
Subject: Re: Another "IDL way to do this" question
Posted by [Jeremy Bailin](#) on Tue, 08 Nov 2011 21:19:30 GMT
[View Forum Message](#) <> [Reply to Message](#)

After some debugging (still will give funny results for extrapolation,
but should work perfectly for interpolation):

```
;  
;+  
; 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.  
;  
; WARNINGS:  
; Untested. Use at your own risk.  
;  
; 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
d0=d[0]-1 ; scalarize it and zero-index

; check that vsize contains a dimension d
if d0 ge nvdimen then message, 'V must contain dimension D.'
; check that X has the right number of elements
if nx ne vsize[d0] 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 d0)
n_allbutd = product(/int, vsize[all_but_d])
v = reform(transpose(temporary(v), [d0,all_but_d]), [nx, n_allbutd])
; do the linear interpolation
xfrac = rebin(xfrac, [nu, n_allbutd], /sample)
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([d0,all_but_d])
output = transpose(reform(temporary(output), [nu, vsize[all_but_d]]),
resorted_dimenlist)

; reform v back to original dimensions so that if it's used outside we
haven't clobbered it
v = reform(v, vsize)

return, output

end

```
