Johnathan von der Heyde – 2019

What is Computational Physics?

Computational Physics (CP), is nothing more complex than the use of computers in modeling physical situations and solving physical problems.

Strangely enough, there isn't ubiquitous consensus on this point. The confusion, as far as I can tell, comes from an outdated insistence that such a practice must conform to the mutually exclusive dichotomy of Theoretical or Experimental Physics. So whereas CP isn't usually done in the lab, it nonetheless often simulates laboratory conditions. Moreover, it can serve to test theories almost as well as it can generate them — hence the confusion.

Though to be fair, as relatively new discipline, the fact that it doesn't easily fit into older categories shouldn't really be a surprise — this is the way things typically go.

Personally speaking, I think CP falls between Theoretical and Experimental Physics, yet it does have theoretical leanings. To get into the matter, we would have to discuss some philosophy of science, which is beyond the scope of this paper. Suffice to say, if science is to be considered the epistemological process by which we refine our model of the world (so as to better predict the future, increase our efficiency, improve our livelihood, etc.), then we must admit to the nuance of that process. That is to say, such a process clearly defines a transformation of the possibly useful (the theoretic) to the empirically useful (the experimental) — a transformation whose intermediate steps are not easily categorizable.

Taxonomy aside, CP is nevertheless best understood by way of example — rather than definition. And since a vast majority of today's physics models numerically solve differential equations, the first example is that of the simple pendulum. This pendulum swings back and forth in 2D, for small angles, just like the pendulum in a Grandfather Clock.

The differential equation for such a system in generalized coordinates is

$$\frac{d^2}{dt^2}\theta(t) = -\frac{g}{\ell}\theta(t)$$

This is a *closed-form* analytical solution (specifically autonomous), although it is approximate. And to make it numerically accessible, we first separate the singular 2nd order differential into two of 1st order. The first of which is basically a definition.

$$\frac{d}{dt}\theta(t) = \omega\big(\theta(t)\big)$$

$$\frac{d}{dt}\omega\big(\theta(t)\big) = -\frac{g}{\ell}\theta(t)$$

We can rearrange this in a more suggestive way, dropping dependencies for brevity

$$d\theta = \omega * dt$$

$$d\omega = -\frac{g}{\ell}\theta * dt$$

We now implement one of the most important concepts in numerical analysis,

$$df(x_i) \approx f(x_i) - f(x_{i-1})$$

Another being the compliment — these are the discrete versions of the otherwise continuous Fundamental Theorem of Calculus (minus relative change in x for slope)

$$\Delta f(x) \approx \sum_{i}^{n} df(x_i)$$

This specific formalism may be new — our functions here are constants, but I won't elaborate on the symbology, just the message. Which is to say, the infinitesimal segment of a function is approximately equal to the difference in its current value relative to its adjacent value, so long as that difference is *sufficiently small*. Likewise, the total change in a function is approximately equal to the sum of all its infinitesimal pieces, so long as n is also *sufficiently large.*

Those familiar with numerical methods will find this obvious, yet it's no trivial thing. Indeed, the accuracy with which numerical solutions approximate analytical ones is entirely contingent on how discretized the parameters are allowed to be. Metaphorically speaking, the more detail or nuance you can encode into your map, the better it may represent the territory.

Moving on, we find

$$\theta_i - \theta_{i-1} = \omega_{i-1} * (t_i - t_{i-1}) = \omega_{i-1} * \Delta t$$
$$\omega_i - \omega_{i-1} = -\frac{g}{\ell}\theta_{i-1} * (t_i - t_{i-1}) = -\frac{g}{\ell}\theta_{i-1} * \Delta t$$

Programmatically, this might look something like this

$$omega {+}{=} -g/\ell \ * theta * timeStep$$
$$theta {+}{=} \ omega * timeStep$$

Which can now be iterated through with sufficiently large n, and sufficiently small $timeStep$ in order to approximate the analytical solution, i.e., the dynamics of the pendulum. Initial conditions are still required, so as to get the recursive algorithm going. Moreover, if we wanted to, we could use this same numerical differentiation technique to analyze the pendulum's acceleration, jerk, etc.

Why do we need Computational Physics?

We need computational physics because it's useful. And it's useful because we cannot think like computers can. Therefore, by using computers to solve problems, we expand our capacities as problem solvers.

Whereas we are generally good at thinking abstractly and inductively (which is to say about global connections and relationships; about thinking itself and the thinking of others; about the probabilistic, the non-monotonic, and the indirect), traditional computers pick up where we lack. Their specialty is thinking specifically, deductively, directly, linearly, and monotonically. For the sake of brevity, I will call such traditional computational thinking techniques, methods, and implementations, "*calculator-esque*".

This specialization primes computers for mathematics, and for physical simulations. That's because of the algebraic, discrete, linear, Boolean, recursive, and monotonic — i.e., calculator-esque, nature of such systems.

That granted, there is much to say about how computational physics helps us solve inexact, non-calculator-esque problems. Indeed that is the main topic of the last section. But before we get there, let's consider many possible examples for why we might need the assistance of CP.

Firstly, as stated, traditional calculator-esque programs are often able to solve problems we humans are not specialized at solving. This includes huge data sets, problems that need to be solved very fast, multivariability, and general complexity. Imagine a bank, which doesn't necessarily do any sort of advanced mathematics, yet still demands its calculations to be done quickly, exactly, and for a large amount of data almost continuously. This is just the job for calculator-esque programs. And in physics, the same idea applies — e.g., for satellite tracking or astrophysical data.

Yet, *for the most part*, computational methods are implemented within physics in order to solve non-calculator-esque problems. I will list a few examples off the cuff:

- Stochastic systems — diffusion, growth, molecular dynamics, state prediction
- Chaotic systems — double pendulums, weather prediction, many-body problems
- Non-conservative (open) systems — entropic systems, far-from-equilibrium
- Nonlinear systems — fluid dynamics, Ising models, any sort of self-interaction
- Probabilistic systems — thermodynamics, quantum mechanics, active inference
- Complex systems — Density Functional Theory, neural networks, model fitting

In the pendulum example, we were lucky in many regards. For one, it represents an incredibly simple system with a closed, albeit nonlinear, exact analytic solution. Yet once we start to add other forces, it quickly becomes intractable.

Therefore we rely on numerical methods, which can handle the ongoing effects of minute changes, over an arbitrary state space.

It should come as no surprise then that most systems do not behave this well. Indeed, analytical solutions are only possible when the state space is very well understood — mathematical analysis (as opposed to numerical analysis) is much like deduction in this regard. So when it comes to more involved systems, it's no wonder we turn to some more human, and less calculator-esque, inductive-like techniques — such as thinking probabilistically, reasoning by default, making stochastic guesses, and refining our solution as we go.

How is Computational Physics used?

Principally, computational physics is used to facilitate numerical analysis, although it can as well benefit problems involving mathematical analysis. One may wonder, why can't computation be applied directly to all mathematical / physical problems? The answer, to be concise, is that it's no simple task — we're working on it. Indeed, the effort to answer that question has created an entire discipline known as Computability Theory.

Some problems are exactly solvable, i.e., can be solved in a finite number of operations, and some aren't. Moreover, it's not often clear whether or not a given problem is or isn't exactly solvable. If one could solve this problem — the problem of whether a given problem P is exactly solvable, then one would be solving the infamous Halting Problem, which would, to say the least, rupture the foundations of Mathematics.

If one were to speak of functional analysis, for sake of example, whereby one's aim might be to figure out what function maps values $\mathcal{A}$ to values $\mathcal{B}$, the main problem is usually twofold: 1. Determining if an exact solution (closed form function $\mathcal{A} \to \mathcal{B}$) exists, 2. If it can exist: determine it, if it can't exist: approximate it.

Computers are not yet resourceful enough to help mathematicians and physicists with 1. That's because 1 generally deals with high-level mathematical analysis, which often involves symbolic manipulation of abstract structures like groups, rings, modules, etc., which are so far largely inaccessible to programmatic language. However, with 2, computational methods may indeed make headway (as we've seen). Of salience is the fact that whether or not an *exact* solution is known to exist, such a solution can be *approximated* with numerical analysis. This process of recursive approximation is what computers are great at, and such approximations may help mathematicians and computer scientists intuit an exact solution — or perhaps the approximations are sufficient, as is often the case for physicists.

Therefore, computational physics is primarily used in the numerical analysis of problems in physics, and is done so by algorithms in almost any program language.

Johnathan von der Heyde – 2019

To Emphasize and Expound

Computational Physics is used by a variety of physicists. In fact, I can't personally think of a single sub-discipline within physics that hasn't yet adopted computational techniques. The *way* CP is used, is of course via programmatic language. Specifically, algorithms (decision procedures) are used iteratively (or recursively) to hone in on a unique state, or to continuously generate and update a simulation. In either case, computers do nothing but what they're told to do, which is just to say they're nothing more than a compactification of old logic — no magic. The importance of this can be summarized in the discipline's old saying "crap goes in, crap comes out".

We need Computational Physics because it extends our capacities well beyond our basic biological problem-solving facilities. Moreover, it allows us to focus on high-level aspects, while the tediums of algebra and calculus are off-loaded to computers. Additionally, as technology progresses, and as societies become ever-more nuanced, computational physics and its associated numerical techniques will be our only method for controlling Big Data and advancing AI. That is to say, the scientific endeavor depends crucially on (sufficiently) accurate physical models.

Computational Physics is the use of computers in solving problems in physics, principally those that are difficult if not impossible to solve analytically or by hand. This primarily includes modeling / simulation techniques — e.g., using a genetic algorithm to predict atomic crystal structure (something I've personally worked on).

So perhaps as a final note, I'll attempt an explanation of the *role* CP plays in a broader, interconnected system.

Mathematicians, as much as the "pure" ones may love the inapplicability of their research, nevertheless continue to expose greater relationships in logic, nature, reason, and organization. And computer scientists are here now doing just the same. These insights go on to inform physicists — who are now almost ubiquitously using computational methods to advance their capabilities, and therefore, with the help of experimentalists and theoreticians alike, we see useful novelty coming from physics almost daily. All this in-turn informs engineers, who go on to design and build better tools and infrastructure for all of us to improve our work.

To put it metaphorically, with science as a mind and technology as its body, we are all generally a self-bolstering system — despite our admitted faults and corruptions.

This is thanks to each part's role. Specifically thanks to each part's differentiability, adaptability, and integrability. And computational physics is, at least to my mind, a truly fascinating place to be.