Subject: Re: Multiplying very high with very low numbers: erfc * exp
Posted by tho.siegert on Wed, 16 Apr 2014 12:09:10 GMT
View Forum Message <> Reply to Message

On Thursday, April 3, 2014 3:21:31 PM UTC+2, alx wrote:
> On Thursday, April 3, 2014 11:35:10 AM UTC+2, tho.s...@gmail.com wrote:
>
>> Hello,
>
>>
>
>> for my MCMC fitting program, I need to evaluate functions of the form (Gaussian with a one sided exponential tail towards lower x-values):
>
>>
>
>>
>
>>
>
>> f(a,b,c,d) * erfc(g(a,b,c,d)) * exp(h(a,b,c,d)) := X * Y * Z = F
>
>>
>
>>
>
>>
>
>> where f,g and h are certain functions of the parameters a,b,c and d.
>
>>
>
>>
>
>>
>
>> It almost always happens that the numbers of these three factors are like:
>
>>
>
>>
>
>>
>
>> F = X * Y * Z = 1e2 * 1e-999 * 1e1000 = 1e3
>
>>
>

>>
>
>>
>
>> Which is a big problem since 1e-999 is represented as 0 and 1e1000 is represented as infinity, thus the result being 0, infinity or nan, but definetly not 1e3.
>
>>
>
>> As a work-around, I went to log-space such that:
>
>>
>
>>
>
>>
>
>> F = exp(ln(F)) = exp(ln(X * Y * Z)) = exp(ln(X) + ln(Y) + ln(Z)) =
>
>>
>
>> = exp(ln(f(a,b,c,d)) + ln(erfc(g(a,b,c,d))) + ln(exp(h(a,b,c,d)))) :=
>
>>
>
>> := exp(Q + W + E)
>
>>
>
>>
>
>>
>
>> Q and E are no problem to evaluate since f() is just a rational function and ln(exp(h())) is just h().
>
>>
>
>> However, W = ln(erfc(g())) contains the same problem as above. If g() is far negative from 0, erfc(g()) is just 2 (and not e.g. 2 - 1e-99). If g() is far positive from 0, erfc(g()) is just 0, returning W as -Inf (as erfc(g()) should actually be something like 1e-99).
>
>>
>
>>
>
>>
>

>> Now, I looked up several representations of the erfc() function in order to build something like a lnerfc - function. I have chosen the erfcc() function in Numerical recipes, Chapter 6, Special Functions (around page 214) which is also given in Wikipedia at http://en.wikipedia.org/wiki/Error_function#Numerical_approx imation
>
>>
>
>> This approximation has two major advantages:
>
>>
>
>> 1) It is represented as proprotional to an exponential function, for which the ln can easily be calculated.
>
>>
>
>> 2) The fractional error is "everywhere less than 1.2e-7".
>
>>
>
>>
>
>>
>
>> Including all these work-arounds, F = X * Y * Z can be calculated to a good enough precision (for me).
>
>>
>
>>
>
>>
>
>> However (again), as you might already think of, it takes a while to calculate F. In a MCMC run, this function has to be evaluated over and over again. If there is more than one such a function present in my data (say N), I need to fit, i.e. evaluate something like:
>
>>
>
>>
>
>>
>
>> sum(F_i, i=0..N)
>
>>
>
>>

>
>>
>
>> over and over again (typically N = 20..30).
>
>>
>
>>
>
>>
>
>> To put it in a nutshell:
>
>>
>
>> I am looking for a speed-up to calculate W = ln(erfc(g(a,b,c,d))).
>
>>
>
>> I know that I can calculate the erfc - function by:
>
>>
>
>> erfc(x) = 1 - sgn(x) * igamma(0.5,x^2)
>
>>
>
>> where igamma is the incomplete gamma-function.
>
>>
>
>> Unfortunately, there is no LNIGAMMA - function in IDL, as for the complete gamma-function (LNGAMMA). As this does not necessarily have to work good then because of the "1 - ".
>
>>
>
>>
>
>>
>
>> I hope you understand the problem and are not overwhelmed by this wall of text.
>
>>
>
>> I appreciate any suggestions.
>
>>
>

>>
>
>>
>
>> Cheers,
>
>>
>
>> Thomas
>
>
>
> I am afraid that IDL will not be able to help you without some reformulation of your problem.
>
> In order to avoid underflow and overflow when computing each of your Y and Z functions, you have to find a derived or approximated expression for their product, which indeed is finite and of order about 10.
>
> You might for instance consider Rational Chebyshev approximations of X*Y, which are often used for computing the "erfcx" function (i.e. exp(x^2)*erfc(x)), whose shape is not far from the one you are dealing with.
>
> Hoping this can help you.
>
> alx.

On Thursday, April 3, 2014 3:21:31 PM UTC+2, alx wrote:
> On Thursday, April 3, 2014 11:35:10 AM UTC+2, tho.s...@gmail.com wrote:
>
>> Hello,
>
>>
>
>> for my MCMC fitting program, I need to evaluate functions of the form (Gaussian with a one sided exponential tail towards lower x-values):
>
>>
>
>>
>
>>
>
>> f(a,b,c,d) * erfc(g(a,b,c,d)) * exp(h(a,b,c,d)) := X * Y * Z = F
>
>>
>

>>
>
>>
>
>> where f,g and h are certain functions of the parameters a,b,c and d.
>
>>
>
>>
>
>>
>
>> It almost always happens that the numbers of these three factors are like:
>
>>
>
>>
>
>>
>
>> F = X * Y * Z = 1e2 * 1e-999 * 1e1000 = 1e3
>
>>
>
>>
>
>>
>
>> Which is a big problem since 1e-999 is represented as 0 and 1e1000 is represented as infinity, thus the result being 0, infinity or nan, but definetly not 1e3.
>
>>
>
>> As a work-around, I went to log-space such that:
>
>>
>
>>
>
>>
>
>> F = exp(ln(F)) = exp(ln(X * Y * Z)) = exp(ln(X) + ln(Y) + ln(Z)) =
>
>>
>
>> = exp(ln(f(a,b,c,d)) + ln(erfc(g(a,b,c,d))) + ln(exp(h(a,b,c,d)))) :=
>
>>

>
>> := exp(Q + W + E)
>
>>
>
>>
>
>>
>
>> Q and E are no problem to evaluate since f() is just a rational function and ln(exp(h())) is just h().
>
>>
>
>> However, W = ln(erfc(g())) contains the same problem as above. If g() is far negative from 0, erfc(g()) is just 2 (and not e.g. 2 - 1e-99). If g() is far positive from 0, erfc(g()) is just 0, returning W as -Inf (as erfc(g()) should actually be something like 1e-99).
>
>>
>
>>
>
>>
>
>> Now, I looked up several representations of the erfc() function in order to build something like a lnerfc - function. I have chosen the erfcc() function in Numerical recipes, Chapter 6, Special Functions (around page 214) which is also given in Wikipedia at http://en.wikipedia.org/wiki/Error_function#Numerical_approx imation
>
>>
>
>> This approximation has two major advantages:
>
>>
>
>> 1) It is represented as proprotional to an exponential function, for which the ln can easily be calculated.
>
>>
>
>> 2) The fractional error is "everywhere less than 1.2e-7".
>
>>
>
>>
>
>>
>

>> Including all these work-arounds, F = X * Y * Z can be calculated to a good enough precision (for me).
>
>>
>
>>
>
>>
>
>> However (again), as you might already think of, it takes a while to calculate F. In a MCMC run, this function has to be evaluated over and over again. If there is more than one such a function present in my data (say N), I need to fit, i.e. evaluate something like:
>
>>
>
>>
>
>>
>
>> sum(F_i, i=0..N)
>
>>
>
>>
>
>>
>
>> over and over again (typically N = 20..30).
>
>>
>
>>
>
>>
>
>> To put it in a nutshell:
>
>>
>
>> I am looking for a speed-up to calculate W = ln(erfc(g(a,b,c,d))).
>
>>
>
>> I know that I can calculate the erfc - function by:
>
>>
>
>> erfc(x) = 1 - sgn(x) * igamma(0.5,x^2)

>
>>
>
>> where igamma is the incomplete gamma-function.
>
>>
>
>> Unfortunately, there is no LNIGAMMA - function in IDL, as for the complete gamma-function (LNGAMMA). As this does not necessarily have to work good then because of the "1 - ".
>
>>
>
>>
>
>>
>
>> I hope you understand the problem and are not overwhelmed by this wall of text.
>
>>
>
>> I appreciate any suggestions.
>
>>
>
>>
>
>>
>
>> Cheers,
>
>>
>
>> Thomas
>
>
>
> I am afraid that IDL will not be able to help you without some reformulation of your problem.
>
> In order to avoid underflow and overflow when computing each of your Y and Z functions, you have to find a derived or approximated expression for their product, which indeed is finite and of order about 10.
>
> You might for instance consider Rational Chebyshev approximations of X*Y, which are often used for computing the "erfcx" function (i.e. exp(x^2)*erfc(x)), whose shape is not far from the one you are dealing with.
>
> Hoping this can help you.
>

> alx.

Okay, since I already actually had a lnerfc function, but was too silly to make it work properly, i post my solution:

```
function lnerfc, x, y

  a = [-1.26551223d, 1.00002368d, 0.37409196d, 0.09678418d, -0.18628806d, 0.27886807d, -1.13520398d, 1.48851587d, -0.82215223d, 0.17087277d]

  t = 1d / (1d + 0.5d * abs(x))

  tau = t * exp( -x*x + (a[0] + t * (a[1] + t * (a[2] + t * (a[3] + t * (a[4] + t * (a[5] + t * (a[6] + t * (a[7] + t * (a[8] + t * a[9])))))))))))

  y = alog(t) + ( -x*x + (a[0] + t * (a[1] + t * (a[2] + t * (a[3] + t * (a[4] + t * (a[5] + t * (a[6] + t * (a[7] + t * (a[8] + t * a[9])))))))))))

  lt0 = where(x lt 0d,/null)

  y[lt0] = y[lt0] + alog(2d / tau - 1d)

  return, y

end
```

It is again taken from Numerical recipes, Chapter 6.2, Special Functions, just translated to logarithm space. This is indeed based on Chebyshev fitting. Thanks alx!

Regards,
Thomas