Subject: Re: declaration of an array and multiplication with a matrix
Posted by JD Smith on Sun, 07 Sep 2003 23:40:49 GMT
View Forum Message <> Reply to Message

On Sat, 06 Sep 2003 04:17:38 -0700, Hassan Iqbal wrote:

> Hi all,
>
> can anyone please tell me what does this expression do:
>
> (w # (fltarr(m)+1)* der)
>
> w= an array of 'n' elements
> der= a matrix of 'n' columns and 'm' rows
>

w # (fltarr(m)+1)

is just a fairly opaque way to take a vector of length n, and construct an array of dimensions (n,m) in which each row holds a copy of said vector. You might also see the equivalent:

w # replicate(1,m)

or even

replicate(1,m) ## w

("#" a "##" are duals under argument interchange). Another way to do this is with the REBIN & REFORM method you've no doubt heard about:

rebin(w,n,m) * der

You can read more about the REBIN/REFORM style of array inflation at David Fanning's site http://dfanning.com/tips/rebin_magic.html. Yet another method constructs an index array of the correct size and converts it to the right signature use various combinations of division and/or MOD:

w[lindgen(n,m) mod n] * der

Why so many ways to skin this particular feline? None of these methods is a truly optimized, native way to recast and redimension arrays for vector calculations. They're all just hacks, to one degree or another. For example, REBIN contains lots of useless (for this purpose) code for decimating arrays using various fancy sampling algorithms. If RSI ever wrote a dedicated "INFLATE" routine, it would probably be noticeably faster (and could combine the REBIN &

REFORM/TRANSPOSE into one easy step).

So why do I use REBIN/REFORM? It can certainly be slightly more verbose in common cases than either the matrix multiply, or index arithmetic methods (though not in this case). It also requires you to specify more of the dimensions explicitly (in this example, we had to have "n" in addition to "m"). However, it scales up to higher dimensions quite trivially, and easily accomodates inflating arrays over "inside" dimensions (neither the first nor the last). Try expanding over the 5th dimension of a 7 dimensional array using lindgen(n,m,p,q,r,s,t) and you'll soon curse the "/" key on your keyboard. As a final advantage, since IDLv5.5 added support for a vector-of-dimensions argument to REBIN, it's really the only technique which can reasonably be used to build generic operators which require arbitrary array inflation over arbitrary dimensions -- see for example my recent posting on computing threaded variances over large arrays. Quasi-anecdotal evidence also indicates it's the fastest method (but not by much, and really only if you use the /SAMPLE keyword to REBIN).

Whatever method you choose, be sure to comment liberally, since this is one place (of many) IDL code can quickly sink into the quagmire of unreadability.

JD
