Subject: Re: hist_nd question
Posted by Wox on Mon, 05 Mar 2007 15:42:35 GMT
View Forum Message <> Reply to Message

Ok, it's not so simple. The problem is it can fall outside the
boundries for 1 dimension (-1 or nbins[i]) and inside for others.
Since the result is summed to h, expanding the boundries with 1 will
not work. Any idea how to do this without using a where in the for
loop?

On Mon, 05 Mar 2007 16:17:09 +0100, Wox <nomail@hotmail.com> wrote:

> The fix is simple: change two lines.
>
> h=(nbins[s[0]-1]-1)<long((V[s[0]-1,*]-mn[s[0]-1])/bs[s[0]-1]) >0L
> =>
> h=nbins[s[0]-1]<long((V[s[0]-1,*]-mn[s[0]-1])/bs[s[0]-1])>-1L
>
> h=nbins[i]*h+((nbins[i]-1)<long((V[i,*]-mn[i])/bs[i])>0L)
> =>
> h=nbins[i]*h+(nbins[i]<long((V[i,*]-mn[i])/bs[i])>-1L)
>
>
>
> But since this function was written be JD and presented on Fanning's
> website, I obviously started doubting myself :-). Hence the question:
> is this a bug or a feature? And by changing these two lines, are there
> negative consequences which I overlooked?

Subject: Re: hist_nd question
Posted by Wox on Mon, 05 Mar 2007 16:05:28 GMT
View Forum Message <> Reply to Message

On Mon, 05 Mar 2007 16:42:35 +0100, Wox <nomail@hotmail.com> wrote:

> Ok, it's not so simple. The problem is it can fall outside the
> boundries for 1 dimension (-1 or nbins[i]) and inside for others.
> Since the result is summed to h, expanding the boundries with 1 will
> not work. Any idea how to do this without using a where in the for
> loop?

This is possible, but maybe there is a more efficient way?

<snip>

```
total_bins=nbins[s[0]-1]      ;Accumulate the size of all bins
```

```
  h=long((V[s[0]-1,*]-mn[s[0]-1])/bs[s[0]-1]) ; 1-dimensional index
  hb = h gt -1L ; true: point in range
  hb and= h lt nbins[s[0]-1]
  for i=s[0]-2,0,-1 do begin
     ;; The scaled indices, s[n]+a[n-1]*(s[n-1]+a[n-2]*(s[n-2]+...
     h*=nbins[i]
     hadd=long((V[i,*]-mn[i])/bs[i])
     hb and= hadd gt -1L
     hb and= hadd lt nbins[i]
     h+=hadd
     total_bins*=nbins[i]
  endfor
  ; set points not in range to -1
  h*=hb
  h-= ~hb
```

<snip>

Subject: Re: hist_nd question
Posted by JD Smith on Mon, 05 Mar 2007 19:16:22 GMT
View Forum Message <> Reply to Message

On Mon, 05 Mar 2007 16:17:09 +0100, Wox wrote:

> Hi all,
>
> I have been using JD's hist_nd before, but I just found something strange.
> Maybe this is already corrected somewhere or maybe this is a "feature",
> but N-dimensional points falling outside of the given min/max boundries,
> are mapped to the first and the last bin. To clarify:

Good catch. The problem is HIST_ND assumes that the 1D indices are unique, such that things outside the bounds of one axis "wrap into" the next dimension. This is not a problem if you let the code derive the N-dimensional bounding box itself, but if you hand specify a range with MIN/MAX, you can feel this issue. HIST_2D solves this by brute force setting to -1 any location which falls outside the bounds. I've modified HIST_ND to do something similar. Please find (and test) the updated version here:

turtle.as.arizona.edu/idl/hist_nd.pro

This version requires IDL v6.1 or later.

An interesting issue arose while making these changes. Comparing the newly slimmed-down loop-based version of the single index creation:

```
h=long((V[s[0]-1,*]-mn[s[0]-1])/bs[s[0]-1])
for i=s[0]-2,0,-1 do h=nbins[i]*h + long((V[i,*]-mn[i])/bs[i])
```

to the fully threaded, loop-free version:

```
h=total(long((V-rebin(mn,s,/SAMP))/rebin(bs,s,/SAMP)) * $
        rebin([1L,product(nbins[0:s[0]-2],/CUMULATIVE,/PRESERVE_TYPE )], $
            s,/SAMP), 1,/PRESERVE_TYPE)
```

I found the former to perform better by 20% or more for large arrays, which results purely from the overhead of REBIN'ing small arrays 3 times, to the size of the input array (e.g. 3 x 5 million for a large array), with it's resulting greater demands on memory. As is often pointed out, small loops which perform lots of work per iteration do not feel the loop penalty, and depending on the exact memory requirements, can actually excel over loop-free methods (yes, you hear me saying this).

JD

Subject: Re: hist_nd question
Posted by David Fanning on Mon, 05 Mar 2007 19:35:33 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> As is often
> pointed out, small loops which perform lots of work per iteration do
> not feel the loop penalty, and depending on the exact memory
> requirements, can actually excel over loop-free methods (yes, you hear
> me saying this).

Craig must be rolling over in his grave. Or is he still out there? Haven't heard from him in a while, it seems. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: hist_nd question
Posted by Kenneth Bowman on Mon, 05 Mar 2007 19:59:03 GMT
View Forum Message <> Reply to Message

In article <pan.2007.03.05.19.16.22.355096@as.arizona.edu>,
 JD Smith <jdsmith@as.arizona.edu> wrote:

> As is often
> pointed out, small loops which perform lots of work per iteration do
> not feel the loop penalty, and depending on the exact memory
> requirements, can actually excel over loop-free methods (yes, you hear
> me saying this).

An old programming rule of thumb I learned when writing Fortran goes:

  "Optimize the innermost loop first."

Cheers, Ken

Subject: Re: hist_nd question
Posted by JD Smith on Mon, 05 Mar 2007 20:49:52 GMT
View Forum Message <> Reply to Message

On Mon, 05 Mar 2007 13:59:03 -0600, Kenneth Bowman wrote:

> In article <pan.2007.03.05.19.16.22.355096@as.arizona.edu>,
> JD Smith <jdsmith@as.arizona.edu> wrote:
>
>> As is often
>> pointed out, small loops which perform lots of work per iteration do
>> not feel the loop penalty, and depending on the exact memory
>> requirements, can actually excel over loop-free methods (yes, you hear
>> me saying this).
>
> An old programming rule of thumb I learned when writing Fortran goes:
>
> "Optimize the innermost loop first."

Right. The difference being that there is no "loop overhead" in
Fortran, so the place to optimize is often harder to intuit with IDL.
My rules of thumb for IDL optimization are:

0. If possible, rethink the algorithm to use vastly fewer resources,
  and never optimize until you've profiled.

1. Minimize the IDL loop overhead by ensuring your innermost loop
  calculation takes at least 1ms (machine dependent).

2. Recast problems to use arrays up to ~1/2 your total physical
   memory.

Sometimes when you don't know how big a problem is going to be, sticking with a solid loop method with lower memory requirements will be the best approach. More commonly, however, rule #2 takes precedence. This is especially true on modern multi-processor systems, where large arrays can be attacked with multiple threads automatically by IDL.

Here's how one might measure the loop overhead:

```
n=10000

a=make_array(VALUE=1.,n+1)
t=systime(1)
for i=1L,n do begin
  a[i]+=a[i-1]
endfor
tl=systime(1)-t

a=make_array(VALUE=1.,n+1)
t=systime(1)
a=total(a,/CUMULATIVE)
tv=systime(1)-t

print,'Loop version:   ',tl
print,'Vector version: ',tv
print,'Loop overhead:  ',(tl-tv)/n

END
```

For me it's roughly 1ms. Much of that overhead arises from creating and operating on individual temporary variables (which are fairly heavyweight in IDL), vs. spinning through a large array in one shot, so it should more properly be called the "loop/variable overhead".

JD

Subject: Re: hist_nd question
Posted by Wox on Tue, 06 Mar 2007 09:42:07 GMT
View Forum Message <> Reply to Message

On Mon, 05 Mar 2007 12:16:22 -0700, JD Smith <jdsmith@as.arizona.edu>
wrote:

<snip>
> I've
> modified HIST_ND to do something similar. Please find (and test) the
> updated version here:
>
> turtle.as.arizona.edu/idl/hist_nd.pro
<snip>

Seems to work as expected. Thanks!

Btw, I'm using it for making SetIntersection multi-dimensional.
Something like this:

```
IDL> print,a
      -1       0       0
       1       0       3
       1       1       1
       1       0       3
IDL> print,b
       1       0       3
       1       1       2
IDL> print,SetIntersection( a, b, /nd, /indices)
          1           3
```

So this is an intersection in 3D and point 1 and 3 from a are in the
intersection.

Although this might not be a good idea when the points in the
intersection are far from eachother. Any thoughts?

Subject: Re: hist_nd question
Posted by Craig Markwardt on Wed, 07 Mar 2007 14:17:11 GMT
View Forum Message <> Reply to Message

David Fanning <news@dfanning.com> writes:
> JD Smith writes:
>
>> As is often
>> pointed out, small loops which perform lots of work per iteration do
>> not feel the loop penalty, and depending on the exact memory
>> requirements, can actually excel over loop-free methods (yes, you hear
>> me saying this).
>
> Craig must be rolling over in his grave. Or is he
> still out there? Haven't heard from him in a while,
> it seems. :-)

Rolling over in my grave? Reports of my death are a bit premature.
One of the people that "often points out" that loops that do a lot of
work per iteration are OK... is me :-)

Yes, I'm reading the IDL newsgroup less these days, sorry. :-(
I have a whole other world of code to puzzle about.

Craig

--
-- --------------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
-- --------------