# The Good, The Bad, And The Useful: Do Things Ever Go Right?

— **Bruce Barkstrom** (brb@ceres.larc.nasa.gov), NASA Langley Research Center

During one of the early EOSDIS design reviews, Dave Emmitt and I were lamenting the fact that the production scenarios being used to scope out the amount of computing power and disk storage didn't really represent the experience of validating large data sets very well. Right after the EOS satellites are launched, we expected the instrument teams to be as busy as they'll ever be, trying to piece together how their instruments are behaving and why the algorithms don't give what their inventors expected. However, the processing scenarios used in the review looked benign—continuous increases in computer power for algorithm testing and integration and gentle turn-ons for reprocessing—no frustrating error diagnosis sessions while Headquarters is breathing down your neck asking for "spectacular results." At the same time, we didn't really have a specific counterproposal to make.

By happenstance, a book I read, called "Just in Time" Production (Manufacturing Systems Engineering by Stanley Gershwin [1994]), has an interesting model for machines that fail and then have to be repaired. Either a production machine is "working," or it is "under repair." I thought "that's like algorithms— either they're working correctly or we have to fix them!" In Gershwin's book, just as with real algorithms, we find machines breaking at random (or, to be a little more precise about our problem, we discover at random that our algorithms aren't working). The time it takes us to fix what's broken is also random: sometimes we can find what we have to change quickly; at other times it takes almost forever.

The model Gershwin describes looks at a machine producing $\mu$ products in a given time. We might call $\mu$ the "rate of production." The probability that this machine fails in a time $\delta t$ is $p\delta t$. When the machine is "down," the probability that it gets fixed and can return to production in a similar time interval is $r\delta t$. The interesting outcome is that over a long time, the average rate of production is

$$q\mu = \frac{r}{r+p}\mu \qquad (1)$$

If we imagine that the machine doesn't necessarily stop production when it fails, but that it produces "defective" products, then we could interpret $q$ as the probability that the machine is producing good products.

Does this make sense for algorithms and data products as well as machines? Well, if we discover a lot of errors in a short time and if it takes us a long time to fix each one, then $p$ is high and $r$ is low—$q$ will be small. In this case, interpreting $q$ as the probability of having a good product makes sense, because we will develop a backlog of errors and what we produce when we run the algorithm isn't likely to be right. On the other hand, if we fix errors quickly and don't discover very many new ones, we're likely to have a lot of confidence that the products are good.

It's also reasonable to expect us to find fewer errors as we go forward in time. In a sense, we expect the pool of errors to be fixed and "the faster we find 'em, the fewer there are left." Thus, a plausible way in which our error discovery rate will behave over time is

$$\frac{dp}{dt} = -\frac{1}{\lambda}p \qquad \text{or} \qquad p = p_0 e^{-t/\lambda} \qquad (2)$$

Here, $p_0$ is the rate at which we discover errors initially. We might call $\lambda$ the "error discovery lifetime." If we use these two equations together, we obtain the "famous" logistic equation

$$q(t) = \frac{1}{1 + p_0 \overline{T}_r \exp(-t/\lambda)} \qquad (3)$$

Instead of using $r$, we have used $\overline{T}_r = 1/r$, the mean time to repair the algorithm. Initially, $q$ has a value of $1/(1 + p_0 \overline{T}_r)$. If we look at the situation several error discovery lifetimes after we start, then $q \approx 1$. Figure 1 shows how $q$ depends on time for a rough estimate of the parameters, $p_0$, $\overline{T}_r$, and $\lambda$.

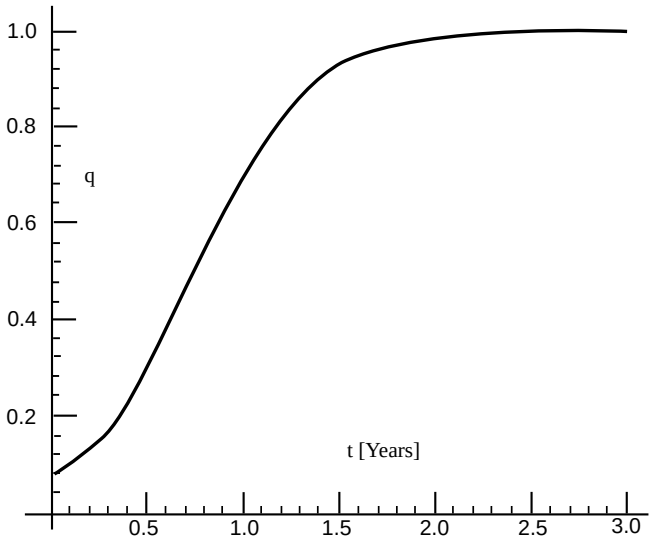


Figure 1. Estimated Probability of Reliable Data from a Single Algorithm. $q$ is the probability of producing "good" data with an algorithm as expressed by equation (3). We have taken $p_0$ = 24 per year, $\overline{T}_r$ = 0.5 years, and $\lambda$ = 0.3 years, based on informal estimates from ERBE experience.

For right now, we'll use some rough estimates of the parameters based on my own memories of the kind of struggle we had getting the Earth Radiation Budget Experiment (ERBE) data into condition for archive. I remember looking at image-like plots we made of those data in the first month or two after launch, and recall several errors we had to fix immediately. From this recollection, it seems that setting $p_0$ = 2 per month (or 24 per year) as the initial error discovery rate puts us in a reasonable ballpark. The second parameter we need is $\overline{T}_r$. In thinking back over the ERBE experience, I recall many errors that were easy to fix— simple one-line problems in the code. However, the mean time to repair is strongly influenced by the errors that were hardest to fix: the "more than one year" struggle to get the Angular Distribution Models right, the six-month ordeal with "striping," the several-year struggle with "offsets." Thus, six months $\overline{T}_r$ seems reasonable. And finally, what do we do about $\lambda$? I'll take it to be about 0.3 years, hoping to catch at least a rough sense of how long it takes our error discovery process to damp out.

**What Happens When Things Don't Go Right**

Of course, if you're a data producer and discover problems, your immediate thought is: "Where's the problem?" After you've gotten some data and looked at it, you're likely to respond: "I need to make some more runs with this program—but with a few changes!!" Life in the Distributed Active Archive Centers (DAACs) and in the Science Computing Facilities (SCFs) is really fun when this happens.

There are several strategies we could apply to error diagnosis and repair. One that we used on ERBE was to concentrate on small samples of data and work on them until we seemed to remove the errors. Then, we would take a larger sample and reexamine that for errors, using the corrections we had developed on the first, small sample of data. Finally, we would try integrating the correction into the operational code and modify the statistics we were tracking to monitor the data product quality. Often, we stopped production while we searched for good ways to fix the algorithms. Of course, this made error detection slower, since then error detection and fixes became a strictly serial processing procedure.

The important thing to note is that error diagnosis and algorithm repair are major activities after launch. Somehow, we feel that the amount of diagnostic processing should be related to how bad the data seems to be and how rapidly we're finding and correcting errors. The good side of this model is that it's simple, it's continuous, and (intentionally) doesn't care about the real details of the repair and production strategy we're using. The bad side of the model is that it doesn't give us an easy way to estimate the three parameters from our previous experience without a fairly detailed study.

In the absence of a detailed historical study or a better theory, we'll make the simplest assumption we can: the amount of work we have to do is proportional to how many errors we've found recently. With the curve in Figure 1, we'll have lots of work early, and then as the algorithm's reliability improves, the amount of additional diagnostic work will decrease as well. Thus, if $\mu$ is the number of jobs we have to run in a given month for "standard production," we estimate the number of jobs we'll have to run at the PI's SCF to be about $(1-q(t))\mu$, and the number at the DAAC to be about $\mu + (1-q(t))\mu = (2-q(t))\mu$. The primary reason the DAAC load increases with this model is that we're accounting for a continuous flow of algorithm repairs, which have to be tested and integrated into the standard production software.

It also seems that it's reasonable to expect the network traffic to increase. No PI is a team unto himself or herself. Other members of an investigation will need to look at the data to make their own judgments of what needs to be done. If this assumption is reasonable, the diagnostic data flow will probably be proportional to $1-q(t)$. Building up a detailed traffic model that estimates how many diagnostic files have to be transfered from the production site to other locations is a matter for another article.

We can also use this model of data quality to estimate when the teams will want to start reprocessing. The usual philosophy seems to be "wait until the data are good enough before we start re-doing products." In more-quantitative terms, if we set a threshold, $q_0$, such that we don't start reprocessing until $q(t_{reproc}) > q_0$, then we wait until

$$t_{reproc} = -\lambda \ln[(1-q_0)/(q_0 p_0 \overline{T}_r)] \qquad (4)$$

If we allow the teams to be careful, and $q_0 = 0.99$, so there's only about a 1% chance of something being wrong, we find $t_{reproc} \approx 2.1\ years$—a calculation we can perform just with a $15 pocket calculator.

**Some Implications**

This model for how algorithms act under stress is a lot simpler than real life. However, it seems to have the right sense in three important ways: a) diagnostic work will be heavy at first and will taper off; b) both the estimated initial data product quality and the length of time needed to get things to the point where reprocessing is justified seem in accord with my experience as a data producer on ERBE; and c) the time to start reprocessing looks "reasonable."

In talking with other EOS data producers who've had experience with large-scale data production, this model also seems to fit with at least a rough characterization of their experience.

One important way in which EOS production differs from this model is that all of the EOS data producers have "algorithmic food chains," in which higher level data products have dependencies on lower level products that we now see to be of evolving quality. A simple way to look at how good the higher level products are is to assume that the errors lower down are independent of the errors higher in the food chain. If we do such a multiplication, we can expect the higher level products to take longer to get right—it takes a while to find out which errors come from products lower down and then it takes more time to fix them. This observation also fits with our ERBE (and other data producer) experience.

This model is also likely to be useful for investigators who want to do science with EOS products (or with other data—there's no reason to think EOS is exceptional in this regard). Clearly, if you want to use Level 1 (radiance) data, the parameter $q$ in Figure 1 is probably descriptive of whether you're likely to have reliable data. If you want to use higher levels, the product reliability is likely to look like some integer power of this curve until the data are reprocessed.

Certainly, there's a lot of work to do. For one thing, the numerical parameters I've suggested here need a stronger basis in what we've recorded from our past history. I have used this model to estimate how many computers we need for the early years of production. Ellen Herring in the Earth Science Data & Information System (ESDIS) Project's System Engineering Office suggested that these estimates need to allow the computer MFLOPS ratings to grow with time. Her

group's experience on the Upper Atmosphere Research Satellite (UARS) may be very helpful in setting up contingencies for EOSDIS. It may even be useful to consider other models to estimate how much diagnostic work we have to do and how that diagnostic work is related to data quality.

**An Opportunity For Reader Involvement**

For those members of the EOS community who want to take a more-active interest in these kinds of problems, here are a few "brain teasers" to work on. I won't offer any prizes, except to suggest that good answers be published here or in some other suitable place:

1. Either suggest an alternative form for this data quality expectation or provide a more-detailed justification for this curve—preferably with some empirical or theoretical justification for any parameters you need to apply the model to practical problems of EOSDIS production. Extra credit for detailed empirical studies with documentation.

2. Confirm or provide a theoretical or empirical counterexample to the suggestions that the number of processing jobs at the DAACs will scale as $(2-q(t))\mu$ and that the diagnostic product flow between the production site and the SCFs will be proportional to $(1-q(t))\mu$. Extra credit to anyone finding the constant of proportionality. Failing mark awarded for mere complaints that the form suggested here is unreasonable.

3. Determine an optimal quality assurance (QA) processing strategy that jointly minimizes the cost of processing and the delay in archiving high-quality data for the following conditions: a) all errors to be found are initial errors of omission in the algorithms; b) continuous generation of new errors due to instrument perturbations such as contamination of calibration sources, or radiation damage and aging of electronics; and c) continuous creation of new errors by code used to fix old errors [this possibility suggested by Rich Ullman].

   Answers to this problem must be accompanied by a *bona fide* proof of optimality. Practical answers or hypotheses should be identified as such, although good reasoning gets partial credit. Without proof, answers are subject to trial by battle with real data from EOS.

4. Provide a closed, analytic form for the expected data quality of a chain of algorithms and products as production and algorithm repair proceed. In the absence of such a solution, establish a plausible form by computer simulation or analytic approximation. If either of these two solutions proves too difficult, at least provide a study of some simple cases.

**References**

Gershwin, Stanley B., 1994: Manufacturing Systems Engineering, Prentice-Hall, Englewood Cliffs, NJ, 501 pp.

Note: The author hopes that this article will be the start of a regular series of columns in the *The Earth Observer* on Engineering EOSDIS. As part of the work the EOS Ad Hoc Working Group on Production (AHWGP) has done to improve our understanding of production, I've begun to build an integrated computer model of EOSDIS and its costs. There are several topics that have surprised me as I've tried to put this model together, and I'd like to share them with the community. If time allows, I may also be able to assemble this material into a more-extended form. ■