Subject: Re: Inaccuracies
Posted by hahn on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe <afl@cdc.noaa.gov> wrote:

> Ok, I am sure this has been discussed before, but let
> me start this thread again. I wish to create a 15-element
> vector which contains the numbers -1.4 to 1.4 by an increment
> of 0.2 I also wish the sum of these elements to be zero
> (No, this isn't the new math). Here is what I tried...

[remainder deleted]

All you must do is to avoid decimal fraction because our computers use binary fractions and truncate the number. Thus you may either multiply each number by a factor large enough to get an integral number and divide when you print/plot the results or you treat 0.2 as 1/5 and work on that. I guess this is what packages such as Mathematica or Maple do.

Norbert Hahn

Subject: Re: Inaccuracies
Posted by lpmix on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

In article <30A7BC4D.7018@cdc.noaa.gov>, Andy Loughe <afl@cdc.noaa.gov> wrote:
>
>
> TRIAL #1
> ========
> IDL> a = findgen(15)*.2 - 1.4
> IDL> print, total(a)
>   7.15256e-07
Stuff deleted**********
>
> Maybe I am missing something here, but this kind of behavior
> makes IDL a bit problematical for scientific use. With only 15
> numbers and double precision arithmetic, I can't believe this
> would fail in FORTRAN or C!
>
> --
> Andrew F. Loughe (afl@cdc.noaa.gov)
> University of Colorado, CIRES * Campus Box 449 * Boulder, CO 80309
> phone: (303) 492-0707   fax: (303) 497-7013

Try
IDL> a = findgen(8)*0.2
IDL> a = [-rotate(a(1:*), 2), a]
IDL> print, total(a, /double)
      0.0000000
I'm not sure what you want to do, but expecting perfect math with
floats is not generally possible.

Paul

 ==========
==========
                          L. Paul Mix
              Distinguished Member of the Technical Staff
        Computational Electromagnetics and Plasma Physics Department

    _/_/_/ _/   _/ _/   SANDIA NATIONAL LABORATORIES   _/_/_/
   _/   _/_/ _/ _/   MS 1186, P.O. Box 5800   _/_/
  _/_/_/   _/ _/ _/ _/   Albuquerque, NM 87185-1186   _/_/_/_/_/_/
     _/ _/ _/_/ _/   E-mail: lpmix@sandia.gov   _/ _/_/ _/
_/_/_/   _/   _/ _/_/_/_/   Phone: (505) 845-7493   _/ _/_/ _/
                          FAX: (505) 845-7890   _/_/_/
 ==========
==========

Subject: Re: Inaccuracies
Posted by Andy Loughe on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

> L. Paul Mix wrote:
> I'm not sure what you want to do, but expecting perfect math with
> floats is not generally possible.

I accept the explanation given by Ken Bowman, but it is hard to
explain the values assigned to
  (1) findgen(15)*.2 -1.4 versus
  (2) dindgen(15)*(.2D)-(1.4D)
  (3) and, taken separately, the results of using the total function
       on (1) and (2) matched with the ability to perform "perfect math"
       with only 13 values.

I am describing a small permutation of inaccuracies here, (1) looks ok,
but (2) does not. total((1)) and total((2)) are not accurate for
the reasons given by Ken.

BTW one IDL user indicated that there was *no* trouble with this math on
his VMS system! Now that is interesting.

Andrew F. Loughe (afl@cdc.noaa.gov)
University of Colorado, CIRES * Campus Box 449 * Boulder, CO 80309
phone: (303) 492-0707 fax: (303) 497-7013

Subject: Re: Inaccuracies
Posted by wclodius on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Some points:

1. It is impossible in finite binary arithmetic to represent any fraction that is not an integer multiple of a power of two. The fpu implementation limits the integer multiples and powers of twos available for such representations to a finite set whose size depends on whether single or double precision is used.

2. Division by 10 involves division by a non-power of two, e.g., the prime number five. Therefore, numbers given to the first decimal place, e.g., 0.1, cannot in general be represented exactly, multiples of 1.0 and 0.5 are exceptions. The representation of most of the numbers therefore will be in error by a fraction of a bit. The sum of numbers given to the first decimal place therefore, cannot, in general, be done exactly in either single or double precision.

3. It is possible that depending. either on the average effects of rounding and on the order of calculations, e.g., whether the sum goes from smallest to largest, or vice versa, the result might accidently be the "correct" answer. On average on such a simple problem, I suspect that single and double precision will loose the same number of bits, but because the double precision has more initially significant bits the relative error in double precision will usually be much smaller than in single precision.

4. It is not generally possible for a compiler or interpreter to know the order of operations which minimizes the error, that requires a knowlege about the detailed properties of the specific set of numbers subject to the operations. It is sometimes possible to use your specific knowlege of the numbers' properties to force the compiler to perform operations in a specific order that minimizes errors, e.g., add the numbers in the order smallest magnitude to largest magnitude to minimize roundoffs, or if the number come in pairs with opposite signs, add the pairs together first. Both methods have runtime performance hits.

5. The conversion in almost any implementation of C, Fortran, or IDL, from the base two internal representation to the base ten output,

typically involves rounding, optimally to within the greater of the least significant bit of the representation or the least significant digit of the output, although that is quality of implementation dependent. This rounding is an additional source of error that might or might not result in the "correct" answer. For free format the least significant bit is comparable in magnitude to the least significant digit in a high quality implementation. For the example proplem, however, this is unlikely to be a significant source of error in the output, as the higher order bits are subtractted out before the output is generated.

--

William B. Clodius Phone (505) 665-9370
Los Alamos Natl. Lab. NIS-1 FAX (505) 665-7395
PO Box 1663, MS-D466 Group Office (505) 667-2701

Subject: Re: Inaccuracies
Posted by thompson on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe <afl@cdc.noaa.gov> writes:

> Ok, I am sure this has been discussed before, but let
> me start this thread again. I wish to create a 15-element
> vector which contains the numbers -1.4 to 1.4 by an increment
> of 0.2 I also wish the sum of these elements to be zero
> (No, this isn't the new math). Here is what I tried...

> TRIAL #1
> ========
> IDL> a = findgen(15)*.2 - 1.4
> IDL> print, total(a)
> 7.15256e-07

> Hmmm! Not so good.

> Maybe I am missing something here, but this kind of behavior
> makes IDL a bit problematical for scientific use. With only 15
> numbers and double precision arithmetic, I can't believe this
> would fail in FORTRAN or C!

Here's a FORTRAN program I tried this on.

```
      program test
c
      total = 0
```

```
      do i = 1,15
        a = (i-1)*0.2 - 1.4
        total = total + a
      enddo
      write (*,*) total
c
      end
```

and the result of running this program?

```
 > a.out
   7.1525574E-07
```

Exactly the same answer as IDL gives! If instead I do it in double precision, I get

```
 > a.out
   4.440892098500626E-015
```

Again, exactly the same answer as IDL! It appears that IDL is working correctly within the confines of the floating point arithmetic of the computer.

Interestingly enough, the same problem done in IDL on an OpenVMS computer gives exactly zero when done in single precision, but 8.8817842e-16 in double precision. VMS uses a different floating point format than the standard IEEE representation used in most modern computers.

The bottom line is that all computers are subject to round-off errors when doing floating point arithmetic. This will occur no matter what software package is used.

Bill Thompson

Subject: Re: Inaccuracies
Posted by Jackel on Tue, 14 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

In article <30A7BC4D.7018@cdc.noaa.gov> Andy Loughe <afl@cdc.noaa.gov> writes:

> Ok, I am sure this has been discussed before, but let
> me start this thread again. I wish to create a 15-element
> vector which contains the numbers -1.4 to 1.4 by an increment
> of 0.2 I also wish the sum of these elements to be zero
> (No, this isn't the new math). Here is what I tried...

Well, this works:

```
  a= (INDGEN(29) - 14) / 10.0d0
```

but that still doesn't address the fundamental problem. Start with

```
  a= 7.0d0 * 0.2d0          which when PRINTed gives  1.4000000
```

then

```
  b= a - 1.4000000d0     when PRINTed gives 2.2204460e-016
```

but

```
  c= a
  b= a - c         gives 0.0
```

(note that c= 1.400000d0, then b=a-c gives a non-zero result). So, it looks like the internal representation of 7.0d0*0.2d0 is not quite 1.4, but for display purposes IDL does a bit of rounding (truncation?).

So, try

```
 PRINT,a,FORMAT='(f27.25)'
```

and get

```
 1.4000000000000001000000000
```

which has a 1d-16 difference. The rest of the puzzle is solved by PRINTing 1.4d0 (with the FORMATing as above):

```
 1.3999999999999999000000000
```

Basically, it looks like a combination of the usual representation error, combined with a short default format for output.

Brian Jackel
University of Western Ontario

Subject: Re: Inaccuracies
Posted by wclodius on Wed, 15 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

In article <48djg2$b85@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) wrote:

> <snip>
> ...Round-off errors are determined solely by the floating point

> processing done by the CPU. It was stated earlier, without testing, that IDL
> was deficient in this repect relative to Fortran or C--I showed in an earlier
> message that this was not the case.

Round off errors are the result of the algorithm, the process of translating this algorithm into CPU and FPU operations, and FPU floating point processing. For simple algorithms, as in the example that prompted this thread, the translation of the algorithm for reasonable implementations is liable to produce the same FPU processing. In such a case you are correct. For more complicated algorithms, with optimizing translations, that need not be the case.

--

William B. Clodius Phone (505) 665-9370
Los Alamos Natl. Lab. NIS-1 FAX (505) 665-7395
PO Box 1663, MS-D466 Group Office (505) 667-2701

Subject: Re: Inaccuracies
Posted by Andy Loughe on Wed, 15 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

William Thompson wrote:

>
> I believe that I was that user. Specifically, I reported that the expected

No, it was someone else who emailed me about this issue.

> answer of 0 was seen when the calculation was done in single precision.
> However, the same was not true in double precision. The only thing this
> demonstrates is that the problems of computer round-off error shows up in
> different ways in VMS than on other computers, and is simply because of the
> difference between the VAX floating point representation and the more standard
> IEEE one. In other words, it's a hardware difference, not a software one.

That is what I have learned as well.

>
> I think the discussion of how computer round-off error manifests itself is
> interesting. However, it's important to remember that this has nothing to do
> with IDL. Round-off errors are determined solely by the floating point
> processing done by the CPU. It was stated earlier, without testing, that IDL
> was deficient in this repect relative to Fortran or C--I showed in an earlier
> message that this was not the case.

>
> William Thompson

Agreed.

--
Andrew F. Loughe (afl@cdc.noaa.gov)
University of Colorado, CIRES * Campus Box 449 * Boulder, CO 80309
phone: (303) 492-0707   fax: (303) 497-7013

Subject: Re: Inaccuracies
Posted by thompson on Wed, 15 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe <afl@cdc.noaa.gov> writes:

>> L. Paul Mix wrote:
>> I'm not sure what you want to do, but expecting perfect math with
>> floats is not generally possible.

> I accept the explanation given by Ken Bowman, but it is hard to
> explain the values assigned to
>   (1) findgen(15)*.2 -1.4 versus
>   (2) dindgen(15)*(.2D)-(1.4D)
>   (3) and, taken separately, the results of using the total function
>   on (1) and (2) matched with the ability to perform "perfect math"
>   with only 13 values.

> I am describing a small permutation of inaccuracies here, (1) looks ok,
> but (2) does not. total((1)) and total((2)) are not accurate for
> the reasons given by Ken.

> BTW one IDL user indicated that there was *no* trouble with this math on
> his VMS system! Now that is interesting.

I believe that I was that user. Specifically, I reported that the expected answer of 0 was seen when the calculation was done in single precision. However, the same was not true in double precision. The only thing this demonstrates is that the problems of computer round-off error shows up in different ways in VMS than on other computers, and is simply because of the difference between the VAX floating point representation and the more standard IEEE one. In other words, it's a hardware difference, not a software one.

I think the discussion of how computer round-off error manifests itself is interesting. However, it's important to remember that this has nothing to do

with IDL. Round-off errors are determined solely by the floating point processing done by the CPU. It was stated earlier, without testing, that IDL was deficient in this repect relative to Fortran or C--I showed in an earlier message that this was not the case.

William Thompson

Subject: Re: Inaccuracies
Posted by C R Shaw on Thu, 16 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe wrote..

> Ok, I am sure this has been discussed before, but let
> me start this thread again. I wish to create a 15-element
> vector which contains the numbers -1.4 to 1.4 by an increment
> of 0.2 I also wish the sum of these elements to be zero
> (No, this isn't the new math). Here is what I tried...
>
> TRIAL #2
> ========
> IDL> a = dindgen(15)*(.2D)-1.4D
> IDL> print, total(a, /double)
> 4.4408921e-15
>
> Ok, this is better but not correct.
> And what are the values of a?
>
> IDL> print, a
> -1.4000000 -1.2000000 -1.0000000 -0.80000000
> -0.60000000 -0.40000000 -0.20000000 2.2204460e-16
> 0.20000000 0.40000000 0.60000000 0.80000000
> 1.0000000 1.2000000 1.4000000
>
> I seem to have lost a zero somewhere, and for me this matters!!!
> Maybe I am missing something here, but this kind of behavior
> makes IDL a bit problematical for scientific use. With only 15
> numbers and double precision arithmetic, I can't believe this
> would fail in FORTRAN or C!

I've had a similar problem with numerical accuracy.
The problem, however, appears to be in the representation
of the values using the IEEE (I think) floating point
standards.

It is also not just limited to IDL programs. The program

below is a C program to do the same calculations...

```
/*
   PROGRAM: test.c
   PURPOSE: Tests the numerical accuracy of C
   HISTORY: Written by Carl Shaw, Nov 1995
*/

#include <stdio.h>

main()
{
 double count, value, total;

 for (count=0.; count < 15. ;count++)
  {
  value=count*0.2-1.4;
  printf("%g  ", value);
  total=total+value;
  }

 printf("\nTotal = %g \n", total);
}
```

And guess what the results are???

-1.4 -1.2 -1 -0.8 -0.6 -0.4 -0.2 2.22045e-16 0.2 0.4
0.6 0.8 1 1.2 1.4

Total = 4.44089e-15 !

Sound familiar?

For most applications though, a value of
0.000000000000004
is close enough to zero!

Mail : Dept. of Pure and Applied Physics,
Queen's University Belfast,
University Road.
Belfast BT7 1NN
E-mail : C.Shaw@QUB.ac.uk
Phone : (01232) 245133 Ext 3045
Fax : (01232) 438918

Subject: Re: Inaccuracies
Posted by Hermann Mannstein on Thu, 16 Nov 1995 08:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe <afl@cdc.noaa.gov> wrote:
> Ok, I am sure this has been discussed before, but let
> me start this thread again. I wish to create a 15-element
> vector which contains the numbers -1.4 to 1.4 by an increment
> of 0.2 I also wish the sum of these elements to be zero
> (No, this isn't the new math). Here is what I tried...
>
>
> TRIAL #1
> ========
> IDL> a = findgen(15)*.2 - 1.4
> IDL> print, total(a)
>   7.15256e-07
>
> Hmmm! Not so good.
>
>
> TRIAL #2
> ========
> IDL> a = dindgen(15)*(.2D)-1.4D
> IDL> print, total(a, /double)
>   4.4408921e-15
>
> Ok, this is better but not correct.
> And what are the values of a?
>
> IDL> print, a
>   -1.4000000   -1.2000000   -1.0000000   -0.80000000
>   -0.60000000   -0.40000000   -0.20000000   2.2204460e-16
>   0.20000000   0.40000000   0.60000000   0.80000000
>   1.0000000   1.2000000   1.4000000
>
> I seem to have lost a zero somewhere, and for me this matters!!!
>
>
>
> TRIAL #3
> ========
> What if I only needed 13 numbers between -1.2 and 1.2.
> IDL> a = findgen(13)*.2 - 1.2
> IDL> print, total(a)
>   0.00000
>
> Now how can I get this to work for 15 numbers?
>

> Maybe I am missing something here, but this kind of behavior
> makes IDL a bit problematical for scientific use. With only 15
> numbers and double precision arithmetic, I can't believe this
> would fail in FORTRAN or C!
>
> --
> Andrew F. Loughe (afl@cdc.noaa.gov)
> University of Colorado, CIRES * Campus Box 449 * Boulder, CO 80309
> phone: (303) 492-0707   fax: (303) 497-7013

It get clearer when you type:

```
IDL> a=indgen(15) - 7
IDL> print,double(a*.2)
     -1.4000000      -1.2000000      -1.0000000     -0.80000001
    -0.60000002     -0.40000001     -0.20000000       0.0000000
     0.20000000      0.40000001      0.60000002      0.80000001
      1.0000000       1.2000000       1.4000000
IDL>
```

but you get:

```
IDL> print,total(double(a*.2))
      0.0000000
```

and

```
IDL> print,total(a*.2D)
  -2.2204460e-16
```

by the way,

```
IDL> a=indgen(13) - 6
IDL> print,double(a*.2)
     -1.2000000      -1.0000000     -0.80000001     -0.60000002
    -0.40000001     -0.20000000       0.0000000      0.20000000
     0.40000001      0.60000002      0.80000001       1.0000000
      1.2000000
IDL>
```

it's the multiplication, and the 'total' works with 13 elements, because the deviations cancel each other

--

Regards,

~~~~~~

Hermann Mannstein   Tel.: +49 8153 28-2503
Institut fuer Physik der Atmosphaere   or -2558
DLR - Oberpfaffenhofen   Fax.: +49 8153 28-1841
~~~~~~

Postfach 1116 \ mailto:H.Mannstein@dlr.de
D-82230 Wessling \ 0 http://www.op.dlr.de/~pa64
Germany _____\/|_____