Subject: Re: Another "IDL way to do this" question
Posted by Jeremy Bailin on Wed, 09 Nov 2011 20:08:55 GMT
View Forum Message <> Reply to Message

On 11/9/11 3:58 AM, Fabzou wrote:
> Wow Jeremy, you produced a nice peace of code here, I will try to
> decipher it!
>
> The problem is that in my case, the abscissae values X are of the same
> dimension as V. In the third dimension case, for example, each column
> [Xn,Yn] has different pressure values, which does not work with your
> assumption that X must have the same number of elements as the dimension
> 3 in V.
>
> Your algorithm is indeed much faster (couldn't check the results, though):
>
> varin = FLTARR(200,200,27,24), interpolation on DIM3:
>
> Mine: 19.391076 sec
> JB : 0.43421888 sec
>
> Do you think I could adapt it?
>
> Thanks a lot,

Ah, I see! Sorry about that. Okay, well I'm sure I can adapt the algorithm. The trick is to effectively create a multi-dimensional version of VALUE_LOCATE. How about this (just done for your case where the dimension is 3, so it doesn't have some of the fancy dimensional footwork):

```
function value_locate_3d, source, data
  compile_opt idl2

  ssize = size(source,/dimen)
  ssize_3d1 = ssize
  ssize_3d1[2] = 1
  ndata = n_elements(data)
  dsize = ssize
  dsize[2] = ndata

  ; change source minimum to be zero (otherwise this will fail if it's
negative)
  ; and find the maximum
  minsource = min(source, dimen=3)
  source += rebin(reform(minsource, ssize_3d1), ssize, /sample)
```

```
   maxsource = max(source)*1.01
   source += rebin(findgen(ssize_3d1)*maxsource, ssize, /sample)

   ; now do the same to data and then use value_locate
   data_uniq = rebin(reform(data, 1,1,ndata,1), dsize, /sample) + $
     rebin(reform(minsource, ssize_3d1) + findgen(ssize_3d1)*maxsource,
dsize, /sample)
   data_in_source_uniq = value_locate(source, temporary(data_uniq))

   ; restore source
   source -= rebin(reform(minsource, ssize_3d1), ssize, /sample) + $
     rebin(findgen(ssize_3d1)*maxsource, ssize, /sample)

   ; and turn into a 1D index into the third dimension
   return, reform((array_indices(source,
temporary(data_in_source_uniq)))[2,*], dsize)
end



function interpol_3d_jb, varin, z_in, loc_param

   compile_opt idl2

   vsize = size(varin, /dimen)
   outsize = vsize
   outsize[2] = n_elements(loc_param)

   ; where do output values lie w/r/t input values
   loc_in_z  = value_locate_3d(z_in, loc_param)
   outsize_3d1 = outsize
   outsize_3d1[[0,1,3]] = 1
   noutput = n_elements(loc_in_z)

   ; prepare indices
   index1 = rebin(indgen(vsize[0],1,1,1),outsize,/samp)
   index2 = rebin(indgen(1,vsize[1],1,1),outsize,/samp)
   index4 = rebin(indgen(1,1,1,vsize[3]),outsize,/samp)

   ; fractional difference between locations and z_in
   xfrac = (rebin(reform(loc_param,outsize_3d1),outsize,/sample) -
z_in[index1,index2,loc_in_z,index4]) / $
     (z_in[index1,index2,loc_in_z+1,index4] -
z_in[index1,index2,loc_in_z,index4])

   ; do the linear interpolation
   output = varin[index1,index2,loc_in_z,index4]*(1.-xfrac) + $
```

varin[index1,index2,loc_in_z+1,index4]*xfrac

return, output

end

-Jeremy.
