Subject: REBIN Question
Posted by pford on Sat, 14 Mar 1998 08:00:00 GMT
View Forum Message <> Reply to Message

REBIN question

Either I have uncovered a bug in REBIN in the Mac version of IDL V 5.0.3 or I donï¿½t fully understand how REBIN works. I want to take a 2-D byte array at_target that is 64X64 in size and make it into a 1-D byte array with the same number of elements and vis versa. The results are not what I am expecting so I used the code below to test it. The displayed images are not even close to each other.

Would someone be kind enough to explain why and how I can do this other than using the code below(test2) the offending section.

Thanks.

```
pro test
window,5,xsize= 128, ysize = 128
window,6,xsize= 128, ysize = 128

at_target= bytarr(64,64)
at_target(0:63,0:63) = 255B
at_target(10:20,10:20) = 200B;

wset,5
tvscl, at_target
wset,6
tvscl, rebin(rebin(at_target, 64*64),64,64)
stop
end

pro test2

window,5,xsize= 128, ysize = 128
window,6,xsize= 128, ysize = 128

at_target = bytarr(64,64)
at_target2 = bytarr(64,64)
at_target(0:63,0:63) = 255B
at_target(10:20,10:20) = 200B;

d1target = btarr(64*64)
n=0
for x=0,63 do begin
```

```
   for y=0,63 do begin
      d1target[n] = at_target[x,y]
      at_target2 [x,y] = d1target[n]
      n = n + 1
   endfor;   for y=0,63 do begin
endfor;  for x=0,63 do begin

wset,5
tvscl, at_target
wset,6
tvscl, at_target2
stop
end
```

--
Patrick Ford
pford@bcm.tmc.edu

Subject: Re: REBIN Question
Posted by Armand J. L. Jongen on Tue, 17 Mar 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Hi Patrick

Patrick Ford, MD wrote:
>
> REBIN question
>
> Either I have uncovered a bug in REBIN in the Mac version of IDL V 5.0.3
> or I don¹t fully understand how REBIN works. I want to take a 2-D byte
> array at_target that is 64X64 in size and make it into a 1-D byte array
> with the same number of elements and vis versa. The results are not what
> I am expecting so I used the code below to test it. The displayed images
> are not even close to each other.
>
> Would someone be kind enough to explain why and how I can do this other
> than using the code below(test2) the offending section.
>
> Thanks.
>

I finally understand what you want to do and what is happening. The
trick is
that rebin does change the contents of the array by bilineair
interpolation when
maximizing a dimension and neighbourhood averaging when minimizing. By
doing

rebin(at_target, 64*64) on a bytarr(64,64) you rescale this array,
thus getting a bytarr(4096,1). BUT! Rebin uses neighborhood averaging
whereby
your code:

> at_target= bytarr(64,64)
> at_target(0:63,0:63) = 255B
> at_target(10:20,10:20) = 200B;

produces a bytarr(4096,1) with roughly

at_target(640:1280,1) EQ 200B

If you then again use rebin(at_target,64,64) this image will be
stretched
in the second dimension whereby making at_target(10:20,*) EQ 200B. So
instead
of a square you end up with a line!

This is not what you want to do. You should use REFORM instead which
will only
change the way in which the array-elements are indexed and NOT alter the
actual
data. Doing this in both instances will give the desired result.

```
pro test
window,5,xsize= 128, ysize = 128
window,6,xsize= 128, ysize = 128

at_target= bytarr(64,64)
at_target(0:63,0:63) = 255B
at_target(10:20,10:20) = 200B;

wset,5
tvscl, at_target
wset,6
; REBIN modifies the data
; tvscl, rebin(rebin(at_target, 64*64),64,64)
; REFORM does NOT modify the data
tvscl, reform(reform(at_target, 64*64),64,64)
end
```

Hope this makes things a bit clear. Cheers,

Armand

--
********** **********

Armand J.L. Jongen Academic Medical Centre
Laser Centre
Phone +31-20-5667418 \\||||// Meibergdreef 9
Fax +31-20-6975594 | ~ ~ | 1105 AZ Amsterdam
E-mail a.j.jongen@amc.uva.nl [| o o |] The Netherlands
**********o00o***(__)***o00o********** **********

Subject: Re: REBIN Question
Posted by pford on Wed, 18 Mar 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Thanks to all who replied. I see that I need to use reform, but now I
understand why rebin was giving the "odd" results.

Patrick Ford, MD
pford@bcm.tmc.edu

Armand J.L.Jongen (a.j.jongen@amc.uva.nl) wrote:
: Hi Patrick

: Patrick Ford, MD wrote:
: > =

: > REBIN question
: > =

: > Either I have uncovered a bug in REBIN in the Mac version of IDL V 5.0.=
: 3
: > or I don=B9t fully understand how REBIN works. I want to take a 2-D by=
: te
: > array at_target that is 64X64 in size and make it into a 1-D byte array=

: > with the same number of elements and vis versa. The results are not wh=
: at
: > I am expecting so I used the code below to test it. The displayed image=
: s
: > are not even close to each other.
: > =

: > Would someone be kind enough to explain why and how I can do this other=

: > than using the code below(test2) the offending section.
: > =

: > Thanks.
: > =

: I finally understand what you want to do and what is happening. The
: trick is =

: that rebin does change the contents of the array by bilineair
: interpolation when
: maximizing a dimension and neighbourhood averaging when minimizing. By
: doing =

: rebin(at_target, 64*64) on a bytarr(64,64) you rescale this array, =

: thus getting a bytarr(4096,1). BUT! Rebin uses neighborhood averaging
: whereby
: your code:

: > at_target=3D bytarr(64,64)
: > at_target(0:63,0:63) =3D 255B
: > at_target(10:20,10:20) =3D 200B;

: produces a bytarr(4096,1) with roughly

: at_target(640:1280,1) EQ 200B

: If you then again use rebin(at_target,64,64) this image will be
: stretched
: in the second dimension whereby making at_target(10:20,*) EQ 200B. So
: instead
: of a square you end up with a line!

: This is not what you want to do. You should use REFORM instead which
: will only
: change the way in which the array-elements are indexed and NOT alter the
: actual
: data. Doing this in both instances will give the desired result.

: pro test
: window,5,xsize=3D 128, ysize =3D 128
: window,6,xsize=3D 128, ysize =3D 128
: =

: at_target=3D bytarr(64,64)
: at_target(0:63,0:63) =3D 255B
: at_target(10:20,10:20) =3D 200B;
: =

: wset,5
: tvscl, at_target

: wset,6
: ; REBIN modifies the data
: ; tvscl, rebin(rebin(at_target, 64*64),64,64)
: ; REFORM does NOT modify the data
: tvscl, reform(reform(at_target, 64*64),64,64)
: end

: Hope this makes things a bit clear. Cheers,

: Armand

: -- =

: ** ************
: Armand J.L. Jongen   Academic Medical Centre
:   Laser Centre
: Phone   +31-20-5667418   \\||||//   Meibergdreef 9
: Fax   +31-20-6975594 | ~ ~ | 1105 AZ Amsterdam
: E-mail a.j.jongen@amc.uva.nl   [| o o |]   The Netherlands
: ****************************o00o***(__)***o00o************* ************

Subject: Re: rebin question
Posted by hradilv.nospam on Fri, 22 Mar 2002 17:11:17 GMT
View Forum Message <> Reply to Message

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

Subject: Re: rebin question
Posted by Craig Markwardt on Fri, 22 Mar 2002 18:31:18 GMT
View Forum Message <> Reply to Message

hradilv.nospam@yahoo.com (Vince) writes:

> 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

Answer: no, but I've wanted one for a long time.

Craig

--
 ---------- ----------

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

Subject: Re: rebin question
Posted by Jonathan Joseph on Fri, 22 Mar 2002 18:40:32 GMT
View Forum Message <> Reply to Message

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.

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.

So, it is a hassle.

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.

Anyone out there thought about this problem before?

-Jonathan

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

Subject: Re: rebin question
Posted by hradilv.nospam on Fri, 22 Mar 2002 18:53:11 GMT
View Forum Message <> Reply to Message

Sorry. I see the hassle now.

I just did a quick search for 'rebin' at
http://www.astro.washington.edu/deutsch/idl/htmlhelp/index.h tml and
maybe you should take a look at 'boxave'?

On Fri, 22 Mar 2002 13:40:32 -0500, Jonathan Joseph <jj21@cornell.edu>
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.
>
> 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.
>
> So, it is a hassle.
>
> 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.
>
> Anyone out there thought about this problem before?
>
> -Jonathan
>
> 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

Subject: Re: rebin question
Posted by Jonathan Joseph on Fri, 22 Mar 2002 18:53:30 GMT
View Forum Message <> Reply to Message

You could certainly write a function that takes an arbitrary number
of arguments up to a limit, then use n_params to see how many you
actually got. It looks like IDL only allows up to 8-dimensional
arrays (which is a goodly number). I only wrote the 2-d case,
but the n-d case should be possible.

-Jonathan

Craig Markwardt wrote:

>
> hradilv.nospam@yahoo.com (Vince) writes:
>
>> 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
>
> Answer: no, but I've wanted one for a long time.
>
> Craig
>
> --
> ---------- ----------
> Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
> Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
> ---------- ----------

Subject: Re: rebin question
Posted by David Fanning on Fri, 22 Mar 2002 19:14:32 GMT
View Forum Message <> Reply to Message

Jonathan Joseph (jj21@cornell.edu) writes:

> You could certainly write a function that takes an arbitrary number
> of arguments up to a limit, then use n_params to see how many you
> actually got. It looks like IDL only allows up to 8-dimensional
> arrays (which is a goodly number). I only wrote the 2-d case,
> but the n-d case should be possible.

I'm not sure "arbitrary" is the right word here.
One can write procedures and functions with a "variable"
number of arguments, up to some arbitrary (to RSI)
number of 100 or 200 or whatever it is. But each
variable *must* be defined on the procedure or function
definition line.

In the case of the function under discussion, you

could define 8 arguments (the first would probably
have to be a required argument, and the rest could
be optional). But you will need some kind of CASE
statement to call REBIN correctly:

```
  CASE N_PARAMS() OF
0: Message, 'Whoops> Wrong!"
     1: A = REBIN(arg1)
2: A = REBIN(arg1, arg2)
    ...
  ENDCASE
```

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: rebin question
Posted by Jonathan Joseph on Fri, 22 Mar 2002 19:20:54 GMT
View Forum Message <> Reply to Message

Thanks, that's seems more reasonable than what I was thinking.
It still uses a loop, but only over the number of elements in the
box. I think I can definitely adapt this strategy to my needs.

-Jonathan

Vince wrote:
>
> Sorry. I see the hassle now.
>
> I just did a quick search for 'rebin' at
>   http://www.astro.washington.edu/deutsch/idl/htmlhelp/index.h tml and
> maybe you should take a look at 'boxave'?
>
> On Fri, 22 Mar 2002 13:40:32 -0500, Jonathan Joseph <jj21@cornell.edu>
> 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.
>>
>> 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.
>>
>> So, it is a hassle.
>>
>> 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.
>>
>> Anyone out there thought about this problem before?
>>
>> -Jonathan
>>
>> 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

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(10000000000000000000ULL,1))
   1.0000000e+19
IDL> print,total(replicate(10000000000000000000ULL,2))
   1.5532559e+18
IDL>   print,rebin(replicate(10000000000000000000ULL,2),1),format=' (G)'
    7.766279631452242E+17
```

Uhh ohh, overflow, but:

```
IDL>  print,rebin(double(replicate(10000000000000000000ULL,2)),1),
 format='(G)'
    1.000000000000000E+19
```

OK, that worked, now how about:

```
IDL>  print,rebin(double(replicate(10000000000000000002ULL,2)),1),
        format='(E30.22)'
  1.0000000000000000000000E+19
```

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

All of these are also using correct (Long-64) integer arithmethic. 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 thorughout 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

Subject: Re: rebin question
Posted by Mark Fardal on Mon, 25 Mar 2002 06:50:21 GMT
View Forum Message <> Reply to Message

hradilv.nospam@yahoo.com (Vince) writes:

> 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

Sure. Something like this:

```
function my_rebin, a, $
  arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9
;continue to heart's content

narg = n_params()-1
command = 'result = round( rebin( float(a)'
for i = 0, narg-1 do begin
  command = command + string(', arg',i,format='(a,i0)')
endfor
command = command + ') )'
junk = execute(command)
return, result
end
```

I imagine there are limits placed on this technique by the maximum number of arguments to an IDL procedure, or the maximum length of a string EXECUTE can handle.

Mark Fardal
University of Victoria

Subject: Re: rebin question
Posted by Mark Hadfield on Mon, 25 Mar 2002 22:07:38 GMT
View Forum Message <> Reply to Message

"Mark Fardal" <fardal@coral.phys.uvic.ca> wrote in message news:yj1zo0xta36.fsf@coral.phys.uvic.ca...
> hradilv.nospam@yahoo.com (Vince) writes:

>> Is it possible to define a function or procedure in IDL that can take
>> an arbitrary number of arguments...
>
> Sure. Something like this:
>
> function my_rebin, a, $
>   arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9
> ;continue to heart's content
>
> narg = n_params()-1
> command = 'result = round(rebin(float(a)'
> for i = 0, narg-1 do begin
>   command = command + string(', arg',i,format='(a,i0)')
> endfor
> command = command + '))'
> junk = execute(command)
> return, result
> end
>
> I imagine there are limits placed on this technique by the maximum
> number of arguments to an IDL procedure, or the maximum length of a
> string EXECUTE can handle.

The maximum number of arguments to an IDL procedure is far higher than
any sane programmer would want to use. The maximum length of an
EXECUTE'd string used to be a significant limitation in earlier
versions of IDL (I think it was 64, later 256) but in 5.4 or 5.5 it
was also increased to an effectively infinite value.

When writing wrapper procedures of this sort (which I do quite
often, for various reasons) I prefer to use this form

```
pro myfoo, p1, p2, ...
   case n_params() of
      0: foo
      1: foo, p1
      2: foo, p1, p2
      ...
   endcase
end
```

I have gone as far as handling 15 positional parameters, but I don't
think I've ever used more than 4.

--
Mark Hadfield
m.hadfield@niwa.co.nz   Ka puwaha et tai nei
http://katipo.niwa.co.nz/~hadfield   Hoea tatou

Subject: Re: rebin question
Posted by Jonathan Joseph on Mon, 25 Mar 2002 22:24:27 GMT
View Forum Message <> Reply to Message

Hi JD,

Now that you are no longer conveniently located, I wonder which takes more of your time: The old going down to your office and getting an explanation from you directly, or posting to this group and you constructing a detailed response :) I guess this way everyone gets the benefit of your words of wisdom.

-Jonathan

JD Smith wrote:
>
> 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(10000000000000000000ULL,1))
>   1.0000000e+19
> IDL> print,total(replicate(10000000000000000000ULL,2))
>   1.5532559e+18
> IDL>   print,rebin(replicate(10000000000000000000ULL,2),1),format=' (G)'
>   7.766279631452242E+17
>
> Uhh ohh, overflow, but:
>
> IDL> print,rebin(double(replicate(10000000000000000000ULL,2)),1),
>   format='(G)'
>   1.000000000000000E+19
>
> OK, that worked, now how about:
>
> IDL> print,rebin(double(replicate(10000000000000000002ULL,2)),1),
>   format='(E30.22)'
>   1.0000000000000000000000E+19
>
> Hmm, we lost that 2: insufficient precision rearing it's ugly head.
>
> All of these are also using correct (Long-64) integer arithmethic. 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 thorughout 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
