Subject: Re: Matrix multiplication again...
Posted by Yngvar Larsen on Tue, 08 May 2012 18:08:02 GMT
View Forum Message <> Reply to Message

On Monday, 7 May 2012 17:40:49 UTC+2, Mats Löfdahl wrote:
> Suppose I have an image (let's say 128x128=16384 pixels) and for each pixel there is a vector with maybe 100 (could be more) elements. I organize this as a variable x with 16384 by 100 elements.
>
> Suppose I also have a 100x100 matrix M (or in general not symmetric but nevermind) and I want to calculate y, which is then also a 16384 by 100 array where
>
> y[i,*] = M ## x[i,*]

Why don't you simply use: y = M##x ? Should work fine.

Or possibly y=transpose(M)##x depending on how you organized your M matrix.

"rows" and "columns" are rather confusing terms in IDL... Which does not stop Excelis VIS from using them in their documentation of # and ##, of course.

I stopped thinking about matrix multiplication the ordinary way (rows and columns) in IDL. Too confusing. To avoid having to remember which dimension is the "row" and which is the "column", I just memorized the following two rules (actually I try to use only the first one if possible):

** Rule #1 **
A ~ fltarr(M, N)
B ~ fltarr(N, P)
=> size(A # B, /dimensions) ~ [M, P]

i.e. second dimension of A is "dotted" with first dimension of B.

** Rule ##2 **
A ~ fltarr(N,M)
B ~ fltarr(P,N)
=> size(A ## B, /dimensions) ~ [P, M]

i.e. first dimension of A is "dotted" with second dimension of B.

PS: To keep things even more confusing, both operators have special cases if one of the arrays is a one dimensional array, and yet another special case if both vectors are one dimensional (an outer product will be calculated.) *sigh*

--
Yngvar

Subject: Re: Matrix multiplication again...
Posted by on Tue, 08 May 2012 18:45:14 GMT
View Forum Message <> Reply to Message

Den tisdagen den 8:e maj 2012 kl. 20:08:02 UTC+2 skrev Yngvar Larsen:
> On Monday, 7 May 2012 17:40:49 UTC+2, Mats Löfdahl wrote:
>> Suppose I have an image (let's say 128x128=16384 pixels) and for each pixel there is a vector with maybe 100 (could be more) elements. I organize this as a variable x with 16384 by 100 elements.
>>
>> Suppose I also have a 100x100 matrix M (or in general not symmetric but nevermind) and I want to calculate y, which is then also a 16384 by 100 array where
>>
>> y[i,*] = M ## x[i,*]
>
> Why don't you simply use: y = M##x ?

Probably because I kept thinking of x as a 3D array, in spite of having reformed it to 2D...

> Should work fine.

Yes, I see that now. Thanks!

> "rows" and "columns" are rather confusing terms in IDL...

I know! I've opted to organize my code so I can always use the ## operator for matrix multiplication. Seems to work so far.

Subject: Re: Matrix multiplication again...
Posted by Yngvar Larsen on Wed, 09 May 2012 15:01:30 GMT
View Forum Message <> Reply to Message

On Tuesday, 8 May 2012 20:45:14 UTC+2, Mats Löfdahl wrote:
> Den tisdagen den 8:e maj 2012 kl. 20:08:02 UTC+2 skrev Yngvar Larsen:
>> On Monday, 7 May 2012 17:40:49 UTC+2, Mats Löfdahl wrote:
>>> Suppose I have an image (let's say 128x128=16384 pixels) and for each pixel there is a vector with maybe 100 (could be more) elements. I organize this as a variable x with 16384 by 100 elements.
>>>
>>> Suppose I also have a 100x100 matrix M (or in general not symmetric but nevermind) and I want to calculate y, which is then also a 16384 by 100 array where
>>>
>>> y[i,*] = M ## x[i,*]
>>
>> Why don't you simply use: y = M##x ?
>
> Probably because I kept thinking of x as a 3D array, in spite of having reformed it to 2D...

Speaking of which: It would be nice if the # and ## operators worked on arrays of more than 2 dimensions (very useful for e.g. tensors or your 3D example). Someting like this:

A ~ fltarr(N_1, N_2, ..., N_m, M)
B ~ fltarr(M, P_1, P_2, ..., P_k)
=> size(A # B, /dimensions) = [N_1, N_2, ..., N_m, P_1, P_2, ..., P_k)

I.e. the last dimension of A is "dotted" with the first dimension of B, and the other dimensions are preserved. Most likely it would not alter IDL's internal implementation of # and ## much.

Right now, we have to reform like in the OPs original problem (here with # instead of ##):

A = reform(A, N_1*N_2*...*N_m, M, /overwrite)
B = reform(B, M, P_1*P_2*...*P_k, /overwrite)
C = A # B
C = reform(C, N_1, N_2, ..., N_m, P_1, P_2, ..., P_k, /overwrite)

And analogously for ##.

I use this pattern all the time. No reason that # and ## should not be able to handle this internally so we don't have to.

>> "rows" and "columns" are rather confusing terms in IDL...
>
> I know! I've opted to organize my code so I can always use the ## operator for matrix multiplication. Seems to work so far.

Right. Like I said in my previous, I opted for # for no particular reason, but ## of course works equally well. One of the arrays will be traversed with a stride in memory either way with these operators. Of course, a third option, let's call it ###, could work like this:

A ~ fltarr(N, M)
B ~ fltarr(N, P)
=> size(A ### B, /dimensions) = [M, P]

This would most likely be the most efficient way to do matrix multiplication since both arrays are traversed with stride one. If I understand the documentation right, the MATRIX_MULTIPLY function might work this way with smart use of the /ATRANSPOSE and/or /BTRANSPOSE keywords.

--
Yngvar