Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by russell.grew on Tue, 29 Jul 2008 00:43:25 GMT
View Forum Message <> Reply to Message

I think we need more information.

The range that the elements can take would be useful. If, for instance
they are integers between 0 and 10 you could use the where command in
conjunction with the count option to check how many times a given
value occurs.

Hope this helps.

Russell.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Brian Larsen on Tue, 29 Jul 2008 06:32:21 GMT
View Forum Message <> Reply to Message

We do need some more information but this is just screaming for
histogram. Have a read through http://www.dfanning.com/tips/histogram_tutorial.html
. Using histogram to see which x's are common you can step through
the reverse_indices and see which y's are then common. There is
probably a more graceful way however.

Cheers,

Brian

-- --------------
Brian Larsen
Boston University
Center for Space Physics
http://people.bu.edu/balarsen/Home/IDL

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Jeremy Bailin on Tue, 29 Jul 2008 15:50:02 GMT
View Forum Message <> Reply to Message

On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
> We do need some more information but this is just screaming for

> histogram. Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
> . Using histogram to see which x's are common you can step through
> the reverse_indices and see which y's are then common. There is
> probably a more graceful way however.
>
> Cheers,
>
> Brian
>
> -- --------------
> Brian Larsen
> Boston University
> Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL

In particular, if you're dealing with integers that don't span too big
a range, use HIST_2D and find the maximum element. If you've got
floats or a wide range, use UNIQ to turn each into an integer on a
small range first.

-Jeremy.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Juggernaut on Wed, 30 Jul 2008 11:54:00 GMT
View Forum Message <> Reply to Message

On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>> We do need some more information but this is just screaming for
>> histogram. Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>> . Using histogram to see which x's are common you can step through
>> the reverse_indices and see which y's are then common. There is
>> probably a more graceful way however.
>
>> Cheers,
>
>> Brian
>
>> -- --------------
>> Brian Larsen
>> Boston University
>> Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
> In particular, if you're dealing with integers that don't span too big
> a range, use HIST_2D and find the maximum element. If you've got
> floats or a wide range, use UNIQ to turn each into an integer on a

> small range first.
>
> -Jeremy.

I think if I were to be working with small datasets....ie not in the
millions of points I would use something like this

```
coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
[3,1],[21,14]]

counter = intarr(9)

FOR i = 0, 8 DO BEGIN
  FOR j = 0, 8 DO BEGIN

    IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++

  ENDFOR
ENDFOR

;- Histogram to find the max bins (no need to measure anything below 2
;- because that would just be a single hit and if all of your pairs
;- only occur once then who cares, right?
hist = histogram(counter, min=2, reverse_indices=ri)
maxHist = max(hist, mxpos)
IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'

;- Use the reverse indices given by histogram to find out exactly
;- where in your counter these maxes are occurring
array_index = (counter[ri[ri[1]:ri[2]-1]])[0]

;- Find where counter is equal to the array index determined by
;- reverse indices
max_index = where(counter EQ array_index)

;- Voila with your max pair
print, coords[*,max_index[0]]
```

Which spits out....

```
20      32
```

This could be tweaked to find the top two or three or whatever as
well.
Hope this helps.

Subject: Re: Finding the Top Two Most Common Coordinates in a

Multi-Dimensional Array
Posted by Juggernaut on Wed, 30 Jul 2008 12:35:43 GMT
View Forum Message <> Reply to Message

On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
> On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>> We do need some more information but this is just screaming for
>>> histogram. Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>> . Using histogram to see which x's are common you can step through
>>> the reverse_indices and see which y's are then common. There is
>>> probably a more graceful way however.
>
>>> Cheers,
>
>>> Brian
>
>>> -- --------------
>>> Brian Larsen
>>> Boston University
>>> Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>> In particular, if you're dealing with integers that don't span too big
>> a range, use HIST_2D and find the maximum element. If you've got
>> floats or a wide range, use UNIQ to turn each into an integer on a
>> small range first.
>
>> -Jeremy.
>
> I think if I were to be working with small datasets....ie not in the
> millions of points I would use something like this
>
```
> coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
> [3,1],[21,14]]
>
> counter = intarr(9)
>
> FOR i = 0, 8 DO BEGIN
>   FOR j = 0, 8 DO BEGIN
>
>     IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>   ENDFOR
> ENDFOR
>
```

> ;- Histogram to find the max bins (no need to measure anything below 2
> ;- because that would just be a single hit and if all of your pairs
> ;- only occur once then who cares, right?
> hist = histogram(counter, min=2, reverse_indices=ri)
> maxHist = max(hist, mxpos)
> IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
> ;- Use the reverse indices given by histogram to find out exactly
> ;- where in your counter these maxes are occurring
> array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
> ;- Find where counter is equal to the array index determined by
> ;- reverse indices
> max_index = where(counter EQ array_index)
>
> ;- Voila with your max pair
> print, coords[*,max_index[0]]
>
> Which spits out....
> 20   32
>
> This could be tweaked to find the top two or three or whatever as
> well.
> Hope this helps.

By the way....you shouldn't hard code these things as you can see
that I've confused everyone by saying there are 9 pairs but there are
actually 10. So replace those hard coded with n_elements(coords[0,*])
or size(coords, /dimensions) to get the correct loop numbers and array
sizes. Still works though.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Jeremy Bailin on Thu, 31 Jul 2008 11:37:16 GMT

View Forum Message <> Reply to Message

On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
> On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>> We do need some more information but this is just screaming for
>>> histogram. Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>> . Using histogram to see which x's are common you can step through
>>> the reverse_indices and see which y's are then common. There is

>>> probably a more graceful way however.
>
>>> Cheers,
>
>>> Brian
>
>>> -- --------------
>>> Brian Larsen
>>> Boston University
>>> Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>> In particular, if you're dealing with integers that don't span too big
>> a range, use HIST_2D and find the maximum element. If you've got
>> floats or a wide range, use UNIQ to turn each into an integer on a
>> small range first.
>
>> -Jeremy.
>
> I think if I were to be working with small datasets....ie not in the
> millions of points I would use something like this
>

```
> coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
> [3,1],[21,14]]
>
> counter = intarr(9)
>
> FOR i = 0, 8 DO BEGIN
>   FOR j = 0, 8 DO BEGIN
>
>     IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>   ENDFOR
> ENDFOR
>
> ;- Histogram to find the max bins (no need to measure anything below 2
> ;- because that would just be a single hit and if all of your pairs
> ;- only occur once then who cares, right?
> hist = histogram(counter, min=2, reverse_indices=ri)
> maxHist = max(hist, mxpos)
> IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
> ;- Use the reverse indices given by histogram to find out exactly
> ;- where in your counter these maxes are occurring
> array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
> ;- Find where counter is equal to the array index determined by
> ;- reverse indices
> max_index = where(counter EQ array_index)
```

>
> ;- Voila with your max pair
> print, coords[*,max_index[0]]
>
> Which spits out....
> 20   32
>
> This could be tweaked to find the top two or three or whatever as
> well.
> Hope this helps.

My version of that would be:

```
min1=min(coords[0,*], max=max1)
min2=min(coords[1,*], max=max2)
arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
bin1=1, min2=min2, max2=max2, bin2=1)
maxval = max(arraymap, maxelement)
print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
[min1,min2]
```

...which avoids loops, and is more obvious to me.

-Jeremy.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Juggernaut on Thu, 31 Jul 2008 14:50:05 GMT
View Forum Message <> Reply to Message

On Jul 31, 7:37 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
>
>
>
>> On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>> On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>>> We do need some more information but this is just screaming for
>>>> histogram. Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>>> . Using histogram to see which x's are common you can step through
>>>> the reverse_indices and see which y's are then common. There is
>>>> probably a more graceful way however.
>
>>>> Cheers,
>

```
>>>> Brian
>
>>>> ------------------------------------------------------------ --------------
>>>> Brian Larsen
>>>> Boston University
>>>> Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>>> In particular, if you're dealing with integers that don't span too big
>>> a range, use HIST_2D and find the maximum element. If you've got
>>> floats or a wide range, use UNIQ to turn each into an integer on a
>>> small range first.
>
>>> -Jeremy.
>
>> I think if I were to be working with small datasets....ie not in the
>> millions of points I would use something like this
>
>> coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
>> [3,1],[21,14]]
>
>> counter = intarr(9)
>
>> FOR i = 0, 8 DO BEGIN
>>   FOR j = 0, 8 DO BEGIN
>
>>     IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>>   ENDFOR
>> ENDFOR
>
>> ;- Histogram to find the max bins (no need to measure anything below 2
>> ;- because that would just be a single hit and if all of your pairs
>> ;- only occur once then who cares, right?
>> hist = histogram(counter, min=2, reverse_indices=ri)
>> maxHist = max(hist, mxpos)
>> IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
>> ;- Use the reverse indices given by histogram to find out exactly
>> ;- where in your counter these maxes are occurring
>> array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
>> ;- Find where counter is equal to the array index determined by
>> ;- reverse indices
>> max_index = where(counter EQ array_index)
>
>> ;- Voila with your max pair
>> print, coords[*,max_index[0]]
>
```

```
>> Which spits out....
>> 20      32
>
>> This could be tweaked to find the top two or three or whatever as
>> well.
>> Hope this helps.
>
> My version of that would be:
>
> min1=min(coords[0,*], max=max1)
> min2=min(coords[1,*], max=max2)
> arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
> bin1=1, min2=min2, max2=max2, bin2=1)
> maxval = max(arraymap, maxelement)
> print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
> [min1,min2]
>
> ...which avoids loops, and is more obvious to me.
>
> -Jeremy.
```

No loops is all and good...but if you put a decimal in coords like
this

```
coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
[20.0,32.3],[10,10],[3,1],[21,14]]
```

your code still spits out (20.0 32.0) where it should spit out (20.0
32.3)
By the way the code I presented up there should have the following
line replaced

```
 array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
```

with

```
 array_index = (counter[ri[ri[mxpos]:ri[mxpos+1]-1]])[0]
```

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Jeremy Bailin on Fri, 01 Aug 2008 11:02:09 GMT

View Forum Message <> Reply to Message

On Jul 31, 10:50 am, Bennett <juggernau...@gmail.com> wrote:
> On Jul 31, 7:37 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
>

```
>>>  On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>>  On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>>>  > We do need some more information but this is just screaming for
>>>>  > histogram.  Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>>>  > .  Using histogram to see which x's are common you can step through
>>>>  > the reverse_indices and see which y's are then common.  There is
>>>>  > probably a more graceful way however.
>
>>>>  > Cheers,
>
>>>>  > Brian
>
>>>>  >  ------------------------------------------------------------ --------------
>>>>  > Brian Larsen
>>>>  > Boston University
>>>>  > Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>>>>  In particular, if you're dealing with integers that don't span too big
>>>>  a range, use HIST_2D and find the maximum element. If you've got
>>>>  floats or a wide range, use UNIQ to turn each into an integer on a
>>>>  small range first.
>
>>>>  -Jeremy.
>
>>>  I think if I were to be working with small datasets....ie not in the
>>>  millions of points I would use something like this
>
>>>  coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
>>>  [3,1],[21,14]]
>
>>>  counter = intarr(9)
>
>>>  FOR i = 0, 8 DO BEGIN
>>>    FOR j = 0, 8 DO BEGIN
>
>>>      IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>>>    ENDFOR
>>>  ENDFOR
>
>>>  ;- Histogram to find the max bins (no need to measure anything below 2
>>>  ;- because that would just be a single hit and if all of your pairs
>>>  ;- only occur once then who cares, right?
>>>  hist = histogram(counter, min=2, reverse_indices=ri)
>>>  maxHist = max(hist, mxpos)
>>>  IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
```

```
>
>>>  ;- Use the reverse indices given by histogram to find out exactly
>>>  ;- where in your counter these maxes are occurring
>>>  array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
>>>  ;- Find where counter is equal to the array index determined by
>>>  ;- reverse indices
>>>  max_index = where(counter EQ array_index)
>
>>>  ;- Voila with your max pair
>>>  print, coords[*,max_index[0]]
>
>>>  Which spits out....
>>>  20      32
>
>>>  This could be tweaked to find the top two or three or whatever as
>>>  well.
>>>  Hope this helps.
>
>>  My version of that would be:
>
>>  min1=min(coords[0,*], max=max1)
>>  min2=min(coords[1,*], max=max2)
>>  arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
>>  bin1=1, min2=min2, max2=max2, bin2=1)
>>  maxval = max(arraymap, maxelement)
>>  print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
>>  [min1,min2]
>
>>  ...which avoids loops, and is more obvious to me.
>
>>  -Jeremy.
>
>  No loops is all and good...but if you put a decimal in coords like
>  this
>
>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>  your code still spits out (20.0 32.0) where it should spit out (20.0
>  32.3)
>  By the way the code I presented up there should have the following
>  line replaced
>   array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>  with
>   array_index = (counter[ri[ri[mxpos]:ri[mxpos+1]-1]])[0]
```

Like I said, if you have floats (or a very large range of integers), you should map them into integers first using SORT and UNIQ...

```
coordsize = size(coords,/dimen)
coords0_sorted = coords[0,sort(coords[0,*])]
map0 = uniq(coords0_sorted)
nmap = n_elements(map0)
new_coords0 = lonarr(coordsize[1])
for i=0l,nmap-1 do new_coords0[where(coords[0,*] eq
coords0_sorted[map0[i]])]=i
```

...and the same for coords[1,*]. There's probably a more efficient way
of doing that, but you get the idea.

-Jeremy.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Juggernaut on Tue, 05 Aug 2008 14:14:23 GMT
View Forum Message <> Reply to Message

```
On Aug 1, 7:02 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>  On Jul 31, 10:50 am, Bennett <juggernau...@gmail.com> wrote:
>
>
>
>>  On Jul 31, 7:37 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>  On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
>
>>>>  On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>>  > On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>>>  > > We do need some more information but this is just screaming for
>>>>  > > histogram.  Have a read throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>>>  > > .  Using histogram to see which x's are common you can step through
>>>>  > > the reverse_indices and see which y's are then common.  There is
>>>>  > > probably a more graceful way however.
>
>>>>  > > Cheers,
>
>>>>  > > Brian
>
>>>>  > >  ------------------------------------------------------------ --------------
>>>>  > > Brian Larsen
>>>>  > > Boston University
>>>>  > > Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
```

```
>
>>>>  > In particular, if you're dealing with integers that don't span too big
>>>>  > a range, use HIST_2D and find the maximum element. If you've got
>>>>  > floats or a wide range, use UNIQ to turn each into an integer on a
>>>>  > small range first.
>
>>>>  > -Jeremy.
>
>>>>  I think if I were to be working with small datasets....ie not in the
>>>>  millions of points I would use something like this
>
>>>>  coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
>>>>  [3,1],[21,14]]
>
>>>>  counter = intarr(9)
>
>>>>  FOR i = 0, 8 DO BEGIN
>>>>    FOR j = 0, 8 DO BEGIN
>
>>>>      IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>>>>    ENDFOR
>>>>  ENDFOR
>
>>>>  ;- Histogram to find the max bins (no need to measure anything below 2
>>>>  ;- because that would just be a single hit and if all of your pairs
>>>>  ;- only occur once then who cares, right?
>>>>  hist = histogram(counter, min=2, reverse_indices=ri)
>>>>  maxHist = max(hist, mxpos)
>>>>  IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
>>>>  ;- Use the reverse indices given by histogram to find out exactly
>>>>  ;- where in your counter these maxes are occurring
>>>>  array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
>>>>  ;- Find where counter is equal to the array index determined by
>>>>  ;- reverse indices
>>>>  max_index = where(counter EQ array_index)
>
>>>>  ;- Voila with your max pair
>>>>  print, coords[*,max_index[0]]
>
>>>>  Which spits out....
>>>>  20      32
>
>>>>  This could be tweaked to find the top two or three or whatever as
>>>>  well.
>>>>  Hope this helps.
```

```
>
>>>  My version of that would be:
>
>>>  min1=min(coords[0,*], max=max1)
>>>  min2=min(coords[1,*], max=max2)
>>>  arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
>>>  bin1=1, min2=min2, max2=max2, bin2=1)
>>>  maxval = max(arraymap, maxelement)
>>>  print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
>>>  [min1,min2]
>
>>>  ...which avoids loops, and is more obvious to me.
>
>>>  -Jeremy.
>
>>  No loops is all and good...but if you put a decimal in coords like
>>  this
>
>>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>>  your code still spits out (20.0 32.0) where it should spit out (20.0
>>  32.3)
>>  By the way the code I presented up there should have the following
>>  line replaced
>>   array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>>  with
>>   array_index = (counter[ri[ri[mxpos]:ri[mxpos+1]-1]])[0]
>
>  Like I said, if you have floats (or a very large range of integers),
>  you should map them into integers first using SORT and UNIQ...
>
>  coordsize = size(coords,/dimen)
>  coords0_sorted = coords[0,sort(coords[0,*])]
>  map0 = uniq(coords0_sorted)
>  nmap = n_elements(map0)
>  new_coords0 = lonarr(coordsize[1])
>  for i=0l,nmap-1 do new_coords0[where(coords[0,*] eq
>  coords0_sorted[map0[i]])]=i
>
>  ...and the same for coords[1,*]. There's probably a more efficient way
>  of doing that, but you get the idea.
>
>  -Jeremy.
```

```
coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
[20.0,32.3],[10,10],[3,1],[21,14]]
```

```
sz = size(coords, /dimensions)

result = rebin(coords,2,sz[1],sz[1])
result2 = rebin(reform(coords,2,1,sz[1]),2,sz[1],sz[1])
indices = array_indices(result/result2,where(result/result2 EQ 1))

hist = histogram(indices[2,*])
maxHist = max(hist, mxpos)

print, coords[*,mxpos]
```

No loops...but definitely limited by size...can't really go with more
than a 7500 indices

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Jeremy Bailin on Wed, 06 Aug 2008 11:28:00 GMT

View Forum Message <> Reply to Message

```
On Aug 5, 10:14 am, Bennett <juggernau...@gmail.com> wrote:
>  On Aug 1, 7:02 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>>  On Jul 31, 10:50 am, Bennett <juggernau...@gmail.com> wrote:
>
>>>  On Jul 31, 7:37 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>>  On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
>
>>>>  > On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>>  > > On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>>>  > > > We do need some more information but this is just screaming for
>>>>  > > > histogram.  Have a read
throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>>>  > > > .  Using histogram to see which x's are common you can step through
>>>>  > > > the reverse_indices and see which y's are then common.  There is
>>>>  > > > probably a more graceful way however.
>
>>>>  > > > Cheers,
>
>>>>  > > > Brian
>
>>>>  > > >  ------------------------------------------------------------ --------------
>>>>  > > > Brian Larsen
```

```
>>>>  > > > Boston University
>>>>  > > > Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>>>>  > > In particular, if you're dealing with integers that don't span too big
>>>>  > > a range, use HIST_2D and find the maximum element. If you've got
>>>>  > > floats or a wide range, use UNIQ to turn each into an integer on a
>>>>  > > small range first.
>
>>>>  > > -Jeremy.
>
>>>>  > I think if I were to be working with small datasets....ie not in the
>>>>  > millions of points I would use something like this
>
>>>>  > coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
>>>>  > [3,1],[21,14]]
>
>>>>  > counter = intarr(9)
>
>>>>  > FOR i = 0, 8 DO BEGIN
>>>>  >   FOR j = 0, 8 DO BEGIN
>
>>>>  >     IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>>>>  >   ENDFOR
>>>>  > ENDFOR
>
>>>>  > ;- Histogram to find the max bins (no need to measure anything below 2
>>>>  > ;- because that would just be a single hit and if all of your pairs
>>>>  > ;- only occur once then who cares, right?
>>>>  > hist = histogram(counter, min=2, reverse_indices=ri)
>>>>  > maxHist = max(hist, mxpos)
>>>>  > IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
>>>>  > ;- Use the reverse indices given by histogram to find out exactly
>>>>  > ;- where in your counter these maxes are occurring
>>>>  > array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
>>>>  > ;- Find where counter is equal to the array index determined by
>>>>  > ;- reverse indices
>>>>  > max_index = where(counter EQ array_index)
>
>>>>  > ;- Voila with your max pair
>>>>  > print, coords[*,max_index[0]]
>
>>>>  > Which spits out....
>>>>  > 20     32
>
>>>>  > This could be tweaked to find the top two or three or whatever as
```

```
>>>>  > well.
>>>>  > Hope this helps.
>
>>>>  My version of that would be:
>
>>>>  min1=min(coords[0,*], max=max1)
>>>>  min2=min(coords[1,*], max=max2)
>>>>  arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
>>>>  bin1=1, min2=min2, max2=max2, bin2=1)
>>>>  maxval = max(arraymap, maxelement)
>>>>  print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
>>>>  [min1,min2]
>
>>>>  ...which avoids loops, and is more obvious to me.
>
>>>>  -Jeremy.
>
>>>  No loops is all and good...but if you put a decimal in coords like
>>>  this
>
>>>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>>>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>>>  your code still spits out (20.0 32.0) where it should spit out (20.0
>>>  32.3)
>>>  By the way the code I presented up there should have the following
>>>  line replaced
>>>   array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>>>  with
>>>   array_index = (counter[ri[ri[mxpos]:ri[mxpos+1]-1]])[0]
>
>>  Like I said, if you have floats (or a very large range of integers),
>>  you should map them into integers first using SORT and UNIQ...
>
>>  coordsize = size(coords,/dimen)
>>  coords0_sorted = coords[0,sort(coords[0,*])]
>>  map0 = uniq(coords0_sorted)
>>  nmap = n_elements(map0)
>>  new_coords0 = lonarr(coordsize[1])
>>  for i=0l,nmap-1 do new_coords0[where(coords[0,*] eq
>>  coords0_sorted[map0[i]])]=i
>
>>  ...and the same for coords[1,*]. There's probably a more efficient way
>>  of doing that, but you get the idea.
>
>>  -Jeremy.
>
>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>  sz = size(coords, /dimensions)
>
>  result = rebin(coords,2,sz[1],sz[1])
>  result2 = rebin(reform(coords,2,1,sz[1]),2,sz[1],sz[1])
>  indices = array_indices(result/result2,where(result/result2 EQ 1))
>
>  hist = histogram(indices[2,*])
>  maxHist = max(hist, mxpos)
>
>  print, coords[*,mxpos]
>
>  No loops...but definitely limited by size...can't really go with more
>  than a 7500 indices
```

That doesn't work if you have individual elements that come up much more often than elements in the most frequent pair... I fooled it with this input:

```
coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14], $
  [20.0,32.3],[10,10],[3,1],[21,14],[10.0,32.3],[10.0,8]]
```

and it comes up with [10.0,32.3].

-Jeremy.

Subject: Re: Finding the Top Two Most Common Coordinates in a Multi-Dimensional Array
Posted by Juggernaut on Thu, 07 Aug 2008 13:21:26 GMT

View Forum Message <> Reply to Message

```
On Aug 6, 7:28 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>  On Aug 5, 10:14 am, Bennett <juggernau...@gmail.com> wrote:
>
>
>
>>  On Aug 1, 7:02 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>  On Jul 31, 10:50 am, Bennett <juggernau...@gmail.com> wrote:
>
>>>>  On Jul 31, 7:37 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>>  > On Jul 30, 7:54 am, Bennett <juggernau...@gmail.com> wrote:
>
>>>>  > > On Jul 29, 11:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
```

```
>>>>  > > > On Jul 29, 2:32 am, Brian Larsen <balar...@gmail.com> wrote:
>
>>>>  > > > > We do need some more information but this is just screaming for
>>>>  > > > > histogram.  Have a read
throughhttp://www.dfanning.com/tips/histogram_tutorial.html
>>>>  > > > > .  Using histogram to see which x's are common you can step through
>>>>  > > > > the reverse_indices and see which y's are then common.  There is
>>>>  > > > > probably a more graceful way however.
>
>>>>  > > > > Cheers,
>
>>>>  > > > > Brian
>
>>>>  > > > >  --------------------------------------------------------- --------------
>>>>  > > > > Brian Larsen
>>>>  > > > > Boston University
>>>>  > > > > Center for Space Physicshttp://people.bu.edu/balarsen/Home/IDL
>
>>>>  > > > In particular, if you're dealing with integers that don't span too big
>>>>  > > > a range, use HIST_2D and find the maximum element. If you've got
>>>>  > > > floats or a wide range, use UNIQ to turn each into an integer on a
>>>>  > > > small range first.
>
>>>>  > > > -Jeremy.
>
>>>>  > > I think if I were to be working with small datasets....ie not in the
>>>>  > > millions of points I would use something like this
>
>>>>  > > coords = [[10,1],[20,32],[5,7],[6,8],[20,32],[2,14],[20,32],[10,10],
>>>>  > > [3,1],[21,14]]
>
>>>>  > > counter = intarr(9)
>
>>>>  > > FOR i = 0, 8 DO BEGIN
>>>>  > >   FOR j = 0, 8 DO BEGIN
>
>>>>  > >     IF array_equal(coords[*,i],coords[*,j]) THEN counter[i]++
>
>>>>  > >   ENDFOR
>>>>  > > ENDFOR
>
>>>>  > > ;- Histogram to find the max bins (no need to measure anything below 2
>>>>  > > ;- because that would just be a single hit and if all of your pairs
>>>>  > > ;- only occur once then who cares, right?
>>>>  > > hist = histogram(counter, min=2, reverse_indices=ri)
>>>>  > > maxHist = max(hist, mxpos)
>>>>  > > IF maxHist EQ 1 THEN print, 'Each pair occurs no more than once'
>
```

```
>>>>  > > ;- Use the reverse indices given by histogram to find out exactly
>>>>  > > ;- where in your counter these maxes are occurring
>>>>  > > array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>
>>>>  > > ;- Find where counter is equal to the array index determined by
>>>>  > > ;- reverse indices
>>>>  > > max_index = where(counter EQ array_index)
>
>>>>  > > ;- Voila with your max pair
>>>>  > > print, coords[*,max_index[0]]
>
>>>>  > > Which spits out....
>>>>  > > 20      32
>
>>>>  > > This could be tweaked to find the top two or three or whatever as
>>>>  > > well.
>>>>  > > Hope this helps.
>
>>>>  > My version of that would be:
>
>>>>  > min1=min(coords[0,*], max=max1)
>>>>  > min2=min(coords[1,*], max=max2)
>>>>  > arraymap = hist_2d(coords[0,*], coords[1,*], min1=min1, max1=max1,
>>>>  > bin1=1, min2=min2, max2=max2, bin2=1)
>>>>  > maxval = max(arraymap, maxelement)
>>>>  > print, array_indices([max1-min1+1,max2-min2+1], maxelement, /dimen)+
>>>>  > [min1,min2]
>
>>>>  > ...which avoids loops, and is more obvious to me.
>
>>>>  > -Jeremy.
>
>>>>  No loops is all and good...but if you put a decimal in coords like
>>>>  this
>
>>>>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>>>>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>>>>  your code still spits out (20.0 32.0) where it should spit out (20.0
>>>>  32.3)
>>>>  By the way the code I presented up there should have the following
>>>>  line replaced
>>>>   array_index = (counter[ri[ri[1]:ri[2]-1]])[0]
>>>>  with
>>>>   array_index = (counter[ri[ri[mxpos]:ri[mxpos+1]-1]])[0]
>
>>>  Like I said, if you have floats (or a very large range of integers),
>>>  you should map them into integers first using SORT and UNIQ...
```

```
>
>>>  coordsize = size(coords,/dimen)
>>>  coords0_sorted = coords[0,sort(coords[0,*])]
>>>  map0 = uniq(coords0_sorted)
>>>  nmap = n_elements(map0)
>>>  new_coords0 = lonarr(coordsize[1])
>>>  for i=0l,nmap-1 do new_coords0[where(coords[0,*] eq
>>>  coords0_sorted[map0[i]])]=i
>
>>>  ...and the same for coords[1,*]. There's probably a more efficient way
>>>  of doing that, but you get the idea.
>
>>>  -Jeremy.
>
>>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14],
>>  [20.0,32.3],[10,10],[3,1],[21,14]]
>
>>  sz = size(coords, /dimensions)
>
>>  result = rebin(coords,2,sz[1],sz[1])
>>  result2 = rebin(reform(coords,2,1,sz[1]),2,sz[1],sz[1])
>>  indices = array_indices(result/result2,where(result/result2 EQ 1))
>
>>  hist = histogram(indices[2,*])
>>  maxHist = max(hist, mxpos)
>
>>  print, coords[*,mxpos]
>
>>  No loops...but definitely limited by size...can't really go with more
>>  than a 7500 indices
>
>  That doesn't work if you have individual elements that come up much
>  more often than elements in the most frequent pair... I fooled it with
>  this input:
>
>  coords = [[10.0,1.0],[20.0,32.3],[5,7],[6,8],[20.0,32.3],[2,14], $
>    [20.0,32.3],[10,10],[3,1],[21,14],[10.0,32.3],[10.0,8]]
>
>  and it comes up with [10.0,32.3].
>
>  -Jeremy.
```

Dang it...well I had to give it a go.