Subject: Re: TOTAL gives totally different result on identical array
Posted by Fabzou on Fri, 08 Jul 2011 10:34:26 GMT
View Forum Message <> Reply to Message

Funny, It looks like the post I wrote a couple of days ago. You should
have a look to this thread :

http://groups.google.com/group/comp.lang.idl-pvwave/browse_t
hread/thread/47482e565173a127/b58f3a3ba7934b93?#b58f3a3ba793 4b93

On 07/08/2011 12:22 PM, M. Suklitsch wrote:
> Hi everybody,
>
>
> I don't exactly know how to start this question, so I'll probably just
> tell you the workflow which leads to a very disturbing result.
>
> I have a netCDF file which contains temperature data for a decade on
> daily time resolution. And I have two different versions of an IDL
> based evaluation tool which should read in the data and do some stuff
> with it.
>
> Now, although I read in exactly the same data (the data array has the
> same size in both IDL sessions with both versions of my tool), TOTAL
> and MEAN give completely different results. And I do not understand
> how that can be, since the data does not contain any NaN values, and
> MIN and MAX of the array are identical in both IDL sessions. The only
> difference between the two IDL sessions are different source codes
> that lead to the point where I do the stuff below. Here is what I get:
>
> Session 1:
> =======
> IDL> ncid=NCDF_OPEN('my_input_file.nc')
> IDL> ncdf_varget, ncid, 'tas', testa
> IDL> help, mean(testa), min(testa), max(testa)
> <Expression> FLOAT = 270.284
> <Expression> FLOAT = 232.614
> <Expression> FLOAT = 317.723
> IDL> print, total(testa)/n_elements(testa)
> 270.284
> IDL> print, n_elements(testa)
> 127124400
> IDL> print, total(testa)
> 3.43597e+10
> IDL> idx=where(testa lt 275., countidx, ncomplement=countnidx)
> IDL> help, countidx, countnidx
> COUNTIDX LONG = 22074445
> COUNTNIDX LONG = 105049955

>
>
> Session 2:
> =======
> IDL> ncid=NCDF_OPEN('my_input_file.nc')
> IDL> ncdf_varget, ncid, 'tas', testa
> IDL> help, mean(testa), min(testa), max(testa)
> <Expression> FLOAT = 67.5711
> <Expression> FLOAT = 232.614
> <Expression> FLOAT = 317.723
> IDL> print, total(testa)/n_elements(testa)
> 67.5711
> IDL> print, n_elements(testa)
> 127124400
> IDL> print, total(testa)
> 8.58993e+09
> IDL> idx=where(testa lt 275., countidx, ncomplement=countnidx)
> IDL> help, countidx, countnidx
> COUNTIDX LONG = 22074445
> COUNTNIDX LONG = 105049955
>
>
> So, the values within the arrays seem to be the same (since the
> counting gives the identical number of elements), yet the TOTAL (and
> MEAN) deviate completely from each other. How can that be? I am really
> confused right now. And insecure about any results I got so far.
>
>
> Regards,
> Martin

Subject: Aw: Re: TOTAL gives totally different result on identical array
Posted by M. Suklitsch on Fri, 08 Jul 2011 12:55:38 GMT
View Forum Message <> Reply to Message

Hi Fabien,

very much the same indeed. Maybe some interesting side notes that I forgot in my original post:

IDL> help,!version,/struct
** Structure !VERSION, 8 tags, length=104, data length=100:
ARCH STRING 'x86_64'
OS STRING 'linux'
OS_FAMILY STRING 'unix'
OS_NAME STRING 'linux'
RELEASE STRING '7.1'

```
   BUILD_DATE      STRING    'Apr 21 2009'
   MEMORY_BITS     INT           64
   FILE_OFFSET_BITS
                   INT           64

IDL> help, testa
TESTA           FLOAT     = Array[174, 200, 3653]
```

So, if I get the discussion in your thread right there's "simply" a precision and/or array size problem with the builtin TOTAL function? That's not really reassuring since I am dealing with arrays of such size all the time.

Subject: Re: TOTAL gives totally different result on identical array
Posted by Liam Gumley on Fri, 08 Jul 2011 14:37:32 GMT
View Forum Message <> Reply to Message

[stuff deleted]

The online documentation for the TOTAL function in IDL warns that this may happen:

---quote begins---
You should be aware that when summing a large number of values, the result from TOTAL can depend heavily upon the order in which the numbers are added. Since the thread pool will add values in a different order, you may obtain a different — but equally correct — result than that obtained using the standard non-threaded implementation. This effect occurs because TOTAL uses floating point arithmetic, and the mantissa of a floating point value has a fixed number of significant digits. The effect is especially obvious when using single precision arithmetic, but can also affect double precision computations. Such differences do not mean that the sums are incorrect. Rather, they mean that they are equal within the ability of the floating point representation used to represent them.
---end quote---

In a previous posting, someone mentioned the Kahan algorithm for computing a numerically stable summation:

http://en.wikipedia.org/wiki/Kahan_summation_algorithm

A translation to IDL looks like this:

```
function kahansum, input
sum = 0.0
c = 0.0
```

```
for i = 0L, n_elements(input) - 1 do begin
  y = input[i] - c
  t = sum + y
  c = (t - sum) - y
  sum = t
endfor
return, sum
end
```

The IDL documentation for TOTAL includes the following example for demonstrating the impact of array ordering on summation:

```
IDL> vec = FINDGEN(100000)
IDL> PRINT, TOTAL(vec) - TOTAL(REVERSE(vec))
    -87552.0
```

However the Kahan summation, while considerably slower, does the summation with much less error:

```
IDL> PRINT, kahansum(vec) - kahansum(reverse(vec))
     0.00000
```

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/

Subject: Re: TOTAL gives totally different result on identical array
Posted by Foldy Lajos on Fri, 08 Jul 2011 15:44:39 GMT
View Forum Message <> Reply to Message

On Fri, 8 Jul 2011, Liam Gumley wrote:

```
>  function kahansum, input
>  sum = 0.0
>  c = 0.0
>  for i = 0L, n_elements(input) - 1 do begin
>   y = input[i] - c
>   t = sum + y
>   c = (t - sum) - y
>   sum = t
>  endfor
>  return, sum
>  end
```

It will be very slow. But it's IDL, vectorize it!

```
; test.pro begin

function kahan_sum, input
sum=0.0
c=0.0
for i=0l, n_elements(input)-1 do begin
    y=input[i]-c
    t=sum+y
    c=(t-sum)-y
    sum=t
endfor
return, sum
end

function kahan_sum_1000, input
sum=fltarr(1000)
c=fltarr(1000)
for i=0l, n_elements(input)-1,1000 do begin
    y=input[i:i+999]-c
    t=sum+y
    c=(t-sum)-y
    sum=t
endfor
return, kahan_sum(sum)
end


pro test
vec=findgen(100000)

t1=systime(1)
s1=kahan_sum(vec)
t1=systime(1)-t1
print, 't1:', t1

t2=systime(1)
s2=kahan_sum_1000(vec)
t2=systime(1)-t2
print, 't2:', t2
print, 'ratio:', t1/t2
print, 'sums and diff:', s1, s2, s2-s1

end

; test.pro end
```

```
IDL> test
t1:     0.056817055
t2:    0.0012319088
ratio:       46.121153
sums and diff:  4.99995e+09  4.99995e+09      0.00000
IDL>
```

(The general case is left to the reader.)

regards,
Lajos

Subject: Re: TOTAL gives totally different result on identical array
Posted by Liam Gumley on Fri, 08 Jul 2011 17:25:24 GMT
View Forum Message <> Reply to Message

On Jul 8, 10:44 am, FÖLDY Lajos <fo...@rmki.kfki.hu> wrote:
> It will be very slow. But it's IDL, vectorize it!

The pairwise summation algorithm is sometimes recommended as a faster solution:

http://en.wikipedia.org/wiki/Pairwise_summation

Here is an IDL implementation (with very little testing!)

```
FUNCTION PAIRWISE_SUM, X
compile_opt idl2
forward_function pairwise_sum
np = 100
nx = n_elements(x)
if (nx le np) then begin
  ;- Naieve summation
  s = total(x, /double)
endif else begin
  ;- Divide and conquer: recursively sum two halves of the array
  m = floor(nx / 2)
  s = pairwise_sum(x[0:m-1]) + pairwise_sum(x[m:*])
endelse
return, s
END
```

Increasing the value of NP makes the algorithm faster, but potentially decreases accuracy. Here's a challenging test case:

```
PRO TEST
```

```
vec = 100 ^ (10.0 * randomu(123456, 10000000))
help, vec
print, 'MIN = ', min(vec), ' MAX = ',
max(vec)

print
print, 'TOTAL result'
print, total(vec) - total(reverse(vec))

print
print, 'TOTAL double precision result'
print, total(vec, /double) - total(reverse(vec), /double)

print
print, 'Kahan sum result'
t1 = systime(1)
result = kahansum(vec) - kahansum(reverse(vec))
t2 = systime(1)
print, result
print, 'Elapsed time (seconds) = ', t2 -
t1

print
print, 'Pairwise sum result'
t1 = systime(1)
result = pairwise_sum(vec) - pairwise_sum(reverse(vec))
t2 = systime(1)
print, result
print, 'Elapsed time (seconds) = ', t2 -
t1

END
```

Results of the test case:

```
IDL> test
% Compiled module: TEST.
VEC             FLOAT     = Array[10000000]
MIN =       1.00000 MAX =   9.99996e+19

TOTAL result
% Compiled module: REVERSE.
 -2.30584e+18

TOTAL double precision result
  -2.5769804e+10
```

Kahan sum result
% Compiled module: KAHANSUM.
      0.00000
Elapsed time (seconds) =   5.7468300

Pairwise sum result
% Compiled module: PAIRWISE_SUM.
       0.0000000
Elapsed time (seconds) =   0.89422798

Cheers,
Liam.
Practical IDL Programming (in print for 10 years!)
http://www.gumley.com/

Subject: Re: TOTAL gives totally different result on identical array
Posted by Fabzou on Fri, 08 Jul 2011 17:46:43 GMT
View Forum Message <> Reply to Message

Thank you very much for these functions.

Do you suggest to use PAIRWISE_SUM systematically? Is it possible to
make a generic function such as:

function robust_total, x

   if (*statement to define*) then return, PAIRWISE_SUM(X) $
     else return, total(X)

end

Thank you for your help!

Fabien

On 07/08/2011 07:25 PM, Liam Gumley wrote:
> On Jul 8, 10:44 am, Fï¿½LDY Lajos<fo...@rmki.kfki.hu> wrote:
>> It will be very slow. But it's IDL, vectorize it!
>
> The pairwise summation algorithm is sometimes recommended as a faster
> solution:
>
> http://en.wikipedia.org/wiki/Pairwise_summation
>
> Here is an IDL implementation (with very little testing!)
>

```
>  FUNCTION PAIRWISE_SUM, X
>  compile_opt idl2
>  forward_function pairwise_sum
>  np = 100
>  nx = n_elements(x)
>  if (nx le np) then begin
>     ;- Naieve summation
>     s = total(x, /double)
>  endif else begin
>     ;- Divide and conquer: recursively sum two halves of the array
>     m = floor(nx / 2)
>     s = pairwise_sum(x[0:m-1]) + pairwise_sum(x[m:*])
>  endelse
>  return, s
>  END
>
>  Increasing the value of NP makes the algorithm faster, but potentially
>  decreases accuracy. Here's a challenging test case:
>
>  PRO TEST
>
>  vec = 100 ^ (10.0 * randomu(123456, 10000000))
>  help, vec
>  print, 'MIN = ', min(vec), ' MAX = ',
>  max(vec)
>
>  print
>  print, 'TOTAL result'
>  print, total(vec) - total(reverse(vec))
>
>  print
>  print, 'TOTAL double precision result'
>  print, total(vec, /double) - total(reverse(vec), /double)
>
>  print
>  print, 'Kahan sum result'
>  t1 = systime(1)
>  result = kahansum(vec) - kahansum(reverse(vec))
>  t2 = systime(1)
>  print, result
>  print, 'Elapsed time (seconds) = ', t2 -
>  t1
>
>  print
>  print, 'Pairwise sum result'
>  t1 = systime(1)
>  result = pairwise_sum(vec) - pairwise_sum(reverse(vec))
>  t2 = systime(1)
```

> print, result
> print, 'Elapsed time (seconds) = ', t2 -
> t1
>
> END
>
> Results of the test case:
>
> IDL> test
> % Compiled module: TEST.
> VEC FLOAT = Array[10000000]
> MIN = 1.00000 MAX = 9.99996e+19
>
> TOTAL result
> % Compiled module: REVERSE.
> -2.30584e+18
>
> TOTAL double precision result
> -2.5769804e+10
>
> Kahan sum result
> % Compiled module: KAHANSUM.
> 0.00000
> Elapsed time (seconds) = 5.7468300
>
> Pairwise sum result
> % Compiled module: PAIRWISE_SUM.
> 0.0000000
> Elapsed time (seconds) = 0.89422798
>
> Cheers,
> Liam.
> Practical IDL Programming (in print for 10 years!)
> http://www.gumley.com/

Subject: Re: TOTAL gives totally different result on identical array
Posted by Foldy Lajos on Fri, 08 Jul 2011 18:33:32 GMT
View Forum Message <> Reply to Message

On Fri, 8 Jul 2011, Liam Gumley wrote:

> On Jul 8, 10:44 am, FÖLDY Lajos <fo...@rmki.kfki.hu> wrote:
>> It will be very slow. But it's IDL, vectorize it!
>
> The pairwise summation algorithm is sometimes recommended as a faster
> solution:
>

> http://en.wikipedia.org/wiki/Pairwise_summation
>
> Here is an IDL implementation (with very little testing!)
>
> FUNCTION PAIRWISE_SUM, X
> compile_opt idl2
> forward_function pairwise_sum
> np = 100
> nx = n_elements(x)
> if (nx le np) then begin
> ;- Naieve summation
> s = total(x, /double)
> endif else begin
> ;- Divide and conquer: recursively sum two halves of the array
> m = floor(nx / 2)
> s = pairwise_sum(x[0:m-1]) + pairwise_sum(x[m:*])
> endelse
> return, s
> END
>

Recursive calls are slow, you can make it much faster:

```
function chunk_sum, x
compile_opt idl2
forward_function chunk_sum
np = 100
nx = n_elements(x)
nchunk = nx / np
sums = dblarr(nchunk+1)
for j=0, nchunk-1 do sums[j] = total(x[j*np:(j+1)*np-1], /double)
left = nx - nchunk * np
if left gt 0 then sums[nchunk] = total(x[nchunk*np, *], /double)
if nchunk gt np then s=chunk_sum(sums) else s=total(sums)
return, s
end


IDL> test
Kahan sum result
      0.00000
Elapsed time (seconds) =       7.1223309

Pairwise sum result
       0.0000000
Elapsed time (seconds) =       1.1735301

Chunk sum result
```

0.0000000
Elapsed time (seconds) = 0.34169912

regards,
Lajos

Subject: Re: TOTAL gives totally different result on identical array
Posted by Liam Gumley on Fri, 08 Jul 2011 18:48:52 GMT
View Forum Message <> Reply to Message

On Jul 8, 12:46 pm, Fabzou <fabien.mauss...@tu-berlin.de> wrote:
> Thank you very much for these functions.
>
> Do you suggest to use PAIRWISE_SUM systematically? Is it possible to
> make a generic function such as:
>
> function robust_total, x
>
> if (*statement to define*) then return, PAIRWISE_SUM(X) $
> else return, total(X)
>
> end
>
> Thank you for your help!
>
> Fabien
>
> On 07/08/2011 07:25 PM, Liam Gumley wrote:
>
>
>
>> On Jul 8, 10:44 am, FÖLDY Lajos<fo...@rmki.kfki.hu> wrote:
>>> It will be very slow. But it's IDL, vectorize it!
>
>> The pairwise summation algorithm is sometimes recommended as a faster
>> solution:
>
>> http://en.wikipedia.org/wiki/Pairwise_summation
>
>> Here is an IDL implementation (with very little testing!)
>
>> FUNCTION PAIRWISE_SUM, X
>> compile_opt idl2
>> forward_function pairwise_sum
>> np = 100
>> nx = n_elements(x)

```
>>  if (nx le np) then begin
>>     ;- Naieve summation
>>     s = total(x, /double)
>>  endif else begin
>>     ;- Divide and conquer: recursively sum two halves of the array
>>     m = floor(nx / 2)
>>     s = pairwise_sum(x[0:m-1]) + pairwise_sum(x[m:*])
>>  endelse
>>  return, s
>>  END
>
>>  Increasing the value of NP makes the algorithm faster, but potentially
>>  decreases accuracy. Here's a challenging test case:
>
>>  PRO TEST
>
>>  vec = 100 ^ (10.0 * randomu(123456, 10000000))
>>  help, vec
>>  print, 'MIN = ', min(vec), ' MAX = ',
>>  max(vec)
>
>>  print
>>  print, 'TOTAL result'
>>  print, total(vec) - total(reverse(vec))
>
>>  print
>>  print, 'TOTAL double precision result'
>>  print, total(vec, /double) - total(reverse(vec), /double)
>
>>  print
>>  print, 'Kahan sum result'
>>  t1 = systime(1)
>>  result = kahansum(vec) - kahansum(reverse(vec))
>>  t2 = systime(1)
>>  print, result
>>  print, 'Elapsed time (seconds) = ', t2 -
>>  t1
>
>>  print
>>  print, 'Pairwise sum result'
>>  t1 = systime(1)
>>  result = pairwise_sum(vec) - pairwise_sum(reverse(vec))
>>  t2 = systime(1)
>>  print, result
>>  print, 'Elapsed time (seconds) = ', t2 -
>>  t1
>
>>  END
```

>
>> Results of the test case:
>
>> IDL> test
>> % Compiled module: TEST.
>> VEC FLOAT = Array[10000000]
>> MIN = 1.00000 MAX = 9.99996e+19
>
>> TOTAL result
>> % Compiled module: REVERSE.
>> -2.30584e+18
>
>> TOTAL double precision result
>> -2.5769804e+10
>
>> Kahan sum result
>> % Compiled module: KAHANSUM.
>> 0.00000
>> Elapsed time (seconds) = 5.7468300
>
>> Pairwise sum result
>> % Compiled module: PAIRWISE_SUM.
>> 0.0000000
>> Elapsed time (seconds) = 0.89422798
>
>> Cheers,
>> Liam.
>> Practical IDL Programming (in print for 10 years!)
>> http://www.gumley.com/

If you just want to compute the total over all dimensions of an array,
you could use it as a drop in replacement for TOTAL. However it would
take a bit more work to support all the optional keywords accepted by
TOTAL.

Liam.

Subject: Re: TOTAL gives totally different result on identical array
Posted by Michael Galloy on Fri, 08 Jul 2011 19:10:51 GMT
View Forum Message <> Reply to Message

On 7/8/11 12:48 PM, Liam Gumley wrote:
> If you just want to compute the total over all dimensions of an array,
> you could use it as a drop in replacement for TOTAL. However it would
> take a bit more work to support all the optional keywords accepted by
> TOTAL.

Certainly no drop in replacement for TOTAL, but a DLM implementation of the Kahan summation:

http://michaelgalloy.com/2011/07/08/dlm-implementing-kahan-s ummation.html

It doesn't implement the keywords of TOTAL, but it does support all numeric IDL types, i.e., in the manner of /PRESERVE_TYPE. It also shows how you can use C macros to implement an algorithm for multiple data types (ugly, but better than copying code).

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL, A Guide to Learning IDL: http://modernidl.idldev.com
Research Mathematician
Tech-X Corporation
