

Re: Floating point arithmetic (epsilon)

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2004-01/1049.html>

From: Kenny Tilton (*ktilton_at_nyc.rr.com*)

Date: 01/20/04

Date: Mon, 19 Jan 2004 23:02:08 GMT

Steven E. Harris wrote:

> *I am writing a small program that employs Newton's method to solve for
> roots of an equation. The function to drive Newton's method is simple;
> my trouble lies in finding a reasonable accuracy threshold to stop the
> refining iterations.*

>

> *The problem is floating-point arithmetic accuracy. When the value
> "delta" in the function below gets too small, the (decf val delta)
> form fails to change the value "val." First, take a look at the
> current implementation:*

>

>

> *(defun newtons-method (func func-prime initial-val max-iterations
> &optional (threshold single-float-negative-epsilon))
> (loop repeat max-iterations
> with val = initial-val
> do (let ((r (funcall func val)))
> (when (< (abs r) threshold)
> (return val))
> (let ((dr (funcall func-prime val)))
> (let ((delta (/ r dr)))
> (when (< (abs delta) threshold)
> (return val))
> (decf val delta))))
> finally return
> (restart-case
> (error 'convergence-failure :value val
> :threshold threshold
> :iterations max-iterations)
> (use-final-value ()
> :report (lambda (stream)
> (format stream "Use final computed value ~A." val))
> val)
> (use-value (other-val)
> :report (lambda (stream)
> (format stream "Input another value instead of ~A." val))
> :interactive (lambda ())*

comp.lang.lisp: Re: Floating point arithmetic (epsilon)

```
> ;; Should the stream be 't' instead?
> (format *query-io* "Use instead of ~A: " val)
> (multiple-value-list (eval (read))))
> other-val)))
>
>
>
> With some tracing in place, a sample invocation looks like this:
>
> ,-----
> | initial val for newton's method: -8.700001
> | newton's method val: -8.700001
> | newton's method r: 0.004221797
> | newton's method dr: -6.665623
> | newton's method delta: -6.333687E-4
> |
> | newton's method val: -8.699368
> | newton's method r: 1.4066696E-5
> | newton's method dr: -6.6235843
> | newton's method delta: -2.1237288E-6
> |
> | newton's method val: -8.699366
> | newton's method r: 1.4305115E-6
> | newton's method dr: -6.623458
> | newton's method delta: -2.1597653E-7
> |
> | newton's method val: -8.699366
> | newton's method r: 1.4305115E-6
> | newton's method dr: -6.623458
> | newton's method delta: -2.1597653E-7
> |
> | [Accept the first restart...]
> |
> | final normal scale: -8.699366
> `-----
>
> Note how all the values (val and delta in particular) settle in and
> don't change through the iterations. I ran some tests (CLISP 2.32) to
> see what's going on:
>
>
>>(type-of -8.699366)
>
> SINGLE-FLOAT
>
>
>>(type-of -2.1597653E-7)
>
> SINGLE-FLOAT
>
>
```

comp.lang.lisp: Re: Floating point arithmetic (epsilon)

```
>>(= -8.699366 (- -8.699366 -2.1597653E-7))
>
> T
>
>
>>(= -8.699366 (- -8.699366 single-float-negative-epsilon))
>
> T
>
>
>>( < (abs -2.1597653E-7) single-float-negative-epsilon)
>
> NIL
>
>
>>( < (abs -2.1597653E-7) single-float-epsilon)
>
> NIL
>
> So 2.1597653E-7, which we'll call "delta," is greater than
> single-float(-negative)-epsilon, suggesting that delta is large enough
> to do useful arithmetic with on single-floats. But subtracting delta
> from our stuck "val" -8.699366 does nothing. I must be
> misunderstanding how to interpret these epsilon constants.
>
> I'm looking for how to express the threshold such that when delta
> becomes so small that adding it to or subtracting it from some other
> number produces no observable difference, we'll give up the iterative
> refinement. That is, we want the smallest threshold such that
>
> (not (= -8.699366 (- -8.699366 threshold)))
>
> is true.
>
> Advice and clarification would be most welcome.
```

Well, I do not know anything about this stuff, but that has never stopped me before:

```
(loop with trials = 200
      repeat trials
      for base = (random 10.0)
      when (= base (- base -2.1597653E-7))
      count 1 into eqs
      finally (print (/ eqs trials)))
```

...returns under ACL6.2/Win32 values between .5+ to .7+

I am thinking about the butterfly effect story. I think it depends on what internal value you get from -8.699366, and how -8.699366 is to being an integral multiple of the epsilon.

Re: Floating point arithmetic (epsilon)

comp.lang.lisp: Re: Floating point arithmetic (epsilon)

I dare say (* 1.5 epsilon) or twice epsilon would always work.

How's that for guesswork?

kenny

>

--

<http://tilton-technology.com>

Why Lisp? <http://alu.cliki.net/RtL%20Highlight%20Film>

Your Project Here! <http://alu.cliki.net/Industry%20Application>