
Subject: Re: rebin question

Posted by [JD Smith](#) on Fri, 22 Mar 2002 19:49:11 GMT

[View Forum Message](#) <> [Reply to Message](#)

Jonathan Joseph wrote:

- >
- > It looks nice doesn't it, and I did write a procedure for the simple
- > case of averaging, but it's not as clean cut as you indicate:
- >
- > 1. first one needs to get the type of the incoming image - you don't
- > want to round the result of a floating point type image - that
- > would give you the wrong result.
- >
- > 2. conversion should be done to double precision floating point
- > (not float) otherwise large long integers will lose precision.
- > loss of precision for large L64 integers will occur even with
- > conversion to double, so they can't be handled properly at all.

Hi JJ,

Since you couldn't walk down the hall to bug me... ;)

This argument is a bit off. When you work in integer precision, all operations occur as integer arithmetic. Thus, your original rebin example of $(5+5+5+5+4)/5=24/5=4$ is an exactly correct integer calculation. REBIN doesn't "averages the pixels, but then instead of rounding to the nearest integer, simply take the integer part of the average", it performs arithmetic at the precision of its inputs. Integer arithmetic truncates, not rounds (try `print,4/5`). You seem to want REBIN to switch back and forth between numeric types (in the way you could do with `float()` and `int()`).

A better illustration is:

```
IDL> print,rebin([[4LL],replicate(5LL,4)],1)
      4
IDL> print,total(replicate(1000000000000000000ULL,1))
1.0000000e+19
IDL> print,total(replicate(1000000000000000000ULL,2))
1.5532559e+18
IDL> print,rebin(replicate(1000000000000000000ULL,2),1),format='(G)'
7.766279631452242E+17
```

Uhh ohh, overflow, but:

```
IDL> print,rebin(double(replicate(1000000000000000000ULL,2)),1),
format='(G)'
```

```
1.0000000000000000E+19
```

OK, that worked, now how about:

```
IDL> print, rebin(double(replicate(1000000000000000002ULL, 2)), 1),  
      format='(E30.22)'  
1.0000000000000000E+19
```

Hmm, we lost that 2: insufficient precision rearing it's ugly head.

All of these are also using correct (Long-64) integer arithmetic. The fact that you can't average together large 64-bit numbers without loss of precision is not a problem with rebin, but with the number representation itself. There simply isn't a big enough floating point type into which to fit this huge integer without loss of precision, and "rounding" is not a defined operation on integer types (if it were, we wouldn't need floats!).

- > 3. need to convert back to the proper type, so your solution
- > should be wrapped by a fix(..., type=type)
- >
- > 4. instead of a rebin, there is now a rebin, two type conversions
- > and a round, which will slow things down and use more memory.
- >

Yes, but these are all essential in your scheme. There's no free lunch. If you'd prefer REBIN to handle all this type conversion itself, it would be hidden from you, but would still suffer the same speed-penalty.

Confer the behavior of total(), which automatically upconverts everything to float() or double(), to avoid overflow (curiously, it didn't quite succeed in one of the examples above). REBIN could do the exact same thing, in the exact same way, but I for one am glad it doesn't. Sometimes I *want* integer arithmetic.

- > So, it is a hassle.

Think of it as an opportunity.

- > But yes, it's still not difficult to write a function to handle the
- > SIMPLE case of averaging for CERTAIN data types. But that does not
- > help with the problem of writing a more general function that handles
- > downsampling using median or downsampling using a mean excluding
- > outliers (pixels with values far from the mean) or downsampling using
- > your favorite method. Doing this quickly in IDL means doing it
- > w/o loops, so while conceptually the problem is not difficult, it
- > does seem somewhat more difficult to do it properly in IDL.

We had a discussion on just this a week or so ago. I have a DLM called "reduce" which does single-dimension reduction, ala `total(array,dimension)`, but with your choice of method (max/min/median/mean/clipped mean/etc.). This could be generalized quite easily to two different swiss-army tools:

1. A smooth/convol-equivalent (preserve size, apply filter).
2. A rebin-equivalent (reduce size).

In fact, a single tool could probably do all three at once. Of course, DLM's are a hassle.

> Anyone out there thought about this problem before?

I think people have pushed up against this problem throughout the history of computing. Usually it's best to spend time reviewing how computers store and manipulate integers and floats. While it is certainly possible to write code which handles arbitrary precision, the tremendous operational overheads of these schemes would have you screaming for your fixed-width ints and floats. It's a tradeoff between speed and flexibility, and it's one we have to work around.

JD

>

> Vince wrote:

>>

>> `print, round(rebin(float([5,5,5,5,4]),1))`

>>

>> Hassle?

>>

>> Maybe you could write a function. Which leads me to a new question:

>>

>> Is it possible to define a function or procedure in IDL that can take an arbitrary number of arguments, e.g.:

>>

>> `function my_rebin, a, arg1, arg2, ...`

>>

>> `return, round( rebin( float(a), arg1, arg2, ... ) )`

>> `end`

>>

>> On Fri, 22 Mar 2002 11:58:41 -0500, Jonathan Joseph <jj21@cornell.edu>

>> wrote:

>>

>>> I figured I would use rebin to downsample an image by averaging the
>>> pixels in blocks of specified size. What I discovered, was that for
>>> integer type images, rebin averages the pixels, but then instead of

```
>>> rounding to the nearest integer, simply takes the integer part of
>>> the average. Hence:
>>>
>>> print, rebin([5,5,5,5,4], 1)
>>>
>>> gives the result of 4, not 5 which is what I would like. I suppose
>>> this is done for speed - to work around the problem, I need to convert
>>> to a floating point type, do the rebin, then round, then convert back
>>> to the proper integer type - a hassle.
>>>
>>> But, I would really like a more generic way of doing downsampling
>>> of this sort, without the high overhead of a loop. Apart from
>>> taking the mean of a block of pixels, I would also like the option
>>> of downsampling using the median of a block of pixels, or using the
>>> mean of a block of pixels disregarding the farthest outlier (or
>>> n outliers).
>>>
>>> Has anyone written IDL code to do downsampling in a more generalized
>>> way than rebin, or have any clever ideas about how to do it quickly?
>>>
>>> Thanks
```
