
Subject: Re: Unique combinations from a 1d array
Posted by [dapoulio](#) on Thu, 15 Jan 2004 16:57:33 GMT
[View Forum Message](#) <> [Reply to Message](#)

"Christopher Lee" <cl@127.0.0.1> wrote in message
news:<20040115.100039.165344818.13691@buckley.atm.ox.ac.uk>...
> In article <MPG.1a6f72c6de3bcc529897a0@news.frii.com>, "David Fanning"
> <david@dfanning.com> wrote:
>
>
>> Darren writes:
>>
>>> Does anyone know of a more efficient means to determine the set of all
>>> unique combinations of 2 from a 1d array? The following is an approach
>>> that works but for large arrays -say 3000 or more elements it is very
>>> slow. Part of the problem is due to memory because the number of paired
>>> comparisons becomes very large ? i.e. for 3000 elements the total
>>> number of combinations is 4498500. Writing the paired difference
>>> results to a temporary file helped considerably, but is still far too
>>> slow. Any ideas would be much appreciated. Here is the code I have:
>>> X = [X1, X2, X3?...Xn+1]
>>> n = n_elements(X)
>>> d = make_array(1, /float)
>>> for i=0, n-1 do for j=0, n-1 do begin
>>> if i le j then begin
>>> d = [d, X[i] - X[j]]
>>> endif
>>> endfor
>>> d = d[1:n-1]
>
> Hi,
>
> I'm with David on what your code actually *does*. Especially since I'm
> not sure if the last line should be 1:n-1 or 1:* (since n_elements(d) >
> n) ? Your 3000 makes 449000 argument says 1:* .
>
> So, incrementally 'improving' your code.
>
> X = [X1,X2,X3,X4,...Xn+1]
> n=n_elements(X)
> d=make_array(type=size(x,/type), dimension=total(findgen(n)))
> c=0L
> for i=0, n-1 do for j=i+1, n-1 do begin
> d[c]=X[i]-X[j]
> c=c+1
> endfor
>
> ;timing results for an N element array are

```

>
> N      yours (s)      mine (s)
> 10     0.0033         0.0028
> 100    0.026          0.011
> 1000   (too long)     0.61
> 10000  *****       61.0
> etc.
>
> Of course, under a few thousand elements there are fun matrix
> methods, i.e
>
> n=n_elements(x)
> y=findgen(n)
> val=x#replicate(1,n) - x##replicate(1,n)
> mask=y#replicate(1,n) - y##replicate(1,n)
> ;upper diagonal of val contains the unique elements I think.
> return, val[where(y gt 0)]
>
> ;
> that one comes in at 0.099s for 1000 points, but there's a health warning
> attached to it, its a memory hog at  $\sim(3*N^2)$  instead of  $\sim(N^2)$ , which doesn't
> sound bad but it is :) I couldn't get results for the 10000 point case, but
> for 2000 (1.0s c.f 2.4s) and 4000 (1.5s c.f 9.4s) it is faster.
>
> Chris.

```

Thanks David and Chris, you're right the code I posted was incorrect and I apologize for it being misleading. The following is the code that I should have posted to demonstrate my problem:

```

X = [X1,X2,X3,X4,...Xn+1]
n = n_elements(X)
d = make_array(1, /float)
for i=0, n-1 do for j=0, n-1 do begin
  if i lt j then begin
    d = [d, X[i] - X[j]]
  endif
endfor
d = d[1:(n_elements(d)-1)]

```

Just to be clear in summarizing -for a 3 element array (i.e [0,1,2]) the total combinations are 3:

```

0 1
0 2
1 2

```

The total combinations can be found using:

$$n!/(n-p)!*p!$$

Where n is the total number of array elements and p is the size of the desired combination in my case p = 2.

Both of the code examples given by Chris do this much more efficiently than what I posted. For 3000 cases, the matrix approach came in at 0.614s and the loop approach at 2.414s on my 2.4 GHz Pentium. However, I believe there was a typo for the matrix approach on the last line which should read `return, val[where(mask gt 0)]` to give the array of paired differences.

Thanks again,

Darren
