in (5.3) to be violated, giving too small a value for S. For this reason we add one to the expression $R + \cdots$ in (5.1). Fortunately, the upper bound in (5.3) cannot be violated. Hence, this incrementation of S cannot be magnified by the violation of the upper bound.

There is a rarely occurring case, which is exemplified by the following diagram.

		p $\quad$ q
a	:	`100000000000000EX`
b	:	$-$ `0999999999999990F`
$c = a - b$	:	`1GG`

In this case $p = 17$ and $q = 3$ so that $S = 1 + R - (p-q) = 1$. It might seem dangerous to round the `1` with the digits `GG` siting next to it, but in practice it works out. For values of `FlapRlevel` less than 14, there is a buffer of zeros.

There are three other cases to consider. When $b = a$, $b - a = 0$, and there is no rounding. When $b = 0$ the ceiling in (5.1) evaluates to zero, so that $S = R$. When b is negative the ceiling evaluates to a value greater than or equal to one so that $S = R$. In the last two cases there can be no cancellation and $S = R$ is the appropriate value.

Acknowledgement

I would like to thank Clever Moler for suggestions that improved both Flap and this paper — in particular, for pointing out the problems that cancellation can cause.

References

[1] C. B. Moler. MATLAB — an interactive matrix laboratory. Technical Report 369, Department of Mathematics and Statistics, University of New Mexico, 1980.

[2] G. W. Stewart. *Matrix Algorithms I: Basic Decompositions*. SIAM, Philadelphia, 1998.