Subject: Re: machine precision
Posted by David Fanning on Mon, 18 May 2009 11:53:36 GMT
View Forum Message <> Reply to Message

Wox writes:

```
>  When checking whether two foating point variables are equal, one has
>  to do this:
>
>  pres = (machar()).eps
>  bequal = abs(f1-f2) lt pres
>
>  This can go wrong however, as illustrated by the example below. Do I
>  need to do error propagation on this? This means that every time f1
>  and f2 are calculated differently, I have to calculate a different
>  uncertainty? This seems like a lot of work, not to mention the machine
>  precision in calculation the propagation of uncertainty... Is there a
>  more general rule of thumb I can use?
>
>  vec1=[1.,2,3,4,5]
>  vec2=vec1
>  pres=(machar()).eps
>  norm1=sqrt(total(vec1^2,1,/pres))
>  norm2=sqrt(total(vec2^2,1,/pres))
>  f1=total(vec1*vec2,/pres) ; inner product
>  f2=norm1*norm2 ; product of the norms
>  ; f1 and f2 must be equal so
>  if abs(f1-f2) ge pres then print,'wrong wrong wrong...'
```

Uh, Wox, I think you need to read this article again:

  http://www.dfanning.com/math_tips/sky_is_falling.html

I think if you try:

```
  vec1 = [1.D, 2, 3, 4, 5]
```

you might find a different result. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: machine precision
Posted by Wout De Nolf on Mon, 18 May 2009 13:04:05 GMT
View Forum Message <> Reply to Message

On Mon, 18 May 2009 05:53:36 -0600, David Fanning <news@dfanning.com>
wrote:

> Uh, Wox, I think you need to read this article again:
>
>   http://www.dfanning.com/math_tips/sky_is_falling.html

Ow boy, sorry for bringing this up again, but in the last example
given in your famous article, why is:

```
f=470.0 - (4.70*100)
d=470d - (4.70d*100)
print,'Bigger than precision:',f,(machar()).eps
print,'Smaller than precision:',d,(machar(/double)).eps
```

I understand that when you give 4.7 to a computer, it stores a number
close to it. When given 4.70d, the same thing happens, only now we're
closer than with single-precision. Why is the machine precision (EPS
from machar) not reflecting this?

Subject: Re: machine precision
Posted by David Fanning on Mon, 18 May 2009 13:30:42 GMT
View Forum Message <> Reply to Message

Wox writes:

> Ow boy, sorry for bringing this up again, but in the last example
> given in your famous article, why is:
>
> f=470.0 - (4.70*100)
> d=470d - (4.70d*100)
> print,'Bigger than precision:',f,(machar()).eps
> print,'Smaller than precision:',d,(machar(/double)).eps
>
> I understand that when you give 4.7 to a computer, it stores a number
> close to it. When given 4.70d, the same thing happens, only now we're
> closer than with single-precision. Why is the machine precision (EPS
> from machar) not reflecting this?

I think the MACHAR values *do* reflect this. But the point
is, when you compare two numbers that are nearly equal
to each other, the *difference* can easily happen in
the significant digits that can be garbage (or, rather,

undefined) in a floating point number. That is to say,
the difference can be in the part of the number that is
beyond the realm of floating point accuracy.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: machine precision
Posted by jameskuyper on Mon, 18 May 2009 13:33:48 GMT

View Forum Message <> Reply to Message

Wox wrote:
> Hi all,
>
> When checking whether two foating point variables are equal, one has
> to do this:
>
> pres = (machar()).eps
> bequal = abs(f1-f2) lt pres
>
> This can go wrong however, as illustrated by the example below. Do I
> need to do error propagation on this? This means that every time f1
> and f2 are calculated differently, I have to calculate a different
> uncertainty?

Yes.

> ... This seems like a lot of work, not to mention the machine
> precision in calculation the propagation of uncertainty... Is there a
> more general rule of thumb I can use?
>
> vec1=[1.,2,3,4,5]
> vec2=vec1
> pres=(machar()).eps
> norm1=sqrt(total(vec1^2,1,/pres))
> norm2=sqrt(total(vec2^2,1,/pres))
> f1=total(vec1*vec2,/pres) ; inner product
> f2=norm1*norm2 ; product of the norms
> ; f1 and f2 must be equal so

At a minimum, you should use pres*(f1^2+f2^2)^0.5 instead of pres for
the following comparison. eps gives you the relative precision, not the
absolute precision; it needs to be scaled by the numbers you're working
with.

> if abs(f1-f2) ge pres then print,'wrong wrong wrong...'

Subject: Re: machine precision
Posted by Wout De Nolf on Mon, 18 May 2009 13:55:25 GMT
View Forum Message <> Reply to Message

On Mon, 18 May 2009 07:30:42 -0600, David Fanning <news@dfanning.com>
wrote:

> I think the MACHAR values *do* reflect this. But the point
> is, when you compare two numbers that are nearly equal
> to each other, the *difference* can easily happen in
> the significant digits that can be garbage (or, rather,
> undefined) in a floating point number. That is to say,
> the difference can be in the part of the number that is
> beyond the realm of floating point accuracy.

If "the difference is in the garbage", isn't it lower than
(machar()).eps ? In the example, it's higher.

Subject: Re: machine precision
Posted by Wout De Nolf on Mon, 18 May 2009 14:02:54 GMT
View Forum Message <> Reply to Message

On Mon, 18 May 2009 13:33:48 GMT, James Kuyper
<jameskuyper@verizon.net> wrote:

> At a minimum, you should use pres*(f1^2+f2^2)^0.5 instead of pres for
> the following comparison. eps gives you the relative precision, not the
> absolute precision; it needs to be scaled by the numbers you're working
> with.

Now this I didn't realize! So it's the precision of the mantisse or
something?

Subject: Re: machine precision
Posted by jameskuyper on Tue, 19 May 2009 01:44:42 GMT
View Forum Message <> Reply to Message

Wox wrote:
> On Mon, 18 May 2009 13:33:48 GMT, James Kuyper
> <jameskuyper@verizon.net> wrote:
>
>> At a minimum, you should use pres*(f1^2+f2^2)^0.5 instead of pres for
>> the following comparison. eps gives you the relative precision, not the
>> absolute precision; it needs to be scaled by the numbers you're working
>> with.
>
> Now this I didn't realize! So it's the precision of the mantisse or
> something?

Yes. If you don't understand why, I recommend re-reading the "The Sky is
Falling!" website that David referred you to, and in particular the
paper it refers to titled "What Every Computer Scientist Should Know
about Floating Point Arithmetic".

Subject: Re: machine precision
Posted by jeffnettles4870 on Tue, 19 May 2009 15:16:34 GMT
View Forum Message <> Reply to Message

On May 18, 9:44 pm, James Kuyper <jameskuy...@verizon.net> wrote:
> Wox wrote:
>> On Mon, 18 May 2009 13:33:48 GMT, James Kuyper
>> <jameskuy...@verizon.net> wrote:
>
>>> At a minimum, you should use pres*(f1^2+f2^2)^0.5 instead of pres for
>>> the following comparison. eps gives you the relative precision, not the
>>> absolute precision; it needs to be scaled by the numbers you're working
>>> with.
>
>> Now this I didn't realize! So it's the precision of the mantisse or
>> something?
>
> Yes. If you don't understand why, I recommend re-reading the "The Sky is
> Falling!" website that David referred you to, and in particular the
> paper it refers to titled "What Every Computer Scientist Should Know
> about Floating Point Arithmetic".

I think I have the beginnings of an understanding of this problem
after having seen the topic pop up so much here in the newsgroup and
after having read David's article several times. But now i'm
wondering how other languages handle this situation? I've tried two
other languages now, perl and VB, and they both seem to not give the
"correct-but-not-expected" results that trip so many people up. For
example, this snippet of perl code:

$i = int(4.70*100);
print "$i\n";

prints 470 whereas the equivalent IDL code: print, fix(4.70*100)
prints 469.

Does anyone have any idea how other languages deal with the "sky is
falling" problem?

Jeff

Subject: Re: machine precision
Posted by Kenneth P. Bowman on Tue, 19 May 2009 18:15:56 GMT
View Forum Message <> Reply to Message

In article
<6c259101-aa56-4964-b01f-cd1c96bdcfc3@m24g2000vbp.googlegroups.com>,
 "Jeff N." <jeffnettles4870@gmail.com> wrote:

> $i = int(4.70*100);
> print "$i\n";
>
> prints 470 whereas the equivalent IDL code: print, fix(4.70*100)
> prints 469.

Have you looked to see exactly what the perl fix function does?
Perhaps it rounds.

In IDL you might want to try ROUND, CEIL, or FLOOR, depending
on exactly what you are trying to accomplish.

People often use LONG when what they really want is ROUND.

Ken Bowman

Subject: Re: machine precision
Posted by jameskuyper on Wed, 20 May 2009 01:41:51 GMT
View Forum Message <> Reply to Message

Jeff N. wrote:
...
> I think I have the beginnings of an understanding of this problem
> after having seen the topic pop up so much here in the newsgroup and
> after having read David's article several times. But now i'm
> wondering how other languages handle this situation? I've tried two

> other languages now, perl and VB, and they both seem to not give the
> "correct-but-not-expected" results that trip so many people up. For
> example, this snippet of perl code:
>
> $i = int(4.70*100);
> print "$i\n";
>
> prints 470 whereas the equivalent IDL code: print, fix(4.70*100)
> prints 469.

You've misunderstood the problem if you think that proves that the
problem doesn't occur in perl. Just because perl gave you the result you
expected in this case doesn't mean that it always will. When I typed
'man perlfun' on my home machine and searched for "int EXPR', I found:

"You should not use this function for rounding: one because it truncates
towards 0, and two because machine representations of floating point
numbers can sometimes produce counterintuitive results. For example,
"int(-6.725/0.025)" produces -268 rather than the correct -269; that’s
because it’s really more like -268.99999999999994315658 instead."

> Does anyone have any idea how other languages deal with the "sky is
> falling" problem?

The "sky is falling problem" is fundamentally a problem with
programmer's expectations being out of line with the way floating point
numbers actually work. The solution is eduction, there's really not a
lot that the language can do about it. For this specific kind of
problem, if you've got a value 'x' that should be a number close to an
integer, but which might be slightly more or slightly less than the
exact integer value, fix(x) is the wrong way to handle it. round() is
more appropriate.

Subject: Re: machine precision
Posted by David Fanning on Wed, 20 May 2009 03:01:29 GMT
View Forum Message <> Reply to Message

James Kuyper writes:

> The "sky is falling problem" is fundamentally a problem with
> programmer's expectations being out of line with the way floating point
> numbers actually work. The solution is eduction

And we are making tremendous progress. I don't even
have to think any more. Any article that gets flagged
by my newsgroup filter for having the phrase "Can someone
please explain this!!!!" is automatically sent a link

to The Sky Is Falling article:

 http://www.dfanning.com/math_tips/sky_is_falling.html

Easy. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: machine precision
Posted by Carsten Lechte on Wed, 20 May 2009 08:46:54 GMT
View Forum Message <> Reply to Message

Jeff N. wrote:
> Does anyone have any idea how other languages deal with the "sky is
> falling" problem?

They default to double precision?

chl

Subject: Re: machine precision
Posted by jameskuyper on Wed, 20 May 2009 10:28:08 GMT
View Forum Message <> Reply to Message

Carsten Lechte wrote:
> Jeff N. wrote:
>> Does anyone have any idea how other languages deal with the "sky is
>> falling" problem?
>
> They default to double precision?

That reduces some of the related problems, has no effect on the others,
and doesn't remove any of them.

Subject: Re: machine precision

Posted by Carsten Lechte on Wed, 20 May 2009 10:44:12 GMT
View Forum Message <> Reply to Message

James Kuyper wrote:
> That reduces some of the related problems, has no effect on the others,
> and doesn't remove any of them.

Yes, it always falls to the programmer to really deal with these issues.

chl

Subject: Re: machine precision
Posted by Wout De Nolf on Wed, 20 May 2009 13:08:47 GMT
View Forum Message <> Reply to Message

Ok, so I was reading the Sky Is Falling paper and the Goldberg paper again. I learned some things I thought I'd share (since this is a recurring issue, despite the Sky Is Falling paper).

A floating point number is stored like this:
f(binary) = sign | exponent | mantissa without leading 1
    sign: 1bit
    exponent: 8bits (11bits when double)
    mantissa: 23bits (52bits when double)

The real number it represents can be found like this
f = sign.mantissa.base^(exponent-bias-n_mantissa)
    sign: -1 when sign-bit=1, +1 when sign-bit=0
    base: 2 (ibeta from MACHAR)
    exponent: 8bit number
    bias: 127 (1023 when double)
    n_mantissa: number of mantissa bits (23, 52 when double)

We will rewrite this as
f = sign.mantissa.eps.base^exp
    eps: base^(-n_mantissa)   (eps from MACHAR)
    exp: exponent-bias

For example: f = 470.
    f(binary) = 0 | 10000111 | 11010110000000000000000
    sign = +1
    exp = 135 - 127 = 8
    mantissa = 15400960
    eps = 2.^(-23)
f(stored) = 15400960*2.^(-15)

The difference between a stored floating point number f1 and its closest neighbour f2:

abs(f1-f2) = eps.(mantissa1.base^exp1-mantissa2.base^exp2)

```
  smallest possible difference when:
  exp1 = exp2 = exp
  mantissa1 = mantissa2 +1
                = eps.base^exp = 1 ulp (unit in last place)
```

The absolute error made when storing a real number is therefore

abserr = abs(freal-f) <= c ulp

where c=1 for truncation and c=0.5 for rounding

The relative error made is

```
relerror = abs(freal-f)/abs(freal)
         <= c.eps.base^exp/abs(freal)
         <= c.eps   (not sure about this last step....)
```

Finally, two numbers are considered equal if

relerr = abs(f1-f2)/(abs(f1)>abs(f2)) <= eps

I'm not really sure about this one either (e.g. what should be in the denominator, what about c,...)

All this doesn't deal with accumulated errors in floating point arithmetic, only with errors introduced by storing a real number.