Subject: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by clement.feller@obspm. on Mon, 13 Nov 2017 14:50:11 GMT
View Forum Message <> Reply to Message

Hello everyone,

I coming back to you for some advice on how to properly generate a grid in an N-D space. I hope that this expression is the proper one in english, but in any case, let me illustrate this by the following exemple:

```
>  a = indgen(3,3) & print, a
 0 1 2
 3 4 5
 6 7 8
```

What I am looking for would be to find the clean and proper IDL way to generate the following sets of combinations:

```
 0,1,2
 0,1,5
 0,1,8
 0,4,2
 0,4,5
 .....
 .....

 6,7,2
 6,7,5
 6,7,8
```

Now, I have found ways to do this for a 2D,3D,4D,5D space with either nested loops (yuck! I know), or with combinations of rebin, reform and transpose.
I've been successfully using those solutions for several weeks, yet I wonder on how to expand this to a general case and in the proper IDL way.

Why and how this is useful to me ?
I am actually trying to evaluate a function with several parameters. Let's call it f(x, p_0, ..., p_n). Given x, I want to evaluate f with multiple sets of parameters.
E.g. to generate a regular grid I can use INTERPOLATE, e.g. for a 2D space with 10 evaluations for each dimension:

```
>  p = [[0.,1.], [0,100]]
>  p = transpose(p)
>  interpolate(p,1./9.*findgen(10))
      0.0000000       0.0000000
     0.11111111       11.111112
     0.22222222       22.222223
     0.33333334       33.333336
     0.44444445       44.444447
```

0.55555558 55.555557
0.66666669 66.666672
0.77777779 77.777779
0.88888890 88.888893
1.0000000 100.00000

If one wants an irregular grid (with 10 evaluations in the first dimension and 5 in the second one), one can use a nested loop and play with indices.

So, once you have this table, how can one generate the proper sets of combinations of indices ? Another way to look at it is that you just want to "multiply" or chunk index your table, i.e. to generate the n vector used in the histogram's i-vector example (http://www.idlcoyote.com/tips/histogram_tutorial.html).

I've playing around with nested indgen, looking for a repetitive motive from the 2D to the 5D space when using rebin, reform, transpose to assemble a generic command. But nothing much so far....

Does anybody out there already had a go with such problem before or any advice ?

I thank you all in advance for your replies.
Cheers,
/C

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by Michael Galloy on Tue, 14 Nov 2017 00:04:58 GMT
View Forum Message <> Reply to Message

On 11/13/17 7:50 AM, clement.feller@obspm.fr wrote:
> Hello everyone,
>
> I coming back to you for some advice on how to properly generate a grid in an N-D space. I hope that this expression is the proper one in english, but in any case, let me illustrate this by the following exemple:
>> a = indgen(3,3) & print, a
> 0 1 2
> 3 4 5
> 6 7 8
>
> What I am looking for would be to find the clean and proper IDL way to generate the following sets of combinations:
> 0,1,2
> 0,1,5
> 0,1,8
> 0,4,2
> 0,4,5
> .....

> .....
>
> 6,7,2
> 6,7,5
> 6,7,8
>
> Now, I have found ways to do this for a 2D,3D,4D,5D space with either nested loops (yuck! I know), or with combinations of rebin, reform and transpose.
> I've been successfully using those solutions for several weeks, yet I wonder on how to expand this to a general case and in the proper IDL way.
>
> Why and how this is useful to me ?
> I am actually trying to evaluate a function with several parameters. Let's call it f(x, p_0, ..., p_n). Given x, I want to evaluate f with multiple sets of parameters.
> E.g. to generate a regular grid I can use INTERPOLATE, e.g. for a 2D space with 10 evaluations for each dimension:
>
>> p = [[0.,1.], [0,100]]
>> p = transpose(p)
>> interpolate(p,1./9.*findgen(10))
>   0.0000000   0.0000000
>   0.11111111   11.111112
>   0.22222222   22.222223
>   0.33333334   33.333336
>   0.44444445   44.444447
>   0.55555558   55.555557
>   0.66666669   66.666672
>   0.77777779   77.777779
>   0.88888890   88.888893
>   1.0000000   100.00000
>
> If one wants an irregular grid (with 10 evaluations in the first dimension and 5 in the second one), one can use a nested loop and play with indices.
>
> So, once you have this table, how can one generate the proper sets of combinations of indices ?
> Another way to look at it is that you just want to "multiply" or chunk index your table, i.e. to generate the n vector used in the histogram's i-vector example (http://www.idlcoyote.com/tips/histogram_tutorial.html).
>
> I've playing around with nested indgen, looking for a repetitive motive from the 2D to the 5D space when using rebin, reform, transpose to assemble a generic command. But nothing much so far....
>
> Does anybody out there already had a go with such problem before or any advice ?
>
> I thank you all in advance for your replies.
> Cheers,

> /C
>

I don't have a good solution. I think the below routine is general, but
slow solution:

https://github.com/mgalloy/mglib/blob/master/src/analysis/mg _find_combinations.pro

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by Markus Schmassmann on Tue, 14 Nov 2017 10:38:37 GMT
View Forum Message <> Reply to Message

On 11/13/2017 03:50 PM, clement.feller@obspm.fr wrote:
> I coming back to you for some advice on how to properly generate a grid in an N-D space. I hope that this expression is the proper one in english, but in any case, let me illustrate this by the following exemple:
>> a = indgen(3,3) & print, a
> 0 1 2
> 3 4 5
> 6 7 8
>
> What I am looking for would be to find the clean and proper IDL way to generate the following sets of combinations:
> 0,1,2
> 0,1,5
> 0,1,8
> 0,4,2
> 0,4,5
> .....
> .....
>
> 6,7,2
> 6,7,5
> 6,7,8
>
> Now, I have found ways to do this for a 2D,3D,4D,5D space with either nested loops (yuck! I know), or with combinations of rebin, reform and transpose.
> I've been successfully using those solutions for several weeks, yet I wonder on how to expand this to a general case and in the proper IDL way.
>

> [...]
>
> I've playing around with nested indgen, looking for a repetitive motive from the 2D to the 5D space when using rebin, reform, transpose to assemble a generic command. But nothing much so far....
>
> Does anybody out there already had a go with such problem before or any advice ?
is this what you are looking for ?

```
array=lindgen(n,long(n)^n)
for k=0,n-1 do array[k,*]= $
    rebin((n*lindgen(n^(k+1))+k) mod (n^2),long(n)^n,/sample)
```

Markus

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by Markus Schmassmann on Tue, 14 Nov 2017 11:29:34 GMT
View Forum Message <> Reply to Message

On 11/14/2017 11:38 AM, Markus Schmassmann wrote:
> On 11/13/2017 03:50 PM, clement.feller@obspm.fr wrote:
>> I coming back to you for some advice on how to properly generate a
>> grid in an N-D space. I hope that this expression is the proper one in
>> english, but in any case, let me illustrate this by the following
>> exemple:
>>> a = indgen(3,3) & print, a
>> 0 1 2
>> 3 4 5
>> 6 7 8
>>
>> What I am looking for would be to find the clean and proper IDL way to
>> generate the following sets of combinations:
>> 0,1,2
>> 0,1,5
>> 0,1,8
>> 0,4,2
>> 0,4,5
>> .....
>> .....
>>
>> 6,7,2
>> 6,7,5
>> 6,7,8
>>
>> Now, I have found ways to do this for a 2D,3D,4D,5D space with either
>> nested loops (yuck! I know), or with combinations of rebin, reform and

>> transpose.
>> I've been successfully using those solutions for several weeks, yet I
>> wonder on how to expand this to a general case and in the proper IDL way.
>>
>> [...]
>>
>> I've playing around with nested indgen, looking for a repetitive
>> motive from the 2D to the 5D space when using rebin, reform, transpose
>> to assemble a generic command. But nothing much so far....
>>
>> Does anybody out there already had a go with such problem before or
>> any advice ?
> is this what you are looking for ?
>
> array=lindgen(n,long(n)^n)
> for k=0,n-1 do array[k,*]= $
> rebin((n*lindgen(n^(k+1))+k) mod (n^2),long(n)^n,/sample)
sorry, for n>5 you have an overflow, correct is:

```
array=bindgen(n,long(n)^n)
for k=0,n-1 do array[k,*]=rebin(byte( $
    (n*lindgen(long(n)^(k+1))+k) mod (n^2) ),long(n)^n, /sample )
```

it's also better on memory, for n=8 the index is only 16MB

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by clement.feller@obspm. on Tue, 14 Nov 2017 13:16:45 GMT
View Forum Message <> Reply to Message

Hello again,

Thanks both of you for your replies.

@Mike: I had looked into this before (I think Jeremy Bailin has published a code similar to yours called combigen.pro), but I then meet difficulties in selecting part of the generated combinations.

@Markus: I say, your code is sleek and nifty. I like your solution.

In the meantime, I had given this problem some more thoughts and I had come up with another slow ugly one that doesn't work for all cases:

```
function gen_indices_comb, m, n
;d I/O:
;d m -> long integer corresponds to the number of row in original table
;d n -> long integer corresponds to the number of columns in original table
;d
```

```
;d vals -> long array listing the vectors of indices to extract the
;d         different possible combinations from the values of the original
;d         table
;d
;d NOTES: SLOW CODE, a mitigation of the values of m and n is REQUIRED
;d        Cases, where m & n are greater than 9, are not to considered
;d        with this code

 nmax = m^n

;c Assemble command generating vector of indices
 cmd = 'tmp = ['
 for ijk=(m-1L),1L,-1L do $
   cmd += ' (lmn/n^'+string(ijk,format='(I03)')+') mod n,'
 cmd += 'lmn mod n ]'

;c initialiase memory
 ini = indgen(m,n)
 tmp = lonarr(m)
 val = lonarr(m, m^n)

;c execute command for each type of combination
 for lmn=0L,(n^m-1L) do begin
   void = execute(cmd)
   if void ne 1 then message, ' > Error generating indices.'
   val[*,ijk] = ini+tmp*m
 endfor

 return, val
end
```

Afn I'm considering this post solved, I'll update it with a definitive version of my solution.

Again thanks your replies,
/C

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by Dick Jackson on Tue, 14 Nov 2017 18:13:58 GMT

View Forum Message <> Reply to Message

Hi Clement,

I found a few efficiencies to improve Markus' method a little, and I think I have a working method to allow unequal numbers of rows and columns—it could be improved, but I have to leave this for the moment.

Report from running the tests:

```
Markus' method
% Time elapsed: 0.15960312 seconds.
n = 7
Memory used (MB):       12.5667

Dick's method 1
% Time elapsed: 0.19358516 seconds.
n = 7
Memory used (MB):       5.49789
Arrays match!

Dick's method 2
% Time elapsed: 0.34163213 seconds.
nc = 7
nr = 7
Memory used (MB):       12.5669
Arrays match!
```

Here is the code:

```
PRO MultiDimCombos

;  https://groups.google.com/forum/#!topic/comp.lang.idl-pvwave /FV6s1s19BJc

n = 7

; Markus

startMem = Memory(/CURRENT)
Tic

array=bindgen(n,long(n)^n)
for k=0,n-1 do $
  array[k,*]=rebin(byte((n*lindgen(long(n)^(k+1))+k) mod (n^2) ), $
                long(n)^n, /sample)

Print
Print, 'Markus'' method'
Toc
highMem = Memory(/HIGHWATER)
Print, 'n = '+StrTrim(n, 2)
Print, 'Memory used (MB): ', (highMem-startMem)/1024./1024
arrayM = array

IF n LE 3 THEN Print, array
```

```
; Dick 1

startMem = Memory(/CURRENT)
Tic

array=bytarr(n,long(n)^n, /NOZERO) ; Changed from bindgen to bytarr(/NOZERO)
for k=0,n-1 do $ ; Removed '*' from destination subscripts
  array[k,0]=Reform(rebin(byte((n*lindgen(long(n)^(k+1))+k) mod (n^2) ), $
                          long(n)^n, /sample), $
                    [1, long(n)^n], /OVERWRITE)

Print
Print, 'Dick''s method 1'
Toc
highMem = Memory(/HIGHWATER)
Print, 'n = '+StrTrim(n, 2)
Print, 'Memory used (MB): ', (highMem-startMem)/1024./1024
arrayD1 = array

IF n LE 3 THEN Print, array

Print, 'Arrays'+([' do not', ''])[Array_Equal(arrayD1, arrayM)]+' match!'

; Dick 2 -- method for unequal numbers of columns and rows

startMem = Memory(/CURRENT)
Tic

; To compare with Markus and Dick1:
nc = n
nr = n

; To test unequal nc and nr:
;nc = 4
;nr = 3

a = bindgen(nc, nr)   ; bindgen is OK to nr = 8
i = lindgen([1, Long(nr)^nc]) ; indgen is OK to nr = 8
array=bytarr(nc,long(nr)^nc, /NOZERO)
for k=0,nc-1 do $
  array[k,0]=a[k,i/(Long(nr)^(nc-1-k)) MOD nr]

Print
Print, 'Dick''s method 2'
Toc
highMem = Memory(/HIGHWATER)
Print, 'nc = '+StrTrim(nc, 2)
Print, 'nr = '+StrTrim(nr, 2)
```

```
Print, 'Memory used (MB): ', (highMem-startMem)/1024./1024
arrayD2 = array

IF nr * nc LE 12 THEN Print, array

IF nc EQ n and nr EQ n THEN $
  Print, 'Arrays'+([' do not', ''])[Array_Equal(arrayD2, arrayM)]+' match!'

END
```

Hope this helps!

Cheers,
-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada --- http://www.d-jackson.com

On Tuesday, 14 November 2017 05:16:47 UTC-8, clement...@obspm.fr wrote:

```
>  Hello again,
>
>  Thanks both of you for your replies.
>
>  @Mike: I had looked into this before (I think Jeremy Bailin has published a code similar to yours called combigen.pro), but I then meet difficulties in selecting part of the generated combinations.
>
>  @Markus: I say, your code is sleek and nifty. I like your solution.
>
>  In the meantime, I had given this problem some more thoughts and I had come up with another slow ugly one that doesn't work for all cases:
>
>  function gen_indices_comb, m, n
>  ;d I/O:
>  ;d m -> long integer corresponds to the number of row in original table
>  ;d n -> long integer corresponds to the number of columns in original table
>  ;d
>  ;d vals -> long array listing the vectors of indices to extract the
>  ;d         different possible combinations from the values of the original
>  ;d         table
>  ;d
>  ;d NOTES: SLOW CODE, a mitigation of the values of m and n is REQUIRED
>  ;d        Cases, where m & n are greater than 9, are not to considered
>  ;d        with this code
>
>   nmax = m^n
>
>  ;c Assemble command generating vector of indices
```

```
>   cmd = 'tmp = ['
>   for ijk=(m-1L),1L,-1L do $
>     cmd += ' (lmn/n^'+string(ijk,format='(I03)')+') mod n,'
>   cmd += 'lmn mod n ]'
>
>  ;c initialiase memory
>   ini = indgen(m,n)
>   tmp = lonarr(m)
>   val = lonarr(m, m^n)
>
>  ;c execute command for each type of combination
>   for lmn=0L,(n^m-1L) do begin
>     void = execute(cmd)
>     if void ne 1 then message, ' > Error generating indices.'
>     val[*,ijk] = ini+tmp*m
>   endfor
>
>   return, val
>  end
>
>  Afn I'm considering this post solved, I'll update it with a definitive version of my solution.
>
>  Again thanks your replies,
>  /C
```

Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by clement.feller@obspm. on Fri, 24 Nov 2017 15:06:23 GMT
View Forum Message <> Reply to Message

Hey Dick,

Thanks for your message, that's indeed a nice improvement on Markus' already sleek code.

I had put this question aside for a few days, and I've come up yesterday evening with what I think is a convenient solution that deals with sets of values of unequal lengths. Here's the script, it is a bit raw solution :/

```
function compute_combinations, vec, s_v
 ;=============================================================
====================
;d AUTHOR:
;d  clement <dot> feller <at> obspm <dot> fr
;d
;d PURPOSE: Building all possible combinations given different sets of values.
;d
```

```
;d CHANGELOG:
;d    23-NOV-2017 v1.0 first light
;d
;d I/O:
;d    vec  -> 1D-array concatenating the differents sets of values.
;d    s_v  -> 1D-array detailling the number of elements per set.
;d  result -> array with all possible combinations given the different sets of
;d               values.
;d
;d DEPENDANCIES: NONE
;d
;d REMARKS: NONE
;d
;d EXAMPLE:
;d IDL> u = [1,2,3]
;d IDL> v = [3.5,4.5]
;d IDL> w = [!pi/3., !pi/2, 2*!pi/3, !pi]
;d IDL> vec = [u,v,w] & s_v = [n_elements(u), n_elements(v), n_elements(w)]
;d IDL> print, compute_combinations(vec,s_v)
;d /* first set of combinations */
;d    1.00000|||   3.50000||    1.04720|
;d    1.00000|||   3.50000||    1.57080|
;d    1.00000|||   3.50000||    2.09440|
;d    1.00000|||   3.50000||    3.14159|
;d    1.00000|||   4.50000||    1.04720
;d    1.00000|||   4.50000||    1.57080
;d    1.00000|||   4.50000||    2.09440
;d    1.00000|||   4.50000||    3.14159
;d /* second set of combinations */
;d    2.00000|||   3.50000      1.04720
;d    2.00000|||   3.50000      1.57080
;d    2.00000|||   3.50000      2.09440
;d    2.00000|||   3.50000      3.14159
;d    2.00000|||   4.50000      1.04720
;d    2.00000|||   4.50000      1.57080
;d    2.00000|||   4.50000      2.09440
;d    2.00000|||   4.50000      3.14159
;d /* third and last set of combinations */
;d    3.00000|||   3.50000      1.04720
;d    3.00000|||   3.50000      1.57080
;d    3.00000|||   3.50000      2.09440
;d    3.00000|||   3.50000      3.14159
;d    3.00000|||   4.50000      1.04720
;d    3.00000|||   4.50000      1.57080
;d    3.00000|||   4.50000      2.09440
;d    3.00000|||   4.50000      3.14159
;d |   -> sequence of 4 elements repeated 3*2 times
;d ||  -> sequence of 2 elements duplicated 4 times and repeated 3 times
```

```
;d ||| -> sequence of 3 elements duplicated 2*4 times
 ;=============================================================
====================
;c Sparing a few cycles and allocating memory for result
 prd__s = product(s_v)
 v_type = size(vec, /type)
 n_cols = n_elements(s_v)
 result = make_array(n_cols, prd__s, type=v_type)

;c Dealing with the first column
 rplct = reform(rebin(indgen(s_v[0]),prd__s), prd__s)
 result[0,*] = (vec[0:s_v[0]])[rplct]

;c Dealing with the last column
 rplct = reform(rebin(indgen(s_v[-1]), reverse(s_v)), prd__s)
 result[-1,*] = (vec[total(s_v[0:-2]):total(s_v[0:*])-1L])[rplct]

;c Dealing with all columns in between
 for ijk=(n_cols-2L),1L,-1L do begin
   nb1 = product(s_v[ijk+1L:*])*s_v[ijk]
   nb2 = product(s_v[0:ijk-1L])
   rplct = reform(rebin(reform(rebin(indgen(s_v[ijk]), nb1),nb1), nb1,nb2), nb1*nb2)
   result[ijk,*] = (vec[total(s_v[0:ijk-1L]):total(s_v[0:ijk])-1L])[rplct]
 endfor

 return, result
end
```

I've tested it multiple times with up to 5 sets of values with different lengths and things seem to work out.
I also tried to see how much memory and time it would require to execute on a simple i5 CPU with 6 sets of indgen(6) and 7 sets of indgen(7) just for fun:

```
IDL> u = indgen(6) & v = indgen(6) & w = indgen(6) & x = indgen(6) & y = indgen(6) & z =
indgen(6)
IDL>  vec = [u,v,w,x,y,z] & s_v = [n_elements(u), n_elements(v), n_elements(w), n_elements(x),
n_elements(y), n_elements(z)]
IDL> mem = memory(/current) & t=systime(/sec) & mat = compute_combinations(vec,s_v) &
print,systime(/sec)-t, 's' & print, (memory(/highwater)-mem)/1024./1024, 'MB'
    0.0057270527s
      1.06842MB


IDL> u = indgen(7) & v = indgen(7) & w = indgen(7) & x = indgen(7) & y = indgen(7) & z =
indgen(7) & a =indgen(7)
IDL> vec = [u,v,w,x,y,z,a] & s_v = [n_elements(u), n_elements(v), n_elements(w), n_elements(x),
n_elements(y), n_elements(z), n_elements(a)]
IDL> mem = memory(/current) & t=systime(/sec) & mat = compute_combinations(vec,s_v) &
```

```
print,systime(/sec)-t, 's' & print, (memory(/highwater)-mem)/1024./1024, 'MB'
      0.074426889s
       20.4207MB
```

I really thank you all guys for your messages, and if you have some more advice or improvement about this script, I'll be glad to read it.

I'll try it some more and I'll put this script on GitHub over the week-end.

Thanks again,
Clement
