Are Computer Simulations Science?

Mats Holmström

Docent Lecture
14 October 2004

Overview

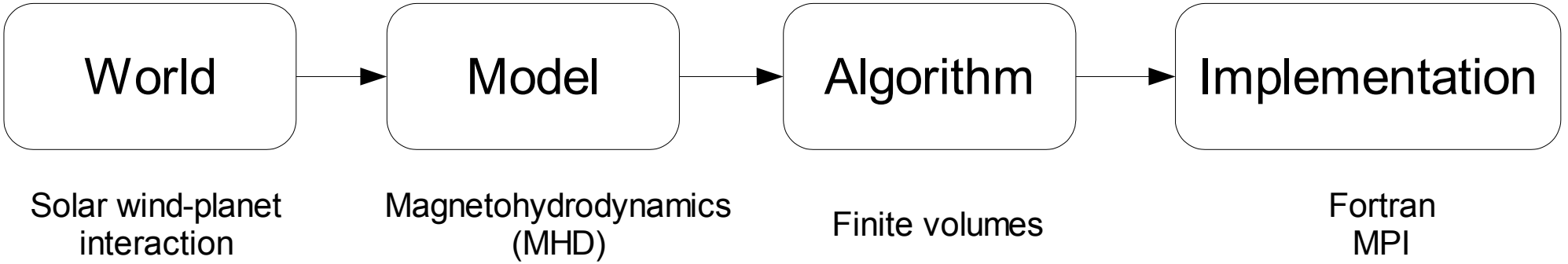
- Traditional physics $\leftrightarrow$ Computer simulations
- Problems with current practices?
- Suggestions for solutions

Traditional Physics Research

- Hard sciences: *Reproducibility*
  - External reality
  - Simple physical laws
  - Independence of the observer
- The scientific method: Theory ↔ Experiment

Stages

Space Physics example

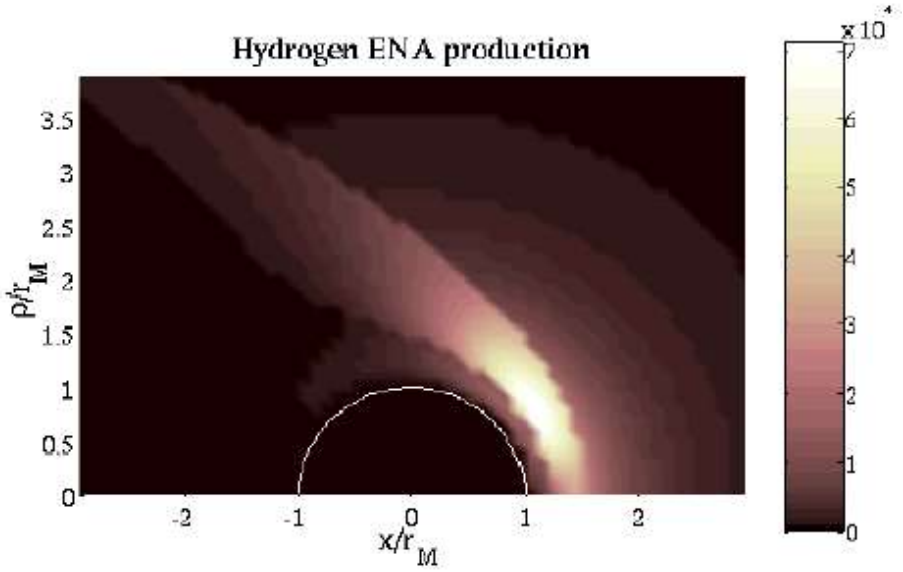


Reproducibility in Computing

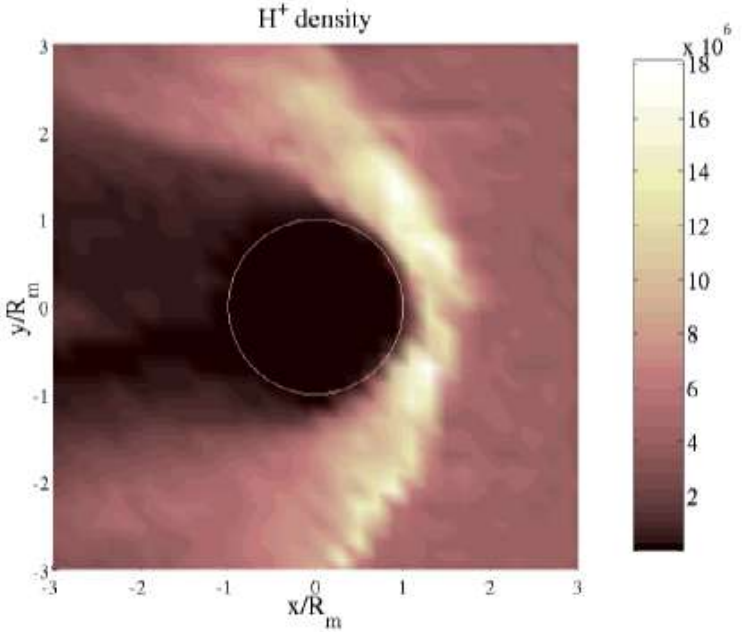
Difficulties:

- For one program:
  Changing environment. All must be known.
- For many programs: We will focus on this.
- Results are often reproduced for different set of parameters/conditions
- The situation (Bruckheit et al.):
    - Researchers cannot reproduce others work
    - Advisors cannot investigate student's problems
    - Researchers cannot reproduce their *own* work

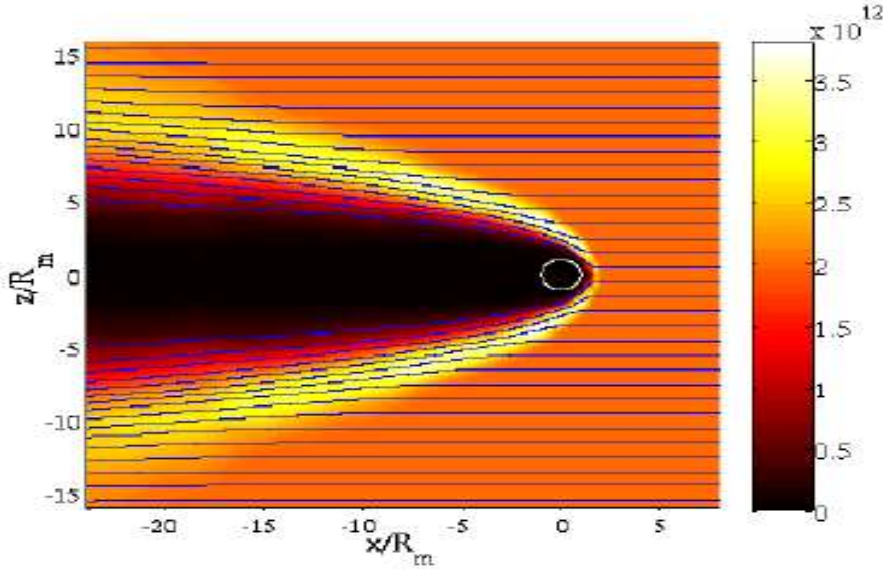
Different Flow Models



Empirical

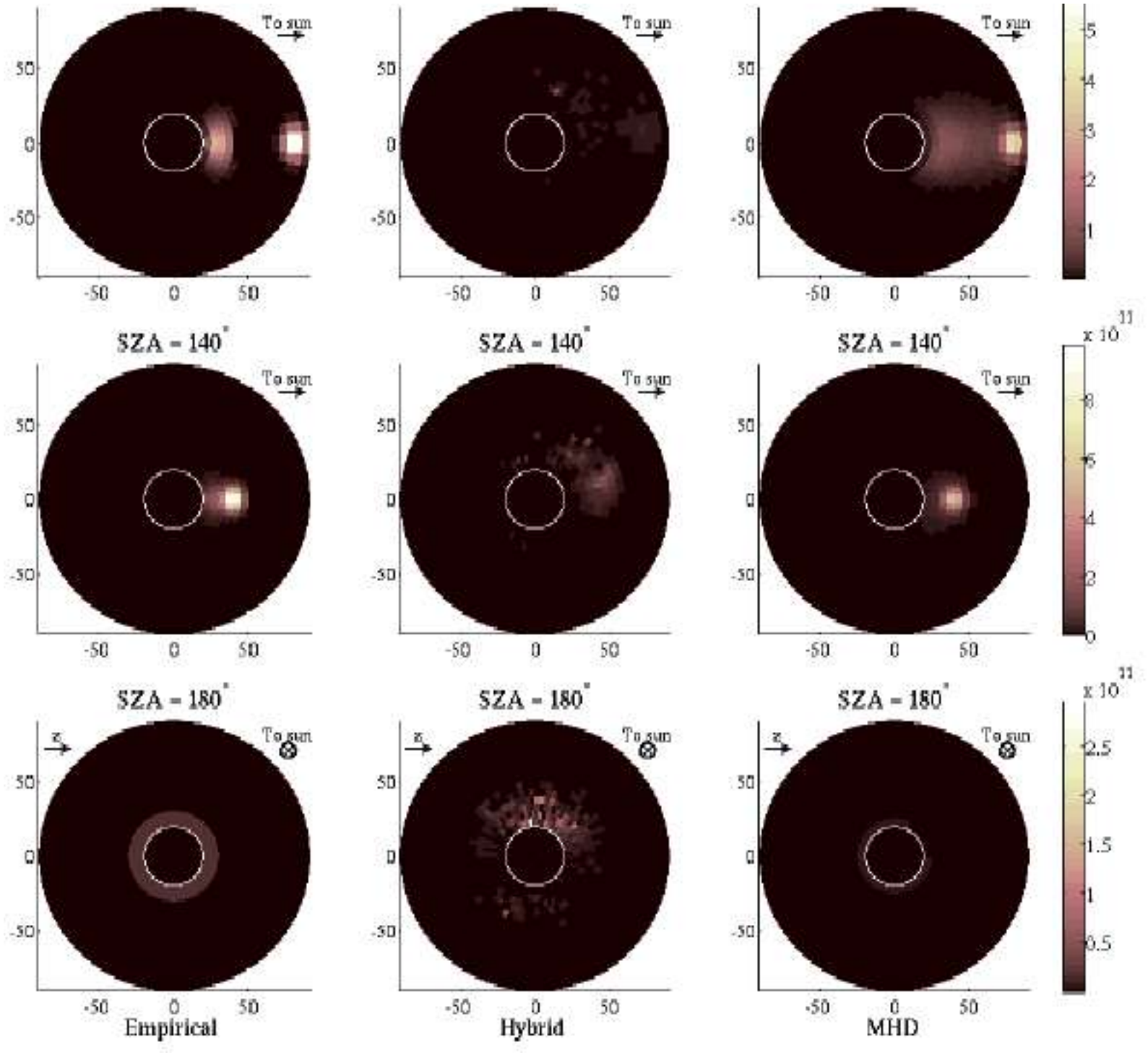


Hybrid



MHD

Comparison of Energetic Neutral Atom Production



How Accurate Is Scientific Software?

Les Hatton and Andy Roberts

Abstract—**This paper describes some results of what, to the authors' knowledge, is the largest N-version programming experiment ever performed. The object of this ongoing four-year study is to attempt to determine just how *consistent* the results of scientific computation really are, and, from this, to estimate accuracy. The experiment is being carried out in a branch of the earth sciences known as seismic data processing, where 15 or so independently developed large commercial packages that implement mathematical algorithms from the same or similar published specifications in the same programming language (Fortran) have been developed over the last 20 years. The results of processing the same input dataset, using the same user-specified parameters, for nine of these packages is reported in this paper. Finally, feedback of obvious flaws was attempted to reduce the overall disagreement. The results are deeply disturbing. Whereas scientists like to think that their code is accurate to the precision of the arithmetic used, in this study, *numerical disagreement grows at around the rate of* 1% *in average absolute difference per* 4000 *lines of implemented code, and, even worse, the nature of the disagreement is nonrandom.* Furthermore, the seismic data processing industry has better than average quality standards for its software development with both identifiable quality assurance functions and substantial test datasets. Comparing the results reported here with other work by Hatton showing broadly similar statically detectable fault rates in software from different disciplines gives strong indications that the software realisations of work in other scientific fields may be a great deal less accurate than many would believe. Against this backdrop, the authors believe that little progress will be made in some sciences until the problem is reduced, particularly in remote sensing, where the answer is generally inaccessible to direct measurement. To this end, the feedback experiments that formed part of the study proved valuable, resulting in significant reductions in disagreement.**

for example, that *resolution* (say, around 0.001% in typical floating point formats) and *accuracy* are synonymous—the widespread use of double precision in some sciences is indicative of the accuracy expectations. The software testing procedures used are left entirely to the authors of the scientific work. Regrettably, as we shall see, scientists are no more successful at writing reliable software than anyone else.

This paper attempts to analyze the scale of the problem in two distinct ways. First, the results of static fault analysis for many different application areas [1] are briefly reviewed. The object of this parallel study was to see if different scientific application areas tended to have the same programming language problems or whether certain areas exhibited a greater or lesser susceptibility than the average to the inadvertent misuse of programming language.

Second, the results of analyzing one particular application area, that of seismic data processing, is studied in depth. This particular application area is probably unique in scientific computation in that it has remained both a highly competitive and a mature environment. During the last 30 years or so, some 15 to 20 proprietary packages with several recognisably different architectures have been developed in effective isolation from each other, *all ostensibly doing the same thing*. During this time, new algorithms have been added and old ones have been rewritten to take advantage of advances in either hardware or software. The authors have chosen to refer to this as an N-version experiment, although there appears to be some controversy over this nomenclature. Here the authors mean the comparison of outputs of N software packages written to

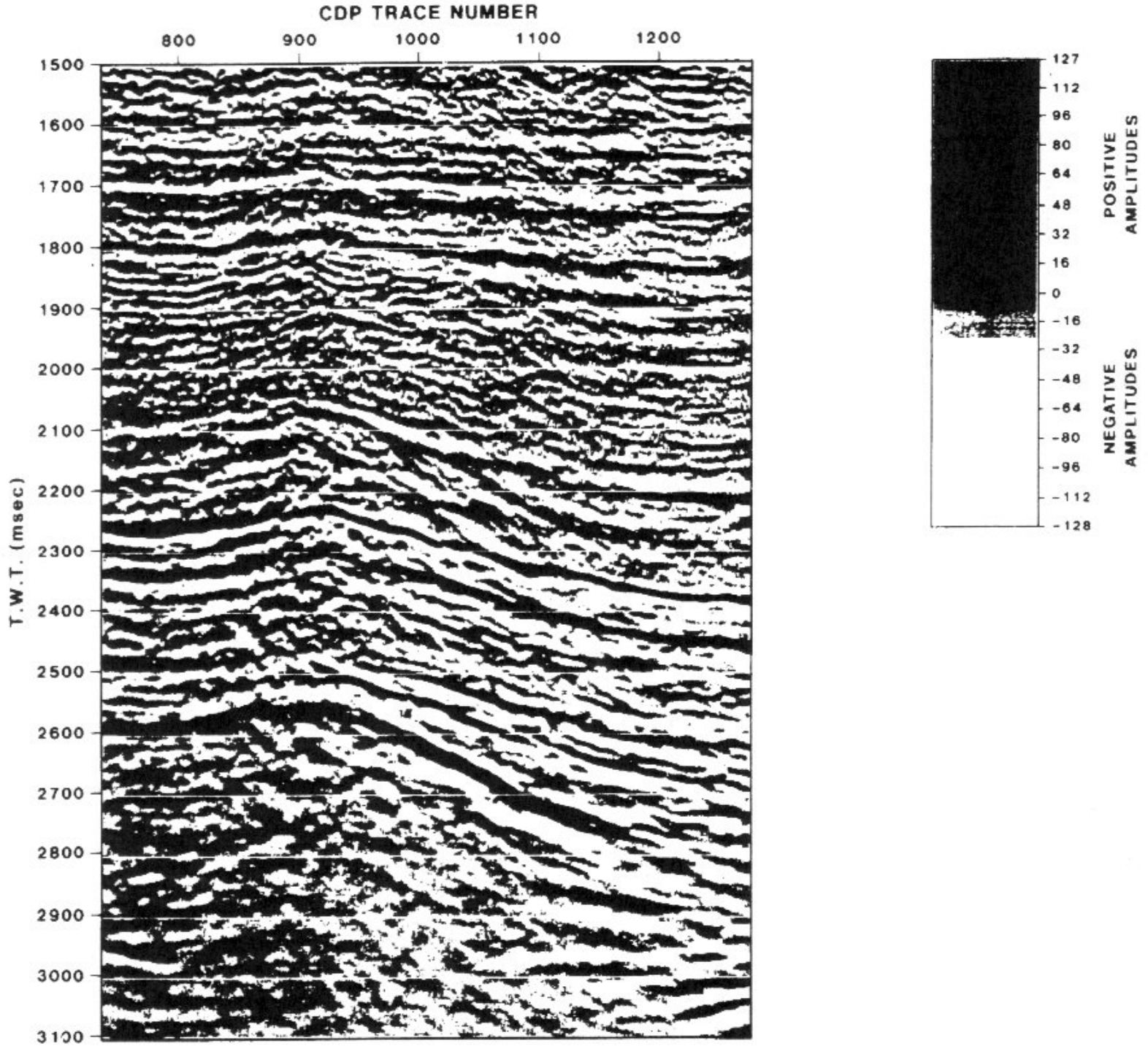


Fig. 2. An example of the final output of a seismic data processing sequence as analyzed by the geologist. The black lines correspond essentially to echoes from strata within the earth and give valuable information concerning the stratigraphy. The horizontal scale is distance along the surface of the earth and the vertical scale is echo travel time.

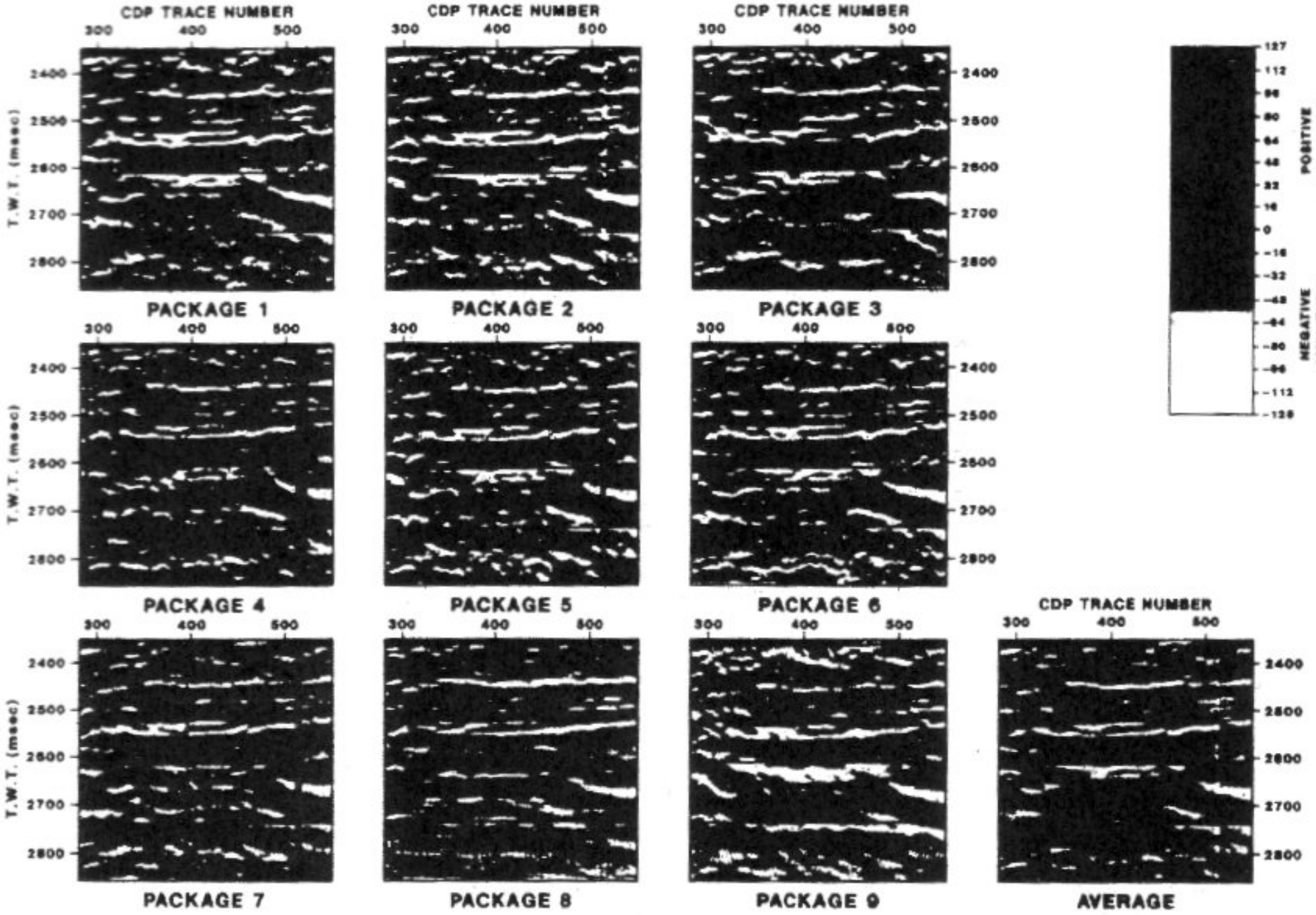


Fig. 10. A collage of the nine different identically processed end-products (calibration point 14) as would be analyzed by a geoscientist. It would be nice to find that they agree to within the single-precision floating-point arithmetic used, i.e., around 0.001%. In practice, differences amount to around 100 000 to 1 000 000 times worse than this. Note that the bottom right cross-section represents the average of all the nine individual cross-sections. Horizontal stripes are timing lines and are the same on each and the vertical stripes correspond to areas of gross departure and have been statistically trimmed.

Reliability in Computing

- Reproducibility does not imply reliability
- Reliable software if
  it always does what is expected of it.

  Problems:
  - bugs
  - numerical inaccuracy/instability
    Traditional numerical analysis: Stability, round-off,...

Reliability — an Example:
Charge Exchange in the Martian Exosphere

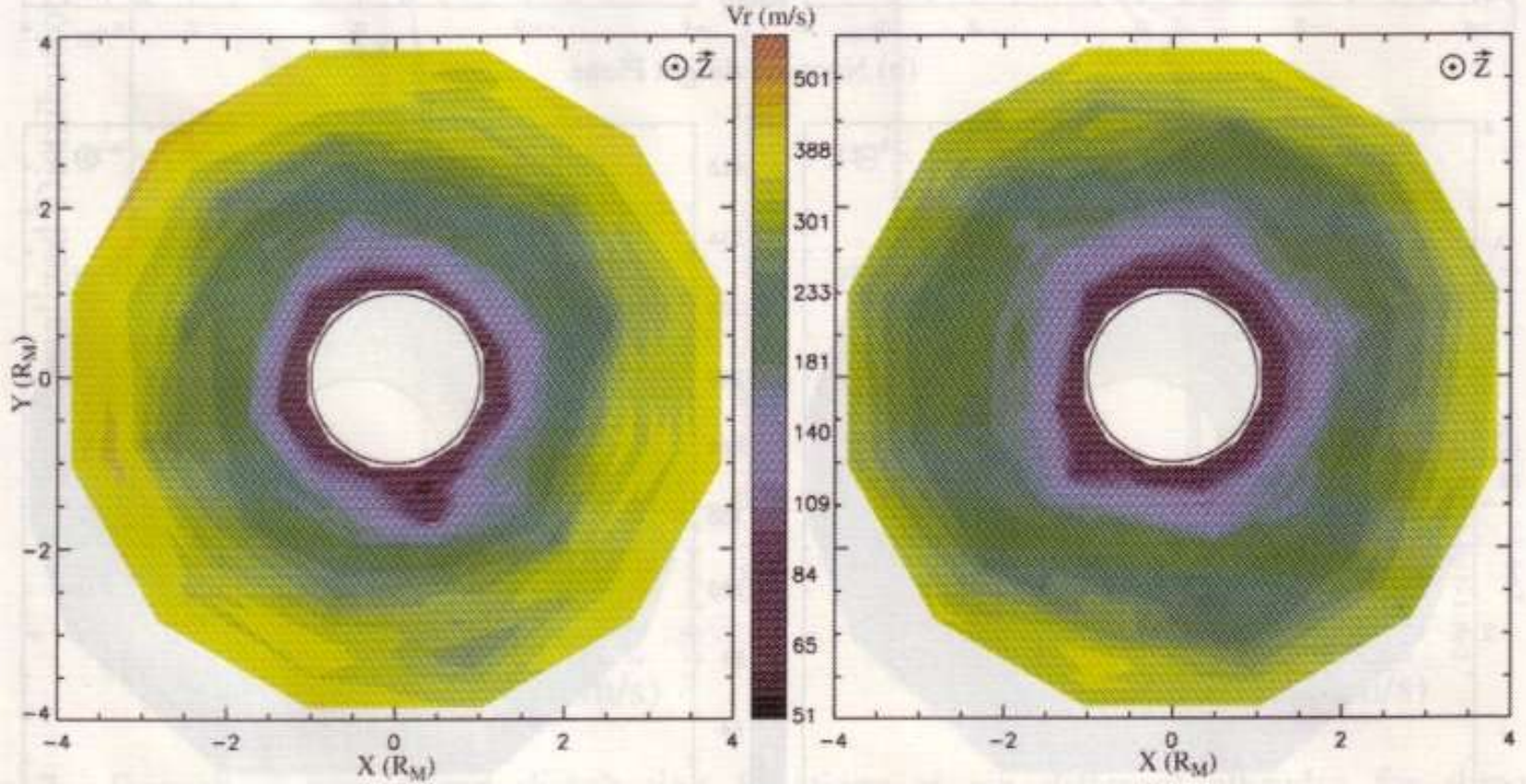


Figure 5. Exosphere radial velocity contours in equatorial cut for two cases: (left) primary exosphere (PRE) and (right) charge-exchanged exosphere (CELSPE). The Mars-Solar-Ecliptic coordinate system is used.

JGR, v108, A10, 2003.

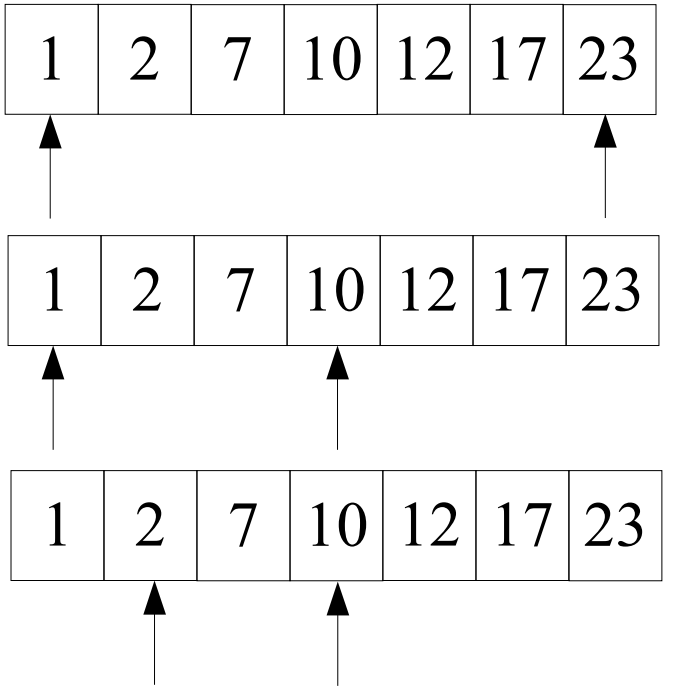
Debate

- "Physics computing is easy, and therefore reliable"
- Counter example: Binary search

Binary Search

- Problem:
- Determine if the sorted array $x[0..n\text{-}1]$ contains the target element t. The answer is stored in p.
  If t is not in the array, p is -1.

- Solution by binary search
  t = 7:



Binary Search

- 10% of professional programmer can implement this without errors in a couple of hours (Bentley)
- The first binary search was published in 1946.
  The first binary search without bugs was published in 1962 (Knuth)

Current Physics Practice

Software usage:

- External software
- Public scientific programs
  (by large communities or small groups)
- Reuse of numerical results
- Unshared resources

Testing

- Usually only by global validation (exceptions: large libraries)
- Comparison with approximation, previous calculations, and experimantal data
- Consistency with conservation laws (mass, energy, ...)
- Peer review

Bug management unusual

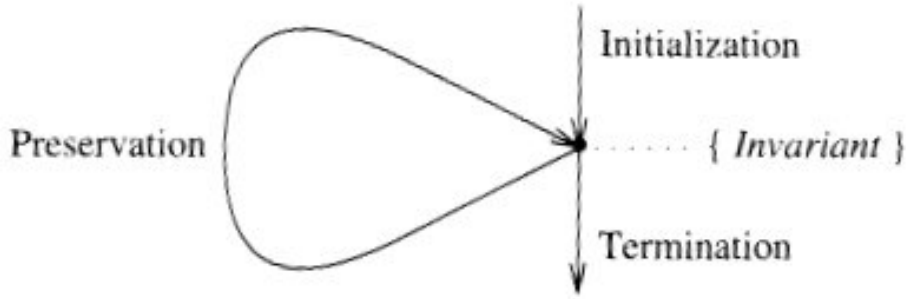
Open source code?

Why keep the source code secret?

- Work
- Fear
- Control
- Commersial value
- No benefits

Valid reasons, but not in tune with science

Iteration Control Structures. To prove the correctness of a loop we must establish three properties:



We first argue that the loop invariant is established by initialization, and then show that each iteration preserves its truth. These two steps show by mathematical induction that the invariant is true before and after each iteration of the loop. The third step is to argue that whenever execution of the loop terminates, the desired result is true. Together these establish that if the loop ever halts, then it does so correctly; we must prove that it does terminate by other means (the halting proof of binary search used a typical argument).

Functions. To verify a function, we first state its purpose by two assertions. Its *precondition* is the state that must be true before it is called, and its *postcondition* is what the function will guarantee on termination. Thus we might specify a C binary search function as follows:

```
int bsearch(int t, int x[], int n)
 /* precondition: x[0] <= x[1] <= ... <= x[n-1]
    postcondition:
        result == -1     =>  t not present in x
        0 <= result < n  =>  x[result] == t
 */
```

Suggestions for solutions

- Publish source code and parameter settings
- Write *specifications*
- Use *source code control* systems
- *Testing*
  - Perturbed parameters and initial conditions
  - Automated testing
  - Automated generation of figures in publications: *Reproducible research*
  - Use *assertions* and *invariants* in the code

Are Computer Simulations Science?

They can be...

References

(1) Computing in Physics: The Challenges of Reproducibility, Reliability, and Complexity, Vincent Sacksteder, preprint, 2003.

(2) How Accurate is Scientific Software?, Les Hatton and Andy Roberts, IEEE Transactions on Software Engineering, v. 20, n. 10, 1994.

(3) Designing Scientific Components,
Paul F. Dubois, Computing in Science and Engineering, IEEE, Sep/Oct 2002.

(4) Making Scientific Computations Reproducible, Matthias Schwab, Martin Karrenbash, and Jon Claerbout, Computing in Science and Engineering, IEEE, Nov/Dec 2000.

(5) WaveLab and reproducible research, Jonathan B. Bruckheit and David L. Donoho, preprint.

(6) Programming Pearls, 2nd ed., Jon Bentley, 2000.