Subject: Re: Wow. exp() difficulties...
Posted by William Clodius on Thu, 24 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

lady of the elves wrote:
>
> I've encountered a really interesting--though annoying--problem. I'm
> trying to use exp(a), where a>88 or so. My calculator can do it...the
> program can't. It gives me a "floating underflow" error.
>
> Can anyone give me more information on how to work this out? I haven't
> yet figured out long numbers...nor how they differ from floating point.
> I don't know if these uncertainties relate, but all of my numbers are
> floating point.
>
> Is it just that the language can't handle the numbers? e^89=4.4*10^38;
> it's rather large--but is it just that it can't handle the size? Or the
> precision? Is it that I need to know more about long numbers?
>
> What *is* a floating underflow?

Ouch!!

While I know that a lack of knowledge about how computers do arithmetic is common, it pains me to see it displayed in a public forum for people who do a lot of computer arithmetic. You are not alone in your ignorance, and there are others that will benefit from what I am about to say, but realize that I am only going to give the most cursory overview and you should really read the manual and some texts on computer arithmetic.

Computers store and retrieve data from memory that (except for a few eperimental machines that will never run IDL) invariably treats memory as a large linear sequence of bits, 2 valued entities. For convenience these bits are grouped into chunks called bytes that are almost invariably 8 bits long. IDL will not run on machines without this definition of bytes. Bytes can be thought of as integers whose values range from -128 to 127 or from 0 to 255 depending on context. In other contexts, on processors in countries with reasonably small character sets, these integer values are treated as mapping to individual characters. In certain contexts bytes are grouped together in sets of two and treated as integers whose values may range from -32768 to 32767. This is the default IDL integer type, called SHORT. In other contexts bytes are grouped together in sets of four and treated as integers whose values range from -2147483648 to 2147483647. This is the IDL integer type called LONG. Most computers provide only these four integer types and they are the only ones IDL provides, although a few machines now also provide integers defined in terms of eight bytes. It is possible

for languages to define integers beyond those intrinsic to the machine, but they have to be emulated through software and suffer a significant performance disadvantage. IDL does not define such integers. (Note the value ranges above assume a twos complement implementation of integers).

In many cases it is desirable to have locations in memory that are treated as representing rational numbers. The first thing to realize is that these rational numbers will at best be approximations to irrational numbers because of the finite memory of the machine. On the vast majority of machines these representations of rational numbers will be in terms of four or eight byte numbers, where the four byte numbers are IDL's FLOAT data type and the eight byte numbers are IDL's DOUBLE data type. The second thing to realize is that, because of the finite size of the storage for these numbers, only a (relatively small) fixed set of rational number can be represented by these types. The machine treats these numbers as consisting of two parts a mantisa, which can be thought of as an integer value, and an exponent, which represents a power of two that multiplies the integer. The third thing to realize is that the useage of a power of two for the exponent means that the processor can not exactly represent many numbers that novices assume are simple enought to be exactly represented, i.e., 1./3. or 0.2. The power of two is desirable because it makes the math easy to implement and ensures uniform and predictable coverage of the real number space. Again it is possible for languages to provide floating point math other than what is provided by the machine, but it requires emulation in software with a significant performance impact. IDL only provides the machine based representations. (NOTE: because calculators often deal with few numbers at a time, they can often devote more storeage space per number and hence can represent numbers with a higher degree of accuracy than most computers.)

The vast majority of processors now define their rational number approximate representation in terms of the IEEE Standard For Binary Floating Point Arithmetic (ANSI/IEEE Std 754-1985). This defines the processor's behavior for certain conditions so that it can reliably report real or potential errors. These errors include generating an infinity (e.g., dividing a finite number by zero), generating an undefined number (e.g., dividing zero by zero), overflow (e.g., mutiplying two large numbers to generate a number too large to be represented), and underflow (e.g., dividing a large number into a small number and generating a finite number that is too small to be represented accurately). Machines with IEEE arithmetic cannot represent e^89 as a FLOAT, but can represent it (in some relatively good approximation) as a DOUBLE. Because you got an underflow error I suspect your calculation either involved e^(-89) or 1./e^89 and not just e^89.

William Kahan's page is a usefull source of additional information

http://HTTP.CS.Berkeley.EDU/~wkahan/

--

William B. Clodius Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2 FAX: (505)-667-3815
PO Box 1663, MS-C323 Group office: (505)-667-5776
Los Alamos, NM 87545 Email: wclodius@lanl.gov