History and motivation

2024-03-14

Very much like Unix in the late 1960s and C in the early 1970s, FeenoX is a third-system effect: I wrote a first hack that seemed to work better than I had expected. Then I tried to add a lot of features and complexities which I felt the code needed. After ten years of actual usage, I then realized

1. what was worth keeping,
2. what needed to be rewritten and
3. what had to be discarded.

The first version was called wasora, the second was "The wasora suite" (i.e. a generic framework plus a bunch of "plugins", including a thermo-mechanical one named Fino) and then finally FeenoX. The story that follows explains why I wrote the first hack to begin with.

It was at the movies when I first heard about dynamical systems, non-linear equations and chaos theory. The year was 1993, I was ten years old and the movie was Jurassic Park. Dr. Ian Malcolm (the character portrayed by Jeff Goldblum) explained sensitivity to initial conditions in a memorable scene, which is worth watching again and again. Since then, the fact that tiny variations may lead to unexpected results has always fascinated me. During high school I attended a very interesting course on fractals and chaos that made me think further about complexity and its mathematical description. Nevertheless, it was not not until college that I was able to really model and solve the differential equations that give rise to chaotic behavior.

Figure 1: Dr. Ian Malcolm (Jeff Goldblum) explains sensitivity to initial conditions.

History and motivation

In fact, initial-value ordinary differential equations arise in a great variety of subjects in science and engineering. Classical mechanics, chemical kinetics, structural dynamics, heat transfer analysis and dynamical systems, among other disciplines, heavily rely on equations of the form

$$\dot{\mathbf{x}} = F(\mathbf{x}, t)$$

During my years of undergraduate student (circa 2004–2007), whenever I had to solve these kind of equations I had to choose one of the following three options:

1. to program an *ad-hoc* numerical method such as Euler or Runge-Kutta, matching the requirements of the system of equations to solve, or
2. to use a standard numerical library such as the GNU Scientific Library and code the equations to solve into a C program (or maybe in Python), or
3. to use a high-level system such as Octave, Maxima, or some non-free (and worse, see below) programs.

Of course, each option had its pros and its cons. But none provided the combination of advantages I was looking for, namely flexibility (option one), efficiency (option two) and reduced input work (partially given by option three). Back in those days I ended up wandering between options one and two, depending on the type of problem I had to solve. However, even though one can, with some effort, make the code read some parameters from a text file, any other drastic change usually requires a modification in the source code—some times involving a substantial amount of work—and a further recompilation of the code. This was what I most disliked about this way of working, but I could nevertheless live with it.

Regardless of this situation, during my last year of Nuclear Engineering, the tipping point came along. Here's a slightly-fictionalized of a dialog between myself and the teacher at the computer lab (Dr E.), as it might have happened (or not):

— (Prof.) Open MATLAB.™
— (Me) It's not installed here. I type `mathlab` and it does not work.
— (Prof.) It's spelled `matlab`.
— (Me) Ok, working. (A screen with blocks and lines connecting them appears)
— (Me) What's this?
— (Prof.) The point reactor equations.
— (Me) It's not. These are the point reactor equations:

$$\begin{cases} \dot{\phi}(t) = \dfrac{\rho(t) - \beta}{\Lambda} \cdot \phi(t) + \sum_{i=1}^{N} \lambda_i \cdot c_i \\ \dot{c}_i(t) = \dfrac{\beta_i}{\Lambda} \cdot \phi(t) - \lambda_i \cdot c_i \end{cases}$$

— (Me) And in any case, I'd write them like this in a computer:

```
phi_dot = (rho-Beta)/Lambda * phi + sum(lambda[i], c[i], i, 1, N)
c_dot[i] = beta[i]/Lambda * phi - lambda[i]*c[i]
```

This conversation forced me to re-think the ODE-solving issue. I could not (and still cannot) understand why somebody would prefer to solve a very simple set of differential equations by drawing blocks and

History and motivation

connecting them with a mouse with no mathematical sense whatsoever. Fast forward fifteen years, and what I wrote above is essentially how one would solve the point kinetics equations with FeenoX.