The representation of missing values in gretl

Allin, 2018-07-23

I'll start with an account of a few pertinent facts.

1. For a long time, gretl has represented "missing value" ("`NA`" for short) by `DBL_MAX`, the largest double-precision (8-byte) floating point number, approximately 1.7977×10^{308}. In this respect gretl follows a long-standing tradition in econometric software of choosing some particular numerical value, unlikely to be confused with valid data, to represent `NA`. Within that class `DBL_MAX` is a good choice since there's zero probability that this could represent a valid value of a socio-economic variable.

2. Double-precision floating point numbers are known in C parlance simply as "doubles", and we'll use that name below. In gretl, floating point values are always stored as doubles.

3. The C library recognizes a few bit-patterns in doubles as special (not regular numerical values). Basically there are three such cases: `nan` ("not-a-number"), `inf` (positive infinity) and `-inf` (negative infinity). These can arise as the result of certain calculations. For example, `log(0)` produces `-inf`, `log(-1)` produces `nan`, and `DBL_MAX * 10` produces `inf`. The reasons behind these results should be fairly obvious.

4. The C library follows well-defined (IEEE) rules in handling such special values. Any arithmetical operation with `nan` as an operand, or mathematical function call with `nan` as argument, will yield `nan` as result. You can flip the sign of an infinity (e.g. by multiplying by -1) but you can't "tame" it; multiplying `inf` by zero gives `nan`.

Against that background, a question: Would we be better off representing `NA` by `nan` in gretl? We'll consider some possible drawbacks and some advantages.

Here's the main argument against. There's surely a conceptual distinction to be made between a "missing value" that arises because an observation wasn't made, a record was lost, or whatever, and a "not-a-number" arising from an invalid calculation such a trying to take the logarithm of a negative number. Equating `NA` and `nan` would efface that distinction. In addition, while the IEEE rules for propagating `nan` in calculation are very close to those we would wish to use in propagating `NA`—just about every calculation involving `NA` should presumably yield `NA`, or perhaps `nan`—there's arguably one exception. If we interpret `NA` as simply an *unobserved* value, then `NA * 0` should be zero, but (reasonably enough) `nan * 0 = nan`.

We'll return to the last point below, but for the moment we switch to the main arguments in favour of redefining gretl's `NA` as `nan`, of which there are two.

1. If we're willing to set aside the case for `NA * 0 = 0`, one obvious advantage of treating `NA` as `nan` is that we then get the IEEE propagation rules "for free". Otherwise we need to set up our own (mostly parallel) propagation rules for `NA`. And we have to be *very* careful never to pass `NA` (= `DBL_MAX`) to any C-library operator or function, on pain of getting totally spurious results. (For example, `DBL_MAX/100` is a perfectly fine numerical value so far as the C library is concerned, but a meaningless one if `DBL_MAX` is standing in for `NA`.)

2. The second argument is really just an extension of the last point. As you might expect, we *do* have our own propagation rules for `NA`—applying to calculation involving series and scalars—and we *are* in fact very careful not to pass `NA` to C-library calculations.[1] If this were just a one-time coding cost that's already been borne, it wouldn't be much of an issue. But it's more than that. A common feature of today's hansl coding is traffic between series and matrices: a dataset holds series, but complicated calculations often involve matrices, and we want to be able to shuttle data between these two representations as seamlessly as possible. But many of gretl's matrix computations are farmed out to LAPACK/BLAS and we cannot "reach into" those calculations and ensure correct propagation of `NA` (= `DBL_MAX`). Therefore, whenever we transfer data from series into matrices we have to check for `NA` and replace with `nan`—an ongoing and quite expensive run-time cost.

We can now reassess the argument canvassed above, *contra* the `DBL_MAX` to `nan` switch. The case was that, in principle, "genuine" missing values and "not-a-number" should be treated as distinct. This might have some force if gretl maintained the distinction consistently, but in fact we don't. For reasons that are certainly debatable but which seemed "good enough" at the time, we decided not to allow `nan` and infinities in series and scalar values: whenever these arise in the course of calculation they are mapped to `NA`. This also means that when the results of matrix calculation are carried back to series, we need to perform the inverse operation of the `NA` → `nan` transformation mentioned above.

So where's all this going? I see three possibilities:

- Leave things as they are. After all, nobody has recently expressed dissatisfaction with the status quo.
- If we think it's really important, make an effort to respect the `NA` versus `nan` distinction more rigorously than hitherto (e.g. stop mapping the results of invalid calculation onto `NA`). I don't think this would be very easy.
- Redefine `NA` to `nan`, thereby saving a non-trivial run-time cost and permitting the removal of a special layer of `NA`-handing in libgretl.

Jack and I discussed the third option earlier this summer and I think we both favour it. I've experimented a bit and I'm fairly confident it would not be a disruptive change.

[1] Though every now and then a bug pops up where we've failed to prevent this!