Subject: Sorting a matrix
Posted by on Fri, 01 Mar 2013 12:45:36 GMT
View Forum Message <> Reply to Message

I'd like to sort a matrix M in such a way that the order of the rows is determined by the value in the first column. When the first column values are the same, the second column should be used, etc.

Something like Craig Marqwardt's multisort (http://cow.physics.wisc.edu/~craigm/idl/down/multisort.pro), with M[0,*] used as key1, M[1,*] as key2, etc. But without actually having to specify the keys one by one. (Never mind which index counts as the column index :o)

As the matrices I have in mind right now are integer arrays and do not have that many possible values (just -1,0,1), I thought about turning the matrix into a 1D string array with the length of each string equal to the number of columns and translating the column values into characters in the string, like A for -1, B for 0, C for 1, and then sorting the string array. But I'd prefer a more general program, in case there is one out there.

Pointers, ideas?

Subject: Re: Sorting a matrix
Posted by Jeremy Bailin on Fri, 01 Mar 2013 16:28:07 GMT
View Forum Message <> Reply to Message

On 3/1/13 7:45 AM, Mats Lï¿½fdahl wrote:
> I'd like to sort a matrix M in such a way that the order of the rows is determined by the value in the first column. When the first column values are the same, the second column should be used, etc.
>
> Something like Craig Marqwardt's multisort (http://cow.physics.wisc.edu/~craigm/idl/down/multisort.pro), with M[0,*] used as key1, M[1,*] as key2, etc. But without actually having to specify the keys one by one. (Never mind which index counts as the column index :o)
>
> As the matrices I have in mind right now are integer arrays and do not have that many possible values (just -1,0,1), I thought about turning the matrix into a 1D string array with the length of each string equal to the number of columns and translating the column values into characters in the string, like A for -1, B for 0, C for 1, and then sorting the string array. But I'd prefer a more general program, in case there is one out there.
>
> Pointers, ideas?
>

I've done something like this before by generating a single unique index... something like this:

matrix = [[4,5,6], [4,6,8], [2,3,4], [4,6,7]]

```
matrixshape = size(matrix, /dimen)
; this gives you the range of each column:
matrixord = lonarr(matrixshape)
for i=0l,matrixshape[0]-1 do matrixord[i,*] = ord(matrix[i,*])
ordmax = max(matrixord, dimen=2)

; what do you need to multiply by to get a unique range?
column_multiply = [reverse(product(reverse(ordmax[1:*]+1), /int,
/cumul)), 1]

; create a unique key and sort on it
sortkey = total(matrixord * rebin(column_multiply,matrixshape, /sample),
/int, 1)
newmatrix = matrix[*, sort(sortkey)]

IDL> print, newmatrix
      2       3       4
      4       5       6
      4       6       7
      4       6       8
```

You'll need ORD(), which is part of JBIU which is currently inaccessible because I'm in the process of moving domains. But here's a stub:

```
function ord, values

nvalues=n_elements(values)
sortvalues = sort(values)
uniqvalues = uniq(values[sortvalues])

nuniq = n_elements(uniqvalues)
ordlist = lindgen(nuniq)

; this is basically the histogram(total(/cumulative)) trick
h = histogram(uniqvalues,bin=1,min=0,reverse=ri)
outp = lonarr(size(values, /dimen))
outp[sortvalues] = ordlist[ri[0:nvalues-1]-ri[0]]

return, outp

end
```

Subject: Re: Sorting a matrix
Posted by on Fri, 01 Mar 2013 16:49:51 GMT
View Forum Message <> Reply to Message

Den fredagen den 1:e mars 2013 kl. 17:28:07 UTC+1 skrev Jeremy Bailin:
> On 3/1/13 7:45 AM, Mats Löfdahl wrote:
>
>> I'd like to sort a matrix M in such a way that the order of the rows is determined by the value in the first column. When the first column values are the same, the second column should be used, etc.
>
>>
>
>> Something like Craig Marqwardt's multisort (http://cow.physics.wisc.edu/~craigm/idl/down/multisort.pro), with M[0,*] used as key1, M[1,*] as key2, etc. But without actually having to specify the keys one by one. (Never mind which index counts as the column index :o)
>
>>
>
>> As the matrices I have in mind right now are integer arrays and do not have that many possible values (just -1,0,1), I thought about turning the matrix into a 1D string array with the length of each string equal to the number of columns and translating the column values into characters in the string, like A for -1, B for 0, C for 1, and then sorting the string array. But I'd prefer a more general program, in case there is one out there.
>
>>
>
>> Pointers, ideas?
>
>>
>
>
>
> I've done something like this before by generating a single unique
>
> index... something like this:
>
>
>
> matrix = [[4,5,6], [4,6,8], [2,3,4], [4,6,7]]
>
>
>
> matrixshape = size(matrix, /dimen)
>
> ; this gives you the range of each column:
>
> matrixord = lonarr(matrixshape)
>
> for i=0l,matrixshape[0]-1 do matrixord[i,*] = ord(matrix[i,*])
>

```
> ordmax = max(matrixord, dimen=2)
>
>
>
> ; what do you need to multiply by to get a unique range?
>
> column_multiply = [reverse(product(reverse(ordmax[1:*]+1), /int,
>
> /cumul)), 1]
>
>
>
> ; create a unique key and sort on it
>
> sortkey = total(matrixord * rebin(column_multiply,matrixshape, /sample),
>
> /int, 1)
>
> newmatrix = matrix[*, sort(sortkey)]
>
>
>
> IDL> print, newmatrix
>
>       2      3      4
>
>       4      5      6
>
>       4      6      7
>
>       4      6      8
>
>
>
> You'll need ORD(), which is part of JBIU which is currently inaccessible
>
> because I'm in the process of moving domains. But here's a stub:
>
>
>
> function ord, values
>
>
>
> nvalues=n_elements(values)
>
> sortvalues = sort(values)
>
```

```
>  uniqvalues = uniq(values[sortvalues])
>
>
>
>  nuniq = n_elements(uniqvalues)
>
>  ordlist = lindgen(nuniq)
>
>
>
>  ; this is basically the histogram(total(/cumulative)) trick
>
>  h = histogram(uniqvalues,bin=1,min=0,reverse=ri)
>
>  outp = lonarr(size(values, /dimen))
>
>  outp[sortvalues] = ordlist[ri[0:nvalues-1]-ri[0]]
>
>
>
>  return, outp
>
>
>
>  end
```

Oh, this looks pretty clever. I think I understand the idea but I'll have to digest the details. If I'm right, you find out for each column how many different values there are and then multiply the index for that column with the proper number to avoid overlap with the other column indices when adding.

Thanks!

Subject: Re: Sorting a matrix
Posted by Jeremy Bailin on Wed, 06 Mar 2013 05:22:01 GMT
View Forum Message <> Reply to Message

> Oh, this looks pretty clever. I think I understand the idea but I'll have to digest the details. If I'm right, you find out for each column how many different values there are and then multiply the index for that column with the proper number to avoid overlap with the other column indices when adding.

Yes, that's exactly right!

-Jeremy.

Subject: Re: Sorting a matrix
Posted by on Thu, 07 Mar 2013 13:32:12 GMT
View Forum Message <> Reply to Message

Den onsdagen den 6:e mars 2013 kl. 06:22:01 UTC+1 skrev Jeremy Bailin:
>> Oh, this looks pretty clever. I think I understand the idea but I'll have to digest the details. If I'm right, you find out for each column how many different values there are and then multiply the index for that column with the proper number to avoid overlap with the other column indices when adding.
>
> Yes, that's exactly right!

OK, good. But I have problems with the ord() function. When called with an array of only zeros, the call to histogram() does not work:

HISTOGRAM: Expression must be an array in this context: UNIQVALUES.
% Execution halted at: ORD 14

Where line 14 is

h = histogram(uniqvalues,bin=1,min=0,reverse=ri)

The reason is that when there is only one value, uniqvalues has only one element and is therefore turned into is a scalar.

The call to histogram works if I chance it to

h = histogram([uniqvalues],bin=1,min=0,reverse=ri)

Could this have bad consequences in other cases or is this what I should do? You described the ord() code you gave as a stub. Does this mean there is a more complete version that maybe checks for this problem (and other problems as well)?

Subject: Re: Sorting a matrix
Posted by Jeremy Bailin on Fri, 08 Mar 2013 00:34:53 GMT
View Forum Message <> Reply to Message

On 3/7/13 7:32 AM, Mats Lï¿½fdahl wrote:
> Den onsdagen den 6:e mars 2013 kl. 06:22:01 UTC+1 skrev Jeremy Bailin:
>>> Oh, this looks pretty clever. I think I understand the idea but I'll have to digest the details. If I'm right, you find out for each column how many different values there are and then multiply the index for that column with the proper number to avoid overlap with the other column indices when adding.
>>
>> Yes, that's exactly right!
>
> OK, good. But I have problems with the ord() function. When called with an array of only zeros,

the call to histogram() does not work:
>
>   HISTOGRAM: Expression must be an array in this context: UNIQVALUES.
> % Execution halted at: ORD   14
>
> Where line 14 is
>
>   h = histogram(uniqvalues,bin=1,min=0,reverse=ri)
>
> The reason is that when there is only one value, uniqvalues has only one element and is therefore turned into is a scalar.
>
> The call to histogram works if I chance it to
>
>   h = histogram([uniqvalues],bin=1,min=0,reverse=ri)
>
> Could this have bad consequences in other cases or is this what I should do? You described the ord() code you gave as a stub. Does this mean there is a more complete version that maybe checks for this problem (and other problems as well)?
>

Ah, interesting! No, I didn't check for that - the code is actually
complete, it's just that the documentation is missing because I didn't
want to obscure the post. ;-)

Thanks for the bugfix!

-Jeremy.

Subject: Re: Sorting a matrix
Posted by cgguido on Fri, 08 Mar 2013 22:24:44 GMT
View Forum Message <> Reply to Message

I was curious to compare this method with a more straightforward way. multisort.pro below does both.
Jeremy's method is twice as fast for large arrays. However, the two methods only give the same result for smallish arrays.

Any ideas what's going on?

```
;;;;;;;;;;;;;; EXAMPLES ;;;;;;;;;;;;;;
multisort, round(randomu(s,20,1e1)*10)
;   1
;   0.00027489662   0.00026106834

multisort, round(randomu(s,20,1e2)*10)
;   0
```

;   0.00072312355   0.00039005280

;;;;;;;;;;;;;; CODE ;;;;;;;;;;;;;;;
```
PRO multisort,  matrix

compile_opt idl2


matrixshape = size(matrix, /dimen)

t1 = systime(1)


; this gives you the range of each column:
matrixord = lonarr(matrixshape)
for i=0l,matrixshape[0]-1 do matrixord[i,*] = ord(matrix[i,*])
ordmax = max(matrixord, dimen=2)

; what do you need to multiply by to get a unique range?
column_multiply = [reverse(product(reverse(ordmax[1:*]+1), /int, /cumul)), 1]

; create a unique key and sort on it
sortkey = total(matrixord * rebin(column_multiply,matrixshape, /sample), /int, 1)
newmatrix = matrix[*, sort(sortkey)]
t2 = systime(1)


newmatrix2 = matrix
FOR i = matrixshape[0]-1, 0, -1 DO BEGIN
  s = bsort(newmatrix2[i, *]);use bsort which maintains order of identical elements
  newmatrix2 = newmatrix2[*, s]
ENDFOR
t3 = systime(1)

print,array_equal( newmatrix,  newmatrix2)

print,  t3-t2,  t2-t1


RETURN
END
```

Subject: Re: Sorting a matrix
Posted by Jeremy Bailin on Fri, 08 Mar 2013 23:51:30 GMT
View Forum Message <> Reply to Message

On 3/8/13 4:24 PM, Gianguido Cianci wrote:
> I was curious to compare this method with a more straightforward way. multisort.pro below does both.
> Jeremy's method is twice as fast for large arrays. However, the two methods only give the same result for smallish arrays.
>
> Any ideas what's going on?
>
> ;;;;;;;;;;;;;; EXAMPLES ;;;;;;;;;;;;;
> multisort, round(randomu(s,20,1e1)*10)
> ; 1
> ; 0.00027489662 0.00026106834
>
> multisort, round(randomu(s,20,1e2)*10)
> ; 0
> ; 0.00072312355 0.00039005280
>
>
> ;;;;;;;;;;;;;; CODE ;;;;;;;;;;;;;;;
> PRO multisort, matrix
>
> compile_opt idl2
>
>
> matrixshape = size(matrix, /dimen)
>
> t1 = systime(1)
>
>
> ; this gives you the range of each column:
> matrixord = lonarr(matrixshape)
> for i=0l,matrixshape[0]-1 do matrixord[i,*] = ord(matrix[i,*])
> ordmax = max(matrixord, dimen=2)
>
> ; what do you need to multiply by to get a unique range?
> column_multiply = [reverse(product(reverse(ordmax[1:*]+1), /int, /cumul)), 1]
>
> ; create a unique key and sort on it
> sortkey = total(matrixord * rebin(column_multiply,matrixshape, /sample), /int, 1)
> newmatrix = matrix[*, sort(sortkey)]
> t2 = systime(1)
>
>
> newmatrix2 = matrix
> FOR i = matrixshape[0]-1, 0, -1 DO BEGIN
> s = bsort(newmatrix2[i, *]);use bsort which maintains order of identical elements
> newmatrix2 = newmatrix2[*, s]
> ENDFOR

> t3 = systime(1)
>
> print,array_equal(newmatrix, newmatrix2)
>
> print, t3-t2, t2-t1
>
>
> RETURN
> END
>

Bwahahah.... oh dear.

The problem is that in this case, column_multiply ends up being a LONG64
in order to fit the maximum value of the PRODUCT. But when you multiply:
  matrixord * rebin(column_multiply, matrixshape, /sample)
the values are larger than the maximum LONG64... and wrap around to
become negative numbers!

```
IDL> print, minmax(matrixord)
           0          10
IDL> print, minmax(column_multiply)
                    1   5054470284992937710
IDL> print, minmax(matrixord * rebin(column_multiply, matrixshape, /sample))
  -8337803503723676196   8596744417517336158
IDL> help, matrixord, column_multiply
MATRIXORD       LONG      = Array[20, 100]
COLUMN_MULTIPLY LONG64    = Array[20]
```

In principle there are 10^20 possible rows, and it's trying to make sure
it has a unique integer for each possibility.

I guess my advice is to cap it at 19 columns and maybe do it in chunks
if there are more columns.

-Jeremy.

Subject: Re: Sorting a matrix
Posted by on Sat, 09 Mar 2013 00:14:49 GMT
View Forum Message <> Reply to Message

Den fredagen den 8:e mars 2013 kl. 23:24:44 UTC+1 skrev Gianguido Cianci:
> I was curious to compare this method with a more straightforward way. multisort.pro below does both.
>
> Jeremy's method is twice as fast for large arrays. However, the two methods only give the same result for smallish arrays.

When the results differ, does any of the methods sort correctly?

Subject: Re: Sorting a matrix
Posted by cgguido on Sat, 09 Mar 2013 00:24:00 GMT
View Forum Message <> Reply to Message

hmmm try it for yourself :-) I check by plotting the data and checking it is non-strictly monotonically increasing:

plot, newmatrix[0,*]; looks strange
plot, newmatrix2[0,*]; looks ok

On Friday, March 8, 2013 6:14:49 PM UTC-6, Mats Löfdahl wrote:
> Den fredagen den 8:e mars 2013 kl. 23:24:44 UTC+1 skrev Gianguido Cianci:
>
>> I was curious to compare this method with a more straightforward way. multisort.pro below does both.
>
>>
>
>> Jeremy's method is twice as fast for large arrays. However, the two methods only give the same result for smallish arrays.
>
>
>
> When the results differ, does any of the methods sort correctly?

Subject: Re: Sorting a matrix
Posted by on Sat, 09 Mar 2013 00:31:08 GMT
View Forum Message <> Reply to Message

Den lördagen den 9:e mars 2013 kl. 01:24:00 UTC+1 skrev Gianguido Cianci:
> hmmm try it for yourself :-) I check by plotting the data and checking it is non-strictly monotonically increasing:
>
>
>
> plot, newmatrix[0,*]; looks strange
>
> plot, newmatrix2[0,*]; looks ok

That's interesting. I've also seen problems with the former method for larger arrays but I haven't tracked them down yet. One thing I would check is if some quantity needs to be larger that what is

accommodated by its integer type.

But it's really late in Sweden now...