Subject: multiplication

Posted by marc schellens[1] on Tue, 28 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Is there a function like TOTAL but for multiplication.
Like the big PI symbol in mathematical notation.
Or this really something for the for loop?

I.E.

a=[1,2,3,...]

result=a[1]*a[2]*a[3]...

thanks,
marc

Subject: Re: multiplication

Posted by meron on Tue, 28 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

In article <38E0A379.34ADB7F7@wizard.net>, James Kuyper <kuyper@wizard.net> writes:
> meron@cars3.uchicago.edu wrote:
>>
>> In article <38E03BDC.868B8396@hotmail.com>, marc <m_schellens@hotmail.com> writes:
>>> Is there a function like TOTAL but for multiplication.
>>> Like the big PI symbol in mathematical notation.
>>> Or this really something for the for loop?
>>>
>>> I.E.
>>>
>>> a=[1,2,3,...]
>>>
>>> result=a[1]*a[2]*a[3]...
>>>
>> if all the elements of a are positive then you can simply do
>>
>> result = exp(total(alog(a)))
> ...
>> If some of the elements are negative, you can still handle it. do
>>
>> dum = where(a lt 0, ndum)
>> sig = (-1)^ndum
>> result = sig*exp(total(alog(abs(a))))
>
> You can't honestly be suggesting that this is a good technique?

Good? No, only not as bad as using "for".

> Ignore for a momement what happens if any element of 'a' is 0.

That's the easiest to deal with. You're already checking for presence of negative elements, can check for zeroes as well. That should be the first thing, in fact, since if even one of the elements is 0, then the result is 0 and you can dispense with the rest of the evaluation.

> That code performs two transcendental function evaluations per element
> of 'a'.

Yep, indeed.

> IDL would have to be very badly engineered (which I suppose is possible),
> for a 'for' loop to execute more slowly than your code.

Well, I run a quick test, comparing the time it takes tto evaluate the product using both methods (it run on an old Vms Alpha, somebody may want to repeat it on a more modern platform. Being lazy, I'm simply filling an array with a constant element, then doing the multiplication. Here is the output

```
IDL> speed, 1.00001, 100, 10
"for" time     =     0.0012000084 res =       1.00100
"exp-log" time =    0.00019999743 res =       1.00100

IDL> speed, 1.00001, 1000, 10
"for" time     =      0.012699997 res =       1.01006
"exp-log" time =     0.0012000084 res =       1.01006

IDL> speed, 1.00001, 10000, 10
"for" time     =       0.12589999 res =       1.10532
"exp-log" time =      0.011699998 res =       1.10532

IDL> speed, 1.00001, 100000, 10
"for" time     =        1.2583000 res =       2.72191
"exp-log" time =       0.12850000 res =       2.72198
```

The first input to SPEED is the array element, the second is the length of the array. the third is just telling SPEED how many times to repeat the test. As you can see, the above was tried for arrays with lengths ranging from 100 to 100000 and calculation using "for" loop is consistently an order of magnitude slower.

Mati Meron | "When you argue with a fool,
meron@cars.uchicago.edu | chances are he is doing just the same"

Subject: Re: multiplication
Posted by John-David T. Smith on Tue, 28 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

James Kuyper wrote:
>
> meron@cars3.uchicago.edu wrote:
>>
>> In article <38E03BDC.868B8396@hotmail.com>, marc <m_schellens@hotmail.com> writes:
>>> Is there a function like TOTAL but for multiplication.
>>> Like the big PI symbol in mathematical notation.
>>> Or this really something for the for loop?
>>>
>>> I.E.
>>>
>>> a=[1,2,3,...]
>>>
>>> result=a[1]*a[2]*a[3]...
>>>
>> if all the elements of a are positive then you can simply do
>>
>> result = exp(total(alog(a)))
> ...
>> If some of the elements are negative, you can still handle it. do
>>
>> dum = where(a lt 0, ndum)
>> sig = (-1)^ndum
>> result = sig*exp(total(alog(abs(a))))
>
> You can't honestly be suggesting that this is a good technique? Ignore
> for a momement what happens if any element of 'a' is 0. That code
> performs two transcendental function evaluations per element of 'a'. IDL
> would have to be very badly engineered (which I suppose is possible),
> for a 'for' loop to execute more slowly than your code.

Only one transcendental is computed for each a, alog(). The exp occurs on the
single value after the total. Results for a 10,000 element random floating
array finely tuned to avoid under or overflow:

```
Loop Method:

Average Time:      0.017213961
  0.00528653

Log Method:

Average Time:     0.0049092293
  0.00528580
```

4 times as fast. Suppose you'd like to do an array with 100,000 double elements... you get:

Loop Method:

```
% Loop limit expression too large for loop variable type.
  <LONG     (       99999)>.
```

Log Method:

```
Average Time:      0.050116260
  7.92382e+10
```

And if you hack it with two nested loops to avoid the loop limit error:

```
c=1. & for j=0L,n/100-1 do for k=0L,99L do c=c*a[j*100L+k]
```

you get:

Hacked Loop Method

```
Average Time:       0.30190306
      0.97063262
```

Log Method:

```
Average Time:      0.068175601
      0.97063262
```

A full 5 times faster.

And now, just for fun, the same data set, but with multiplication computed in a heavily optimized C program. The core of the C code is simply the straightforward: "for(i=0;i<N;i++) res*=a[i]"; The result:

```
Got 0.97063262 (Average Time: 0.001710 s)
```

Ouch! Another speedup of by a factor of 40!

Morals: IDL loops are pitifully slow, and you can't loop over very large arrays without trickery, and for many operations, compiled C is *significantly* faster.

JD

--
 J.D. Smith |*| WORK: (607) 255-5842
 Cornell University Dept. of Astronomy |*| (607) 255-6263

304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: multiplication
Posted by Craig Markwardt on Tue, 28 Mar 2000 08:00:00 GMT

View Forum Message <> Reply to Message

Carsten Dominik <dominik@astro.uva.nl> writes:

> Well, it depends very much on the size of the array. Loops in IDL are
> indeed very slow. Try the following: Set N to a large number
> (e.g. 10 000 000) and execute the following lines:
>
> x=fltarr(n)*0.+1.000001 & p=1 & for i=0.,1.*n_elements(x)-1 do p=p*x[i] & print,p
>
> x=fltarr(n)*0.+1.000001 & p=exp(total(alog(x)))&print,p
>
> You'll get a surprise, I promise.

One way to speed things up is to use some sort of a divide and conquer algorithm. Which is to say, divide the array into two segments and multiply them element-by-element. Keep doing this until you get down to a single element.

```
FUNCTION CMPRODUCT, ARRAY
        X = ARRAY
        N = N_ELEMENTS(X)
        WHILE N GT 1 DO BEGIN
            IF (N MOD 2) EQ 1 THEN X(0) = X(0) * X(N-1)  ;; When N is odd!!
            N2 = FLOOR(N/2)
            X = X(0:N2-1) * X(N2:*)  ;; Don't worry if N is odd here.

            ;; X keeps shrinking by a factor of two each time
            N = N2
        ENDWHILE
 RETURN,X(0)
END
```

Disadvantages are that it may be slower when n_elements(array) is small. Also, the round-off error can grow to significance, as I think Carsten was trying to say, but this will happen with most approaches unfortunately. Double precision can help.

Craig

--

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

Subject: Re: multiplication
Posted by Carsten Dominik on Tue, 28 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

>>>> > "JK" == James Kuyper <kuyper@wizard.net> writes:

JK> meron@cars3.uchicago.edu wrote:
>>
>> In article <38E03BDC.868B8396@hotmail.com>, marc
>> <m_schellens@hotmail.com> writes:
>>> Is there a function like TOTAL but for multiplication. Like the
>>> big PI symbol in mathematical notation. Or this really something
>>> for the for loop?
>>>
>>> I.E.
>>>
>>> a=[1,2,3,...]
>>>
>>> result=a[1]*a[2]*a[3]...
>>>
>> if all the elements of a are positive then you can simply do
>>
>> result = exp(total(alog(a)))
JK> ...
>> If some of the elements are negative, you can still handle it. do
>>
>> dum = where(a lt 0, ndum) sig = (-1)^ndum result =
>> sig*exp(total(alog(abs(a))))

JK> You can't honestly be suggesting that this is a good technique?
JK> Ignore for a momement what happens if any element of 'a' is
JK> 0. That code performs two transcendental function evaluations per
JK> element of 'a'. IDL would have to be very badly engineered (which
JK> I suppose is possible), for a 'for' loop to execute more slowly
JK> than your code.

Well, it depends very much on the size of the array. Loops in IDL are
indeed very slow. Try the following: Set N to a large number
(e.g. 10 000 000) and execute the following lines:

```
x=fltarr(n)*0.+1.000001 & p=1 & for i=0.,1.*n_elements(x)-1 do p=p*x[i] & print,p
```

```
x=fltarr(n)*0.+1.000001 &  p=exp(total(alog(x)))&print,p
```

You'll get a surprise, I promise.

- Carsten

```
--
Carsten Dominik <dominik@astro.uva.nl>          \ _ /
Sterrenkundig Instituut "Anton Pannekoek"        |X|               _
Kruislaan 403; NL-1098 SJ Amsterdam             /| |\    _  _     _/ \
phone +31 (20) 525-7477; FAX +31 (20) 525-7484 ___|o|____/ ~~ \___/    ~~~~~
```

Subject: Re: multiplication
Posted by James Kuyper on Tue, 28 Mar 2000 08:00:00 GMT

View Forum Message <> Reply to Message

```
meron@cars3.uchicago.edu wrote:
>
>  In article <38E03BDC.868B8396@hotmail.com>, marc <m_schellens@hotmail.com> writes:
>> Is there a function like TOTAL but for multiplication.
>> Like the big PI symbol in mathematical notation.
>> Or this really something for the for loop?
>>
>> I.E.
>>
>> a=[1,2,3,...]
>>
>> result=a[1]*a[2]*a[3]...
>>
>  if all the elements of a are positive then you can simply do
>
>  result = exp(total(alog(a)))
...
>  If some of the elements are negative, you can still handle it.  do
>
>  dum = where(a lt 0, ndum)
>  sig = (-1)^ndum
>  result = sig*exp(total(alog(abs(a))))
```

You can't honestly be suggesting that this is a good technique? Ignore for a momement what happens if any element of 'a' is 0. That code performs two transcendental function evaluations per element of 'a'. IDL would have to be very badly engineered (which I suppose is possible), for a 'for' loop to execute more slowly than your code.

Subject: Re: multiplication
Posted by James Kuyper on Wed, 29 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

```
meron@cars3.uchicago.edu wrote:
>
>  In article <38E0A379.34ADB7F7@wizard.net>, James Kuyper <kuyper@wizard.net> writes:
>> meron@cars3.uchicago.edu wrote:
...
>>>  dum = where(a lt 0, ndum)
>>>  sig = (-1)^ndum
>>>  result = sig*exp(total(alog(abs(a))))
>>
>> You can't honestly be suggesting that this is a good technique?
>
>  Good?  No, only not as bad as using "for".
>
>> Ignore for a momement what happens if any element of 'a' is 0.
>
>  That's the easiest to deal with.  You're already checking for presence
>  of negative elements, can check for zeroes as well.  That should be
>  the first thing, in fact, since if even one of the elements is 0, then
>  the result is 0 and you can dispense with the rest of the evaluation.
>
>> That code performs two transcendental function evaluations per element
>> of 'a'.
>
>  Yep, indeed.
>
>>  IDL would have to be very badly engineered (which I suppose is possible),
>> for a 'for' loop to execute more slowly than your code.
>
>  Well, I run a quick test, comparing the time it takes tto evaluate the
>  product using both methods (it run on an old Vms Alpha, somebody may
>  want to repeat it on a more modern platform.  Being lazy, I'm simply
>  filling an array with a constant element, then doing the
>  multiplication.  Here is the output
>
>  IDL> speed, 1.00001, 100, 10
>  "for" time     =     0.0012000084 res =       1.00100
>  "exp-log" time =    0.00019999743 res =       1.00100
>
>  IDL> speed, 1.00001, 1000, 10
>  "for" time     =      0.012699997 res =       1.01006
>  "exp-log" time =     0.0012000084 res =       1.01006
>
>  IDL> speed, 1.00001, 10000, 10
>  "for" time     =       0.12589999 res =       1.10532
>  "exp-log" time =      0.011699998 res =       1.10532
```

>
> IDL> speed, 1.00001, 100000, 10
> "for" time   =   1.2583000 res =   2.72191
> "exp-log" time =   0.12850000 res =   2.72198
>
> The first input to SPEED is the array element, the second is the
> length of the array. the third is just telling SPEED how many times to
> repeat the test. As you can see, the above was tried for arrays with
> lengths ranging from 100 to 100000 and calculation using "for" loop is
> consistently an order of magnitude slower.

OK - I'd not bothered testing before, I didn't realize the disadvantage
of for loops was that large. Point taken.

Subject: Re: multiplication
Posted by Harald Frey on Thu, 30 Mar 2000 08:00:00 GMT

View Forum Message <> Reply to Message

"J.D. Smith" wrote:

>
> Loop Method:
>
> % Loop limit expression too large for loop variable type.
>   <LONG   (99999)>.
>
> Log Method:
>
> Average Time: 0.050116260
>   7.92382e+10
>
>
> Morals: IDL loops are pitifully slow, and you can't loop over very large arrays
> without trickery, and for many operations, compiled C is *significantly* faster.
>
> --
> J.D. Smith |*| WORK: (607) 255-5842
> Cornell University Dept. of Astronomy |*| (607) 255-6263
> 304 Space Sciences Bldg. |*| FAX: (607) 255-5875
> Ithaca, NY 14853 |*|

You can ideed loop over very large arrays. But I think what you tried to do was something
like

for i=0,1000000l do j=i

% Loop limit expression too large for loop variable type.

```
  <LONG     (      1000000)>.
% Execution halted at:  $MAIN$
```

But if you change your code slightly you get a good result:

```
for i=0l,1000000l do j=i
```

Harald

```
===========================================================
Harald U. Frey
Space Sciences Lab          phone: 510-643-3323
University of California   fax:   510-643-2624
Berkeley, CA 94720-7450    email: hfrey@ssl.berkeley.edu
```