Subject: Locate an underflow
Posted by francesco.spada on Tue, 22 May 2001 21:59:09 GMT
View Forum Message <> Reply to Message

Hi,

I've got this error

% Program caused arithmetic error: Floating underflow

Is it possible to know where it appens without putting a million of
stop or breakpoint in my program?

Thank you

Ciao

Francesco

Subject: Re: Locate an underflow
Posted by davidf on Wed, 23 May 2001 15:16:56 GMT
View Forum Message <> Reply to Message

William Thompson (thompson@orpheus.nascom.nasa.gov) writes:

> What I would really like is something that would differentiate between
> underflow errors and all other exceptions. I think that most users don't know
> what an underflow "error" is, and their initial reaction is that something bad
> is going on.

Hear! Hear!

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Locate an underflow
Posted by Paul van Delst on Wed, 23 May 2001 16:59:29 GMT
View Forum Message <> Reply to Message

William Thompson wrote:
>
> m.hadfield@niwa.cri.nz ("Mark Hadfield") writes:
>
>> From: "Francesco" <francesco.spada@jrc.it>
>>> I've got this error
>>>
>>> % Program caused arithmetic error: Floating underflow
>>>
>>> Is it possible to know where it appens without putting a million of
>>> stop or breakpoint in my program?
>
>> Set the !EXCEPT system variable to 2.
>
>> From IDL 5.4 documentation:
>
>> !EXCEPT
>
>> An integer variable that controls when IDL checks for invalid
>> mathematical computations (exceptions), such as division by zero.
>> The three allowed values are:
>
>> Value Description
>
>> 0 Never report exceptions.
>
>> 1 Report exceptions when the interpreter is returning to an
>> interactive prompt (the default).
>
>> 2 Report exceptions at the end of each IDL statement. Note that
>> this slows IDL by roughly 5% compared to setting !EXCEPT=1.
>
>> For more information on invalid mathematical computations and
>> error reporting, see Math Errors. The value of !EXCEPT is used by
>> the CHECK_MATH function to determine when to return errors. See
>> CHECK_MATH for details.
>
>> ---
>> Mark Hadfield
>> m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield
>> National Institute for Water and Atmospheric Research
>
> What I would really like is something that would differentiate between
> underflow errors and all other exceptions. I think that most users don't know
> what an underflow "error" is, and their initial reaction is that something bad
> is going on.

Hmm. I do see your point, but if I grab someone else's code (not just IDL code BTW) the

first thing I do is run their supplied test case (I hope there is one) with all warning
flags on (for IDL, !EXCEPT = 2; for Fortran or similar, set the platform specific compiler
switch to trap under/overflows, divide by zero, etc.).

If, on running said code, I get a crapload of underflow errors, it's an indication that
that either a) the code hasn't been tested very well or b) the programmer didn't really
think about the problem enough (and I'm guilty of both of these.... most of the time
actually). If there are (usually harmelss) underflow errors, how do I know that there
won't be other more serious errors at some point for different input?

Writing code for operational use (i.e. has to work all the time and when it doesn't has to
handle the error) has made me think a lot more carefully about what numbers are getting
crunched and/or squashed in my code.

> Even when you do know that underflow "errors" are (almost always) harmless,
> continuously getting these messages is highly annoying. I'd use !EXCEPT=0 to
> suppress underflow messages, except that it would also suppress other
> potentially more serious error messages as well.

You state that there are cases where underflows are not harmless (via the "almost always"
qualifier) but then go on to say you would like to suppress their reporting? Why not fix
the error so you don't get underflows for that rare case where they CAN cause a problem?

paulv

--
Paul van Delst A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP Drink deep, or taste not the Pierian spring;
Ph: (301)763-8000 x7274 There shallow draughts intoxicate the brain,
Fax:(301)763-8545 And drinking largely sobers us again.
Alexander Pope.

Subject: Re: Locate an underflow
Posted by Craig Markwardt on Wed, 23 May 2001 18:25:11 GMT

View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:

> Hmm. I do see your point, but if I grab someone else's code (not
> just IDL code BTW) the first thing I do is run their supplied test
> case (I hope there is one) with all warning flags on (for IDL,
> !EXCEPT = 2; for Fortran or similar, set the platform specific
> compiler switch to trap under/overflows, divide by zero, etc.).
>
> If, on running said code, I get a crapload of underflow errors, it's
> an indication that that either a) the code hasn't been tested very

> well or b) the programmer didn't really think about the problem
> enough (and I'm guilty of both of these.... most of the time
> actually). If there are (usually harmelss) underflow errors, how do
> I know that there won't be other more serious errors at some point
> for different input?

Yah, but consider the difference between the following bits of code:

```
1>  y = exp(-x^2)

2>  u = x^2
2>  sz = size(x)
2>  isdouble = sz(sz(0)+1) EQ 5
2>  mask = u LT alog(machar(double=isdouble).xmax)
2>  y = mask*exp(-u*mask)
```

Both sets of code accomplish the same thing, computing a gaussian
function, except the second one avoids bogus underflow error messages.
Which one do you think I'd rather write? :-) Which one shows the
original mathematical intent more ?

Craig

--
 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: Locate an underflow
Posted by thompson on Wed, 23 May 2001 21:01:19 GMT

View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:

> ... If, on running said code, I get a crapload of underflow errors, it's an
> indication that that either a) the code hasn't been tested very well or b) the
> programmer didn't really think about the problem enough ...

I disagree. It's exceedingly easy to get underflow errors, and extremely
difficult to program around them. For example, a simple Gaussian

```
 Y = A*exp(-((X-X0)/Sig)^2)
```

is almost guaranteed to generate underflow errors. At some point this is going
to be indistinguishable from zero. You'd have to jump through hoops to avoid

getting the completely useless underfloat messages.

William Thompson

Subject: Re: Locate an underflow
Posted by thompson on Wed, 23 May 2001 21:13:27 GMT
View Forum Message <> Reply to Message

Sorry to chime in again. I should have included this in my previous post.

Paul van Delst <paul.vandelst@noaa.gov> writes:

> ... You state that there are cases where underflows are not harmless (via the
> "almost always" qualifier) but then go on to say you would like to suppress
> their reporting? Why not fix the error so you don't get underflows for that
> rare case where they CAN cause a problem? ...

That's because, while I can imagine in theory that there might be a situation
where one would want to be informed of a floating underflow error, I've never
actually run into one in practice. But underflow error messages are
ubiquitous.

William Thompson

Subject: Re: Locate an underflow
Posted by Paul van Delst on Thu, 24 May 2001 14:21:50 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> Paul van Delst <paul.vandelst@noaa.gov> writes:
>
>> Hmm. I do see your point, but if I grab someone else's code (not
>> just IDL code BTW) the first thing I do is run their supplied test
>> case (I hope there is one) with all warning flags on (for IDL,
>> !EXCEPT = 2; for Fortran or similar, set the platform specific
>> compiler switch to trap under/overflows, divide by zero, etc.).
>>
>> If, on running said code, I get a crapload of underflow errors, it's
>> an indication that that either a) the code hasn't been tested very
>> well or b) the programmer didn't really think about the problem
>> enough (and I'm guilty of both of these.... most of the time
>> actually). If there are (usually harmelss) underflow errors, how do
>> I know that there won't be other more serious errors at some point

```
>>  for different input?
>
>  Yah, but consider the difference between the following bits of code:
>
>  1>  y = exp(-x^2)
>
>  2>  u = x^2
>  2>  sz = size(x)
>  2>  isdouble = sz(sz(0)+1) EQ 5
>  2>  mask = u LT alog(machar(double=isdouble).xmax)
>  2>  y = mask*exp(-u*mask)
>
>  Both sets of code accomplish the same thing, computing a gaussian
>  function, except the second one avoids bogus underflow error messages.
>  Which one do you think I'd rather write? :-)
```

The one that avoids errors? :o)

> Which one shows the original mathematical intent more ?

If the code is commented I fail to see the problem. Really. How about

```
  y = gaussian_function( x )
```

which encapsulates all the checking? (Prefixed with the original author's initials of
course to avoid namespace collisions... :o)

To be fair, the entire debate has no meaning without some context. I agree with you (and
William Thompson) completely for "regular" stuff - you know, the day to day computing that
everyone does. However, for applications (e.g. flight control software, numerical weather
prediction, etc.) upon which a lot more is at stake (e.g. lives, property damage, etc) I
think it's worth the extra (not much more) effort. And when you've done it once, you just
re-use the same function/routine/procedure/whatever.

paulv

```
--
Paul van Delst           A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP        Drink deep, or taste not the Pierian spring;
Ph: (301)763-8000 x7274  There shallow draughts intoxicate the brain,
Fax:(301)763-8545        And drinking largely sobers us again.
                                   Alexander Pope.
```

Subject: Re: Locate an underflow
Posted by Paul van Delst on Thu, 24 May 2001 14:36:01 GMT
View Forum Message <> Reply to Message

William Thompson wrote:
>
> Paul van Delst <paul.vandelst@noaa.gov> writes:
>
>> ... If, on running said code, I get a crapload of underflow errors, it's an
>> indication that that either a) the code hasn't been tested very well or b) the
>> programmer didn't really think about the problem enough ...
>
> I disagree. It's exceedingly easy to get underflow errors, and extremely
> difficult to program around them. For example, a simple Gaussian
>
> Y = A*exp(-((X-X0)/Sig)^2)
>
> is almost guaranteed to generate underflow errors. At some point this is going
> to be indistinguishable from zero. You'd have to jump through hoops to avoid
> getting the completely useless underfloat messages.

Not really. what about something like (assuming double precision):

```
  tolerance = (MACHAR(/DOUBLE)).EPS

  y = DBLARR( N_ELEMENTS( X ) )
  xarg = ((X-X0)/Sig)^2
  index = WHERE( xarg < tolerance, count )
  IF ( count GT 0 ) THEN $
    y = A*exp(-xarg[index])
```

Or, as I mentioned in my reply to Craig:

```
  y = A * gaussian_function( (X-X0)/Sig )
```

(or similar) which contains all the bits and pieces for checking. Craig also provided a
method of avoiding the underflows.

paulv

--
Paul van Delst   A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP   Drink deep, or taste not the Pierian spring;
Ph: (301)763-8000 x7274 There shallow draughts intoxicate the brain,
Fax:(301)763-8545   And drinking largely sobers us again.
                           Alexander Pope.

Subject: Re: Locate an underflow
Posted by thompson on Thu, 24 May 2001 15:42:39 GMT
View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:

> William Thompson wrote:
>>
>> Paul van Delst <paul.vandelst@noaa.gov> writes:
>>
>>> ... If, on running said code, I get a crapload of underflow errors, it's an
>>> indication that that either a) the code hasn't been tested very well or b) the
>>> programmer didn't really think about the problem enough ...
>>
>> I disagree. It's exceedingly easy to get underflow errors, and extremely
>> difficult to program around them. For example, a simple Gaussian
>>
>>   Y = A*exp(-((X-X0)/Sig)^2)
>>
>> is almost guaranteed to generate underflow errors. At some point this is going
>> to be indistinguishable from zero. You'd have to jump through hoops to avoid
>> getting the completely useless underfloat messages.

> Not really. what about something like (assuming double precision):

```
>   tolerance = (MACHAR(/DOUBLE)).EPS

>   y = DBLARR( N_ELEMENTS( X ) )
>   xarg = ((X-X0)/Sig)^2
>   index = WHERE( xarg < tolerance, count )
>   IF ( count GT 0 ) THEN $
>     y = A*exp(-xarg[index])
```

And that's not jumping through hoops???

> Or, as I mentioned in my reply to Craig:

```
>   y = A * gaussian_function( (X-X0)/Sig )
```

> (or similar) which contains all the bits and pieces for checking. Craig also provided a
> method of avoiding the underflows.

When IDL first starting printing out tons of underflow errors, and users
started complaining about them, I also wrote out a routine very much like what
you suggest, called SAFE_EXP(), simply to avoid the messages. In the end,
though, it didn't help, because one ends up multiplying arrays with very tiny
numbers in them with other arrays with very tiny numbers in them, and you end
up with an underfloat anyway. They're all over the place. In the end, you
just give up and ignore them.

If somebody has a real need to make sure that underfloat errors are not
occuring in their programs, then there should be a mechanism to check for that,
as there is. But must people don't care, and find the messages useless and
annoying. There should be a way of suppressing them, if you want. RSI has
supplied a way of doing that, but only by suppressing *ALL* error messages.
All I'm asking for is an option for selectively suppressing the underfloat
messages.

Underfloat messages are like the kid who kept crying wolf. Eventually, you
start ignoring them.

William Thompson

Subject: Re: Locate an underflow
Posted by George N. White III on Fri, 25 May 2001 12:08:26 GMT
View Forum Message <> Reply to Message

On 23 May 2001, William Thompson wrote:

> Paul van Delst <paul.vandelst@noaa.gov> writes:
>
>> ... If, on running said code, I get a crapload of underflow errors,
>> it's an indication that that either a) the code hasn't been tested
>> very well or b) the programmer didn't really think about the problem
>> enough ...

Sound advice.

> I disagree. It's exceedingly easy to get underflow errors, and extremely
> difficult to program around them. For example, a simple Gaussian
>
> Y = A*exp(-((X-X0)/Sig)^2)
>
> is almost guaranteed to generate underflow errors. At some point this
> is going to be indistinguishable from zero. You'd have to jump
> through hoops to avoid getting the completely useless underfloat
> messages.

I do things like this every day, but only in the privacy of my own
computer. I try to avoid it in code that other people will use or
even code that I might want to use next year.

In the good old days underflow errors from things like the above
example often were harmless, but:

a) you need to be confident that the program doesn't rely on the

the assumption that exp(-x) is always positive, and

b) underflow can be very expensive on modern hardware. In cases
where underflow doesn't alter the results of the calculation,
you are often dealing with something like "a+c*exp(-x)" where
"c*exp(-x)" is so small that the f.p. representation of a
doesn't change for values of x much small than the values that
produce f.p. underflow in exp(-x).

I've seen a factor of 10 speedup for a program running on SGI MIPS by
replacing y=a+c*exp(-x) with a version that skips the evaluation of
exp(-x) when c*exp(-x) is too small to affect the f.p. representation
of a. For IA-64 the effect is likely to be much more severe
because the f.p. exception handling loads a software f.p. emulator so the
operation producing the exception can be replayed to gather data on
the cause of the exception.

Combined multiply-add instructions (y=a+b*x) aren't uncommon, and in the
future you may well see single instructions for things like y=a+b*exp(x).
If you get an exception in such an instruction, how do you determine which
operation was responsible? Exception handling is an increasingly tricky
thing to implement, and isn't much tested by current benchmark codes, so
the chances are good that it won't work right on new hardware or after a
software upgrade or just because the vendor changed the compiler flags
governing exception handling and you didn't take time to study the
release notes saying that exception handling only works for certain
(low) optimization levels.

In most cases a simple range test will avoid underflow when evaluating
a Gaussian. Not only does this mean that the program will have more
predictable performance on a variety of hardware, it makes it much
easier to examine the other situations where undeflow occurs to
ensure that they are harmless.

--
George N. White III <gnw3@acm.org> Bedford Institute of Oceanography