
Subject: Re: odd behaviour from `array_equal()` with NaN, Inf values

Posted by [Craig Markwardt](#) on Mon, 23 Jan 2012 05:46:12 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Jan 22, 11:42 pm, wallabadah <write.to.wpow...@gmail.com> wrote:

> I've come across the following behaviour while tracking down odd behaviour from NaN and Inf values. I'm trying to use `array_equal()` to check that arrays contain the same content - but it bombs when the arrays contain NaN. I've tracked it down to the process of copying an array containing NaN values to a new variable. For example:

>

> IDL> a = findgen(5)

> IDL> a[1] = !values.f_nan

> IDL> print, array_equal(a, a)

> 1

> IDL> b = a

> IDL> print, array_equal(a, b)

> 0

> IDL> print, array_equal(b, b)

> 1

>

> Repeating the process with `!values.f_infinity` behaves as expected:

>

> IDL> a = findgen(5)

> IDL> a[1] = !values.f_infinity

> IDL> print, array_equal(a, a)

> 1

> IDL> b = a

> IDL> print, array_equal(a, b)

> 1

> IDL> print, array_equal(b, b)

> 1

>

> Is this a bug in `make_array()` or some artifact of how values are copied to new variables?? Is it reproducible on different platforms (I'm on Mac OS X, IDL 8.1).

Greetings--

I think it's a subtle bug in `ARRAY_EQUAL()`. `ARRAY_EQUAL(A,A)` *should* return 0 for your case.

It is a property of NAN that it can never equal itself so - perhaps surprisingly -

`(A[1] EQ A[1])` is false. Try it yourself!

I suspect that the IDL folks decided to put an special case optimization in to the code which always returned true when the same variable is passed in both parameter positions. Unfortunately, this fails when one more more elements is NAN.

Craig

Subject: Re: odd behaviour from array_equal() with NaN, Inf values

Posted by [Carsten Lechte](#) on Mon, 23 Jan 2012 12:37:11 GMT

[View Forum Message](#) <> [Reply to Message](#)

On 23/01/12 06:46, Craig Markwardt wrote:

- > It is a property of NAN that it can never equal itself so - perhaps
- > surprisingly -
- > (A[1] EQ A[1]) is false. Try it yourself!

I think this is a case where adherence to the IEEE standard is not necessarily what the user wants. There should be a version of ARRAY_EQUAL where NaN==NaN, for cases where one wants to know if two arrays contain the same data; similar to MEAN(), which gives the opportunity to ignore NaNs, and, with it, to shoot oneself in the foot.

chl

Subject: Re: odd behaviour from array_equal() with NaN, Inf values

Posted by [wallabadah](#) on Mon, 23 Jan 2012 23:27:25 GMT

[View Forum Message](#) <> [Reply to Message](#)

Here's a quick wrapper that checks for NaNs in the input, and returns true for the above test case. I haven't tested it extensively, but it seems to solve my current problem.

```
function wjp_array_equal, op1, op2, no_typeconv = no_typeconv
  compile_opt idl2

  op1NaN = where(finite(op1, /NaN), op1NaNCount, complement = op1Finite)
  op2NaN = where(finite(op2, /NaN), op2NaNCount, complement = op2Finite)

  ; if no NaN values are present, use the normal array_equal() function
  if op1NaNCount eq 0 && op2NaNCount eq 0 then $
    return, array_equal(op1, op2, no_typeconv = no_typeconv)

  ; if counts of NaN values are equal, and results of the two where()
  ; functions are the same, compare the non-NaN values using array_equal()
  if op1NaNCount eq op2NaNCount && array_equal(op1Finite, op2Finite) then $
    return, array_equal(op1[op1Finite], op2[op2Finite], no_typeconv = no_typeconv)

  ; otherwise return 0
  return, 0
```

end
