Subject: Re: rounding errors
Posted by James Kuyper on Wed, 02 May 2001 15:06:19 GMT
View Forum Message <> Reply to Message

"Dominic R. Scales" wrote:
>
> Craig Markwardt wrote:
>>
>> In principle you should not have to worry about those extra random
>> digits. Since your original data are floating point, then they
>> contain no more than 7 digits of precision. Any digits beyond the 7th
>> are simply *undefined*. If you want higher precision then you should
>> recreate your file using double precision arithmetic from the start.
>>
>> Even if you use double precision then you will still have random
>> digits at the end, but just farther out:
>>
>> IDL> print, 2.56989d, format='(D30.20)'
>> 2.56989000000000000767
>>
>> This is a consequence of how numbers are represented in the base-two
>> number system. Generally speaking the relative uncertainty will be
>> (MACHAR()).EPS for floating point in IDL.
>>
>> Craig
>
> Randall Skelton wrote:
>>
>> Perhaps I am confused, but if you want the data to be represented as
>> doubles, you should read it directly into a double array
>> ('array=dblarr(1000)' or something similar)
>>
>> While you can legitimately extend the precision of floating point numbers
>> to those of doubles you must always remember the underlying IEEE
>> definition which states floats only have 6 digits precision while doubles
>> have 15 digits of precision. When you recast a float into a double, you
>> expect decimal digits 6-15 will be noise because bits beyond the float
>> precision can truly be anything. Asking IDL to make a floating point
>> number into double precision with 'zero padding' like you suggest is like
>> asking IDL to know what decimal digits 6-15 were before you cast them as
>> floats. Using strings as an intermediate type does avoid the problem
>> you describe but it also shows a genuine misunderstanding of storage
>> types.
>>
>> For the record, I had no idea that IDL requires you to explicitly state
>> 'a=2.348339d0' instead of a=double(2.348339).
>>
>> Randall

>>
>> PS: If you are still having trouble with this consider a simple C program:
>>
>
> Dear Craig, dear Randall
>   thanks for your answers.
>
>   Unfortunately, the input data is a fixed format I get from an external
>   source and can not change. I fully appreciate and understand the precision
>   difference in significant digits between float and double, be it in idl,
>   c, or any other programming language, as it is inherently diffucult
>   (read impossible) to map an infinite set of numbers to a finite realm of
>   4/8 byte and not miss any.
>   The case I am trying to make is this: why aren't the missing digits, i.e.
>   the ones added by the cast, set to 0.? As I go along with my calculations,

There are no "extra" digits being set to 0, because there aren't any digits, extra or otherwise. Every single bit in the mantissa is being used to represent the number you requested as accurately as possible as a binary fraction. It simply can't be represented as an exact binary fraction. This isn't unusual; it's true for most numbers that can be represented by a decimal fraction. This isn't unique to binary formats; no matter what base you use, most numbers can't be represented exactly in a finite number of digits, even if you restrict yourself to rational numbers.

This is why many early computers experimented with various binary coded decimal formats; basically, each half of a byte was used to store a single decimal digit. Decimal numbers could be represented exactly. However, that advantage was not sufficient to sustain them. These schemes have been pretty much abandoned in favor of pure binary formats. I can't vouch for the reason, but I suspect that the hardware implementation of math operations on BCD formats was woefully inefficient compared to using true binary formats.