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,