“You’re doing simulations with your students, so why are you using *Mathematica*?”

Phil Ramsden, Department of Mathematics, Imperial College of Science, Technology and Medicine, London, UK

Abstract

At Imperial College, staff from the Mathematics and Chemistry departments have set up computer-based “Mathematics Laboratories” for first year Chemistry undergraduates. We’ve provided several types of activity in these laboratories, but one of the most important kinds involves students’ setting up and studying models and simulations of chemical conditions and processes.

Dedicated simulation software exists for this sort of thing, much of it offering a lot of presentational sophistication and dynamic interactivity. But actually, all our simulations are implemented entirely in *Mathematica*; moreover, our presentation style is simple and stark in the extreme.

Mathematica works for us as simulation and modelling software because it allows us to be open in our approach to design. All the code that is used to set up and run the simulations is visible to the students, and indeed we require them to engage with it directly. The relationships between the underlying mathematics, the chemistry that is being modelled, and the computer code that implements the model can be explicit rather than hidden. Students can be encouraged to adopt a critical attitude towards mathematical models and to develop their own ideas and approaches.

On the other hand, *Mathematica*’s power and range enable us to set up fairly realistic and uncontrived problems, and to place some mathematical elements of these problems firmly in the background if we choose.

I illustrate these strengths of *Mathematica* for our purposes using the example of a series of assessed problems. These concern the forces between ions (modelled as Newtonian point charges) and the chemical dynamics to which these forces give rise. I show examples of closed and open-ended tasks, and present excerpts of students’ work on these tasks.

The “Maths Labs”

The use of *Mathematica* in the Chemistry department at Imperial College began in 1994. Then, as now, Chemistry and Mathematics staff cooperated in the design of the computer-based course materials and in the way they were implemented. In those days, however, we in the Mathematics department had very much the dominant role in the authoring process. No doubt partly for that reason—partly, too, because of our brief as what was then a government-funded project—there is very little *chemistry* in those five-year-old materials. “Learn the mathematics here, then apply it in your other courses” was the—perhaps unspoken—model.

But we gradually became aware of a problem with this approach. Based on observations of the materials in use, and our reading in the educational research literature, we began to have doubts about whether an approach based exclusively on “learn, then apply” was the best we could do. Students find transferring knowledge from one area to another notoriously difficult, and although “learn, then apply” is very valuable, indeed probably indispensible, in higher education, we felt it was legitimate to wonder whether it was enough on its own. We began to look for ways of building in explicit links between mathematics and chemistry right from the start of the course, and we became convinced that *Mathematica* offered us a way of doing that—a way that compromised the integrity of neither discipline.

Briefly put, our idea was this. Explicitly linking mathematics and chemistry as you go along is, in general, difficult, because first year students at the start of their courses rarely know enough mathematics for the exercise to be meaningful, and if they are restricted to trivial and contrived chemistry problems, the point is lost. However, if the students have at their disposal powerful mathematical software, then it is possible to give them access to problems of more intrinsic interest to the would-be chemist. In this paper, I report on our experience with the "Mathematics Laboratories" we have been running with this aim in mind.

The term "Mathematics Laboratory" was one we coined early on, but with hindsight one can see that it was something of a misnomer five years ago. Then, it was simply meant to suggest an approach to teaching that encouraged students to pose, and to test, conjectures. The usage has actually grown more apt with time, though, as we have introduced a measure of explicit scientific content into the first year mathematics course in Chemistry. For an important part of their time on the course, students work on problems that genuinely belong to the field of physical chemistry, but they bring mathematical and computational tools to bear on them.

Actually, various kinds of activity go on in the "Maths Labs": learning how to use *Mathematica*, for instance, or performing statistical analyses of experimental data, to cite two examples I shall not be discussing at all in this paper. What I want to focus on here is what I have described in broad terms above: the study of chemical problems, with the help of *Mathematica*, to motivate and aid the learning of mathematical topics.

How the course is organised

The mathematics course in the First Year in Chemistry consists of the Maths Labs, plus plenary lectures and non computer-based tutorials. In the lectures and tutorials there is, on the whole, less explicit chemistry content than there is in the Labs. Each Lab session lasts for three hours, and students attend one a week while the course is active (a total of eleven weeks, split among the three terms of the academic year). In addition, students have open access to the computers at certain other times if they need it.

The first two weeks are taken up by a six-hour introduction to *Mathematica*. Even here, we have tried to make the content explicitly chemical: animations are covered through a study of molecular vibrations, for example, and three-dimensional plotting is applied to simple quantum wavefunctions representing, say, orbitals in atomic hydrogen. But in these introductory sessions the chemistry is there purely in order to help motivate the "core" work, and although this is a function it serves throughout the course, we ask rather more of it—and of the students—from Week 3 onwards.

The rest of the course consists of seven modules, some of which are designed to represent three hours' work, and some six. In each, one or more topics in mathematics are approached via one or more problems drawn from a branch of physical chemistry.

	Mathematical topic(s) studied:	**Physical chemistry problem(s) relating to:**
Lab 3	Vectors: position and translation vectors; vector addition.	Space crystals and lattices.
Lab 4	Resultants.	Chemisorption on to a crystal lattice.
Lab 5	Scalar product of vectors; integration.	Energies of dissociation and adsorption; stability of equilibria.
Lab 6	Differential equations (numerical solution); integration.	Reaction dynamics; forces, potentials and energy (classical picture)
Lab 7	Differential equations (analytic solution).	Reaction kinetics (rates and orders).
Lab 8	Exact differentials; line integrals.	Thermodynamics: enthalpy, internal energy, entropy.
Lab 9	Differential equations (eigenvalue problems); continuity and smoothness.	Quantum chemistry: electrons in linear dye molecules.

Choosing Mathematica

Mathematica is the system of choice for all undergraduate mathematical computing in the Chemistry Department at Imperial. We have opted to teach students to use the same, unified, computational environment for all their mathematical work, rather than having them use one package for one kind of task and one for another. Sometimes, this means we need to compromise; where data analysis is concerned, for instance, there might be a good case for using a slick statistical package instead of *Mathematica*. But the theme of this paper is my belief that the Maths Labs themselves have required no such compromise: that even in those areas where one might think there was a case for a dedicated system, *Mathematica* was actually the ideal choice.

I want to make this case by focusing on Labs 3 and 6. I've chosen these because they perhaps represent the sternest test of my case. *Mathematica*, it might be argued, is an odd choice for Lab 3, because the main activity in Lab 3 is creating crystal lattices on screen, and there are several packages you can buy that are designed for that purpose and that are a lot slicker to use. Likewise, it might be suggested that Lab 6, which centres on an animated reaction simulation, might benefit from a more specialised simulation tool, either written by us or bought in. I want to argue that actually *Mathematica* was exactly the right tool in both cases.

Moreover, I want to argue for a very simple, stripped-down mode of use: *Mathematica* "out of the can", with no additional packages, and all code visible to (and editable by) the user. This approach is different from the way we did things a few years back (see [1], for instance, or our student text [2]), and is actually closer to what the Calculus&*Mathematica* team were arguing for, and we were arguing against, at the time. I would still hesitate to raise "out of the can" to a universal maxim, but our experience in designing these materials has certainly brought home to us the arguments in favour of it.

Lab 3: Vectors and crystals

Maths Lab 3, the first after the two-session introduction to *Mathematica*, deals with the topic of vectors: specifically, with the use of vector addition to characterise space lattices and crystals. Students get

- a printed *Mathematica* notebook, Mathslab3.nb, containing text and code;
- access to an online copy of this notebook;
- access to an online version of the notebook containing only the code.

You can download a copy of Mathslab3.nb by clicking on the following link, which is repeated in the bibliography:

http://metric.ma.ic.ac.uk/chem/Mathslab3.nb

Figure 1 shows where Mathslab3.nb leads: the graphic that students are asked to produce as the final exercise in the assignment. It's a unit cell of the gallium arsenide crystal; the smaller spheres on the vertices and faces (shown red if you're viewing in colour) represent gallium atoms, and the larger (green) spheres within the cell, arsenic atoms. I've shown the unit cell in two forms: the one on the left is the working version, in which simple `Point` primitives have been used to create the spheres, and that on the right is the presentation version, in which these have been replaced by translated `Sphere` objects, built using the `Graphics‘Shapes‘` package.

Specialist molecular modelling software exists for generating graphics like these, including at least one *Mathematica* package that I'm aware of (there may be more). Using these, users can quickly create on-screen models of molecules and crystals: models whose graphical quality is often superior to Figure 1 (certainly to the working version, which is all our students are formally required to produce) and which, in many cases, they can dynamically manipulate on the screen. We made a deliberate choice to use none of this software, and indeed, as I say, we didn't even write a package. We require our students to build up this graphic pretty much from scratch, and a laborious business it is.

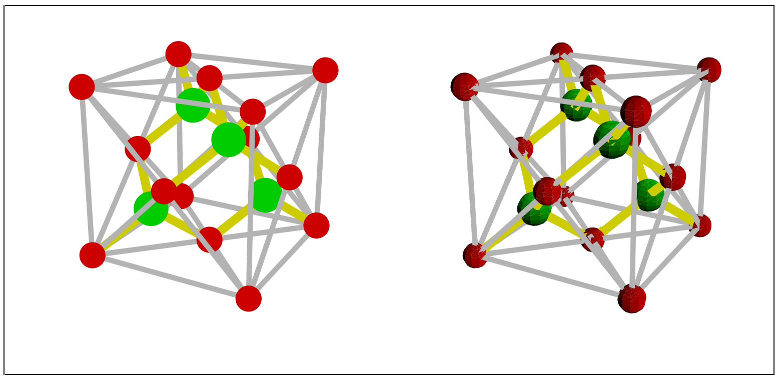
Figure 1.
Conventional unit cell of the gallium arsenide crystal, working and presentation versions.

First, they have to generate the grey bounding lines whose intersections define the lattice points. There are slick ways of doing that in *Mathematica*, but we haven't used any of them: the code for these lines is

```
unitCell[fcc]={AbsoluteThickness[2], GrayLevel[0.7],
  Line[{{0, 0, 0}, {0, 0, 1}}], Line[{{0, 0, 0}, {0, 1, 0}}],
  Line[{{0, 0, 0}, {0, 1, 1}}], Line[{{0, 0, 0}, {1, 0, 0}}],
  Line[{{0, 0, 0}, {1, 0, 1}}], Line[{{0, 0, 0}, {1, 1, 0}}],
  Line[{{0, 0, 1}, {0, 1, 0}}], Line[{{0, 0, 1}, {0, 1, 1}}],
  Line[{{0, 0, 1}, {1, 0, 0}}], Line[{{0, 0, 1}, {1, 0, 1}}],
  Line[{{0, 0, 1}, {1, 1, 1}}], Line[{{0, 1, 0}, {0, 1, 1}}],
  Line[{{0, 1, 0}, {1, 0, 0}}], Line[{{0, 1, 0}, {1, 1, 0}}],
  Line[{{0, 1, 0}, {1, 1, 1}}], Line[{{0, 1, 1}, {1, 0, 1}}],
  Line[{{0, 1, 1}, {1, 1, 0}}], Line[{{0, 1, 1}, {1, 1, 1}}],
  Line[{{1, 0, 0}, {1, 0, 1}}], Line[{{1, 0, 0}, {1, 1, 0}}],
  Line[{{1, 0, 0}, {1, 1, 1}}], Line[{{1, 0, 1}, {1, 1, 0}}],
  Line[{{1, 0, 1}, {1, 1, 1}}], Line[{{1, 1, 0}, {1, 1, 1}}]
  };
```

They then have to build the lattice points (where the grey lines intersect, and where they'll eventually be putting the gallium atoms) as explicit linear combinations of three standard *primitive vectors*, namely `{1/2, 1/2, 0}`, `{1/2, 0, 1/2}` and `{0, 1/2, 1/2}`. The next task is to create a *molecular basis*, consisting of a gallium atom and an arsenic atom, and place a translated copy of this basis at each of the lattice points. Some of the arsenic atoms then fall outside the unit cell, and these have to be removed. Finally, the covalent bonds between the gallium and arsenic atoms (shown as yellow lines in the figure) have to be created and added to the picture. For some parts of this task, we make very strong suggestions about what to do and how to do it; other parts are somewhat less structured, and students are require to develop their own strategies.

Why do it that way, one might ask? Why not hide some of this machinery in a package, or behind a function that automates some of it? Why use mathematical software at all, when specialist applications exist, aimed at chemists and chemistry students?

However, this is not a course in crystallography, but in mathematics for chemists. The "machinery" is precisely what's important here. Moreover, the appeal of *Mathematica* is that it offers us, the designers, so much freedom to decide what we automate and what we make an explicit part of the task. For example, we felt that building the lattice points out of linear combinations of the primitive vectors was central to the aims of this piece of work, so we provided no code for this bit except a simple visualisation tool based on the `Graphics‘PlotField3D‘` package. On the other hand, clearing away those arsenic atoms that happen to lie outside the unit cell is really just tidying up, and we therefore provide students with a suggested way of doing this that's slightly slicker and more automatic:

```
inside[{atom_,{x_,y_,z_}}] := TrueQ[(0≤x≤1)&&(0≤y≤1)&&(0≤z≤1)]
unitCellAtoms = Select[unitCellAtoms, inside]
```

This code snippet, with its use of a logical function to `Select` from a list, is actually a little difficult for many students this early in the First Year (the exceptions tending to be those with a liking or aptitude for programming). Nonetheless, we made the decision not to hide it, and the same is true of all the code in the Maths Labs, some of which, especially in the later sessions, is a good deal more involved than the above. This design constraint, voluntarily submitted to, places a fairly strict limit on how sophisticated our models and simulations can be, something that becomes especially clear in the next section.

Lab 6: Simulating an ionic reaction

Maths Lab 6 is available at

http://metric.ma.ic.ac.uk/chem/Mathslab6.nb

This link is repeated in the bibliography.

Here the main focus is on the idea of differential equations and their solutions. The Lab is built around a reaction simulation that is effectively a Newtonian three-body problem, and therefore not exactly integrable in general; the solution process, which is numerical, is hidden inside *Mathematica*'s `NDSolve` function, and we don't ask these First Year Chemistry students to explore that.

The assignment involves simulating a reaction between a free chloride ion and a sodium chloride dipole. For simplicity, the ions are treated as Newtonian point charges, and their movement is restricted to one dimension.

Again, setting things up is rather involved. First, the students need to set some physical constants: the permittivity of the vacuum, Avogadro's number, and so on. Then, they define a couple of force functions, representing empirically derived classical approximations to what is actually, of course, a quantum phenomenon. These cover interactions between particles of like charge…

$$F_{mm}[r\_] = -\frac{e^2}{4\pi\varepsilon_0 r^2}$$

and unlike charge…

$$F_{pm}[r\_] = \frac{e^2}{4\pi\varepsilon_0}\left(\frac{1}{r^2} - \frac{(r_0)^8}{r^{10}}\right)$$

respectively. After setting a duration for the "reaction", and a set of initial positions and velocities for the three ions, the student is finally ready to solve the equations of motion of the ions:

$$\begin{aligned}
&\texttt{xrules = NDSolve[\{}\\
&\quad \texttt{x1''[t] ==}\ \frac{F_{pm}[x2[t]-x1[t]] + F_{mm}[x3[t]-x1[t]]}{m_{Cl}},\\
&\quad \texttt{x2''[t] ==}\ \frac{-F_{pm}[x2[t]-x1[t]] + F_{mm}[x3[t]-x2[t]]}{m_{Na}},\\
&\quad \texttt{x3''[t] ==}\ \frac{-F_{mm}[x3[t]-x1[t]] - F_{pm}[x3[t]-x2[t]]}{m_{Cl}},
\end{aligned}$$

```
    x1[0] == x1₀, x2[0] == x2₀, x3[0] == x3₀,
    x1′[0] == v1₀, x2′[0] == v2₀, x3′[0] == v3₀},
    {x1,x2,x3}, {t,0,T}, MaxSteps -> 2000];
```

The solutions that come from this call to NDSolve are then used to define position and velocity functions, which go to set up a fairly crude animated "simulation":

```
threeIonFrame[t_] := Show[Graphics[{AbsolutePointSize[15],
  {RGBColor[0.8, 0.8, 0], Point[{pos1[t], 0}]},
  {RGBColor[0.4, 0.4, 0.4], Point[{pos2[t], 0}]},
  {RGBColor[0.8, 0.8, 0], Point[{pos3[t], 0}]},
  PlotRange->{{-15r₀, 15r₀}, Automatic}]

Table[threeIonFrame[t], {t, 0, T, T/100}]
```

Some frames from this animation are shown in Figure 2. Here, the starkness of the representation, and its inferiority to what could be achieved either with more sophisticated *Mathematica* code or with dedicated simulation software, are perhaps even more apparent than in Lab 3. This graphical crudeness is forced upon us because we want to keep our code simple enough for students to read—and, just as importantly, *edit*, and use as a model for their own code. One of the things that *Mathematica*, used in this way, offers us is the opportunity to make clear the connection between the Newtonian equations of motion of the three ions, clearly apparent in the above code, and the behaviour of the system. If we hadn't used a computer algebra system at all, or if (as is fashionable in some quarters) we'd used one but hidden the algebra part beneath a more "friendly" user interface, this advantage would have been lost. It would also, we believe, have been somewhat compromised even if we'd put some of the code in a package. We might also note in passing that a system whose typesetting capabilities aren't as good as those of *Mathematica*—and those do, of course, exist—would have made rather less obvious the key fact that the equations of motion are simply rearrangements of Newton's second law.

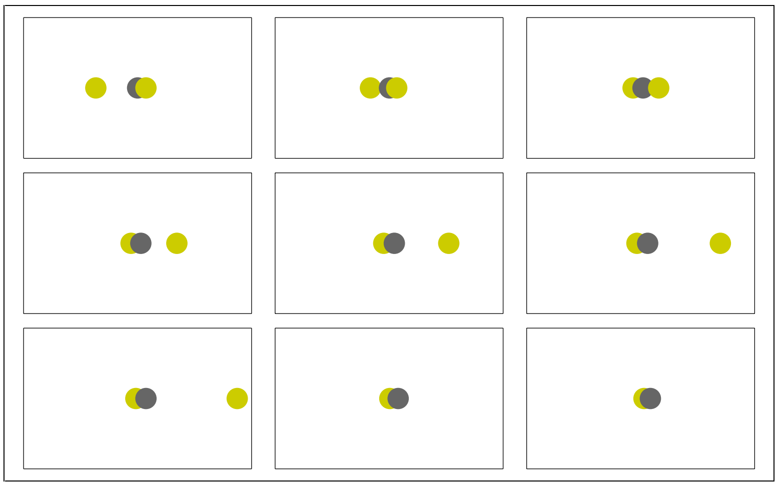

Figure 2.
Frames from the "worked example" animation in Maths Lab 6

After typing in "our" code, the students have four things to do. First, they're asked to write some code of their own for viewing the system's behaviour by means of a *graph* instead of an animation. Next, they have to vary the initial positions and velocities of the particles systematically in order to make the system behave—apparently, anyway—in at least three ways:

- the chloride ion that was dissociated becomes bound, and vice versa (a reaction occurs, in other words);

- the dissociated chloride ion interacts with the dipole but is repelled, and becomes dissociated once more;
- the interaction between the three ions continues for ever, with neither chloride ion becoming dissociated from the other two.

The third task is to test the model itself: specifically, to check that energy appears to be conserved. This involves converting the forces into potentials, and sampling (or graphing) the total mechanical energy of the system during the reaction. So: although the details of the numerical solution method are kept hidden behind `NDSolve`, students are asked to use *Mathematica* to make sure that the dynamical behaviour that comes out of `NDSolve` is physically plausible. In other words, the very system that generated the data can be used in a natural way to examine that data critically. With *Mathematica*, a black box needn't be completely black.

Finally, students are asked to devise, from scratch, a *Mathematica*-based test for ionic dissociation and use this to find out whether the three reaction outcomes listed above have actually been realised in their work, or merely apparently so. The test does not have to be perfect and foolproof, but it must take one further than merely observing the reaction. Figure 3 shows three graphs from the work of a certain student, who approached this problem by partitioning the potential energy of the system among the three particles (in a *fairly* plausible way), then graphing the total mechanical energy of each ion against time. In the figure, we see this graph for one of the chloride ions—the one on the left of the frame in Figure 2—in the context of each of the three reaction outcomes listed above. A particle is dissociated, in this student's analysis, if it has a total mechanical energy greater than zero; as you can see, the test seems to work quite well.

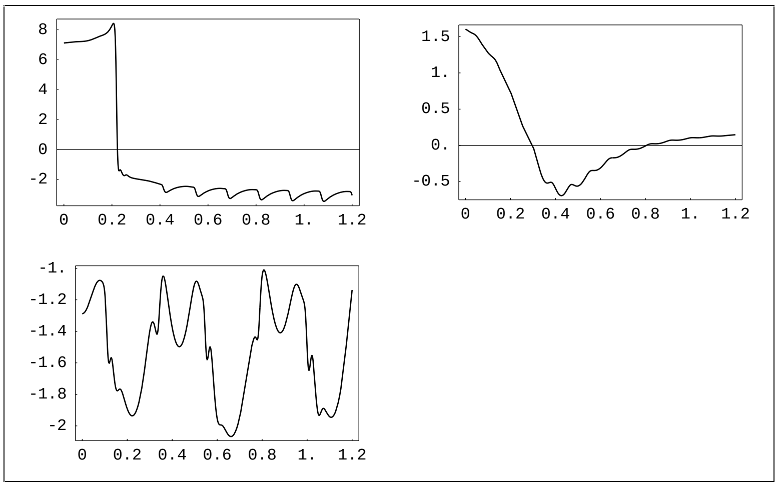

Figure 3 (student's work).
Graphs of "total energy" ($\times 10^{-19}$ J) against time ($\times 10^{-12}$ s) for the left-hand chloride ion in the context of the three reaction outcomes described above.

We were, in general, very pleased by the standard of work on this rather challenging task, which increased our confidence that we had been right to use *Mathematica* in the first place and, in particular, right to use it like this.

Concluding remarks

The model of "applied mathematics" teaching that says "learn the mathematics, then apply it" is a powerful one, because it enables mathematical ideas to be taught independently of a particular application and therefore to feed into a number of such applications. However, it runs foul of the well-known difficulty that students have in linking different parts of their knowledge.

Powerful computer software is helpful if you want to link mathematics to a scientific application without having to trivialise the science. If you have access to such software, you're a good deal less constrained by the narrow range of pen-and-paper techniques in which students may be confident and proficient at any particular time. For instance, there are *some* chemically interesting systems that can be modelled using the types of differential equations that students can solve exactly, but there aren't many, and the one in Lab 6, in particular, would be off limits without computer power.

If we deploy that computer power in an *open* way (which *Mathematica* lends itself to), students can come to wield it themselves, and even use it in creative, original and unforeseen ways. It's natural to want to protect novice students from *Mathematica*'s syntactical complexities, but there is such a thing as being overprotective. Where *Mathematica*, used "out of the can", comes into its own is in those situations where we want our computational representations to reflect (a) the key concepts of the problem, (b) the underlying mathematics and (c) the connections between the two.

References and links

1. Kent, P., Ramsden, P. and Wood, J., '*Mathematica* for valuable and viable computer-based learning', in Keränen, V. and Mitic, P., ed., *Mathematics with Vision: Proceedings of the First International Mathematica Symposium*, Computational Mechanics Publications, Southampton UK 1995. Available online at http://metric.ma.ic.ac.uk/articles/Southampton.html
2. Kent, P., Ramsden, P. and Wood, J., *Experiments in Undergraduate Mathematics: a Mathematica-Based Approach*, Imperial College Press, London 1996. Details and packages online at http://metric.ma.ic.ac.uk/book/

Imperial College's Chemistry Mathematics Laboratories:
Lab 3: http://metric.ma.ic.ac.uk/chem/Mathslab3.nb
Lab 6: http://metric.ma.ic.ac.uk/chem/Mathslab6.nb

Educational papers and articles by the author and his colleagues:
http://metric.ma.ic.ac.uk/articles/