Re: Finding the inverf

Source: http://sci.tech−archive.net/Archive/sci.math/2007−12/msg02425.html

- *From*: Ray Koopman <koopman@xxxxxx>
- *Date*: Tue, 11 Dec 2007 17:19:46 −0800 (PST)

On Dec 8, 5:50 pm, David W. Cantrell <DWCantr...@xxxxxxxxxxx> wrote:

> Ray Koopman <koop...@xxxxxx> wrote:
>
>> On Dec 2, 9:43 pm, morg...@xxxxxxxxx wrote:
>>
>>> Could someone tell me how to calculate inverf (x) if I am
>>> given
>>> the value of erf (x).
>>
>> Here's a relatively simple approximation for inverf[x]:
>>
>> With[{t = −2*Log[1−|x|]}, Sign[x]*Sqrt[(t −
>> Log[1 + t + (.01167845*t+.1066561)*t^2/
>> ((.02118035*t+.3710243)*t+1)])/2]]
>>
>> The maximum absolute error is about 3.6*10^−5
>> for all x in (−1,1) in 64−bit IEEE format.
>
> Thanks for that approximation. I have some comments and what some might
> consider to be an improvement.
>
> First, I believe your statement about the error to be correct. (N.B. I've
> forgotten much about floating−point arithmetic, so anything I say related
> to that here should be "taken with a grain of salt".) The reason that your
> statement is correct is that numbers only very slightly less than 1 are not
> representable in 64−bit IEEE format.

Actually, I wrote too quickly and claimed a bit too much, literally.
As I mentioned in my reply to Axel, the approximation was originally
developed to invert the standard normal cdf. The coefficients were
derived to minimize the maximum absolute error for max[p,q] < 1 in
machine numbers. Since the largest number < 1 that can be represented
exactly is 1 − 2^−53, and erf = p−q, the stated error bound of 3.6*
10^−5 holds only for |erf| <= 1 − 2^−52; at |erf| = 1 − 2^−53 the
absolute error is 3.7*10^−5.

> But forgetting floating−point and just
> considering the interval (−1, 1) of real numbers, the maximum error of your
> approximation is about 1.2*10^−4, occurring when |x| is about 1 − 10^−79 .

One of the nice features of this family of approximations (which really should be thought of as inverting erfc rather than erf −− just change the start to inverfc[y] = With[{t = −2*Log@Min[y,2−y]}, Sign[1−y]* ...]) is that the error naturally −> 0 as y −> 0. If we optimize the coefficients for all y in (0,1], then the maximum error rises to only 5.96*10^−5, with extrema at y = {.87, .379, .0160, 9.18*10^−10, 1.027*10^−146}.

> Sometimes we prefer to consider relative, rather than absolute, error.
> That's what I would normally choose to do for erf. Your approximation has
> relative error exceeding 8*10^−4 at x = 0.
>
> With the constraint that relative error = 0 at x = 0 and then
> minimizing |relative error| on the real interval (−1, 1), for an
> approximation of erf(x) in the form
>
> With[{t = −2*Log[1−|x|]},
> Sign[x]*Sqrt[(t − Log[1 + t + (a*t + b)*t^2/((c*t + d)*t+1)])/2]]
>
> we find that b = (4 − pi)/8, implied by the constraint, and
> a = 0.01857224, c = 0.03453839 and d = 0.44739441 .

For the relative error to behave nicely near zero, it is also necessary to change Log[1 + t + ...] to log1p[t + ...], where log1p[x] returns log[1+x] accurately for x near 0.

> This gives |relative error| < 3.7*10^−5 . (If we were to drop the
> constraint, then worst relative error could be reduced a bit further.
> However, if for nothing but aesthetics, I prefer to keep the constraint.)

I guess beauty is in the eye of the beholder. In general, I think the choice of a minimand ought to be based on the intended use of the approximation. In a statistical context where we're inverting the normal cdf, I sometimes worry about the approximate value of z becoming separated from the fact that the value is an approximation, and being used to recreate the original p or q in situations where the relative error in min[p,q] is important. In such cases the weighting should go the other way, with equal−magnitude errors in the approximate z being more serious for large |z| than for small |z|.

> BTW, with my coefficients above, if we were to approximate the inverse
> of the complementary error function, inverfc(x), on (0, 2), my bound
> on |relative error| would remain unchanged, of course. But with your
> original coefficients, if we were to approximate inverfc(x) on (0, 2)
> in 64−bit IEEE format, your error bound would increase (since
> floating−point allows representation of numbers much closer to 0
> than to 1).

Apples vs oranges?

.