Subject: Re: Center of mass???
Posted by J.D. Smith on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Jonathan Joseph wrote:
>
> Well, I'm not going to take up JD's challenge,
> but are you all sure you are answering the right
> question?
>
> I mean sure, great, if you happen to have
> an MxNx.... array of masses then you've got
> everything you need. But when I first read
> Anders' post, I thought, "gee that sounds simple."
>
> I thought of N masses at N locations,
>
> m = 1D array of N masses
> pos = D x N array of locations of the masses in D dimensions
>
> then:
>
> s=size(pos, /dimensions)
> mm = m ## replicate(1,s(0))
> cm = total(pos * mm, 2) / total(m)
>
> Please someone correct me if I'm wrong.
> Also, Is there a better way of multiplying
> an MxN array by a one dimensional array of
> length N such that each row of the MxN array
> is multiplied by the corresponding element
> of the one dimensional array?

You can use :

rebin(reform(m,1,N),D,N,/SAMP)*pos

but the array multiplication method also works. The relative speeds are system dependent.

As far as your method of C.O.M. calculation, it's clearly good when you have such a DxN array, or even a sparse array of almost all zeroes (which you can safely ignore in the calculation). However, the application I imagine is some plane or cube or hypercube of data for which the C.O.M. is required. In that case, to use your method, you'd have to inflate your data by a factor of D. That is, you're paying for all those repeated indices being multiplied *before* totalling the data. This is an Index first rather than total first method.

Here is a replacement routine using your idea which takes regular MxNx... arrays
of data and makes the DxN index array.

```
function com2, arr,DOUBLE=dbl
   s=size(arr,/DIMENSIONS)
   d=n_elements(s)
   n=n_elements(arr)
   inds=lonarr(d,n,/NOZERO)

   fac=1
   for i=0,d-1 do begin
      inds[i,*]=lindgen(n)/fac mod s[i]
      fac=fac*s[i]
   endfor

   return, total(inds*rebin(reform(arr,1,n),d,n,/SAMP),2,DOUBLE=dbl)/ $
    total(arr,DOUBLE=dbl)
end
```

You see the work here is in generating the index array and inflating the data
array. I compared this routine to my other one for a random array of size
10x10x10x10x10. The results were:

```
Index First Method Time: 0.45424998
Total First Method Time: 0.048227489
```

How about 1024x1024:

```
Index First Method Time: 1.9181580
Total First Method Time: 0.23625147
```

And for something really ludicrous... 5x5x5x5x5x5x5x5x5

```
Index First Method Time: 2.7887635
Total First Method Time: 0.44504005
```

So you see, even for many dimension, for which the Total First routine is
currently inefficient, it is always faster. And for big arrays, like
100x100x100x20, I couldn't even get the Index First method to run (memory
issues). Too much copying of data.

JD

--
 J.D. Smith |*| WORK: (607) 255-5842
 Cornell University Dept. of Astronomy |*| (607) 255-6263

```
304 Space Sciences Bldg.            |*|       FAX: (607) 255-5875
Ithaca, NY 14853                  |*|
```