Dimensionful Matrices

Dan Piponi

February 2024 (originally April 2016)

1 Introduction

Programming languages and libraries for numerical work tend not to place a lot of emphasis on the types of their data. For example Matlab, R, Octave, Fortran, and Numpy (but not the now defunct Fortress all tend to treat their data as plain numbers meaning that any time you have a temperature and a mass, say, there is nothing to prevent you adding them.

I've been wondering how much dimensions (in the sense of dimensional analysis and units could help with numerical programming. As I pointed out on G+ recently (which is where I post shorter stuff these days), you don't have to limit dimensions to the standard ones of length, mass, time, dollars and so on. Any scale invariance in the equations you're working with can be exploited as a dimension giving you a property that can be statically checked by a compiler.

There are quite a few libraries to statically check dimensions and units now. For example Boost.Units for C++, units for Haskell and even quantities for Idris.

2 A matrix that breaks things

Even if a language supports dimensions, it's typical to define objects like vectors and matrices as homogeneous containers of quantities. But have a look at the Wikipedia page on the metric tensor. There is a matrix

$$\eta = \begin{pmatrix} -c^2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which has the curious property that 3 entries on the diagonal seem to be dimensionless while the first entry is a squared velocity with dimension L^2T^{-2}. This will break many libraries that support units. An obvious workaround is to switch to use natural units, which is much the same as abandoning the usefulness of dimensions. But there's another way, even if it may be tricky to set up with existing languages.

3 Heterogeneous vectors and matrices

According to a common convention in physics, a 4-vector $x = (t = x^0, x^1, x^2, x^3)$ has dimensions $[x] = (T, L, L, L)$ where I'm using the convention that we can represent the units of a vector or matrix simply as a vector or matrix of dimensions, and here T is time and L is length. The metric tensor is used like this: $ds^2 = x^i \eta_{ij} x^j$ (where I'm using the Einstein summation convention so the i's and j's are summed over). If we think of ds^2 having units of length squared (it is a pseudo-Riemannian *metric* after all) then it makes sense to think of η having dimensions given by

$$[\eta] = \begin{pmatrix} L^2T^{-2} & LT^{-1} & LT^{-1} & LT^{-1} \\ LT^{-1} & 1 & 1 & 1 \\ LT^{-1} & 1 & 1 & 1 \\ LT^{-1} & 1 & 1 & 1 \end{pmatrix}$$

We can write this more succinctly as

$$[\eta] = (LT^{-1}, 1, 1, 1) \otimes (LT^{-1}, 1, 1, 1)$$

where $\otimes$ is the usual outer product.

I'll use the notation $a : A$ to mean a is of type A. So, for example,

$$(t, x^1, x^2, x^3) : (T, L, L, L).$$

I'll also use pointwise notation for types such as

$$(A, B, C, D) * (E, F, G, H) = (AE, BF, CG, DH)$$

and

$$(A, B, C)^{-1} = (A^{-1}, B^{-1}, C^{-1}).$$

Now I can give some general rules. If $m : M$ is a matrix, $x : X$ and $y : Y$ are vectors, and $s : S$ is a scalar, then $y = mx$ only makes sense if $M = Y \otimes X^{-1}$. Similarly the "inner product" $x^T my = s$ only makes sense if $M = X^{-1} \otimes Y^{-1}S$.

4 Generic vectors and matrices

Although these kinds of types might be useful if you're dealing with the kind of heterogeneous matrices that appear in relativity, there's another reason they might be useful. If you write code (in the imaginary language that supports these structures and understands dimensions and units) to be as generic as possible in the types of the vector and matrix entries, failures to type check will point out parts of the code where there are hidden assumptions, or even errors, about scaling.

For example, consider a routine to find the inverse of a 3 by 3 matrix. Writing this generically as possible means we should write it to operate on a matrix of type $(A, B, C) \otimes (D, E, F)$, say. The result should have type $(D, E, F)^{-1} \otimes (A, B, C)^{-1}$. If this type checks when used with a suitably powerful type checker then it means that if we replace the units for type A, say, with units twice as large, it should have no effect on the result, taking into account those units. In this case, it means that if we multiply the numbers of the first row of the input by 0.5 then the numbers of the first column of the output should get multiplied by 2. In fact this is a basic property of matrix inverses. In other words, this mathematical property of matrix inverses is guaranteed by a type system that can handle units and heterogeneous matrices. It would be impossible to write a matrix inverter that type checks and fails to have this property. Unfortunately it's still possible to write a matrix inverter that type checks and is incorrect some other way. Nonetheless this kind of type system would put a very big constraint on the code and is likely to eliminate many sources of error.

5 An example, briefly sketched

I thought I'd look at an actual example of a matrix inverter to see what would happen if I used a type checker like the one I've described. I looked at the conjugate gradient method. At the Wikipedia page, note the line

$$\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_{k+1}}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$$

This would immediately fail to type check because if r is of generic vector type $R = (R_1, \ldots, R_n)$ then R_1^2 isn't the same type as R_2^2 so they can't be added. I won't go into any of the details but the easiest way to patch up this code to make it type check is to introduce a new matrix P of type $R^{-1} \otimes R^{-1}$ and besides using it to make this inner product work (replacing the numerator by $\mathbf{r}_k^T P \mathbf{r}_{k+1}$) we also use P anywhere in the code we need to convert a vector of type R to a vector of type R^{-1}. If you try to do this as sparingly as possible you'll end up with a modified algorithm. But at first this seems weird. Why should this matrix inverse routine rely on someone passing in a second matrix to make it type check? And what is this new algorithm anyway? Well scroll down the Wikipedia page and you get to the *preconditioned* conjugate gradient algorithm. The extra matrix we need to pass in is the preconditioner. This second algorithm would type check. Preconditioned conjugate gradient, with a suitable preconditioner, generally performs better than pure conjugate gradient. So in this case we're getting slightly more than a check on our code's correctness. The type checker for our imaginary language would give a hint on how to make the code perform better. There's a reason for this. The original conjugate gradient algorithm is implicitly making a choice of units that sets scales along the axes. These determine the course taken by the algorithm. It's not at all clear that picking these scalings randomly (which is in effect what you're doing if you throw a random problem at the algorithm) is any good. It's better to pick a preconditioner adapted to the scale of the problem and the type checker is hinting (or would be if it existed) that you need to do this. Compare with the gradient descent algorithm whose scaling problems are better known.

6 But which language?

I guess both Agda and Idris could be made to implement what I've described. However, I've a hunch it might not be easy to use in practice.