



Following are notes for the talk given at the Twin Cities SAS Users Group, March 21, 1991. Graphics are omitted, but some additional material is included in response to questions raised at the meeting.

SAS Efficiency

- Definition
- When does it matter?
- System vs Personal
- I. Process only the data you need

(Due to lack of time, only topics above this line covered in talk. The remaining topics, however, include many opportunities for improved efficiency and code quality.)

- II. Execute only statements you need
- III. Know your defaults
- IV. Control sorting
- V. Write clean code

Efficiency - what does it mean?

- Computer efficiency - a trade-off
- Human vs Computer

Efficiency notes

Computer efficiency - tradeoff examples:

CPU vs I/O: use of disk-caching on PC SAS; use of numeric variable lengths less than 8 bytes.

Additional MEM can save CPU and I/O time in sorting. Use of Expanded MEM in PC saves PC and I/O.

Mainframe: very large memory requests shorten clock time (but may make you a bad citizen!)

Human vs computer:

SAS is intended to save human time at the expense of computer time (as compared to FORTRAN, C, etc.)

Some practices recommended in SAS programming tips use more programmer time to save computer time - I AVOID these ideas in this talk.

Among the tips discussed here, good programming practice IS NO HARDER than bad.

Minimize variable lengths

Or, what precision do I really need?

Length	Significant Digits Retained	Largest IntegerLargest Integer Represented ExactlyRepresented Exactly On PC'sOn IBM Mainframes	
		(not allowed on PC)	255
2	2		
3	4	8,192	65,535
4	6	2,097,152	16,777,215
5	9	536,870,912	4,294,967,295
6	11	137,428,953,472	1,099,511,627,775
7	13	35,184,372,088,832	281,474,946,710,655
8	16	9,007,198,254,740,992	72,057,594,037,827,935

Within a dataset, all numeric variables are 8 bytes in length. This means that variables coming from datasets that have shorter lengths are "padded" to 8 bytes. In SAS, *all numbers* are represented as floating point. Floating point numbers consist of three parts: a sign bit, an exponent, and the mantissa (which is always a binary number).

On IBM mainframes, the sign bit and exponent occupy the entire first byte. The exponent is a *power of 16*. This means that in estimating precision you should recognize that up to three leading

bits of the mantissa may be zero. Therefore, the number of significant bits in mantissa is at least: $8 \cdot (\text{bytes} - 1) - 3$, which may be simplified to $8 \cdot \text{bytes} - 11$. Example: a 4 byte number has at least $8 \cdot 4 - 11 = 21$ significant bits.

On PC's, $1\frac{1}{2}$ bytes are devoted to sign and exponent, giving the exponent 11 bits. Here, the exponent is a *power of 2*. Also, since a "normalized" floating point number would always have a leading 1 bit in the mantissa, the designers of the PC floating point format realized that they could actually *omit* the leading bit (and treat it as 1). Therefore, the number of bits in the mantissa of a PC floating point number is $8 \cdot \text{bytes} - 12 + 1$, or $8 \cdot \text{bytes} - 11$. Coincidentally, that is the same as for the IBM mainframe floating point number.

In estimating *decimal* significant digits, multiply the number of mantissa bits by .3. (For the mathematically inclined, .3 is $\log_{10} 2$.) Example: a 4-byte number has 21 mantissa bits, as we have seen. It therefore yields $21 \cdot .3 = 6.3$, or a little better than 6 significant decimal digits.

Now realize that the way a number is represented in a floating point format depends on the base. In fact, some fractional numbers can be represented exactly in one base but not in another. Example: $1/5$ is represented exactly in base 10 (0.2), but $1/3$ is not (0.333333...). In base 16 (IBM mainframe) or base 2 (PC), many common terminating decimal values are non-terminating. Examples: 0.5, 0.25, 0.75 are all terminating as binary fractions because they have only powers of 2 in their denominator ($1/2$, $1/4$, $3/4$ respectively). But 0.1, 0.2, 0.3 are all non-terminating, because their fractional representation involves a power of 5 in the denominator. Conclusion: *almost all* real numbers (or even rational numbers) are non-terminating in any particular base! The following example illustrates the obvious fact that when you compare the representation of the same number in two different lengths it usually won't match.

SAS Institute Cries Wolf!

In SAS Programming Tips, p. 78: "Caution do not shorten numeric variables containing fractions." Here's an example showing effect of differing lengths:

```
1 data fred;
2 length x 3;
3 drop i;
```

```
4 do i = 1 to 9;
5   x = ranuni(0);
6   y = x;
7   if x ne y then put x = y = ; /*None!*/
8   output;
9   end;
10 run;
11 data _null_;
12 set fred;
13 if x ne y then put x = y = ; /*Most all!*/
14 run;
X=0.80444 Y=0.80451
X=0.72131 Y=0.72140
X=0.10884 Y=0.10885
X=0.51330 Y=0.51334
X=0.71118 Y=0.71126
X=0.57812 Y=0.57815
X=0.09953 Y=0.09954
X=0.70019 Y=0.70025
X=0.06288 Y=0.06288
```

Understanding how numbers are represented in floating point formats, we see why this happens. In general, making an EQ or NE comparison of fractional values can cause trouble, but in fact there is little practical need for such comparisons. If you do need to make such comparisons, it should be done within some tolerance, like this: if $\text{abs}(x-y) \leq \text{tol}$ then ... (where "tol" is either a literal, like .001, a macrovariable, or a regular variable.

Read only fields you need

Acceptable

```
data oct90;
  infile file-spec;
  input jday rcode $ sales exps salhrs exphrs
  invin invout;
run;
```

More efficient

```
data oct90;
  infile file-spec;
  input @1 jday
        @13 sales
        @42 invout;
run;
```

NOTE: in macro-based systems, it is tricky, but feasible, to accomplish this in a general way.

Read selection fields first

Acceptable

```
data temp;
  infile file spec;
  input month [and lots more fields];
run;
data sales.aug90;
  set temp;
  if month eq 8;
run;
```

Better

```
data sales.aug90;
  infile file spec;
  input month [and lots more fields];
  if month eq 8;
run;
```

Best

```
data sales.aug90;
  infile file-spec;
  input month @;
  if month eq 8 then input [lots more fields];
  else do;
    input; /* Release input line */
    return;
  end;
run;
```

Store data in SAS datasets

Appears to be 2 to 5 times faster getting data from a SAS dataset than from a flat file.

Keep summaries of large SAS data sets

Summaries are usually only a fraction the size of the original dataset, so doesn't add much storage to do this, and avoids much computation.

Store only necessary variables

Acceptable

```
data good.stuff;
  infile filespec;
  input x y;
  x2 = x*x;
  y2 = y*y;
  xy = x*y;
run;
```

More efficient

```
data good.stuff;
```

```
infile filespec;
```

```
input x y;
```

```
run;
```

(Variables x2, y2, and xy can easily be calculated whenever they are needed.)

Process only variables needed

Acceptable

```
data stocks.gainers;
  set stocks.oct;
  keep tot numshrs company;
  if net gt 0 then
    ....
run;
```

More efficient

```
data stocks.gainers;
  set stocks.oct(keep = tot numshrs company
    net);
  drop net;
  if net gt 0 then ....
run;
```

(Very effective if many variables in stocks.gainers.)

Create all data subsets at one time

Acceptable

```
data dog;
  set master;
  if beast eq 'DOG';
run;
data cat;
  set master;
  if beast eq 'CAT';
run;
[etc.]
```

Much more efficient

```
data dog cat mouse ...;
  set master;
  select(beast);
    when('DOG') output dog;
    when('CAT') output cat;
    ....
  end;
run;
```

Consider "parallel" dataset

for repetitive information

Example: a master file has data by ZIP code. Another file has county data. Rather than merging the two files, *keep them separate!* Any time both are needed, a "merge master county; by ctyfips;" may be used rather than a "set bigfile;".

Read "SAS Programming Tips", but be skeptical!

Lots more good stuff! BUT ...

Tip 4.12: Create indexes when appropriate.

A SUGI paper found "appropriate" is rare

Tip 5.21: Set the lower bound of arrays to 0.

Very confusing, unless you name your variables x0, x1, x2, ... rather than x1, x2, x3, ...

Tip 5.23: Use the stored program facility.

SUGI paper says time savings small, and its very confusing for large programs, since *only* datasteps are compile-stored, and each data step must be stored separately.

Tip 7.8: Compress large SAS data sets.

SUGI paper found time increases *several times* when operating on compressed data sets.

Tip 7.10: Store numbers as 1-byte character values.

Saves disk storage. For one-digit numbers, the character representation is 1 byte rather than 2 or 3. The tip also shows how to store numbers up to 255 in 1 byte character.

BUT UGLY!

Author information

David Sommer is the president of Optimal Systems, a consulting company specializing in SAS applications. He is available for workshops and consulting on this topic and others. Call 612/ 831-5724.