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

Subject: Re: Dimensional Juggling Tutorial
Posted by Samuel Haimov on Tue, 03 Apr 2001 18:19:32 GMT
View Forum Message <> Reply to Message

A message from one of these lurking occasional readers (I wish I could manage my time better):

Just to give my 2 cents of help to those that would like a routine that replicates arrays using the ideas discussed by JD.

I'm including here a function called replicarr. It also uses a modified version of rebin (I called rebin1), which is also included. As usual, no
warranty of any kind is given.

Hope this might be useful to somebody. I'm still dragging my feet to put my
idl routines for public use (like anybody cares :-( )

Cheers, Sam (haimov@uwyo.edu)

_____

;$Id: replicarr.pro,v 2.0 2000/03/20 16:28:40 haimov Exp haimov $

```
;+
; NAME:
;       REPLICARR
;
; PURPOSE:
;       This function replicates N-dim. array to (N+1)-dim. array
;       (e.g, vector to matrix). Scalars can be strings or any numerical
type.
;       Vectors can be any numerical type. Matrices and higer dim.
arrays must
;       be non-complex numerical.
;
; CATEGORY:
;       idltools/general
;
; CALLING SEQUENCE:
;       result=replicarr(x,dn)
;
; INPUTS:
;       X:      scalar or array to be replicated
;       DN:     size of the new (added) dimension
;
; OUTPUTS:
;       The result has one more DN-size dimension than X.
;       Every element along DN is a copy of X.
;
; KEYWORD PARAMETERS:
;       DIM:    Position of the new dimension; 1, 2, ..., LastXdim+1
;               1          - replication in row stack
;               LastXdim+1 - replication in column stack  (default)
;       HELP:   Shows this text
;
; SIDE EFFECT:
;       Scalar is copied DN times to a vector (uses replicate function)
;       Matrix is copied DN times to a 3-D array in a manner determined
by DIM
;       N_dimensional arrays are coppied DN times to N+1_dim array along
DIM
;
;       This function does not work for string arrays or complex arrays
;       with more than 1 dimension
;
; EXAMPLE:
;       x=findgen(3,3)
;       y=replicarr(x,2)
;       z=replicarr(x,2,dim=1)
;
; MODIFICATION HISTORY:
```

```
;       Written by:  Samuel Haimov, October 1997
;       Feb 2000, S.H. -- optimized replication by using rebin1 function
;                         added help keyword
;-


function replicarr,x,dn,dim=dim,help=help

on_error,2

if keyword_set(help) then begin
  doc_library,'replicarr'
  return,''
endif

if n_params() eq 0 then begin
   message,"Usage: result=replicarr(x,dn)",/info
   return,''
endif

; Check the input

xsize=size(x)
xdim=xsize(0)
nxsize=n_elements(xsize)

if xsize(nxsize-2) eq 0 then message,'Input variable is undefined'
if dn lt 1 then message,'Additional dimension size is less than 1'
if n_elements(dim) eq 0 then dim=xdim+1

; For scalars or vectors use replicate or matrix multiplication

  if xdim eq 0 then return,replicate(x,dn) $
  else if xdim eq 1 and xsize(nxsize-2) lt 7 then $
     if dim eq 1 then return, x##replicate(1,dn) else return,
x#replicate(1,dn)

; For 2D or more dimensions use rebin1 and reform

  if xsize(nxsize-2) gt 6 then message,'Input variable is not numerical'
$
  else if xsize(nxsize-2) eq 6 then $
     message,'Complex type is not allowed for multi-dimensional arrays'

  ; Create arrays of the output dimensions

  if dim eq 1 then begin
    dims=[dn,xsize[1:xdim]]
```

```
   dimr=[1,xsize[1:xdim]]  ; use dimr to add the new dimension by using
reform
  endif else if dim eq xdim+1 then begin
   dims=[xsize[1:xdim],dn]
   dimr=[xsize[1:xdim],1]
  endif else begin
   dims=[xsize[1:dim-1],dn,xsize[dim:xdim]]
   dimr=[xsize[1:dim-1],1,xsize[dim:xdim]]
  endelse

  ; Rebin the output

  return,rebin1(reform(x,dimr),dims)

end
 ------------------------------------------------------------------------
-------------------------

;$Id: rebin1.pro,v 2.0 2000/03/20 16:28:40 haimov Exp haimov $
;+
; NAME:
;       REBIN1
;
; PURPOSE:
;       The REBIN1 function resizes a vector/array to dimensions given
in DIMS
;       This function is identical to RSI function REBIN but accepts an
input
;       array for the dimensions instead of a list of the dimensions.
;
; CATEGORY:
;       idltools/general
;
; CALLING SEQUENCE:
;       result=rebin1(Array,Dims, [/SAMPLE, /HELP])
;
; INPUTS:
;       Dims:   Array with the dimensions of the resulting resampled
array
;               (Dims = [D1,D2,...,DN]).  The dimensions must be integer
;               multiples or factors of the corresponding original
dimension
;               Max num. of dimensions (dims size) is 8
;
; KEYWORD PARAMETERS:
;       SAMPLE: Normally, REBIN uses bilinear interpolation when
magnifying
;               and neighborhood averaging when minifying.  Set the
```

```
keyword to
;             use nearest neighbor sampling for both magnification and
;             minification. Bilinear interpolation gives higher
quality
;             results but requires more time.
;       HELP:   Shows this text
;-
; MODIFICATION HISTORY:
;       Written by:  Samuel Haimov, Feb 2000
;

function rebin1, array,dims,sample=sample,help=help

on_error,2

if keyword_set(help) then begin
  doc_library,'rebin1'
  return,''
endif

if n_params() lt 2 then begin
   message,"Usage: result=rebin(array,dims)",/info
   return,''
endif

case n_elements(dims) of
   0 : begin
         message,"DIMS is not defined.",/info
         return,''
       end
   1 : return,rebin(array,dims[0],sample=sample)
   2 : return,rebin(array,dims[0],dims[1],sample=sample)
   3 : return,rebin(array,dims[0],dims[1],dims[2],sample=sample)
   4 :  return,rebin(array,dims[0],dims[1],dims[2],dims[3],sample=sa mple)
   5 :
 return,rebin(array,dims[0],dims[1],dims[2],dims[3],dims[4],s ample=sample)
   6 :
 return,rebin(array,dims[0],dims[1],dims[2],dims[3],dims[4],d ims[5], $
                 sample=sample)
   7 :
 return,rebin(array,dims[0],dims[1],dims[2],dims[3],dims[4],d ims[5], $
                 dims[6],sample=sample)
   8  :
 return,rebin(array,dims[0],dims[1],dims[2],dims[3],dims[4],d ims[5], $
                 dims[6],dims[7],sample=sample)
   ELSE: begin
          message,"Output array have too many dimensions",/info
          return,''
```

```
        end
endcase

end
```

JD Smith wrote:
>
> 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

Subject: Question (Re: Dimensional Juggling Tutorial)
Posted by marc schellens[1] on Thu, 05 Apr 2001 05:04:36 GMT
View Forum Message <> Reply to Message

I have a simple question, more or less related to
the rebin stuff:

Assume the following routine, which shoud calculate the pairwise
distance of two array of 3-dim vectors:

```
array=indgen(3,21)

a=array[*,0:5]
b=array[*,6:*]

nA=n_elements(a)/3
nB=n_elements(b)/3
```

```
distArr2=fltarr(nA,nB)
for i1=0,nT1-1 do begin

    distArr2[i1,*]=sqrt(total((b-rebin(a[*,i1],3,nA))^2,1))
endfor
```

so far so good, the routine does the job, but is there
a possibility to eliminate the for loop completely?

Greetings,
:-) marc
