Subject: Re: Adding x,y events to a 2d array (quickly)
Posted by Dick Jackson on Fri, 08 Nov 2013 08:22:22 GMT
View Forum Message <> Reply to Message

Michael Galloy wrote, On 2013-11-07, 3:18pm:
> On 11/7/13, 3:20 PM, Dick Jackson wrote:
>> Phillip Bitzer wrote, On 2013-11-07, 12:16pm:
>>> On Thursday, November 7, 2013 1:27:00 PM UTC-6, Dick Jackson wrote:
>>>
>>>> I seem to recall someone explaining this behaviour before, and thanks to
>>>>
>>>> Russell, I realize one good way of getting *part* of what you
>>>> (reasonably!) want
>>>>
>>>> to do. If all of your 'e' values were equal, then you can find how
>>>> many counts
>>>>
>>>> of each (x,y) pair exist by using Hist_ND:
>>>>
>>>> (http://tir.astro.utoledo.edu/idl/hist_nd.pro)
>>>>
>>>> IDL> Print, Hist_ND(Transpose([[1,1,2],[1,1,2]]), 1, Min=0)
>>>>
>>>> But, in general, to add a varying set of 'e' values to those (x,y)
>>>> locations...
>>>>
>>>> I have to think a bit...
>>>>
>>>
>>> I've got you covered....
>>>
>>> Oliver, reverse indices are your friend here, as Russell alluded to.
>>> Get the two-dimensional histogram, slightly modified from Dick's version:
>>>
>>> h = HIST_ND([TRANSPOSE(x), TRANSPOSE(y)], 1, MIN=0,
>>> REVERSE_INDICES=ri)
>>>
>>> Since you said you have large arrays, I transpose each individually,
>>> and then concatenate.
>>>
>>> Now, go through the reverse indices:
>>>
>>> totalE = FLTARR(SIZE(h, /DIM))
>>> FOR i=0, N_ELEMENTS(h)-1 do if h[i] GT 0 THEN totalE[i]= TOTAL(
>>> e[ri[ri[i]:ri[i+1]-1]])
>>>
>>> print, totalE
>>>   0.00000   0.00000   0.00000

```
>>>        0.00000      20.0000      0.00000
>>>        0.00000      0.00000      10.0000
>>>
>>>  This is the basic idea. It can be sped up by only looping over the
>>>  elements of h with non-zero counts (as opposed to "skipping" them as I
>>>  did here).
>>>
>>>  Here's some highly recommended reading on histograms:
>>>  http://www.idlcoyote.com/tips/histogram_tutorial.html
>>
>>  Histograms and reverse-indices are amazingly powerful and the right way
>>  to go in many tough problems, but I think Oliver is looking for a
>>  solution avoiding loops (I am too!). If a loop solution were OK, the
>>  last block here would be more direct, with no need for histograms:
>>
>>  x=[1,1,2]
>>  y=[1,1,2]
>>  e=[10,11,12]
>>
>>  counts=fltarr(3,3)
>>  counts(x,y)++
>>  Print, 'counts:'
>>  Print, counts    ; Shows that three increments by 1 were done
>>
>>  totalenergy=fltarr(3,3)
>>  totalenergy(x,y)+=e
>>  Print, 'totalenergy:'
>>  Print, totalenergy ; It appears that only two increments by 10 were done
>>
>>  totalenergy2=fltarr(3,3)
>>  FOR i=0, N_Elements(x)-1 DO totalenergy2(x[i],y[i])+=e[i]
>>  Print, 'totalenergy2:'
>>  Print, totalenergy2 ; All three increments were done
>>
>>  ... which gives us:
>>
>>  counts:
>>       0.000000     0.000000     0.000000
>>       0.000000      2.00000     0.000000
>>       0.000000     0.000000      1.00000
>>  totalenergy:
>>       0.000000     0.000000     0.000000
>>       0.000000      11.0000     0.000000
>>       0.000000     0.000000      12.0000
>>  totalenergy2:
>>       0.000000     0.000000     0.000000
>>       0.000000      21.0000     0.000000
>>       0.000000     0.000000      12.0000
```

```
>>
>>  Still looking for the "IDL way" (read: "ideal way") to do this...
>>
>
>  Phillip's method loops over the bins in the histogram, so should be reasonable.
>  My MG_HIST_ND does the same thing:
>
>  IDL> x = [1, 1, 2]
>  IDL> y = [1, 1, 2]
>  IDL> weights = [10., 11., 12.]
>  IDL>
>  IDL> h = mg_hist_nd([transpose(x), transpose(x)], weights=weights, min=0,
>  bin_size=1, unweighted=unweighted)
>  IDL> print, h
>        0.00000      0.00000      0.00000
>        0.00000      21.0000      0.00000
>        0.00000      0.00000      12.0000
>  IDL> print, unweighted
>            0          0          0
>            0          2          0
>            0          0          1
>
>  Get MG_HIST_ND on GitHub:
>
>      https://github.com/mgalloy/mglib/blob/master/src/analysis/mg _hist_nd.pro
>
>  Mike
```

Mike,

That's an amazing routine (thank you!), and the Weights option provides exactly
the functionality and result Oliver is looking for. However, in most of my test
cases it seems to be less efficient than the simple loop in time or space. (it's
good when the 2-D counts array is small and you don't mind using lots of memory)
Here's my test, with no idea if it covers similar scale to Oliver's application!

```
PRO IncrementTest

FOREACH size, [100, 1000, 10000] DO $ ; Width, height of (square) counts array
   FOREACH nPts, [1E6, 1E7, 4E7] DO BEGIN ; Number of points to create

   x = Long(RandomU(42L, nPts) * size)
   y = Long(RandomU(56L, nPts) * size)
   e = Long(RandomU(98L, nPts) * 10) + 1 ; Random from 1-10

   Print
   Print
   Help, size, nPts

   Print
   Print, 'MG_Hist_ND method'
   m0 = Memory(/Current)
   Tic
   countsMGhistND = mg_hist_nd([transpose(x), transpose(y)], weights=e, $
                          min=0, bin_size=1) ; , unweighted=unweighted)
   Toc
   Print, (Memory(/Highwater)-m0)/(1024.^2), ' MB used'

   Print
   Print, 'Loop method'
   m0 = Memory(/Current)
   Tic
   countsLoop = LonArr(size, size) ; FltArr(size, size)
   FOR i=0, N_Elements(x)-1 DO countsLoop(x[i],y[i]) += e[i]
   Toc
   Print, (Memory(/Highwater)-m0)/(1024.^2), ' MB used'

   Print
   Print, 'Results are ' + $
     (Array_Equal(countsLoop, countsMGhistND) ? '' : 'not ') + 'equal!'

ENDFOREACH

END
```

... and results, with "better" values labeled with "*****" :

```
IDL> incrementtest
% Compiled module: INCREMENTTEST.


SIZE          INT       =      100
NPTS           FLOAT     =  1.00000e+006

MG_Hist_ND method
% Time elapsed: 0.23399997 seconds. *****
      19.0744 MB used

Loop method
% Time elapsed: 0.35899997 seconds.
    0.0382977 MB used              *****

Results are equal!
```

```
SIZE           INT       =     100
NPTS           FLOAT     =  1.00000e+007

MG_Hist_ND method
% Time elapsed: 2.3920002 seconds. *****
      190.736 MB used

Loop method
% Time elapsed: 3.4849999 seconds.
    0.0382271 MB used            *****

Results are equal!


SIZE           INT       =     100
NPTS           FLOAT     =  4.00000e+007

MG_Hist_ND method
% Time elapsed: 10.017000 seconds. *****
      762.940 MB used

Loop method
% Time elapsed: 14.156000 seconds.
    0.0382271 MB used            *****

Results are equal!


SIZE           INT       =    1000
NPTS           FLOAT     =  1.00000e+006

MG_Hist_ND method
% Time elapsed: 1.3910000 seconds.
      26.7042 MB used

Loop method
% Time elapsed: 0.51600003 seconds. *****
      3.81478 MB used             *****

Results are equal!


SIZE           INT       =    1000
NPTS           FLOAT     =  1.00000e+007

MG_Hist_ND method
```

```
% Time elapsed: 5.6570001 seconds.
      190.736 MB used

Loop method
% Time elapsed: 4.1250000 seconds. *****
      3.81478 MB used            *****

Results are equal!


SIZE          INT       =    1000
NPTS          FLOAT     =  4.00000e+007

MG_Hist_ND method
% Time elapsed: 16.332000 seconds.
      762.940 MB used

Loop method
% Time elapsed: 16.274000 seconds. *****
      3.81478 MB used            *****

Results are equal!


SIZE          INT       =   10000
NPTS          FLOAT     =  1.00000e+006

MG_Hist_ND method
% Time elapsed: 18.353000 seconds.
      1159.67 MB used

Loop method
% Time elapsed: 1.5000000 seconds. *****
      381.470 MB used            *****

Results are equal!


SIZE          INT       =   10000
NPTS          FLOAT     =  1.00000e+007

MG_Hist_ND method
% Unable to allocate memory: to make array.
   Not enough space
```

Oliver, I hope this helps!

--

Cheers,
-Dick

Dick Jackson Software Consulting
Victoria, BC, Canada
www.d-jackson.com