Subject: Re: Generating a grid in the 3D,4D,5D...N space - Advice/Combinatory/Matrices
Posted by lecacheux.alain on Fri, 24 Nov 2017 19:20:55 GMT
View Forum Message <> Reply to Message

Le vendredi 24 novembre 2017 16:06:25 UTC+1, clement...@obspm.fr a écrit :
> 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

Bonjour,

If I well understood your problem, I find that the result for your particular example can be obtained simply by:

result = reverse([w[lindgen(1,n) mod w.length], v[(lindgen(1,n)/w.length) mod v.length], u[lindgen(1,n)/w.length/v.length]])

(note that "reverse" and "(1,n)" tricks are only used for getting a display similar to yours).

In order to generalize to any number of element sets of any type (but compatible with available memory space !), I would suggest a function as follows:

```
function compute_combinations, elts
  compile_opt IDL2
  len = elts.map(lambda(x: x.length)) ;lengths of sublists
  n = len.reduce(lambda(x,y: y*x))    ;number of combinations
  result = list()
  foreach elt,elts,i do result.Add, elt[lindgen(n) mod len[i]]
  return, result.ToArray(/PROMOTE)    ;assume all elements are numbers
end
```

In input, I am using the IDL list: elts = list(u,v,w) (with your example).

Hoping that this can help you.

alain.
