Subject: [Q]: ID analog to FORTRAN "sign" function
Posted by Rostyslav Pavlichenko on Sat, 07 Oct 2000 05:34:21 GMT
View Forum Message <> Reply to Message

I need help, I am a new to the IDL.

Does the IDl have something close to Fortran SIGN (DSIGN... so on...)
functions

IN FORTRAN:
Elemental Intrinsic Function (Generic):
Returns the absolute value of the first argument times the sign of the
second argument.

Syntax:
=======
result = SIGN (a, b)
 a (Input) Must be of type integer or real.

 b Must have the same type and kind parameters as a.

Results:
=========
The result type is the same as a.
The value of the result is
| a | if b >= zero
and -| a | if b < zero.

--

Best regards, and thank you in advance

==========

Dr Rostyslav Pavlichenko

Research Center for
Development of Far Infrared Region
Fukui University

Bunkyo 3-9-1, Fukui 910-8507 Japan

phone: [+81] 776 27 8972
fax: [+81] 776 27 8752
fax: [+81] 776 27 8750

mailto:slavik@maxwell.apphy.fukui-u.ac.jp

==========

Subject: Re: [Q]: ID analog to FORTRAN "sign" function
Posted by Mark Hadfield on Mon, 09 Oct 2000 21:45:57 GMT
View Forum Message <> Reply to Message

"Alex Schuster" <alex@pet.mpin-koeln.mpg.de> wrote in message
news:39E1C067.16F7488E@pet.mpin-koeln.mpg.de...
> Rostyslav Pavlichenko wrote:
>
>> Does the IDl have something close to Fortran SIGN (DSIGN... so on...)
>> functions
>> ...
>> result = SIGN (a, b)
>> a (Input) Must be of type integer or real.
>>
>> b Must have the same type and kind parameters as a.
>>
>> Results:
>> =========
>> The result type is the same as a.
>> The value of the result is
>> | a | if b >= zero
>> and -| a | if b < zero.
>> ...
> No, but you can easily write it:
>
> function sign, a, b
> if (b ge 0) then $
> return, abs(a) $
> else $
> return, -abs(a)
> end

The following is more compact and works when b is an array

```
return, abs(a) * (fix(b ge 0) - fix(b lt 0))
```

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Hi! I'm a .signature virus! copy me into your .signature file to help me

spread!

Subject: Re: [Q]: ID analog to FORTRAN "sign" function
Posted by Phillip David on Thu, 12 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Dick Jackson wrote:
>
> Do I dare offer one more? Subscript lookups seem faster than arithmetic
> operations, making this one faster, more compact and no less cryptic! :-)
>
> Return, Abs(a) * ([-1, 1])[b GE 0]

to which I reply:

If this one really works, then why not go even one step further?

Return, ([-Abs(a), Abs(a)])[b GE 0]

or

Return, ([a, -a])[(a*b) LT 0]

I haven't timed either of these to find out if they're truly better (as
I don't have IDL on my newsgroup computer), but they MIGHT work
better...

Phillip

Subject: Re: [Q]: ID analog to FORTRAN "sign" function
Posted by Dick Jackson on Thu, 12 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Mark Hadfield" <m.hadfield@niwa.cri.nz> wrote in message
news:971127957.784431@clam-ext...
> "Alex Schuster" <alex@pet.mpin-koeln.mpg.de> wrote in message
> news:39E1C067.16F7488E@pet.mpin-koeln.mpg.de...
>> Rostyslav Pavlichenko wrote:
>>
>>> Does the IDl have something close to Fortran SIGN (DSIGN... so on...)
>>> functions
>>> ...
>>> result = SIGN (a, b)
>>> a (Input) Must be of type integer or real.
>>>

```
>>>         b  Must have the same type and kind parameters as a.
>>>
>>>  Results:
>>>  =========
>>>  The result type is the same as a.
>>>  The value of the result is
>>>  | a | if b >= zero
>>>  and -| a | if b < zero.
>>> ...
>>  No, but you can easily write it:
>>
>>  function sign, a, b
>>    if ( b ge 0 ) then $
>>      return, abs( a ) $
>>    else $
>>      return, -abs( a )
>>  end
>
>  The following is more compact and works when b is an array
>
>      return, abs(a) * (fix(b ge 0) - fix(b lt 0))
```

Do I dare offer one more? Subscript lookups seem faster than arithmetic operations, making this one faster, more compact and no less cryptic! :-)

```
   Return, Abs(a) * ([-1, 1])[b GE 0]
```

A bit faster still, if you know the expected type of a and b, to avoid an extra type conversion:

```
   Return, Abs(a) * ([-1.0, 1.0])[b GE 0]
```

or

```
   Return, Abs(a) * ([-1.0D, 1.0D])[b GE 0]
```

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / Voice/Fax: +1-403-242-7398

Subject: Re: [Q]: ID analog to FORTRAN "sign" function
Posted by Mark Hadfield on Thu, 12 Oct 2000 20:50:59 GMT

View Forum Message <> Reply to Message

"Dick Jackson" <dick@d-jackson.com> wrote in message
news:8MkF5.12$953.172@read1...
> Do I dare offer one more? Subscript lookups seem faster than arithmetic
> operations, making this one faster, more compact and no less cryptic! :-)
>
> Return, Abs(a) * ([-1, 1])[b GE 0]

Good one!

> A bit faster still, if you know the expected type of a and b, to avoid an
> extra type conversion:
>
> Return, Abs(a) * ([-1.0, 1.0])[b GE 0]

I think that's a little *too* clever. I just tried multiplying a float array
with 10^7 elements by 1 and then by 1.0. Time taken = 0.52 seconds in both
cases.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Hi! I'm a .signature virus! copy me into your .signature file to help me
spread!

Subject: Re: [Q]: ID analog to FORTRAN "sign" function
Posted by Dick Jackson on Fri, 13 Oct 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Phillip David wrote:
> Dick Jackson wrote:
>>
>> Do I dare offer one more? Subscript lookups seem faster than arithmetic
>> operations, making this one faster, more compact and no less cryptic!
:-)
>>
>> Return, Abs(a) * ([-1, 1])[b GE 0]
>
> to which I reply:
>
> If this one really works, then why not go even one step further?
>
> Return, ([-Abs(a), Abs(a)])[b GE 0]

>
> or
>
> Return, ([a, -a])[(a*b) LT 0]

Right, these would work fine for scalar a and b, with negligible time taken
in any case. I was looking for the most efficient way when we need this to
work on large arrays a and b.

About the 1 vs 1.0 debate, Mark Hadfield wrote:
> I think that's a little *too* clever. I just tried multiplying a float
array
> with 10^7 elements by 1 and then by 1.0. Time taken = 0.52 seconds in both
> cases.

This is getting interesting. For reference, here are a couple of handy timer
routines I use:

```
;---

PRO TStart ; Timer Start
; Save current time for use by TReport
COMMON Timer, t0
t0 = SysTime(1)
END

;---

PRO TReport ; Timer Report
; Print elapsed time since last TStart
COMMON Timer, t0
Print, Format='(D10.3," seconds.")',SysTime(1)-t0
END

;---
```

Here's some testing runs from my Win2000 PC:

```
IDL> a=randomu(seed,1000000)-0.5
IDL> b=randomu(seed,1000000)-0.5
IDL> tstart & for i=1,10 do c=Abs(a) * ([-1, 1])[b GE 0] & treport
    3.250 seconds.
IDL> tstart & for i=1,10 do c1=Abs(a) * ([-1.0, 1.0])[b GE 0] & treport
    2.813 seconds.
```

I think the time saving here is not in the multiplying itself, but in the
time building an integer array, then converting it to float. In this case
it's 10^6 ones/minus-ones, perhaps in your case it was converting only a

single 1 to 1.0, then multiplying it.

Fascinating, isn't it? I'd be happy to hear further refinements!

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / Voice/Fax: +1-403-242-7398