Subject: Re: Another "IDL way to do this" question
Posted by Jeremy Bailin on Tue, 08 Nov 2011 20:08:46 GMT
View Forum Message <> Reply to Message

On 11/8/11 9:50 AM, Fabzou wrote:
> Dear IDLers,
>
> I have to interpolate modelled fields from eta to pressure coordinates,
> which is a relatively easy thing to do when using a simple linear
> approach. Since my 3D grid is irregular "only" on the third dimension
> (the fourth dimension being time), well I did not find any other
> solution than looping on the three regular dimensions and use INTERPOL
> for each vertical column.(interpol_3d in the code posted below)
>
> This works fine, but is not very very fast.
>
> Then I tried to loop on the time dimension outside the "main loop"
> (interpol_3d_alt in the code posted below), but results are only
> slightly better, as shown if I run the test program compare_interp:
>
> % Compiled module: INTERPOL_3D.
> % Compiled module: INTERPOL_3D_ALT.
> % Compiled module: COMPARE_INTERP.
> IDL> compare_interp
> Method1: 19.713714
> Method2: 18.455919
>
> Do you have an idea how to make this faster? I thought about other types
> of 3D interpolation, but they all seem so "overkill" for such a simple
> problem...
>
> Thanks a lot
>
> Fab

I would do the interpolation by hand over the relevant dimension. Here's
a somewhat general solution - I haven't tested it, and I *know* it fails
on the edge cases, so check it against results you know first, but it
should be much much faster than what you're doing:

```
;+
; NAME:
;   INTERPOL_D
;
; PURPOSE:
;   Perform linear interpolation on an irregular grid, a la INTERPOL(V,
X, U), but only performing
;   the interpolation over one particular dimension.
```

```
;
; CATEGORY:
;   Math
;
; CALLING SEQUENCE:
;   Result = INTERPOL_D(V, X, U, D)
;
; INPUTS:
;   V:    Input array.
;
;   X:    Abscissae values for dimension D of array V. Must have the
same number
;         of elements as the appropriate dimension of V.
;
;   U:    Abscissae values for the result. The result will have the same
number of
;         elements as U in dimension D.
;
;   D:    Dimension over which to interpolate, starting at 1.
;
; OUTPUTS:
;   INTERPOL_D returns an array with the same dimensions as V except
that dimension D
;   has N_ELEMENTS(U) elements in dimension D.
;
; MODIFICATION HISTORY:
;   Written by:   Jeremy Bailin, 8 November 2011
;
;-
function interpol_d, v, x, u, d

vsize = size(v,/dimen)
nvdimen = n_elements(vsize)
nx = n_elements(x)
nu = n_elements(u)
if n_elements(d) ne 1 then message, 'D must have only one element.' else
d=d[0]  ; scalarize it

; check that vsize contains a dimension d
if d gt nvdimen then message, 'V must contain dimension D.'
; check that X has the right number of elements
if nx ne vsize[d] then message, 'X must have the same number of elements
as dimension D of array V.'

; where do U (output) values lie w/r/t X (input) values?
u_in_x_low = value_locate(x, u)
; fractional distance between u_in_x_low and u_in_x_low+1
xfrac = (u-x[u_in_x_low])/(x[u_in_x_low+1]-x[u_in_x_low])
```

```
; reform V so that dimension D is in the first dimension and all other
dimensions are in a single
; dimension afterward
all_but_d = where(indgen(nvdimen) ne d)
n_allbutd = product(/int, vsize[all_but_d])
v = reform(transpose(temporary(v), [d,all_but_d]), [nx, n_allbutd])
; do the linear interpolation
output = v[u_in_x_low,*]*(1.-xfrac) + v[u_in_x_low+1,*]*xfrac
; and reform back so that the interpolated dimension is where it should be
resorted_dimenlist = sort([d,all_but_d])
output = reform(transpose(temporary(output), resorted_dimenlist), vsize)

return, output

end
```