Subject: Re: How to test for a vector/matrix of constants?
Posted by JD Smith on Wed, 06 Dec 2006 23:53:04 GMT
View Forum Message <> Reply to Message

On Wed, 06 Dec 2006 22:31:30 +0100, Fï¿½LDY Lajos wrote:

>
> On Wed, 6 Dec 2006, Paul van Delst wrote:
>
>> From Braedley
>> if n_elements(uniq(array, sort(array))) eq 1 then ;flag it
>> -> No float comparison
>
> Do you think sort works without comparison? :-)
>
> regards,
> lajos

It's not SORT, but UNIQ which is the problem. Here's how UNIQ works (see the code !DIR/lib/uniq.pro):

```
 indices = where(q ne shift(q,-1), count)
```

For every one of the mentioned methods, float equality is checked.

I know a lot has been said about the issue of floating point representation (search "sky is falling float"), but I thought I'd have a stab.

There's nothing special about float comparison, other than the fact that it's hard to write down a floating point number uniquely, by hand as a literal in your code, as ASCII in a text file, etc. This issue just doesn't exist for integers, for which there is a one-to-one mapping between integer representation in ASCII text and binary representation on disk (as long as the number is within the integer range).

A float is equal to another float only when their exact representation in memory is equal. Note that the following is not an issue:

```
IDL> print,array_equal(replicate(4.923,100),4.923)
   1
```

It will *always* be true for any floating number specified like this, even ones which cannot be represented at the given precision:

```
IDL>  print,array_equal(replicate(4.923098790790790879087908790879087, 100), $
        4.923098790790790879087908790879087)
```

```
   1
```

Problems only begin to occur when you start to rely on your
(incorrect) intuitive expectation of how a computer *should* handle a
floating point number:

```
IDL> print,array_equal(replicate(4.923,100),1.00 + 1.02 + 1.045 + 1.858)
   0
```

But, why not, you ask, since:

```
IDL> print, 1.00 + 1.02 + 1.045 + 1.858
      4.92300
```

Well, not all of those numbers (if considered as exact values drawn
from the inifinite set of all real numbers) can actually be
represented uniquely as a float. The classic example of this issue is
the decimal number 0.9, which in binary representation is:

0.11100110011001100110011001100110011.....

and repeating so on forever. Unfortunately, you can only allocate so
many bits for a float, so:

```
IDL> print, 0.9 eq (0.6 + 0.3)
   0
IDL> print,FORMAT='(F0.8)',0.9, 0.6 + 0.3
0.89999998
0.90000004
```

Note that this doesn't mean that 0.9 and numbers like it will *never*
equal to the "sum of their parts". Sometimes you get lucky and the
bit truncation works out the same::

```
IDL> print, 0.9 eq (0.5 + 0.4)
   1
```

Here's a nice example in C to get a better handle on this, which will
let you peek behind the curtain and see the real bit representation of
a given floating point number:

```
#include <stdio.h>
int main() {
  float f=0.9,f1;

  printf("Literal -- %f: %lx\n",f,*(unsigned int *) &f);

  f1=(float) 0.3 + (float) 0.6;
```

```
  printf("0.6+0.3 -- %f: %lx\n",f1,*(unsigned int *) &f1);

  printf("Floats are %s\n",f1==f?"Equal":"Not Equal");

}
```

For convenience it prints the floats as 4 bytes of hexadecimal instead of 32 bits of binary, and it uses a de-referencing trick to get at the actual binary data for the float. Here are the results

```
% ./compare_float
Literal -- 0.900000: 3f666666
0.6+0.3 -- 0.900000: 3f666667
Floats are Not Equal
```

Note how I had to cast the literal numbers 0.3 and 0.6 to floats, since by default most C compilers promote literals to double in arithmetic before assigning to float. IDL does this casting natively as well (which is why you aren't guaranteed to get the same answer with IDL and C).

Note also how the bit representation of the floats is not equal. Here they are in binary:

```
3f666666: 11111101100110011001100110
3f666667: 11111101100110011001100111
```

Just a single bit difference, but what a difference it is!

If you really want to test floats which arrived in a given array from various sources for "equality", you need to specify what "equality" means (if something other than the computer's quite naive definition of equality as "exact same representation in binary format"). You might try:

```
ARRAY_EQUAL(abs(a-a[0]) lt epsilon,1b)
```

where epsilon is your maximum allowed difference within which floats should be considered "equal". A good number might be (machar()).eps, i.e.:

```
IDL> print,abs(0.9 - (0.6+0.3)) lt (machar()).eps
   1
```

Ahh, all is right with the world.

JD
