Subject: Matrix algebra and index order, A # B vs A ## B
Posted by   on Mon, 26 Mar 2012 11:33:02 GMT
View Forum Message <> Reply to Message

IDL has two operators for matrix multiplication, # and ##. The former assumes the matrices involved have colum number as the first index and row number as the second, i.e., A_{rc} = A[c,r] with mathematics on the LHS and IDL on the RHS. The latter operator makes the opposite assumption, A_{rc} = A[r,c].

I believe much headache can be avoided if one chooses one notation and sticks with it. If it were only me, I'd choose the A_{rc} = A[r,c] notation. But it isn't only me, because I like to take advantage of IDL routines written by others. So, has there merged some kind of consensus among influential IDL programmers (those that write publicly available routines that are widely used - thank you BTW!) for which convention to use?

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by David Fanning on Mon, 26 Mar 2012 14:35:57 GMT
View Forum Message <> Reply to Message

Mats Löfdahl writes:

> It's that bad? :o)

Well, it gets better after a couple of beers. :-)

```
>
> One thing that had me wondering is the documentation for Craig Markwardt's qrfac routine:
>
>
> ; Given an MxN matrix A (M>N), the procedure QRFAC computes the QR
> ; decomposition (factorization) of A.  This factorization is useful
> ; in least squares applications solving the equation, A # x = B.
> ; Together with the procedure QRSOLV, this equation can be solved in
> ; a least squares sense.
> ;
> ; The QR factorization produces two matrices, Q and R, such that
> ;
> ;     A = Q ## R
> ;
> ; where Q is orthogonal such that TRANSPOSE(Q)##Q equals the identity
> ; matrix, and R is upper triangular.
```

See, this is the real problem. Is Craig explaining the linear
algebra to us, or is he giving us the IDL rendition of the
linear algebra? Or, did Craig write the documentation after
the fact and is really trying to forget what he *actually*
did in the program? All I know is that anytime I read the

word "matrix" I can tell I'm about to get a headache.

> The ## operator for the matrix-matrix multiplications but # for
> matrix-vector multiplication! But then I thought this might be
> IDL 1D arrays being interpreted as row vectors so x # A is
> actually just another way of writing A ## transpose(x). And
> the former would be more efficient. Am I on the right
> track here...?

Probably. But I can't be bothered. I've got to go
find some aspirin. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by Craig Markwardt on Mon, 26 Mar 2012 22:41:18 GMT
View Forum Message <> Reply to Message

On Monday, March 26, 2012 9:45:51 AM UTC-4, Mats Löfdahl wrote:
> On Monday, March 26, 2012 3:00:05 PM UTC+2, David Fanning wrote:
>> Mats Löfdahl writes:
>>
>>> IDL has two operators for matrix multiplication, # and ##.
>>> The former assumes the matrices involved have colum number as
>>> the first index and row number as the second, i.e., A_{rc} =
>>> A[c,r] with mathematics on the LHS and IDL on the RHS. The
>>> latter operator makes the opposite assumption, A_{rc} = A[r,c].
>>>
>>> I believe much headache can be avoided if one chooses one
>>> notation and sticks with it. If it were only me, I'd choose
>>> the A_{rc} = A[r,c] notation. But it isn't only me, because
>>> I like to take advantage of IDL routines written by others.
>>> So, has there emerged some kind of consensus among influential
>>> IDL programmers (those that write publicly available
>>> routines that are widely used - thank you BTW!) for
>>> which convention to use?
>>

>> Yes, the consensus that has emerged is that no operation
>> is more fraught with ambiguity, anguish, and frustration
>> than trying to translate a section of linear algebra code
>> from a paper or textbook (say on Principle Components
>> Analysis) to IDL than almost anything you can imagine!
>> It's like practicing backwards writing in the mirror.
>>
>> And, of course, while you are doing it you have the
>> growing realization that there is no freaking way you
>> are EVER going to be able to write the on-line
>> documentation to explain this dog's dish of a program
>> to anyone else. :-(
>>
>> The solution, of course, is to stick with the ##
>> notation for as long as it makes sense, then throw
>> in a couple of # signs whenever needed to make the
>> math come out right. :-)
>
> It's that bad? :o)
>
> One thing that had me wondering is the documentation for Craig Markwardt's qrfac routine:
>
>
> ; Given an MxN matrix A (M>N), the procedure QRFAC computes the QR
> ; decomposition (factorization) of A. This factorization is useful
> ; in least squares applications solving the equation, A # x = B.
> ; Together with the procedure QRSOLV, this equation can be solved in
> ; a least squares sense.
> ;
> ; The QR factorization produces two matrices, Q and R, such that
> ;
> ;   A = Q ## R
> ;
> ; where Q is orthogonal such that TRANSPOSE(Q)##Q equals the identity
> ; matrix, and R is upper triangular.
>
> The ## operator for the matrix-matrix multiplications but # for matrix-vector multiplication! But then I thought this might be IDL 1D arrays being interpreted as row vectors so x # A is actually just another way of writing A ## transpose(x). And the former would be more efficient. Am I on the right track here...?

I believe I double checked the notation of QRFAC when I wrote it way back when.

Maybe you need to read this part of the documentation as well....

```
;       Note that the dimensions of A in this routine are the
;       *TRANSPOSE* of the conventional appearance in the least
;       squares matrix equation.
```

The transposed matrix means you flip all the #'s: # <--> ##.

I realize this is very confusing, but unfortunately I inherited this code from somewhere else (MPFIT), so it retains the warts of the original.

By the way, there's an example provided with the documentation, which you could test the notation for yourself.

Craig

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by   on Tue, 27 Mar 2012 07:45:40 GMT

View Forum Message <> Reply to Message

Den tisdagen den 27:e mars 2012 kl. 00:41:18 UTC+2 skrev Craig Markwardt:
> On Monday, March 26, 2012 9:45:51 AM UTC-4, Mats Löfdahl wrote:
>> On Monday, March 26, 2012 3:00:05 PM UTC+2, David Fanning wrote:
>>> Mats Löfdahl writes:
>>>
>>>> IDL has two operators for matrix multiplication, # and ##.
>>>> The former assumes the matrices involved have colum number as
>>>> the first index and row number as the second, i.e., A_{rc} =
>>>> A[c,r] with mathematics on the LHS and IDL on the RHS. The
>>>> latter operator makes the opposite assumption, A_{rc} = A[r,c].
>>>>
>>>> I believe much headache can be avoided if one chooses one
>>>> notation and sticks with it. If it were only me, I'd choose
>>>> the A_{rc} = A[r,c] notation. But it isn't only me, because
>>>> I like to take advantage of IDL routines written by others.
>>>> So, has there emerged some kind of consensus among influential
>>>> IDL programmers (those that write publicly available
>>>> routines that are widely used - thank you BTW!) for
>>>> which convention to use?
>>>
>>> Yes, the consensus that has emerged is that no operation
>>> is more fraught with ambiguity, anguish, and frustration
>>> than trying to translate a section of linear algebra code
>>> from a paper or textbook (say on Principle Components
>>> Analysis) to IDL than almost anything you can imagine!
>>> It's like practicing backwards writing in the mirror.
>>>
>>> And, of course, while you are doing it you have the
>>> growing realization that there is no freaking way you
>>> are EVER going to be able to write the on-line
>>> documentation to explain this dog's dish of a program
>>> to anyone else. :-(

>>>
>>> The solution, of course, is to stick with the ##
>>> notation for as long as it makes sense, then throw
>>> in a couple of # signs whenever needed to make the
>>> math come out right. :-)
>>
>> It's that bad? :o)
>>
>> One thing that had me wondering is the documentation for Craig Markwardt's qrfac routine:
>>
>>
>> ; Given an MxN matrix A (M>N), the procedure QRFAC computes the QR
>> ; decomposition (factorization) of A. This factorization is useful
>> ; in least squares applications solving the equation, A # x = B.
>> ; Together with the procedure QRSOLV, this equation can be solved in
>> ; a least squares sense.
>> ;
>> ; The QR factorization produces two matrices, Q and R, such that
>> ;
>> ;   A = Q ## R
>> ;
>> ; where Q is orthogonal such that TRANSPOSE(Q)##Q equals the identity
>> ; matrix, and R is upper triangular.
>>
>> The ## operator for the matrix-matrix multiplications but # for matrix-vector multiplication! But then I thought this might be IDL 1D arrays being interpreted as row vectors so x # A is actually just another way of writing A ## transpose(x). And the former would be more efficient. Am I on the right track here...?
>
> I believe I double checked the notation of QRFAC when I wrote it way back when.
>
> Maybe you need to read this part of the documentation as well....
>
> ;   Note that the dimensions of A in this routine are the
> ;   *TRANSPOSE* of the conventional appearance in the least
> ;   squares matrix equation.

Yes, but that doesn't help much when "the conventional appearance" is not defined...

> The transposed matrix means you flip all the #'s: # <--> ##.
>
> I realize this is very confusing, but unfortunately I inherited this code from somewhere else (MPFIT), so it retains the warts of the original.
>
> By the way, there's an example provided with the documentation, which you could test the notation for yourself.

Yes, of course. Sorry, I realize I gave the impression that I had problems running the qrfac program. I don't, trial and error solved that problem. But it got me thinking about it and I thought it might be nice to find out the most common convention and then perhaps stay sane by writing wrappers around routines that use the opposite convention (if I stumble upon any). At least when that can be done without much time penalty.

Anyway, if my notation on the math side is right I believe qrfac uses the A[r,c] notation. So that's one data point in favor of the ## operator. But the comment about "the conventional appearance" is then a data point against?

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by Craig Markwardt on Tue, 27 Mar 2012 13:11:17 GMT
View Forum Message <> Reply to Message

On Tuesday, March 27, 2012 3:45:40 AM UTC-4, Mats Löfdahl wrote:
> Den tisdagen den 27:e mars 2012 kl. 00:41:18 UTC+2 skrev Craig Markwardt:
>> On Monday, March 26, 2012 9:45:51 AM UTC-4, Mats Löfdahl wrote:
>>> On Monday, March 26, 2012 3:00:05 PM UTC+2, David Fanning wrote:
>>>> Mats Löfdahl writes:
>>>>
>>>> > IDL has two operators for matrix multiplication, # and ##.
>>>> > The former assumes the matrices involved have colum number as
>>>> > the first index and row number as the second, i.e., A_{rc} =
>>>> > A[c,r] with mathematics on the LHS and IDL on the RHS. The
>>>> > latter operator makes the opposite assumption, A_{rc} = A[r,c].
>>>> >
>>>> > I believe much headache can be avoided if one chooses one
>>>> > notation and sticks with it. If it were only me, I'd choose
>>>> > the A_{rc} = A[r,c] notation. But it isn't only me, because
>>>> > I like to take advantage of IDL routines written by others.
>>>> > So, has there emerged some kind of consensus among influential
>>>> > IDL programmers (those that write publicly available
>>>> > routines that are widely used - thank you BTW!) for
>>>> > which convention to use?
>>>>
>>>> Yes, the consensus that has emerged is that no operation
>>>> is more fraught with ambiguity, anguish, and frustration
>>>> than trying to translate a section of linear algebra code
>>>> from a paper or textbook (say on Principle Components
>>>> Analysis) to IDL than almost anything you can imagine!
>>>> It's like practicing backwards writing in the mirror.
>>>>
>>>> And, of course, while you are doing it you have the
>>>> growing realization that there is no freaking way you
>>>> are EVER going to be able to write the on-line
>>>> documentation to explain this dog's dish of a program
>>>> to anyone else. :-(
>>>>

>>>> The solution, of course, is to stick with the ##
>>>> notation for as long as it makes sense, then throw
>>>> in a couple of # signs whenever needed to make the
>>>> math come out right. :-)
>>>
>>> It's that bad? :o)
>>>
>>> One thing that had me wondering is the documentation for Craig Markwardt's qrfac routine:
>>>
>>>
>>> ; Given an MxN matrix A (M>N), the procedure QRFAC computes the QR
>>> ; decomposition (factorization) of A. This factorization is useful
>>> ; in least squares applications solving the equation, A # x = B.
>>> ; Together with the procedure QRSOLV, this equation can be solved in
>>> ; a least squares sense.
>>> ;
>>> ; The QR factorization produces two matrices, Q and R, such that
>>> ;
>>> ;   A = Q ## R
>>> ;
>>> ; where Q is orthogonal such that TRANSPOSE(Q)##Q equals the identity
>>> ; matrix, and R is upper triangular.
>>>
>>> The ## operator for the matrix-matrix multiplications but # for matrix-vector multiplication! But then I thought this might be IDL 1D arrays being interpreted as row vectors so x # A is actually just another way of writing A ## transpose(x). And the former would be more efficient. Am I on the right track here...?
>>
>> I believe I double checked the notation of QRFAC when I wrote it way back when.
>>
>> Maybe you need to read this part of the documentation as well....
>>
>> ;   Note that the dimensions of A in this routine are the
>> ;   *TRANSPOSE* of the conventional appearance in the least
>> ;   squares matrix equation.
>
> Yes, but that doesn't help much when "the conventional appearance" is not defined...
>
>> The transposed matrix means you flip all the #'s: # <--> ##.
>>
>> I realize this is very confusing, but unfortunately I inherited this code from somewhere else (MPFIT), so it retains the warts of the original.
>>
>> By the way, there's an example provided with the documentation, which you could test the notation for yourself.
>
> Yes, of course. Sorry, I realize I gave the impression that I had problems running the qrfac program. I don't, trial and error solved that problem. But it got me thinking about it and I thought it

might be nice to find out the most common convention and then perhaps stay sane by writing wrappers around routines that use the opposite convention (if I stumble upon any). At least when that can be done without much time penalty.
>
> Anyway, if my notation on the math side is right I believe qrfac uses the A[r,c] notation. So that's one data point in favor of the ## operator. But the comment about "the conventional appearance" is then a data point against?

The statement in the QRFAC documentation is not about array multiplication notation, it's about the typical dimensions of the "A" matrix.

Normally in QR factorization, an "A" matrix is tall and thin. Something like DBLARR(N,M) where M > N.

QRFAC accepts the transpose of this matrix, i.e. short and fat. DBLARR(M,N)

My understanding is that
 ## - is what you should always use for standard matrix multiplication (array-array or array-vector)
 # - is what you should use when ## doesn't work right.

Craig

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by David Fanning on Tue, 27 Mar 2012 13:27:35 GMT
View Forum Message <> Reply to Message

Craig Markwardt writes:

> My understanding is that
> ## - is what you should always use for standard matrix multiplication (array-array or array-vector)
> # - is what you should use when ## doesn't work right.

I'm relieved to know Craig and I are on the same page here!
There are times when you start fooling around with this
stuff and you think you are going crazy. ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/

Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by on Wed, 28 Mar 2012 10:45:09 GMT
View Forum Message <> Reply to Message

Den tisdagen den 27:e mars 2012 kl. 15:27:35 UTC+2 skrev David Fanning:

> I'm relieved to know Craig and I are on the same page here!

That's enough of a consensus for me. :o)

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by on Thu, 05 Apr 2012 14:55:05 GMT
View Forum Message <> Reply to Message

I'm writing some code where matrix algebra is at the heart of things so I really wanted to understand the conventions and convince myself that I can use them in a consistent way.

I realized a major problem for me was not being able to inspect the matrices properly. The IDL print command can print arrays but because the arrays are interpreted as matrices differently depending on which convention is used, I was never sure of what I was looking at. So I wrote a subroutine that prints a matrix the way it is interpreted, based on the multiplication operator (which I think is a convenient way of specifying column-major/row-major). That really helped me make sense of it so I put it and some examples on a web page here:
http://www.solarphysics.kva.se/~mats/idl/matrixalgebra/

I also satisfied myself that I understand the output of LA_SVD and QRfac and that they use the ## operator convention (with a little workaround for QRfac), see the same page.

OK, so this helped me understand what I'm doing. Does it address the frustration that you all have expressed in this thread or are there other issues that I just haven't realized yet?

/Mats

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by David Fanning on Thu, 05 Apr 2012 15:22:34 GMT
View Forum Message <> Reply to Message

Mats Löfdahl writes:

> I'm writing some code where matrix algebra is at the heart of things so I really wanted to understand the conventions and convince myself that I can use them in a consistent way.
>

> I realized a major problem for me was not being able to inspect the matrices properly. The IDL print command can print arrays but because the arrays are interpreted as matrices differently depending on which convention is used, I was never sure of what I was looking at. So I wrote a subroutine that prints a matrix the way it is interpreted, based on the multiplication operator (which I think is a convenient way of specifying column-major/row-major). That really helped
me make sense of it so I put it and some examples on a web page here:
http://www.solarphysics.kva.se/~mats/idl/matrixalgebra/

Years ago there was a joint product created between RSI
and IMSL called IDL-IMSL. IMSL produced a very powerful
mathematical library, including a number of matrix
algebra routines. This problem of conventions plagued
the partnership (an entire generation of IMSL engineers
either retired early or ended up the Houston Home for
the Mentally Insane), and a number of these "print"
and "read" routines were created then. I just looked
for them in the "obsolete" directory, but couldn't
find them. They really did nothing more than add
to the confusion, so I'm not surprised they are no
longer with us.

> Does it address the frustration that you all have expressed in this
> thread or are there other issues that I just haven't realized yet?

Alas, anything written on this subject seems to add
to the frustration, rather than alleviate it, but maybe
that's just me. ;-)

I applaud your efforts in any case! :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by Kenneth P. Bowman on Thu, 05 Apr 2012 16:48:02 GMT
View Forum Message <> Reply to Message

In article <MPG.29e766a33cd1eb35989a43@news.giganews.com>,

David Fanning <news@idlcoyote.com> wrote:

>
>> I'm writing some code where matrix algebra is at the heart of things so I
>> really wanted to understand the conventions and convince myself that I can
>> use them in a consistent way.

The "Manipulating Arrays" section of the documentation is some help.

As is so often the case with IDL, the ultimate solution to understanding
how it actually works is to create a trivial example and make sure
that you understand it.

This is easier than trying to figure out row-major, column-major, etc.

Such as

```
IDL> a = FINDGEN(3,3)
IDL> x = REPLICATE(1.0, 3)
IDL> print, x
      1.00000      1.00000      1.00000
IDL> print, TRANSPOSE(a)
      0.00000      3.00000      6.00000
      1.00000      4.00000      7.00000
      2.00000      5.00000      8.00000
IDL> print, a#x
      9.00000      12.0000      15.0000
IDL> print, a
      0.00000      1.00000      2.00000
      3.00000      4.00000      5.00000
      6.00000      7.00000      8.00000
IDL> print, a##x
      3.00000
      12.0000
      21.0000
```

Ken Bowman

Subject: Re: Matrix algebra and index order, A # B vs A ## B
Posted by   on Thu, 05 Apr 2012 17:29:33 GMT
View Forum Message <> Reply to Message

Kenneth P. Bowman:
> David Fanning:

```
>
>>  Mats Löfdahl writes:
>>
>>>  I'm writing some code where matrix algebra is at the heart of things so I
>>>  really wanted to understand the conventions and convince myself that I can
>>>  use them in a consistent way.
>
>  The "Manipulating Arrays" section of the documentation is some help.
>
>  As is so often the case with IDL, the ultimate solution to understanding
>  how it actually works is to create a trivial example and make sure
>  that you understand it.
>
>  This is easier than trying to figure out row-major, column-major, etc.
>
>  Such as
>
>  IDL> a = FINDGEN(3,3)
>  IDL> x = REPLICATE(1.0, 3)
>  IDL> print, x
>       1.00000      1.00000      1.00000
>  IDL> print, TRANSPOSE(a)
>       0.00000      3.00000      6.00000
>       1.00000      4.00000      7.00000
>       2.00000      5.00000      8.00000
>  IDL> print, a#x
>       9.00000      12.0000      15.0000
>  IDL> print, a
>       0.00000      1.00000      2.00000
>       3.00000      4.00000      5.00000
>       6.00000      7.00000      8.00000
>  IDL> print, a##x
>       3.00000
>       12.0000
>       21.0000
```

I guess everybody has to understand various concepts in their own way. To me, your example just demonstrates one thing I tried to avoid: having to memorize when an array can be printed as it is and look like the matrix it is intended to represent and when it has to be transposed.

/Mats