Subject: Re: performing multiple histograms without loops
Posted by MC on Tue, 02 Feb 2010 09:48:18 GMT
View Forum Message <> Reply to Message

On Feb 2, 6:25 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> I thought it would be worth expanding on the technique that I used in
> response
> to Ed's question, because it's a very useful one. The basic idea is
> this:
> suppose I want to use HISTOGRAM not on one a single set of N data
> points, but
> independently on multiple (say M) sets each of N data points. The
> simple solution
> is to use a for loop, but if M is large the IDL loop penalty soon
> becomes a problem. Is it possible to avoid a loop?
>
> The answer is yes, and the trick is to modify the data in each of the
> M data
> sets so that they don't overlap, and then use a single HISTOGRAM on
> all
> of them at once.
>
> For a concrete example, let's say that we have 5 data sets, each with
> 10
> data points that are small integers (for convenience - if your data
> doesn't
> look like this but contains individual values, then you can use a
> combination
> of UNIQ and VALUE_LOCATE to turn it into this form, or just
> VALUE_LOCATE
> if you need to bin different values together).
>
> set1 = [4,1,2,3,1,3,2,2,1,1]
> set2 = [2,3,1,3,4,2,2,0,0,4]
> set3 = [2,2,0,3,4,1,2,3,1,1]
> set4 = [1,4,2,4,1,4,2,4,3,3]
> set5 = [0,4,1,2,1,4,2,2,3,4]
> set6 = [3,4,1,0,0,1,1,0,2,1]
> datasets = [[set1],[set2],[set3],[set4],[set5],[set6]]
>
> datasets is now a 10x6 array with all of the data:
>
> IDL> print, datasets
> 4 1 2 3 1 3 2 2
> 1 1
> 2 3 1 3 4 2 2 0
> 0 4
> 2 2 0 3 4 1 2 3

> 1 1
> 1 4 2 4 1 4 2 4
> 3 3
> 0 4 1 2 1 4 2 2
> 3 4
> 3 4 1 0 0 1 1 0
> 2 1
>
> If we want to print out the histogram of each one, the traditional way
> would be
> to put a HISTOGRAM command inside a for loop:
>
> datasetsize=size(datasets,/dimen)
> ndatasets=datasetsize[1]
> minval=min(datasets)
> maxval=max(datasets)
> for i=0,ndatasets-1 do print, histogram(datasets[*,i], min=minval,
> max=maxval)
> 0 4 3 2 1
> 2 1 3 2 2
> 1 3 3 2 1
> 0 2 2 2 4
> 1 2 3 1 3
> 3 4 1 1 1
>
> But we can do it within a single histogram by adding 5 to all values
> in set2
> so that it runs from 5 to 9, 10 to all the values in set3 so that it
> runs
> from 10 to 14, etc.
>
> datasets_new = datasets - minval
> dataspan = maxval-minval+1
> datasets_new += rebin(transpose(lindgen(ndatasets)*dataspan), size
> (datasets,/dimen))
>
> IDL> print, datasets_new
> 4 1 2 3 1
> 3
> 2 2 1 1
> 7 8 6 8 9
> 7
> 7 5 5 9
> 12 12 10 13 14
> 11
> 12 13 11 11
> 16 19 17 19 16
> 19

> 17 19 18 18
> 20 24 21 22 21
> 24
> 22 22 23 24
> 28 29 26 25 25
> 26
> 26 25 27 26
>
> Now, if we perform a histogram of datasets_new, the 1s from set1 don't
> interfere
> with the 1s from set2 (which are now 6s), or the 1s from set3 (which
> are
> now 11s), etc. A single histogram will effectively perform a histogram
> of each set independently:
>
> h_new = histogram(datasets_new, min=0, max=ndatasets*dataspan-1,
> bin=1)
>
> But the 6 histograms are all jammed up against each other inside h!
> How
> do we get them out?
>
> h_new = reform(h_new, dataspan, ndatasets)
>
> IDL> print, h
> 0 4 3 2 1
> 2 1 3 2 2
> 1 3 3 2 1
> 0 2 2 2 4
> 1 2 3 1 3
> 3 4 1 1 1
>
> We can compare to the loop version and see that it does indeed give
> the
> right answer.
>
> "Alright," you say, "but the main reason I use HISTOGRAM is because of
> REVERSE_INDICES. How do I get those out?"
>
> If you want the reverse indices for data set j, they can be easily
> extracted from the reverse indices of the full histogram. If we
> create the histogram as:
>
> h_new = reform(histogram(datasets_new, min=0,
> max=ndatasets*dataspan-1, bin=1, $
> reverse_indices=ri), dataspan, ndatasets)
>
> then the i-vector (in JD's terminology) for data set j runs from

> ri[j*dataspan] to ri[(j+1)*dataspan-1]. These can be used directly
> to index the o-vector. To get the original element in data set j
> from the value in the o-vector, subtract j*datasetsize[0]. For
> example,
> where are the 3's in datasets[*,1]?
>
> IDL> print, ri[ri[1*dataspan+3]:ri[1*dataspan+3+1]-1] - 1*datasetsize
> [0]
> 1 3
>
> A more interesting question is "where are the 3's in all of the
> datasets"?
> This can in fact be done without loops! First, let's look at the loop
> version:
>
> for i=0,ndatasets-1 do begin
> h = histogram(datasets[*,i], min=minval, max=maxval,
> reverse_indices=ri1)
> if h[3] gt 0 then print, ri1[ri1[3]:ri1[3+1]-1]
> endfor
> 3 5
> 1 3
> 3 7
> 8 9
> 8
> 0
>
> Using a combination of chunk indexing and "chunk index generation"
> (i.e. the solution to Wox's problem of a few weeks ago):
>
> n = h_new[lindgen(ndatasets)*dataspan+3]
> h2=histogram(total(n>0,/CUMULATIVE,/int)-1,/
> BINSIZE,MIN=0,REVERSE_INDICES=ri2)
> nh=n_elements(h2)
> chunkind=ri2[0:nh-1]-ri2[0]
> l1=((l=lindgen(((nm=ndatasets>(max(n))))),nm) mod nm))[where((l lt $
> (rebin(transpose(n),nm,nm,/sample))))]
> where3 = [[chunkind],[ri[ri[chunkind*dataspan+3]+l1] mod datasetsize
> [0]]]
>
> where3 is now a N3 x 2 array containing the data set and index within
> that
> data set of every value of 3 (of which there are N3=10 in this
> example):
>
> IDL> print, transpose(where3)
> 0 3
> 0 5

> 1 1
> 1 3
> 2 3
> 2 7
> 3 8
> 3 9
> 4 8
> 5 0
>
> -----
> One caveat with this method is that it can be quite wasteful of
> memory.
> The full histogram contains dataspan x ndatasets entries, but really
> only
> ndatasets^2 of them can be non-zero. If dataspan is much larger than
> ndatasets,
> as might be the case if the values from each data set don't appear in
> the
> other data sets, then you might run into memory problems pretty
> quickly.
> You can try to work around this a bit by using UNIQ to compress the
> values
> that get fed into the histogram, but it makes it much more complicated
> to extract
> the original information back out.
>
> -Jeremy.

Sorry but it not at all clear to me that this is a good idea, you have to search all the datasets to make sure you get no overlaps and determine the offsets (which have to be added) and may also run into integer problems for large integer data sets. Comments?

Cheers

Subject: Re: performing multiple histograms without loops
Posted by Wout De Nolf on Tue, 02 Feb 2010 10:11:48 GMT
View Forum Message <> Reply to Message

On Mon, 1 Feb 2010 21:25:32 -0800 (PST), Jeremy Bailin
<astroconst@gmail.com> wrote:

> I thought it would be worth expanding on the technique that I used in
> response
> to Ed's question, because it's a very useful one.

Nothing like a nice vectorized loop in the morning!

Did you compare the speed of your approach to the loop-approach? For reasonably sized datasets I expect the loop to be faster. This was also true for the chunk index generation you mentioned.

Thanks for sharing,

Wox

Subject: Re: performing multiple histograms without loops
Posted by Jeremy Bailin on Wed, 03 Feb 2010 13:43:13 GMT
View Forum Message <> Reply to Message

On Feb 2, 4:48 am, MC <morefl...@gmail.com> wrote:
> On Feb 2, 6:25 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>
>
>> I thought it would be worth expanding on the technique that I used in
>> response
>> to Ed's question, because it's a very useful one. The basic idea is
>> this:
>> suppose I want to use HISTOGRAM not on one a single set of N data
>> points, but
>> independently on multiple (say M) sets each of N data points. The
>> simple solution
>> is to use a for loop, but if M is large the IDL loop penalty soon
>> becomes a problem. Is it possible to avoid a loop?
>
>> The answer is yes, and the trick is to modify the data in each of the
>> M data
>> sets so that they don't overlap, and then use a single HISTOGRAM on
>> all
>> of them at once.
>
>> For a concrete example, let's say that we have 5 data sets, each with
>> 10
>> data points that are small integers (for convenience - if your data
>> doesn't
>> look like this but contains individual values, then you can use a
>> combination
>> of UNIQ and VALUE_LOCATE to turn it into this form, or just
>> VALUE_LOCATE
>> if you need to bin different values together).
>

>> set1 = [4,1,2,3,1,3,2,2,1,1]
>> set2 = [2,3,1,3,4,2,2,0,0,4]
>> set3 = [2,2,0,3,4,1,2,3,1,1]
>> set4 = [1,4,2,4,1,4,2,4,3,3]
>> set5 = [0,4,1,2,1,4,2,2,3,4]
>> set6 = [3,4,1,0,0,1,1,0,2,1]
>> datasets = [[set1],[set2],[set3],[set4],[set5],[set6]]
>
>> datasets is now a 10x6 array with all of the data:
>
>> IDL> print, datasets
>> 4 1 2 3 1 3 2 2
>> 1 1
>> 2 3 1 3 4 2 2 0
>> 0 4
>> 2 2 0 3 4 1 2 3
>> 1 1
>> 1 4 2 4 1 4 2 4
>> 3 3
>> 0 4 1 2 1 4 2 2
>> 3 4
>> 3 4 1 0 0 1 1 0
>> 2 1
>
>> If we want to print out the histogram of each one, the traditional way
>> would be
>> to put a HISTOGRAM command inside a for loop:
>
>> datasetsize=size(datasets,/dimen)
>> ndatasets=datasetsize[1]
>> minval=min(datasets)
>> maxval=max(datasets)
>> for i=0,ndatasets-1 do print, histogram(datasets[*,i], min=minval,
>> max=maxval)
>> 0 4 3 2 1
>> 2 1 3 2 2
>> 1 3 3 2 1
>> 0 2 2 2 4
>> 1 2 3 1 3
>> 3 4 1 1 1
>
>> But we can do it within a single histogram by adding 5 to all values
>> in set2
>> so that it runs from 5 to 9, 10 to all the values in set3 so that it
>> runs
>> from 10 to 14, etc.
>
>> datasets_new = datasets - minval

>> dataspan = maxval-minval+1
>> datasets_new += rebin(transpose(lindgen(ndatasets)*dataspan), size
>> (datasets,/dimen))
>
>> IDL> print, datasets_new
>> 4 1 2 3 1
>> 3
>> 2 2 1 1
>> 7 8 6 8 9
>> 7
>> 7 5 5 9
>> 12 12 10 13 14
>> 11
>> 12 13 11 11
>> 16 19 17 19 16
>> 19
>> 17 19 18 18
>> 20 24 21 22 21
>> 24
>> 22 22 23 24
>> 28 29 26 25 25
>> 26
>> 26 25 27 26
>
>> Now, if we perform a histogram of datasets_new, the 1s from set1 don't
>> interfere
>> with the 1s from set2 (which are now 6s), or the 1s from set3 (which
>> are
>> now 11s), etc. A single histogram will effectively perform a histogram
>> of each set independently:
>
>> h_new = histogram(datasets_new, min=0, max=ndatasets*dataspan-1,
>> bin=1)
>
>> But the 6 histograms are all jammed up against each other inside h!
>> How
>> do we get them out?
>
>> h_new = reform(h_new, dataspan, ndatasets)
>
>> IDL> print, h
>> 0 4 3 2 1
>> 2 1 3 2 2
>> 1 3 3 2 1
>> 0 2 2 2 4
>> 1 2 3 1 3
>> 3 4 1 1 1
>

>> We can compare to the loop version and see that it does indeed give
>> the
>> right answer.
>
>> "Alright," you say, "but the main reason I use HISTOGRAM is because of
>> REVERSE_INDICES. How do I get those out?"
>
>> If you want the reverse indices for data set j, they can be easily
>> extracted from the reverse indices of the full histogram. If we
>> create the histogram as:
>
>> h_new = reform(histogram(datasets_new, min=0,
>> max=ndatasets*dataspan-1, bin=1, $
>> reverse_indices=ri), dataspan, ndatasets)
>
>> then the i-vector (in JD's terminology) for data set j runs from
>> ri[j*dataspan] to ri[(j+1)*dataspan-1]. These can be used directly
>> to index the o-vector. To get the original element in data set j
>> from the value in the o-vector, subtract j*datasetsize[0]. For
>> example,
>> where are the 3's in datasets[*,1]?
>
>> IDL> print, ri[ri[1*dataspan+3]:ri[1*dataspan+3+1]-1] - 1*datasetsize
>> [0]
>> 1 3
>
>> A more interesting question is "where are the 3's in all of the
>> datasets"?
>> This can in fact be done without loops! First, let's look at the loop
>> version:
>
>> for i=0,ndatasets-1 do begin
>> h = histogram(datasets[*,i], min=minval, max=maxval,
>> reverse_indices=ri1)
>> if h[3] gt 0 then print, ri1[ri1[3]:ri1[3+1]-1]
>> endfor
>> 3 5
>> 1 3
>> 3 7
>> 8 9
>> 8
>> 0
>
>> Using a combination of chunk indexing and "chunk index generation"
>> (i.e. the solution to Wox's problem of a few weeks ago):
>
>> n = h_new[lindgen(ndatasets)*dataspan+3]
>> h2=histogram(total(n>0,/CUMULATIVE,/int)-1,/

>> BINSIZE,MIN=0,REVERSE_INDICES=ri2)
>> nh=n_elements(h2)
>> chunkind=ri2[0:nh-1]-ri2[0]
>> l1=((l=lindgen(((nm=ndatasets>(max(n)))),nm) mod nm))[where((l lt $
>> (rebin(transpose(n),nm,nm,/sample))))]
>> where3 = [[chunkind],[ri[ri[chunkind*dataspan+3]+l1] mod datasetsize
>> [0]]]
>
>> where3 is now a N3 x 2 array containing the data set and index within
>> that
>> data set of every value of 3 (of which there are N3=10 in this
>> example):
>
>> IDL> print, transpose(where3)
>> 0 3
>> 0 5
>> 1 1
>> 1 3
>> 2 3
>> 2 7
>> 3 8
>> 3 9
>> 4 8
>> 5 0
>
>> -----
>> One caveat with this method is that it can be quite wasteful of
>> memory.
>> The full histogram contains dataspan x ndatasets entries, but really
>> only
>> ndatasets^2 of them can be non-zero. If dataspan is much larger than
>> ndatasets,
>> as might be the case if the values from each data set don't appear in
>> the
>> other data sets, then you might run into memory problems pretty
>> quickly.
>> You can try to work around this a bit by using UNIQ to compress the
>> values
>> that get fed into the histogram, but it makes it much more complicated
>> to extract
>> the original information back out.
>
>> -Jeremy.
>
> Sorry but it not at all clear to me that this is a good idea, you have
> to search all the datasets to make sure you get no overlaps and
> determine the offsets (which have to be added) and may also run into
> integer problems for large integer data sets. Comments?

>
> Cheers

I'm not sure what you mean about the first point - do you mean that you think the time it takes to calculate the dataspan and offsets will offset the loop saving? I've run some timing tests in response to Wox's question that show that it wins under lots of reasonable conditions.

It's definitely true that you'll need to watch out for overflowing your variable type - I've switched everything to ulong64s, which should be sufficient for most cases, and it doesn't slow things down.

-Jeremy.

Subject: Re: performing multiple histograms without loops
Posted by Jeremy Bailin on Wed, 03 Feb 2010 14:12:47 GMT
View Forum Message <> Reply to Message

> Did you compare the speed of your approach to the loop-approach? For
> reasonably sized datasets I expect the loop to be faster. This was
> also true for the chunk index generation you mentioned.

Here's some test code (just does the histogram, since the reverse_indices come out in different forms and it will depend what exactly you want to do with them). The scaling depends on ndatasets, datalen, and datarange. I had Ed's problem in mind, where he said each data set corresponded to a pixel in a large image, so there could be millions of data sets, and datalen seemed low since he was willing to append them all by hand. First, the punch line:

ndatasets	datalen	datarange	Loop-speed	Vectorized-speed
10	10	4	5.8e-5	3.0e-5
10000	10	4	0.014	0.0031
10000000	10	4	12	3.5
10	10000	4	0.0010	0.0029
10	10000000	4	0.81	2.8
1000	1000	4	0.0044	0.031
10	10	100	5.4e-5	4.2e-5
10000	10	100	0.014	0.0086
10000000	10	100	11.5	9.1
10	10000	100	0.0011	0.0030
10	10000000	100	0.80	2.8
10000	10	10000	0.39	0.40
10	10000	10000	0.0016	0.0035

So, if datarange is low and ndatasets is larger than datalen, the vectorized version wins. If datalen is larger than ndatasets, the loop wins no matter what. If datarange is large, the loop usually wins or comes close (but note that datarange never needs to be larger than the product of ndatasets and datalen - if it is, use uniq+value_locate to remap the values into a smaller range). The best approach definitely depends on your problem!

Here's the code:

```
ndatasets=10l
datalen=10l
datarange=4l

datasets = round(randomu(seed,datalen,ndatasets)*datarange)
; needed for both approaches, so leave it out of timing
minval=min(datasets, max=maxval)

; loop version
t0=systime(/sec)
for i=0l,ndatasets-1 do h=histogram(datasets[*,i], min=minval,
max=maxval)
t1=systime(/sec)
print, 'Loop approach: ',t1-t0

; vectorized version
t0=systime(/sec)
dataspan=maxval-minval
datasets -= minval
h = reform(histogram(datasets + rebin(transpose(ul64indgen(ndatasets)
*dataspan), $
  size(datasets,/dimen)), min=0, max=ulong64(ndatasets)*dataspan-1),
dataspan, $
  ndatasets)
t1=systime(/sec)
print, 'Vectorized approach: ',t1-t0
```

-Jeremy.

Subject: Re: performing multiple histograms without loops
Posted by JDS on Wed, 03 Feb 2010 21:47:48 GMT
View Forum Message <> Reply to Message

On Feb 2, 12:25 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> I thought it would be worth expanding on the technique that I used in

> response
> to Ed's question, because it's a very useful one. The basic idea is
> this:
> suppose I want to use HISTOGRAM not on one a single set of N data
> points, but
> independently on multiple (say M) sets each of N data points. The
> simple solution
> is to use a for loop, but if M is large the IDL loop penalty soon
> becomes a problem. Is it possible to avoid a loop?
>
> The answer is yes, and the trick is to modify the data in each of the
> M data
> sets so that they don't overlap, and then use a single HISTOGRAM on
> all
> of them at once.

It's a good method... in fact, you've essentially rediscovered the algorithm employed by HIST_ND (which itself elaborated on IDL's own HIST_2D). The M sets of N data points constitute a second dimension (which don't really need to correspond to physical dimensions of any sort... they could just be "M different sets numbered 0..M-1"). If you follow through the dimensional analogy, you can also more easily extract the "sub-histogram" by dumping the raw output into an appropriately dimensioned array.

JD

Subject: Re: performing multiple histograms without loops
Posted by rogass on Thu, 04 Feb 2010 12:05:35 GMT
View Forum Message <> Reply to Message

Hi,
maybe you can speed up your vectorized approach by avoding the transpose step and by using the SAMPLE keyword within rebin to: rebin(ul64lindgen(1,ndatasets)*dataspan,size.... ,/SAMPLE) For large vectors (replicate({ : })).(0) may work sometimes faster.

Greets

CR

Subject: Re: performing multiple histograms without loops
Posted by Jeremy Bailin on Thu, 04 Feb 2010 19:21:59 GMT
View Forum Message <> Reply to Message

On Feb 4, 7:05 am, chris <rog...@googlemail.com> wrote:
> Hi,
> maybe you can speed up your vectorized approach by avoding the
> transpose step and by using the SAMPLE keyword within rebin to:
> rebin(ul64lindgen(1,ndatasets)*dataspan,size.... ,/SAMPLE) For large
> vectors (replicate({ : })).(0) may work sometimes faster.
>
> Greets
>
> CR

Indeed! Changing the rebin makes a huge difference... for example,
with ndatasets=10000000, datalen=10, datarange=100, I get a factor of
3 improvement.

10000000 10 100 11.5 9.1

turns into

10000000 10 100 11.6 3.2

I didn't try the full grid of parameters, but if that's consistent
across the board, that brings the vectorized approach up to at least
as good as the loop approach over almost the entire grid, and quite a
bit better over most of it.

-Jeremy.
