Subject: Re: newbie sees iterative solution, wants to know the IDL Way
Posted by Chris[6] on Mon, 21 Jul 2008 09:08:49 GMT
View Forum Message <> Reply to Message

On Jul 20, 6:28 pm, Tom Roche <tlro...@gmail.com> wrote:
```
>  <best viewed in monospace font/>
>
>  From what I read, iteration is not the way to go with IDL. So I'm
>  wondering, how to translate the following iterative solution:
>
>  I'm reading netCDF data with dimensions={time, lat, lon}, from which I
>  want to get the "seasonality" such that, for each spacetime, I get the
>  proportion of the total over time at that point in space. E.g. if I
>  had input data for 4 space points (j,k) @ 2 time points (i):
>
>  (k=0) j=0 j=1
>    i=0: 0   1
>    i=1: 2   3
>
>  (k=1)
>    i=0: 4   5
>    i=1: 6   7
>
>  it would map to
>
>        0    1/4
>        1    3/4
>
>        2/5  5/12
>        3/5  7/12
>
>  My coding background is such that I quickly see an iterative solution
>  like
>
>    nTime=size of time dimension;
>    nLat=size of latitude dimension;
>    nLon=size of longitude dimension;
>
>    create array summing[nLat][nLon];
>    for (int i = 0; i < nTime; i++)
>      for (int j = 0; j < nLat; j++)
>        for (int k = 0; k < nLon; k++)
>          // gotta handle input=NaN
>          if (isNumber(input[i][j][k]))
>            summing[j][k] += input[i][j][k];
>
>    create array seasonality[nTime][nLat][nLon];
>    for (int i = 0; i < nTime; i++)
>     for (int j = 0; j < nLat; j++)
>       for (int k = 0; k < nLon; k++)
>         // gotta handle input=NaN, summing={0, NaN}
>         if (isNumber(input[i][j][k]) &&
>             isNumber(summing[j][k]) &&
>             (input[i][j][k] != 0))
>           seasonality[i][j][k] = input[i][j][k] / summing[j][k];
>
>  Is there a better, IDLer way?
>
>  TIA, Tom Roche <Tom_Ro...@pobox.com>
```

A couple of things:
First, you have your diagram mislabeled. In IDL, the first index is the column. I will assume you still want to sum over time, but the correct diagrams (and answers) will now be:

```
(k=0) i=0 i=1
>   j=0: 0   1
>   j=1: 2   3
>
>  (k=1)
>   j=0: 4   5
>   j=1: 6   7
>
>  output:
>
>       0    1
>       2/5    3/5
>
>       4/9  5/9
>       6/13  7/13
```

To do this, try:

```
1) sz=size(input)
2) summing=total(input,1,/nan)
3)  summing=rebin(reform(summing,1,sz[2],sz[3]),sz[1],sz[2],sz[3 ])
4) output=input/summing
5) badval=where(output eq 0, ct)
6) if ct ne 0 then output[badval]=0
```

Explanation:
-size returns information on the size of the input. In particular, sz[i] gives the length of the ith dimension
-Line 2 sums the input over dimension 1, treating NAN's as missing data. This returns a j by k array
-Line 3 is the quirky IDL line. Reform redistributes the elements of

an array along different dimensions. It turns the j by k array into a
1 by j by k cube slice. Rebin will copy the array along the first
dimension, making an i by j by k cube. Read up at
http://www.dfanning.com/tips/rebin_magic.html
-Line 4 divides the input and summing cubes element by element.
-Lines 5 and 6 take care of cases where the output has a zero element.
This only happens when the answer should be zero, so we set it
manually.

Note that line three is messier than it has to be, because the summing
happens over the 1st dimension. If time were on the third dimension,
you could replace lines 2 and 3 by
summing=total(input,3,/nan)
summing=rebin(summing,sz[1],sz[2],sz[3])

cheers,
chris