# Science relies on computer modelling – so what happens when it goes wrong?

April 1 2016, by Jeremy Gibbons

Modelling three bodies interacting with each other is harder than it may seem. Credit: Kevin Gill/Flickr, CC BY-SA

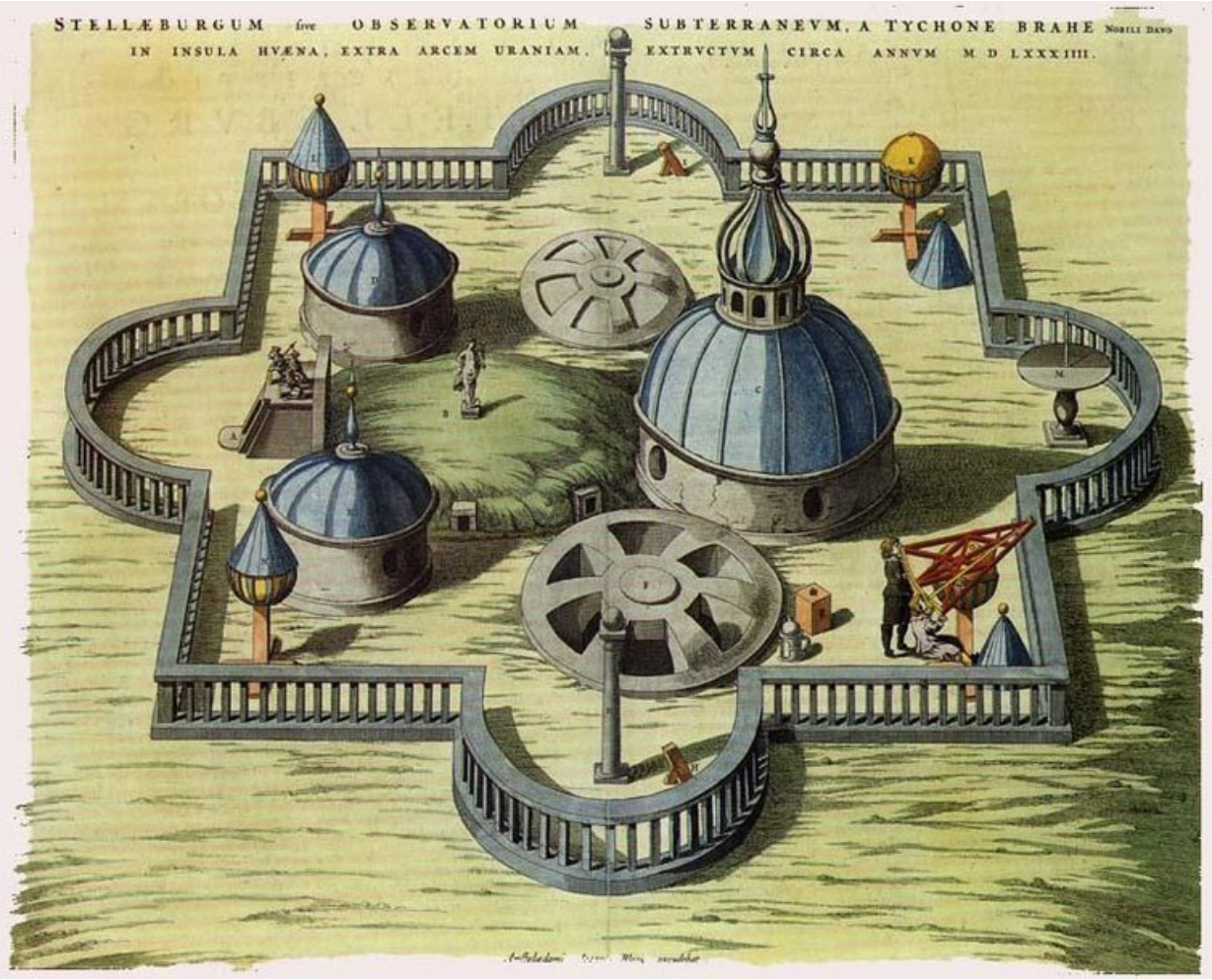
From the transforming discovery of penicillin to the theories of relativity and quantum mechanics, science progressed with mind-boggling speed even before there were computers. Much of this is down to the robustness of the scientific method: scientific results are validated by being replicated and extended by other scientists.

But the way we do science is changing – we now rely increasingly on complex computer models to understand nature. And it turns out that these models can be nearly impossible to reproduce – meaning an important touchstone of science is being challenged. So what are the real-world repercussions of this change and what can we do about it?

Pre-modern science – known as "natural philosophy" – was empirical. Empirical science uses past observations to make predictions about the future, which may then be tested. Tycho Brahe, a 16th-century Danish astronomer, managed to make accurate and comprehensive observations of the heavens in this way.

Modern science, however, is theoretical. Theoretical science also makes predictions, but it derives them from mathematical models rather than from prior observations. Think of Isaac Newton's laws of motion, such as the inverse square law of gravitation.

For example, there is an equation describing the orbit of the Earth around the sun. This equation can be used to build a computer model into which you can just plug certain variables and see how the solution changes. You could just plug in a future date and read off the position of the Earth at that date. You could also use the same program to model other planetary systems – it's all based on the same mathematics. All you have to do is plug in different masses and various other properties of the bodies involved.




Drawing of Tycho Brahe's observatory in Denmark. Credit: Willem Blaeu/wikimedia

Such mathematical equations are great when they are available – but often they are not. For example, we know that there is no simple equation that solves the so-called "three-body problem", which describes three bodies orbiting around and influencing each other by gravitational forces – like the moon, Earth and sun.

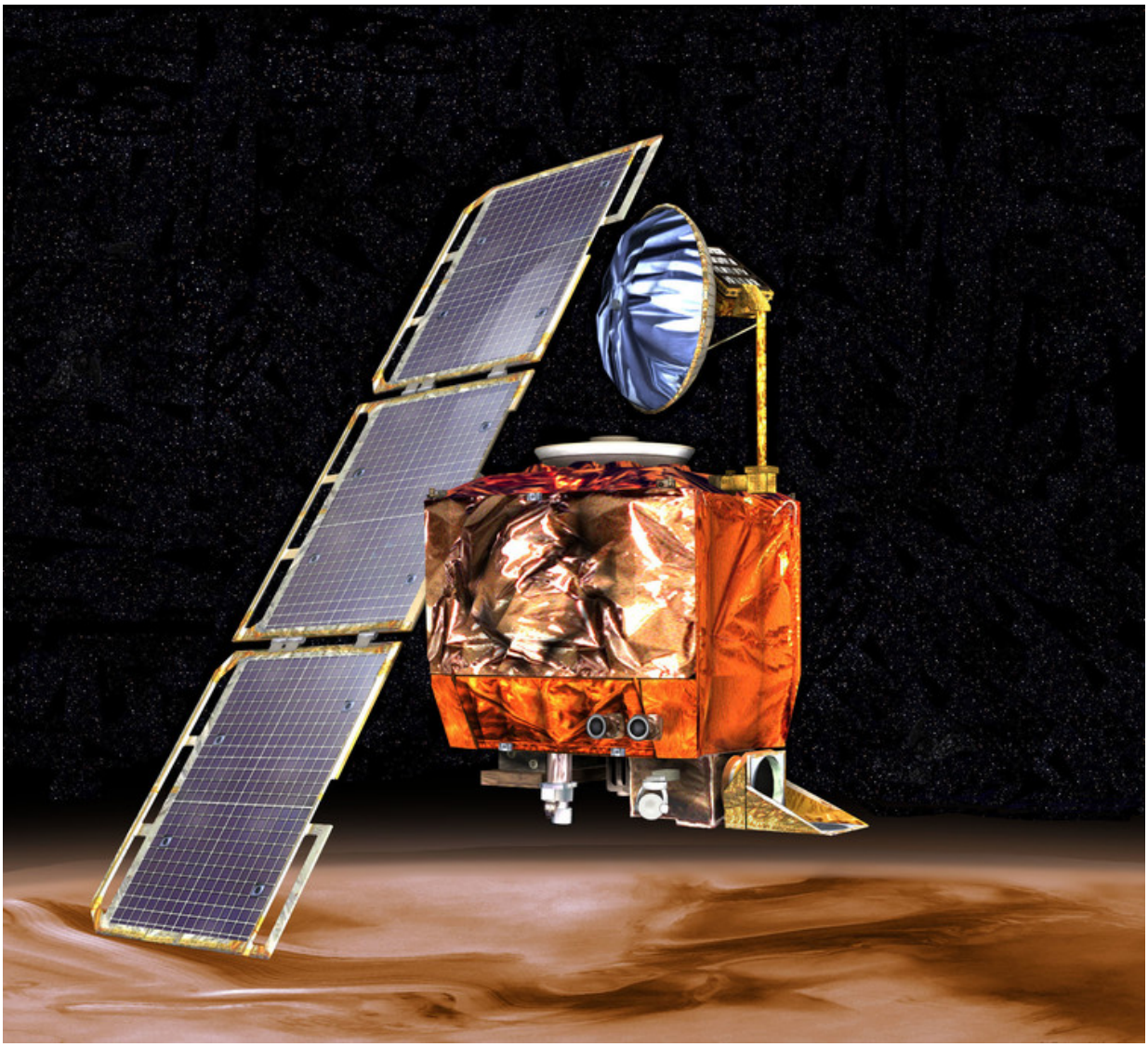
Much of current science deals with even more complicated systems, and

similarly lacks exact solutions. Such models have to be "computational" – describing how a system changes from one instant to the next. But there is no way to determine the exact state at some time in the future other than by "simulating" its evolution in this way. Weather forecasting is a familiar example; until the advent of computers in the 1950s, it was impossible to predict future weather faster than it actually happened.

Current science typically consists of devising a mathematical model that describes a complicated system, then turning this into a computational simulation, and running the simulation to make predictions in order to validate the model.

## When modelling fails

Modelling is used across scientific fields – ranging from astrophysics and climate prediction to bioinformatics and economics. But there is increasing debate about the fact that this science is difficult to validate through reproduction.



The Mars Climate Orbiter got lost in space and ended up disintegrating in Mars' atmosphere. Credit: NASA

It turns out that simply describing experimental methods in words is not enough. That's partly because natural languages such as English are simply too vague for describing computations precisely. There is, after all, a reason why programmers use programming languages. One of the biggest challenges in software development is in converting vague

requirements into precise specifications of behaviour.

Humans – even scientists – are after all fallible. Transforming any information into a program almost invariably introduces bugs along the way. For example, many scientists depend on data exploration tools such as spreadsheets, which are designed for ease of use and not for robustness. It is very easy simply to sum up the wrong range of cells in a spreadsheet, without getting any warnings. This was one of the methodological flaws in a paper that the US Republican Party used to base their pro-austerity policies on.

Similarly a recent study on the 15,770 spreadsheets that were made public during the investigation into the US corporation Enron showed that 24% of the spreadsheets containing at least one formula had obvious bugs, such as adding up blank cells.

In the natural sciences, the Mars Climate Observer, a space probe launched in 1998 to study the climate on Mars, was lost a year later because one part of the control software mistakenly used imperial instead of metric units. Another study of nine independent implementations of the same geoscience experiment – using the same dataset, algorithms, and programming language – showed very little agreement in the results obtained.

What's more, even if the reader of a research paper can successfully interpret the writer's precise meaning, and then faultlessly translate it into a program, there are still pitfalls in executing it. One particularly tricky class of problems arises from how computers handle numbers: although they can manipulate integers such as 42 and -17 with perfect accuracy, standard techniques for manipulating real numbers such $\pi \approx 3.14$ and $\sqrt{2} \approx 1.414$ permit only approximate accuracy. These approximations mean that apparently equivalent ways of computing the same value can yield different results.

So, what can be done? If even expert software developers cannot reliably produce correct software, what hope is there for amateur programmers like scientists?

One line of work is to produce tools for designing "domain-specific" programming languages, each tailored to a particular class of problem, such as the behaviour of agents in economic markets or the diffusion of drugs across cells. These aim to make it much easier for specialists to describe computations directly in familiar terms, rather than having to encode them indirectly in a general-purpose programming language.

A second approach seeks to design more expressive but still user friendly "type systems" for programs. These would make it easier to catch "silly" errors, such as blank cells in spreadsheets, or mixing up values in different units. It cannot rule out all logic errors though. A third line is to develop usable libraries of code for exact arithmetic, avoiding the problems of approximation.

There is every chance these approaches can help fix the problem going forward, or at least eliminate some of the risk. After all, the world needs science and scientists need computers – that's not likely to change anytime soon.



Source: The Conversation