Subject: Comparing 2 arrays
Posted by teich on Sun, 26 Aug 2007 04:18:39 GMT
View Forum Message <> Reply to Message

Hi,

I would like to know if the data in two different 1 dimensional arrays
are the same or not. This should be for every index. For example, if
A=[0,1,2,3] and B=[0,1,3,2], then I would want the answer to be that A
and B are not equal even though they happen to have the same values in
different order. A problem with comparing each element is that the
numbers are floating point so I would prefer not to compare with
'EQ'. Has anyone any suggestions?

Alternatively, my arrays are actually from different structures. Is
there a way to compare structures? For example, data1.A vs data2.B in
the above example.

Thanks

Howie

Subject: Re: Comparing 2 arrays
Posted by Allan Whiteford on Tue, 28 Aug 2007 16:10:46 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Jean H. writes:
>
>
>> to get back to a previous discussion we had a few month ago about being
>> "sufficiently close to zero", shouldn't it be (data1.A - data2.B) LT
>> epsilon * data1.A , with epsilon=(machar()).eps?
>
>
> OK, I found that discussion and read it eight or ten times until
> I finally understood it. (Probably why I forgot it before.)
>
> I've put a significantly edited discussion of this
> problem here:
>
>   http://www.dfanning.com/code_tips/comparearray.html
>
> In my preferred solution now, I choose a number that
> is "sufficiently close" to zero like this:
>

```
>     epsilon = (MACHAR()).eps
>     NUMBER = (array_1 > array_2) * epsilon
>
>  Then, the comparison between arrays is done like this:
>
>     IF Total(Abs(array_1 - array_2) LT NUMBER) EQ N_Elements(array_1) $
>        THEN RETURN, 1 ELSE RETURN, 0
>
>  Additional comments welcome if you want to argue further. :-)
>
>  Cheers,
>
>  David
```

David,

Not that I wish to argue but... this looks like it will fail on elements
which are identically zero.

Changing the 'lt' to 'le' will probably sort it.

Thanks,

Allan

Subject: Re: Comparing 2 arrays
Posted by David Fanning on Tue, 28 Aug 2007 16:18:20 GMT
View Forum Message <> Reply to Message

Allan Whiteford writes:

> Not that I wish to argue but... this looks like it will fail on elements
> which are identically zero.
>
> Changing the 'lt' to 'le' will probably sort it.

Ah, good point. Guess I won't apply for that job as a code
tester. :-(

Fixed now.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.

Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Comparing 2 arrays
Posted by Jean H. on Tue, 28 Aug 2007 16:20:42 GMT
View Forum Message <> Reply to Message

perfect!! ... we had to move all the equipment form our lab so I
couldn't participate in the debate (never send a post on Sunday if you
don't know what you will be doing on Monday :-)

Thanks all for the clarity of the answer/article!

David, in your example (in this message), you use
NUMBER = (array_1 > array_2) * epsilon
and
NUMBER = (array_1 > array_2) * epsilon * 2 in your article...

Also, but that is for pure curiosity, could it be possible to read
directly the mantissa and the power (see David's article if it sounds
like a foreign language) and to compare them for our 2 numbers? ... 1)
look a the power, if they have any difference, the two values can not be
equal. Then look at the mantissa and asses if the digit values are the
same or not...

Jean

David Fanning wrote:
> Jean H. writes:
>
>> to get back to a previous discussion we had a few month ago about being
>> "sufficiently close to zero", shouldn't it be (data1.A - data2.B) LT
>> epsilon * data1.A , with epsilon=(machar()).eps?
>
> OK, I found that discussion and read it eight or ten times until
> I finally understood it. (Probably why I forgot it before.)
>
> I've put a significantly edited discussion of this
> problem here:
>
>   http://www.dfanning.com/code_tips/comparearray.html
>
> In my preferred solution now, I choose a number that
> is "sufficiently close" to zero like this:
>

```
>     epsilon = (MACHAR()).eps
>     NUMBER = (array_1 > array_2) * epsilon
>
>  Then, the comparison between arrays is done like this:
>
>     IF Total(Abs(array_1 - array_2) LT NUMBER) EQ N_Elements(array_1) $
>        THEN RETURN, 1 ELSE RETURN, 0
>
>  Additional comments welcome if you want to argue further. :-)
>
>  Cheers,
>
>  David
```

Subject: Re: Comparing 2 arrays
Posted by David Fanning on Tue, 28 Aug 2007 16:52:54 GMT
View Forum Message <> Reply to Message

Jean H. writes:

```
>  David, in your example (in this message), you use
>  NUMBER = (array_1 > array_2) * epsilon
>  and
>  NUMBER = (array_1 > array_2) * epsilon * 2  in your article...
```

I just thought there should be some multiple to account
for possible accumulative round-off errors. Two was an
arbitrary selection, but based on something I read in the
previous discussion. Naturally, I decided on this after
I wrote the note describing it. :-)

> Also, but that is for pure curiosity, could it be possible to read
> directly the mantissa and the power (see David's article if it sounds
> like a foreign language) and to compare them for our 2 numbers? ... 1)
> look a the power, if they have any difference, the two values can not be
> equal. Then look at the mantissa and asses if the digit values are the
> same or not...

I don't think IDL has easy access to the numbers as they
are stored. I guess you could do it yourself, but I'm
pretty sure it involves "twos complement" arithmetic,
which I've never been sufficiently motivated to learn. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Comparing 2 arrays
Posted by Paul Van Delst[1] on Tue, 28 Aug 2007 18:43:18 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
```
> Jean H. writes:
>
>> to get back to a previous discussion we had a few month ago about being
>> "sufficiently close to zero", shouldn't it be  (data1.A - data2.B) LT
>> epsilon * data1.A ,  with epsilon=(machar()).eps?
>
> OK, I found that discussion and read it eight or ten times until
> I finally understood it. (Probably why I forgot it before.)
>
> I've put a significantly edited discussion of this
> problem here:
>
>    http://www.dfanning.com/code_tips/comparearray.html
>
> In my preferred solution now, I choose a number that
> is "sufficiently close" to zero like this:
>
>    epsilon = (MACHAR()).eps
>    NUMBER = (array_1 > array_2) * epsilon
>
> Then, the comparison between arrays is done like this:
>
>    IF Total(Abs(array_1 - array_2) LT NUMBER) EQ N_Elements(array_1) $
>       THEN RETURN, 1 ELSE RETURN, 0
>
> Additional comments welcome if you want to argue further. :-)
```

Sure! :o)

I think you should also pass a scaling factor, ala,

```
FUNCTION FLTARRAYS_EQUAL, array_1, array_2, ULP=ulp
  ....
  IF ( N_ELEMENTS(ulp) EQ 0 ) THEN ulp=1.0
  ....
  NUMBER = (array_1 > array_2) * epsilon * ulp
  ....
END
```

Also, there needs to be differentiation for singel and double precision so you can
determine epsilon correctly and set ulp to a suitable default (1.0 or 1.0d0).

cheers,

paulv

>
> Cheers,
>
> David

Subject: Re: Comparing 2 arrays
Posted by David Fanning on Tue, 28 Aug 2007 18:56:15 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

```
> I think you should also pass a scaling factor, ala,
>
> FUNCTION FLTARRAYS_EQUAL, array_1, array_2, ULP=ulp
>   ....
>   IF ( N_ELEMENTS(ulp) EQ 0 ) THEN ulp=1.0
>   ....
>   NUMBER = (array_1 > array_2) * epsilon * ulp
>   ....
> END
>
> Also, there needs to be differentiation for singel and double precision so you can
> determine epsilon correctly and set ulp to a suitable default (1.0 or 1.0d0).
```

I usually save these extra touches for programs I add to
the Coyote Library and document heavily, not for ones I
throw away into the Tip Examples pile. But I have a feeling
this one is destined for the Library anyway.

I've seen you use ULP as a variable name before.
Is this something like VEGEMITE? Or does it actually
mean something to you? I'd probably call it FUDGE
if I was making up the name myself. :-)

Can double precision *possibly* make a difference in this program in practice?

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Comparing 2 arrays
Posted by Jean H. on Tue, 28 Aug 2007 19:18:29 GMT
View Forum Message <> Reply to Message

> Can double precision *possibly* make a difference in this
> program in practice?
>
> Cheers,
>
> David

we could be comparing doubles, not only floats?!

Jean

Subject: Re: Comparing 2 arrays
Posted by Paul Van Delst[1] on Tue, 28 Aug 2007 19:33:50 GMT
View Forum Message <> Reply to Message

```
David Fanning wrote:
> Paul van Delst writes:
>
>> I think you should also pass a scaling factor, ala,
>>
>> FUNCTION FLTARRAYS_EQUAL, array_1, array_2, ULP=ulp
>>   ....
>>   IF ( N_ELEMENTS(ulp) EQ 0 ) THEN ulp=1.0
>>   ....
>>   NUMBER = (array_1 > array_2) * epsilon * ulp
>>   ....
>> END
>>
>> Also, there needs to be differentiation for singel and double precision so you can
>> determine epsilon correctly and set ulp to a suitable default (1.0 or 1.0d0).
```

>
> I usually save these extra touches for programs I add to
> the Coyote Library and document heavily, not for ones I
> throw away into the Tip Examples pile. But I have a feeling
> this one is destined for the Library anyway.
>
> I've seen you use ULP as a variable name before.
> Is this something like VEGEMITE? Or does it actually
> mean something to you? I'd probably call it FUDGE
> if I was making up the name myself. :-)

From my Fortran95 function header:

```
! NAME:
!       Compare_Float
!
! PURPOSE:
!       Function to compare floating point scalars and arrays with adjustible
!       precision tolerance.
!
! CALLING SEQUENCE:
!       Result = Compare_Float( x, y,    &  ! Input
!                               ULP=ULP )  ! Optional input
!
! INPUT ARGUMENTS:
!       x, y:        Two congruent floating point data objects to compare.
!                    UNITS:      N/A
!                    TYPE:       REAL(Single)   [ == default real]
!                                  OR
!                                REAL(Double)
!                                  OR
!                                COMPLEX(Single)
!                                  OR
!                                COMPLEX(Double)
!                    DIMENSION:  Scalar, or any allowed rank array.
!                    ATTRIBUTES: INTENT(IN)
!
! OPTIONAL INPUT ARGUMENTS:
!       ULP:         Unit of data precision. The acronym stands for "unit in
!                    the last place," the smallest possible increment or decrement
!                    that can be made using a machine's floating point arithmetic.
!                    A 0.5 ulp maximum error is the best you could hope for, since
!                    this corresponds to always rounding to the nearest representable
!                    floating-point number. Value must be positive - if a negative
!                    value is supplied, the absolute value is used.
!                    If not specified, the default value is 1.
!                    UNITS:      N/A
!                    TYPE:       INTEGER
!                DIMENSION:  Scalar
!                ATTRIBUTES: OPTIONAL, INTENT(IN)
```

> Can double precision *possibly* make a difference in this
> program in practice?

Oh my goodness, emphatically yes, most definitely. I use the above functionality nearly every day comparing tangent-linear to adjoint model output. By definition, the results should be the same to within numerical precision -- which they have to be to pass my tests. The dynamic range of my inputs vary by many orders of magnitude (such that "numerical precision" could be represented by anything from around 1.0e-05 to 1.0e-23.)

To paraphrase a cliche and wrap it up in some hyperbole: when those butterflies start flapping their wings, you better hope your model uses double precision floating point arithmetic or your hurricane might not go where you think.

Seriously, though, I'm a physical scientist, not a computer one. Given enough time and resources (which, for physical scientists is usually much, much, much less than computer ones) I could probably come up with smart algorithms that work just as well in single as double precision. But my brain would start oozing out of my earholes, and some smart youngster who grew up using student editions of matlab would soon take my place. Using double precision by default is a good fat-fingered insurance policy (even if frowned upon by the computer cognoscenti).

insert multiple :o)'s here as required. :o)

cheers,

paulv

Subject: Re: Comparing 2 arrays
Posted by James Kuyper on Tue, 28 Aug 2007 20:02:33 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
...
> I've seen you use ULP as a variable name before.
> Is this something like VEGEMITE?

See <http://en.wikipedia.org/wiki/Unit_in_the_last_place>

Subject: Re: Comparing 2 arrays
Posted by Allan Whiteford on Wed, 29 Aug 2007 08:04:27 GMT

David Fanning wrote:
> Jean H. writes:
>
>
>> to get back to a previous discussion we had a few month ago about being
>> "sufficiently close to zero", shouldn't it be (data1.A - data2.B) LT
>> epsilon * data1.A , with epsilon=(machar()).eps?
>
>
> OK, I found that discussion and read it eight or ten times until
> I finally understood it. (Probably why I forgot it before.)
>
> I've put a significantly edited discussion of this
> problem here:
>
> http://www.dfanning.com/code_tips/comparearray.html
>
> In my preferred solution now, I choose a number that
> is "sufficiently close" to zero like this:
>

```
>    epsilon = (MACHAR()).eps
>    NUMBER = (array_1 > array_2) * epsilon
```

>
> Then, the comparison between arrays is done like this:
>

```
>    IF Total(Abs(array_1 - array_2) LT NUMBER) EQ N_Elements(array_1) $
>       THEN RETURN, 1 ELSE RETURN, 0
```

>
> Additional comments welcome if you want to argue further. :-)
>
> Cheers,
>
> David

David,

Not sure about this point but wouldn't:

```
NUMBER =  (abs(array_1) > abs(array_2)) * epsilon
```

make more sense?

Certainly it would make more sense to me in terms of both dealing with
the comparison of two negative numbers (in this case NUMBER will end up
being negative which can't be good!) or even when comparing a small
positive number with a large negative number.

Thanks,

Allan

Subject: Re: Comparing 2 arrays
Posted by Carsten Lechte on Wed, 29 Aug 2007 10:31:56 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> I've seen you use ULP as a variable name before.
> Is this something like VEGEMITE? Or does it actually
> mean something to you?

ulp stands for "units in the last place." See this excellent
article on all the bad stuff that can befall the hapless
floating point user:

http://docs.sun.com/source/806-3568/ncg_goldberg.html

chl

Subject: Re: Comparing 2 arrays
Posted by David Fanning on Wed, 29 Aug 2007 13:17:04 GMT
View Forum Message <> Reply to Message

Allan Whiteford writes:

> Not sure about this point but wouldn't:
>
> NUMBER = (abs(array_1) > abs(array_2)) * epsilon
>
> make more sense?

I'm about ready to give up on open source software development!
Sure, you get better programs. But it is damn hard to get
any real work done. :-(

Cheers,

David

P.S. On the other hand, there's more room to spread the blame
around. That's a good thing!

--

David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Comparing 2 arrays
Posted by ph le sager on Wed, 29 Aug 2007 17:32:39 GMT
View Forum Message <> Reply to Message

For somewhat similar reasons, I came up with the following function. I
find myself using it in lot of routines to replace array subtraction.
There is probably better way to do it, but it works fine, and I can
tell then where the two arrays are not equal.

```
;+
; NAME:     FLOAT_DIFF
;
;           ...usefull only if you specify EPS > machine precision,
;           or use STRICT. Keep reading...
;
; PURPOSE: - to take into account the accuracy in number
;           when differentiating data.
;
;     In a computer, differences are zero if they are less than the
;     precision of the float (or double) representation. See idea in:
;
;     http://www.ibiblio.org/pub/languages/fortran/ch1-8.html#02
;
;     Here, the idea is to overwrite the machine precision. Basically,
;     DATA1 - DATA2 = 0 if less than the relative-EPS, where you can
;     define EPS. For example, if your data accuracy is 1% (which is a
;     lot less than the machine accuracy), you would type:
;
;             DIFF = FLOAT_DIFF(data1,data2,eps=0.01)
;
;     This is really useful to disregard insignificant differences
;     when comparing datasets.
;
;
; INPUTS: - data1, data2 : data to differentiate. If they are
;           not supplied(!), NAN is return. If they have different
;           size the smallest size of the two array is used, like for
;           a regular array difference.
;
;
; OUTPUT: - array same size as the smallest input data size
;             output  = array1 - array2
```

```
;
; CATEGORY:  math, testing
;
; CALLING SEQUENCE:
;
;         result = float_diff(array1, array2, eps=eps)
;
; KEYWORD PARAMETERS:
;
;         DOUBLE : set if everything is done with double precision
;
;         EPS : to redefine the floating (OR DOUBLE) point
;              precision. If not set, the machine precision for
;              floating (OR DOUBLE) is used, which will give the
;              same result as
;                     res = DATA1 - DATA2
;
;         STRICT: set if you want the precision to be specified in
;                absolute value. By default the
;                relative precision is used, that is difference is
;                compared to:
;
;                    EPS * MAX( ABS(VALUE1), ABS(VALUE2) )
;
;                Setting /Strict, differences are compared to EPS.
;
; LIMITATION -
;
; MODIFICATION HISTORY: phs, 11/08/06 - v1.0
;
;-

FUNCTION FLOAT_DIFF, DATA1,  DATA2,  $
             EPS=EPS, STRICT=STRICT, DOUBLE=DOUBLE, _extra=extra


   ;; on error return to caller
   on_error, 2


   ;; Basic inputs checking
   dbl = keyword_set(double)
   eps = n_elements(eps) eq 0 ? (machar(double=dbl)).eps : eps[0]

   if n_params() ne 2 then begin
      message, 'not enough input to FLOAT_DIFF... return NaN',  $
        /continue
      return, dbl ? !values.d_nan : !values.f_nan
```

```
  endif


  ;; Differences (zero if below accuracy)
  valid = keyword_set(strict) ? EPS : EPS*(abs(data1) > abs(data2))
  return, ( abs(data1-data2) gt valid ) * (data1-data2)

END
```