Subject: Dimensional Juggling Tutorial
Posted by John-David T. Smith on Sun, 01 Apr 2001 23:01:26 GMT
View Forum Message <> Reply to Message

I've sensed lots of confusion recently on how exactly reform and rebin work together to allow dimensional manipulations. It often looks cryptic, but it's actually pretty simple. The basic idea is as follows:

To "vectorize" some operations, you often need to "inflate" a vector to something larger than its former self.

Example:

```
IDL> a=indgen(2,3)
IDL> print,a
       0       1
       2       3
       4       5
IDL> b=randomu(sd,3)
IDL> print,b
    0.662640     0.991186     0.479801
```

Suppose we'd like to multiply each column of "a" by "b". If we had a new array, "b2", the same size as "a", but with a copy of "b" in all its columns, we could perform the multiplication trivially. How do we get such a beast? Rebin is the answer. First point to memorize: rebin only will inflate numeric-type arrays to integer multiples of its present dimensions (each dimension can have a different integer). I.e. something with dimensions [2,3] could become [2,6] or [4,3] or [4,6], but not [3,2].

We can also add new trailing dimensions with rebin, as long as all dimensions before it follow this rule. E.g. [2,3] could become [2,3,5] without trouble, but not [3,2,5]. (You can think of this by imagining a vector/array has implicity as many *trailing* shallow (see below) dimensions as you want. IDL often truncates these, but also auto-creates them as necessary, as in this case!)

Back to the task at hand. Our "b" vector only has a single dimension, 3. Now consider:

```
IDL> print,rebin(b,3,2)
    0.662640     0.991186     0.479801
    0.662640     0.991186     0.479801
```

Aha, our first dimension has remained the same, but now we have two identical rows. Fine, you say, but we wanted identical *columns*. Well, as we've seen, we can't just say:

```
IDL> print,rebin(b,2,3)
% REBIN: Result dimensions must be integer factor of original dimensions
```

The problem here is we're trying to change the single dimension "3" (the 1st of "b") to dimension "2". Not gonna happen. How can we proceed? If only b had dimensions [1,3] : [2,3] certainly follows the "integer multiple" rules then. That leading unit dimension makes this what I call a "column vector", a terminology some people object to, but which is descriptive nonetheless. I like to call dimensions of size 1 "shallow". E.g., I say, "b has a shallow leading dimension".

We can add shallow dimensions with, you guessed it, reform:

```
IDL> print, reform(b,1,3)
    0.662640
    0.991186
    0.479801
```

Aha, printing like a column now. Now we put these two things together:

```
IDL> print, rebin(reform(b,1,3),2,3)
    0.662640    0.662640
    0.991186    0.991186
    0.479801    0.479801
```

Two columns side by side. Just what we needed! And of course our multiplication is trivial now:

```
IDL>  print, rebin(reform(b,1,3),2,3)*a
     0.00000    0.662640
     1.98237     2.97356
     1.91920     2.39901
```

This isn't the only technique, but I think it's the most straightforward. Another common approach is to make an array of indices (using, say lindgen) of the correct size, and then use some arithmetic (% and /) to force the indices to be correct, then use it as a subscript into the original array. It gets complicated fast for higher dimensions.

N.B. There are other ways to make "column vectors". Examples:

```
transpose(b)
rotate(b,1)
1#b
b##1
```

You'll often see these in place of reform(b,1,3), but don't be confused, they do exactly the same thing: prepend a leading shallow dimension. They are nice because they relieve you from having to know they length of b (3 here). However, they only work for creating column vectors, though: i.e. up to 2D data.

What if you have 3D data, and you'd like to inflate things over the third dimension? Well, you guessed it, we'll need another shallow dimension:

```
IDL> print, size(reform(b,1,1,3),/DIMENSIONS)
           1           1           3
```

And we can thread this "down" through the depth of a data cube, just as easily:

```
IDL> print, rebin(reform(b,1,1,3),3,3,3)
    0.662640    0.662640    0.662640
    0.662640    0.662640    0.662640
    0.662640    0.662640    0.662640

    0.991186    0.991186    0.991186
    0.991186    0.991186    0.991186
    0.991186    0.991186    0.991186

    0.479801    0.479801    0.479801
    0.479801    0.479801    0.479801
    0.479801    0.479801    0.479801
```

Imagine those three groups as "planes" in a data cube, and you can see our vector is now filling downwards.

you should also easily see how to thread across all rows on every plane:

```
IDL> print, rebin(b,3,3,3)
```

or columns on every plane

```
IDL> print, rebin(reform(b,1,3),3,3,3)
```

With these tricks you should be able to perform all kinds of complex vector operations.

And now we get to an advanced application, which isn't yet easy. Could we write a generic routine to do this for any given dimension: i.e. could we say "replicate this vector over dimension #4". Yes, but not as easily as you might hope.

The key is the ability to specify dimensions all together, as a vector, instead of as individual arguments. E.g.

IDL> print, reform(b,[1,1,3]) ; Notice the [and]

Now we can custom make the second argument to have as many leading shallow dimensions as are needed. To complete the operation, you'd need rebin to have the same functionality. Alas, it does not. Why? No reason. Typical RSI incomplete implementation. I know Craig agrees with me here.

So, RSI, if you're listening, why not allow rebin to interpret it's first argument as a vector also? Or use a keyword DIMENSION ala make_array? (And while I'm at it -- nothing like two different mechanisms for exactly the same thing, there).

Perhaps I'll also write a histogram tutorial, revealing all my tricks. Then I could pass the torch to Pavel...

JD
