Subject: Re: Can I do this without using loops?
Posted by David Ritscher on Mon, 10 Jun 1996 07:00:00 GMT
View Forum Message <> Reply to Message

S Bhattacharyya wrote:
>
> Regarding loops, I am kinda in the same boat. My advisor
> keeps complaining about how slow our code runs...We don't seem to
> know any better around here :-)
>
> Q1) I have a generic array foo(x,y). I'd like to divide each column
> by its max. Can this be done without looping ?
>
> Q2) I have a generic array foo=fltarr(a,b). I'd like to copy findgen(b)
> into every column. Any way of doing this without loops ?

Can these be done without loops???
No. Unfortunately, the IDL/PVWave subscript syntax is relatively powerful, but not that powerful. I keep hoping that one or both of the two languages (i.e., companies) will decide to improve in this area, providing a syntax that specifies 'loop over' and 'extract as vector', i.e., something that communicates to the interpreter which dimensions of an array should be extracted and passed in entirety to something, such as a function, and over which dimensions looping should be performed. As an example syntax, the following would extract the greatest value of each column of the array A:

```
column_maxes = fltarr( 1, n_elements(A(0,*)))
column_maxes(loop_over:i) = max( A(loop_over:i, extract_as_vector) )
```

where 'i' is a dummy variable that synchronizes the looping.

Back to the real world: The following are possible solutions under the current syntax limitations. They concur with and expand upon the comments of Prof. Kenneth P. Bowman.

> Q1) I have a generic array foo(x,y). I'd like to divide each column
> by its max. Can this be done without looping ?

Take, for example:

```
xsize = 3
ysize = 5
foo = findgen(xsize, ysize)
```

Here's the simplest solution:

```
for i = 0L, xsize-1 do foo(i, 0) = foo(i, *) / max(foo(i, *))
```

However, this accesses the arrays column-by-column, which can lead to

slow operation with large arrays. Then the following would be more
efficient, and could reduce page faulting to ~1/3 since it is
necessary to search through the columns only once, not three times:
```
  maxes = fltarr(xsize)
  for i = 0L, xsize-1 do maxes(i) = max(foo(i, *))
  for j = 0L, ysize-1 do foo(0, j) = foo(*, j) / maxes
```

Note that it might also be necessary to check if the max of a column is zero.

> Q2) I have a generic array foo=fltarr(a,b). I'd like to copy findgen(b)
> into every column. Any way of doing this without loops ?
Still no. :-(

simple and efficient:
```
  foo = fltarr(3, 5)
  for j = 0L, ysize-1 do foo(*, j) = float(j)
```

If you were doing it over the columns instead of the rows, (i.e.,
copying findgen(a) into each row), the following would then be more
efficient, since again it would work row-by-row:
```
  one_row = findgen(xsize)
  for j = 0L, ysize-1 do foo(0, j) = one_row
```

For cases where a similar operation is to be performed many times, it
can be useful to create an indexing array that aids in carrying out
the procedure. For example, if the procedure of Q1 were to be
repeated a number of times on different arrays of the same dimension,
the result of Q2 could be used as an indexing array to make the
repititions more efficient (but note the additional demand on
memory!).
```
 ; for example, use the following 'foo':
 xsize = 3
 ysize = 5
 foo = findgen(xsize, ysize)
 ;
 ; Create an index for accessing inv_maxes vector repeatedly:
 inv_maxes_index = make_array(size=size(foo))
 for i = 0L, xsize-1 do inv_maxes_index(i, *) = i
 ;
 ; Repeat the following for each array to be processed (foo, foo2, etc.):
 maxes = fltarr(xsize)
 for i = 0L, xsize-1 do maxes(i) = max(foo(i, *))
 ; check for a column max of '0', that will cause divide problems:
 if ((where(maxes EQ 0.0))(0) NE -1) then $
     message, 'column found with max = 0'
 inv_maxes = 1. / maxes
 ; Now perform the scaling, scanning through the elements of 'foo' and
```

```
 ; repeatedly through the inv_maxes vector:
 foo = foo * inv_maxes(inv_maxes_index)
```

Thus the scaling is performed as a single matrix multiply.

Note that in IDL, the handy TEMPORARY function can make the last line above more efficient, by not making a second copy of 'foo':

```
 foo = temporary(foo) * inv_maxes(inv_maxes_index)
```

Be careful with the above indexes - with an interesting 'enhanced feature' of PVWave and IDL, when you index an array with an array, instead of the normal array checking, elements going out of bounds are simply 'truncated' to the last element. For example:

```
 test = indgen(2)
 test_index = indgen(9)
 print, test(test_index)
      0       1       1       1       1       1       1       1       1
```

--

David Ritscher
Raum 47.2.401
Zentralinstitut fuer Biomedizinische Technik
Albert-Einstein-Allee 47
Universitaet Ulm
D-89069 ULM
Germany

Tel: ++49 (731) 502 5313
Fax: ++49 (731) 502 5315
internet: david.ritscher@zibmt.uni-ulm.de