Why should we compare models?

Mats Holmström

Swedish Institute of Space Physics

matsh@irf.se

SWIM 08
San Diego
January 25, 2008

Overview

- Background
- Traditional physics ↔ Computer simulations
- Problems with current practices?
- Suggestions for solutions
- Why should we compare models?
- Future activities?

Background

- Published papers on the Mars-solar wind iteraction show small and large differences in, e.g., ion escape rates, ion flux morphology, ...
- Is the reason due to
  - Model? MHD? Hybrid?
  - Boundary conditions and source terms? Solar wind conditions? Exosphere model? Ionosphere model?
  - Numerical issues? Grid resolution? Number of particles? Simulation time?
  - Errors? In the implementation? Model problems?

Mars-Solar Wind Models

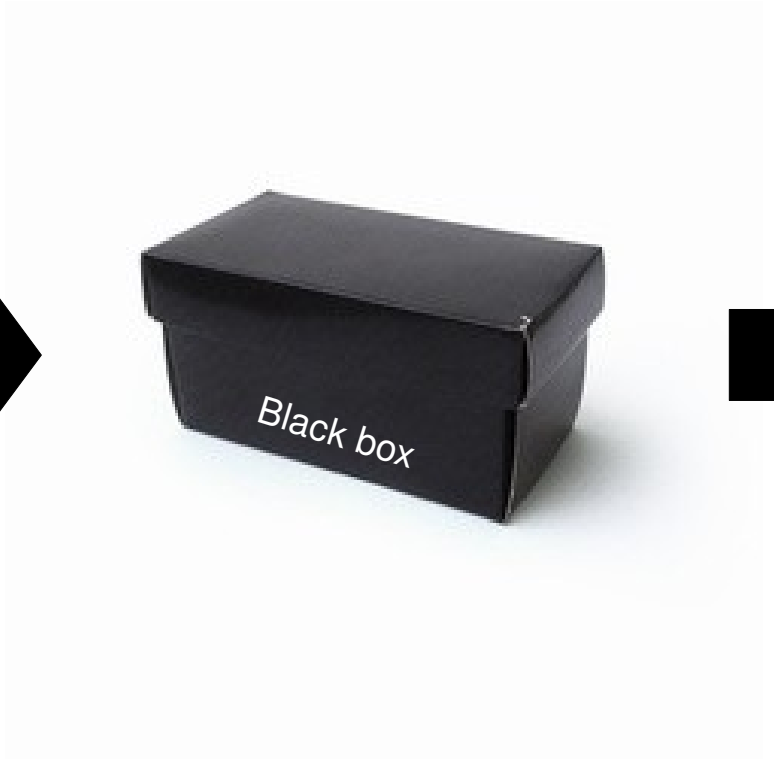
Mars-Solar Wind Models

Input

Solar wind

Exosphere

...



Output

Escape rates

...

Traditional Physics Research

- Hard sciences: *Reproducibility*
  - External reality
  - Simple physical laws
  - Independence of the observer

Reproducibility in Computing

Difficulties:

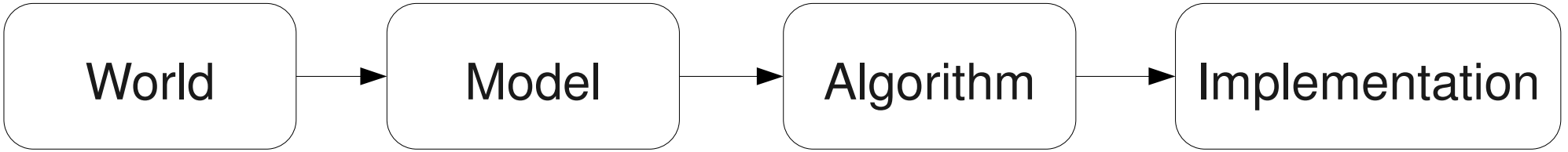
- For one program:
  Changing environment. All must be known.
- For many programs: We will focus on this.
- Results are often reproduced for different set of parameters/conditions
- The situation (Bruckheit et al.):
  - Researchers cannot reproduce others work
  - Advisors cannot investigate student's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

Reproducibility in Solar System Simulations

- Of course results reproducible, e.g., a hybrid simulation of the Mars-Solar wind interaction with a Chamberlain exosphere, and .... as described in the paper.
- Anyone should be able to make a new version of the modell, thus reproducing the results
- Problem: Details matter in computing...

Stages

Space Physics example



Solar wind-planet interaction

Magnetohydrodynamics (MHD)

Finite volumes

Fortran
MPI

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

Example

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

Example

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

Example

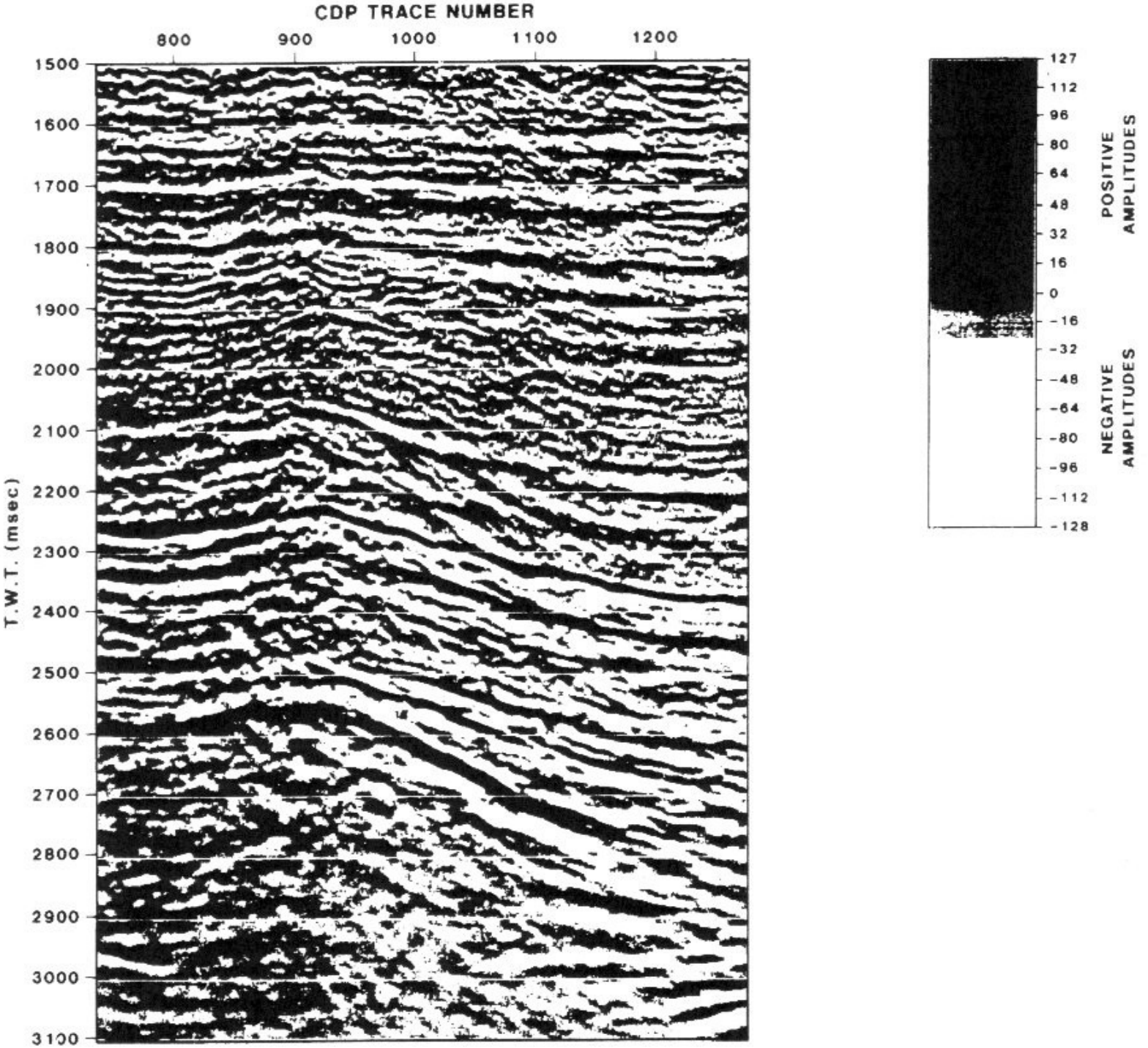


Fig. 2. An example of the final output of a seismic data processing sequence as analyzed by the geologist. The black lines correspond essentially to echoes from strata within the earth and give valuable information concerning the stratigraphy. The horizontal scale is distance along the surface of the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

Example

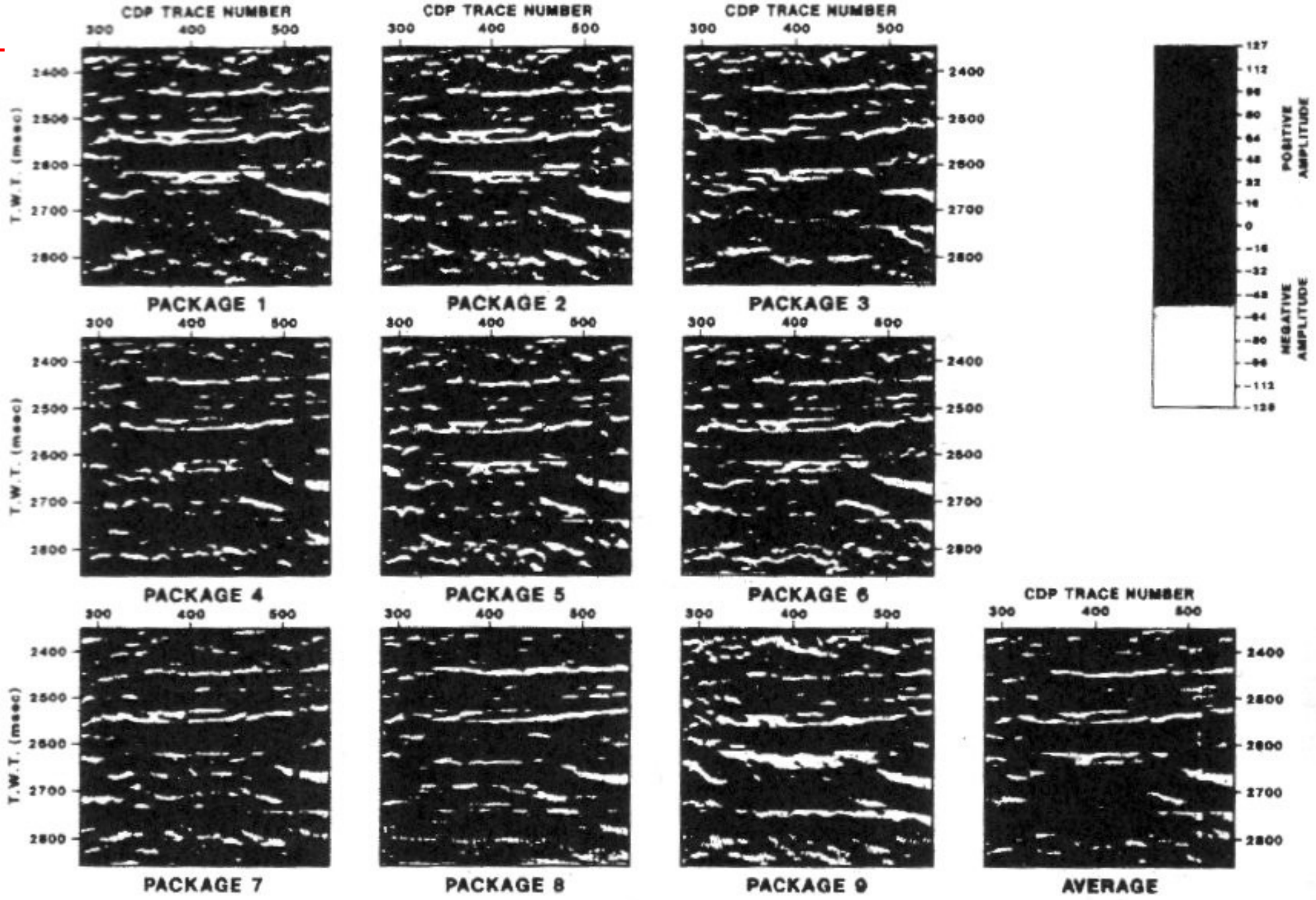


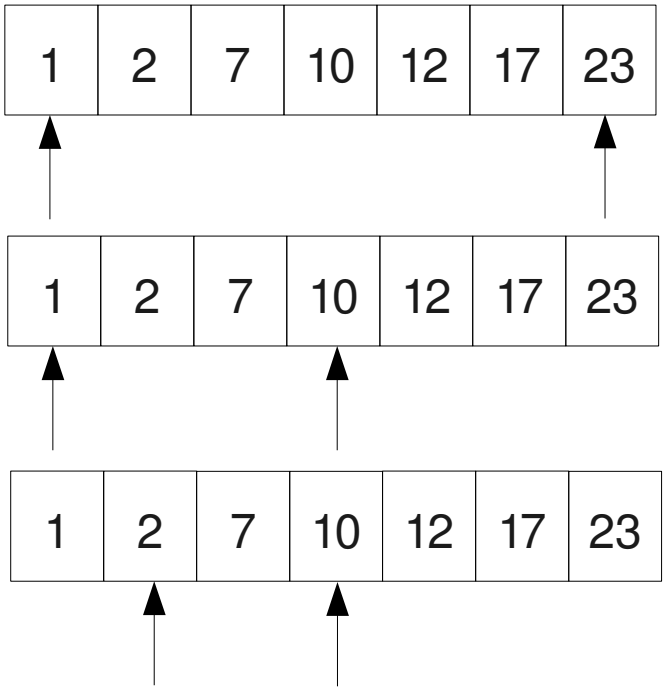
Fig. 10. A collage of the nine different identically processed end-products (calibration point 14) as would be analyzed by a geoscientist. It would be nice to find that they agree to within the single-precision floating-point arithmetic used, i.e., around 0.001%. In practice, differences amount to around 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.

Why? Are code errors a problem?

- "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)

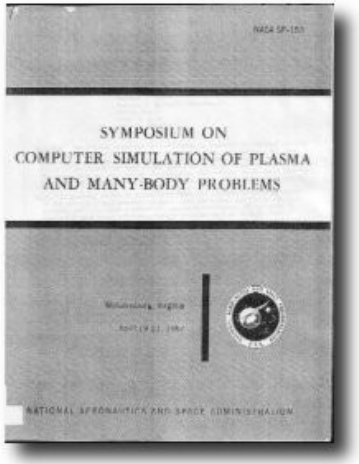
Open source code?

Why keep the source code secret?

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

Valid reasons, but not in tune with science

A 40 year old debate



When an institution publishes a scientific paper, or even a report, the division head or professor won't let it go out without its being checked by colleagues and refereed by himself. So equally, I would advocate that we should write programs and subroutines that we are not afraid to give to other people. We should try to make each program as readable as a mathematical textbook, and should try to adopt the principle that all important programs are published. If a program is never published and if it is not intelligible, how are we to known that the results are right? In a few years, some of these programs may produce a great amount of data and some very interesting physical results, which are analogous to astronomic observations or to records of careful scientific measurements - say on the earth's magnetic field. If this data is based on programs of quite unknown validity, nobody can check the results and I think that the scientific literature may get corrupted. So I feel that we should actually publish all our programs in an intelligible form, and set very high standards comparable to those of the rest of science.

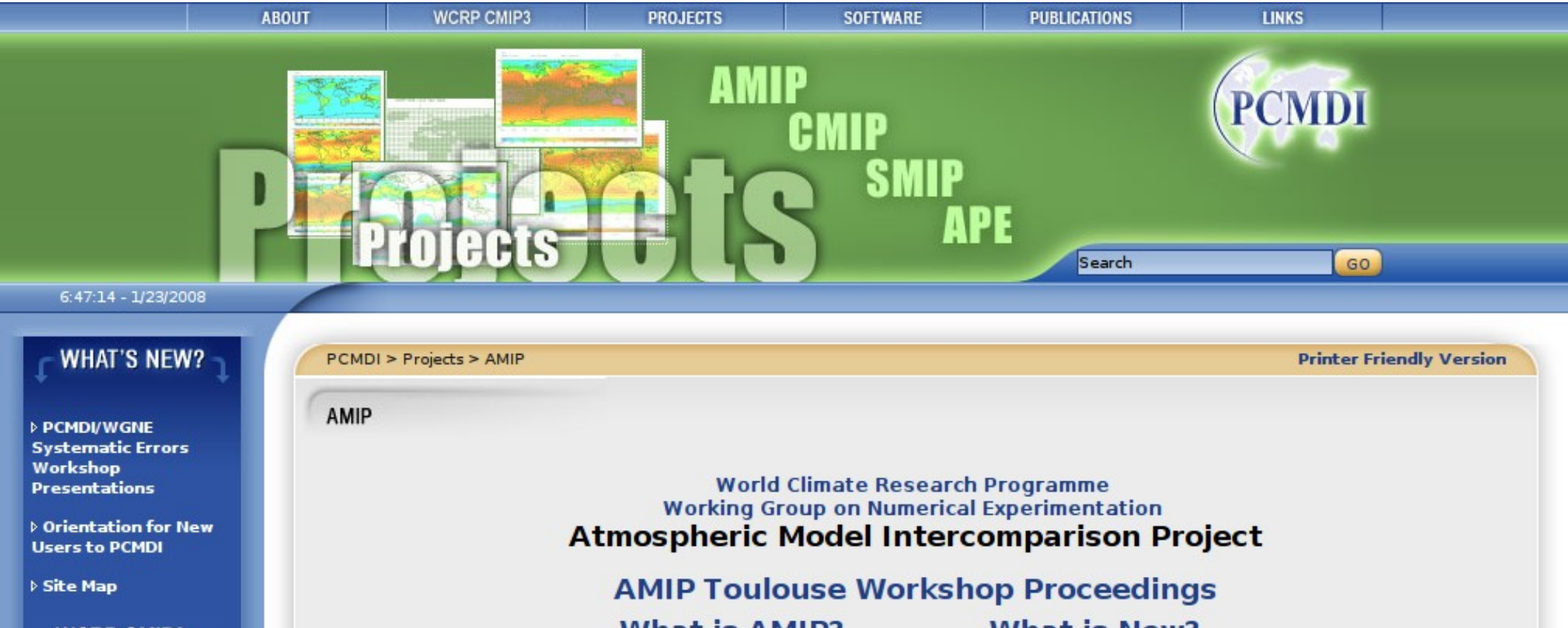
Keith V. Roberts, Panel Discussion on: Applications and Organization of the new Field of Computational Physics, Symposium on Computer Simulation of Plasma and Many-Body Problems, Williamsburg, Virginia, April 19-21, **1967**, NASA SP-153.

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*

Model Intercomparisons

- A first step toward reproducible simulations
- Many examples from other fields, e.g.,



Why should we compare models?

Our proposed strategy to tackle the problem of reproducibility and standardization of numerical models in planetary science is to initiate model intercomparison activities. The benefits of model intercomparison activities are many, among them

(1) Ensures that the science is reproducible

(2) Model errors will be found

(3) Standardized data formats for the simulation outputs, making simulation results easier available to data analysts that are not directly involved in simulations. This will also stimulate the comparison between observations and the results of numerical models.

(4) Standardized software interfaces, services, and libraries. This will enable software reuse, e.g., of visualization tools and data I/O libraries. It will also simplify the coupling of models, e.g., magnetosphere, exosphere, or thermosphere models.

(5) Collaboration between different research groups, using different codes, will increase and provide new research groups in the field with a starting point and reference solutions, and

(6) New diagnostics will be found, i.e. new ways of looking at model results.

Model Intercomparisons

- An iterative process.

  Continue after this meeting with

  - Workshops
  - Website
  - ...
- Two possible options to continue the effort
  - ISSI team proposal
  - EU FP7 Europlanet Proposal

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.