Subject: Re: faster then where possible?
Posted by rogass on Mon, 11 May 2009 09:00:00 GMT
View Forum Message <> Reply to Message

```
On 8 Mai, 16:14, cmanc...@gmail.com wrote:
>  On May 8, 7:58 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>
>
>>  On May 7, 11:06 am, rog...@googlemail.com wrote:
>
>>>  Hi,
>>>  i'm searching for some alternative approaches to compute the following
>>>  "much" faster:
>
>>>  -> matrix1 has m columns and n rows, matrix2 has 2 columns and n rows
>>>  -> the values in matrix2 are NOT in matrix1, but within the min-max-
>>>  range of matrix1
>
>>>  szm1=size(matrix1,/dimensions)
>>>  szm2=size(matrix2,/dimensions)
>>>  index={ind:ptr_new()}
>>>  indices=replicate(index,szm2[1])
>
>>>  for j=0ull,szm1[1] do begin
>>>     helpindex= where(matrix1[*,j] ge matrix2[0,j] and matrix1[*,j] le
>>>  matrix2[1,j],c)
>>>     if c gt 0 then begin
>>>           indices[j]    = ptr_new(uintarr(c))
>>>           (*indices)[j]=helpindex
>>>      endif else continue
>>>  endfor
>
>>>  It seems to be a typical Nearest-Neighbor-Problem, but all alternative
>>>  approaches I tried were always slower. Maybe someone here has a good
>>>  idea?
>
>>>  Thank you and best regards
>
>>>  Christian
>
>>  I don't suppose the data in the rows of matrix1 are sorted? If so, you
>>  could use VALUE_LOCATE to figure out the bounds.
>
>>  -Jeremy.
>
```

> Yeah, this sounds like a job for value_locate + histogram, which can combine to make histograms with irregularly spaced bins (which, I think, is basically what you're doing). To use value_locate you just have to have the list you are searching within sorted. This is how you would make a histogram with irregularly spaced bins:
>
> ```
> bin_mins = findgen(15)+randomu(seed,15)*.2 - .1
> vals_to_bin = randomu(seed,200)*15
>
> find = value_locate( bin_mins, vals_to_bin )
> hist = histogram( find, reverse_indices=ri )
> nbins = n_elements(hist)
>
> for i=0,nbins-1 do begin
>     if ri[i+1] eq ri[i] then continue ; no data in this bin
>     inds = ri[ ri[i]:ri[i+1]-1 ]
> endfor
> ```
>
> So you use value_locate to find which elements belong to which minimum value, and then you use reverse_indices to pick out the indexes of the elements in each bin. Again, this depends on your list of bin minimums being sorted. So if your minimum value from matrix2[0,j] is equal to the maximum value in matrix2[1,j] (in the sense that the maximum for one bin is the minimum for the next), then the above code will exactly solve your problem, and the inds variable above has the exact same content as the helpindex variable in your code. If the maximum value (matrix2[1,j]) is smaller, such that the maximum value in a bin is smaller than the minimum value of the next bin, then you can just throw an additional where() in the for loop above. Since you only have to search over a small subset of your data set, rather than the full data set, this should still be much faster. I.e. the for loop above would change to:
>
> ```
> for i=0,nbins-1 do begin
>     if ri[i+1] eq ri[i] then continue ; no data in this bin
>     inds = ri[ ri[i]:ri[i+1]-1 ]
>     t = where( vals_to_bin[inds] lt some_max_value, c )
>     helpindex = inds[t]
> endfor
> ```
>
> In the case that there is overlap between bins, such that the maximum for a bin is larger than the minimum of the next bin... well in that case the above code wouldn't work at all :(Value locate always puts each item in exactly one bin, so if things can potentially be in more than one bin it clearly won't work...- Zitierten Text ausblenden -
>
> - Zitierten Text anzeigen -

Dear all,
thank you for your suggestions, especially for the idea due to value_locate. I will test it carefully.

Best regards

Christian