Subject: indexing
Posted by Jeremy Bailin on Thu, 02 Apr 2009 01:12:54 GMT
View Forum Message <> Reply to Message

I swear I've seen this come up within the past few months, but I can't find it, so here goes:

Let's say I have a 3D array - think of it as x,y,z if you want. I have a list of x,y pairs and I want to perform on operation on a given range in z (say z1:z2) and each x,y pair. For a simple example, let's say I just want them all up. So, if the following were my x,y pairs:
3 4
2 1
3 7
and my z range was 1:3 then I want
array[3,4,1]+array[3,4,2]+array[3,4,3]
+array[2,1,1]+array[2,1,2]+array[2,1,3]
+array[3,7,1]+array[3,7,2]+array[3,7,3]

Is there a simple representation for this? My standard solution, if pair is an npair x 2 array containing the x,y pairs, looks something like this:
nz=z2-z1+1
zindices=rebin(reform(z1+lindgen(nz),1,nz), npair,nz)
xindices=rebin(pair[*,0],npair,nz)
yindices=rebin(pair[*,1],npair,nz)
answer = total(array[xindices,yindices,zindices])

...but if nz and npair are large, generating all of those 2D index arrays is really wasteful. The following also works:

answer = total(array[pair[*,0],pair[*,1],z1:z2] * rebin(identity (npair,npair,nz)))

but again generates 2 intermediate npair x npair x nz arrays that are wasteful if npair is large.

Any takers?

-Jeremy.

Subject: Re: indexing
Posted by Jeremy Bailin on Thu, 02 Apr 2009 19:20:24 GMT
View Forum Message <> Reply to Message

On Apr 2, 12:49 pm, "Dick Jackson" <d...@d-jackson.com> wrote:
> Hi all,

>
> Good one, Chris. How about this for getting the subset:
>
> PRO IndexXYPairsOverZ
>
> a = Transpose(IndGen(10,10,10), [2,1,0]) ; Array where, e.g. a[3,0,4]=304
>
> xy = [[3,4],[2,1],[3,7]]
> z0 = 1
> z1 = 3
>
> dims=Size(a,/Dim)
> a = Reform(a, dims[0]*dims[1], dims[2], /Overwrite) ; Reshape a temporarily
> xyIndices = xy[0,*]+xy[1,*]*dims[0]
> subset = a[xyIndices, z0:z1]
> a = Reform(a, dims, /Overwrite) ; Restore a's shape
>
> Help, subset
> Print, subset
>
> END
>
> Printed result:
>
> SUBSET INT = Array[3, 3]
> 341 211 371
> 342 212 372
> 343 213 373
> Note that the Reforms will take effectively no time at all, and you only
> make one list of indices.
>
> Hope this helps.
>
> Cheers,
> -Dick
>
> --
> Dick Jackson Software Consulting http://www.d-jackson.com
> Victoria, BC, Canada +1-250-220-6117 d...@d-jackson.com
>
> "Jeremy Bailin" <astroco...@gmail.com> wrote in message
>
> news:a50cae0f-63e5-47b7-a44e-b5363114855a@r37g2000yqn.google groups.com...
> On Apr 2, 1:34 am, Chris <beaum...@ifa.hawaii.edu> wrote:
>
>> I would have done something like this, accomplishing the summation in
>> two steps
>

>> x = [3, 2, 3]
>> y = [4, 1, 7]
>> sum = total(array[*,*,1:3], 3) ; summed along z direction
>> answer = total(sum[x,y]) ; summed over x,y pairs
>
>> chris
>
> Yeah, that definitely works nicely for total (I like the fact that it
> doesn't need any internal index vectors, and that the intermediate
> stage is a constant size independent of npair and nz), which is what
> I'm doing today. But it wouldn't work for the more generic case where
> I want to do something else to those values - for example, I've needed
> to do the equivalent with median before, and that won't work as a 2-
> step process.
>
> -Jeremy.

Ah, now this is exactly the sort of thing I was looking for! Excellent job, Dick! As with any elegant solution, now that I look at it, I'm kicking myself for not figuring it out earlier... but it's a nice trick to add to the IDL Way Arsenal. :-)=

-Jeremy.

---

Subject: Re: indexing
Posted by JDS on Wed, 08 Apr 2009 22:35:56 GMT
View Forum Message <> Reply to Message

On Apr 2, 3:20 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 2, 12:49 pm, "Dick Jackson" <d...@d-jackson.com> wrote:
>
>
>
>> Hi all,
>
>> Good one, Chris. How about this for getting the subset:
>
>> PRO IndexXYPairsOverZ
>
>> a = Transpose(IndGen(10,10,10), [2,1,0]) ; Array where, e.g. a[3,0,4]=304
>
>> xy = [[3,4],[2,1],[3,7]]
>> z0 = 1
>> z1 = 3
>
>> dims=Size(a,/Dim)
>> a = Reform(a, dims[0]*dims[1], dims[2], /Overwrite) ; Reshape a temporarily

```
>>  xyIndices = xy[0,*]+xy[1,*]*dims[0]
>>  subset = a[xyIndices, z0:z1]
>>  a = Reform(a, dims, /Overwrite)                    ; Restore a's shape
>
>>  Help, subset
>>  Print, subset
>
>>  END
>
```

This is an really cool way of doing it, but it's nothing that straightforward REBIN can't handle:

```
 nxy=dims[0]*dims[1] & nz=z1-z0+1
 t=[nz,n_elements(xy)/2]
 indices=rebin(xy[0,*]+dims[0]*xy[1,*],t) + rebin(nxy*(z0+lindgen
(nz)),t)
```

I regard the computation of an index array as a *benefit* not a liability here and in many cases. The reason? IDL happily computes its own array of indices for you behind the scenes when you use the higher-order indexing function, e.g. a[xyIndices, z0:z1]. Especially problematic are statements like a[*,*,1:3] (as above) which don't just make an index vector, but one which is much larger than needed, wasting time and memory. The advantage of computing the index vector yourself is that you can re-use it without throwing it away and computing it all over again (which is what IDL would do).

What was extremely interesting about this problem was the relative performance of these two methods for very large lists of xy indices and large arrays. For small to moderate lists of xy pairs, the REFORM method Dick presented was roughly 2x faster for me. However, as the size of the xy list got large, the REBIN method catches up and eventually overtakes the REFORM method. Over about 5 million xy pairs by 100 z planes, the REBIN method keeps getting faster compared to the IDL-native calculations of the indices. Probably a memory usage difference, or perhaps related to the use of the thread pool (dual proc system). Still, getting IDL to do all the index computation almost entirely internally, as Dick's method does, seems to be a real benefit at least for some problem sizes.

JD