It has shown that without intervention RcppNumerical does not handle integration over infinite ranges.

On the question was asked how to numerically integrate a function over a infinite range in Rcpp, e.g. by using RcppNumerical. As an example, the integral

∫∞−∞dxexp(−(x−μ)42)∫−∞∞dxexp⁡(−(x−μ)42)

was given. Using RcppNumerical is straight forward. One defines a class that extends Numer::Func for the function and an interface function that calls Numer::integrate on it:

*// [[Rcpp::depends(RcppEigen)]]*

*// [[Rcpp::depends(RcppNumerical)]]*

**#include <RcppNumerical.h>**

**class** **exp4**: **public** Numer::Func {

**private**:

**double** mean;

**public**:

exp4(**double** mean\_) : mean(mean\_) {}

**double** **operator**()(**const** **double**& x) **const** {

**return** exp(-pow(x-mean, 4) / 2);

}

};

*// [[Rcpp::export]]*

Rcpp::NumericVector **integrate\_exp4**(**const** **double** &mean, **const** **double** &lower, **const** **double** &upper) {

exp4 **function**(mean);

**double** err\_est;

**int** err\_code;

**const** **double** result = Numer::integrate(function, lower, upper, err\_est, err\_code);

**return** Rcpp::NumericVector::create(Rcpp::Named("result") = result,

Rcpp::Named("error") = err\_est);

}

This works fine for finite ranges:

integrate\_exp4(4, 0, 4)

**#***# result error*

**#***# 1.077900e+00 9.252237e-08*

However, it produces NA for infinite ones:

integrate\_exp4(4, -Inf, Inf)

**#***# result error*

**#***# NaN NaN*

This is disappointing, since base R’s integrate() handles this without problems:

exp4 <- **function**(x, mean) exp(-(x - mean)^4 / 2)

integrate(exp4, 0, 4, mean = 4)

**## 1.0779 with absolute error < 1.3e-07**

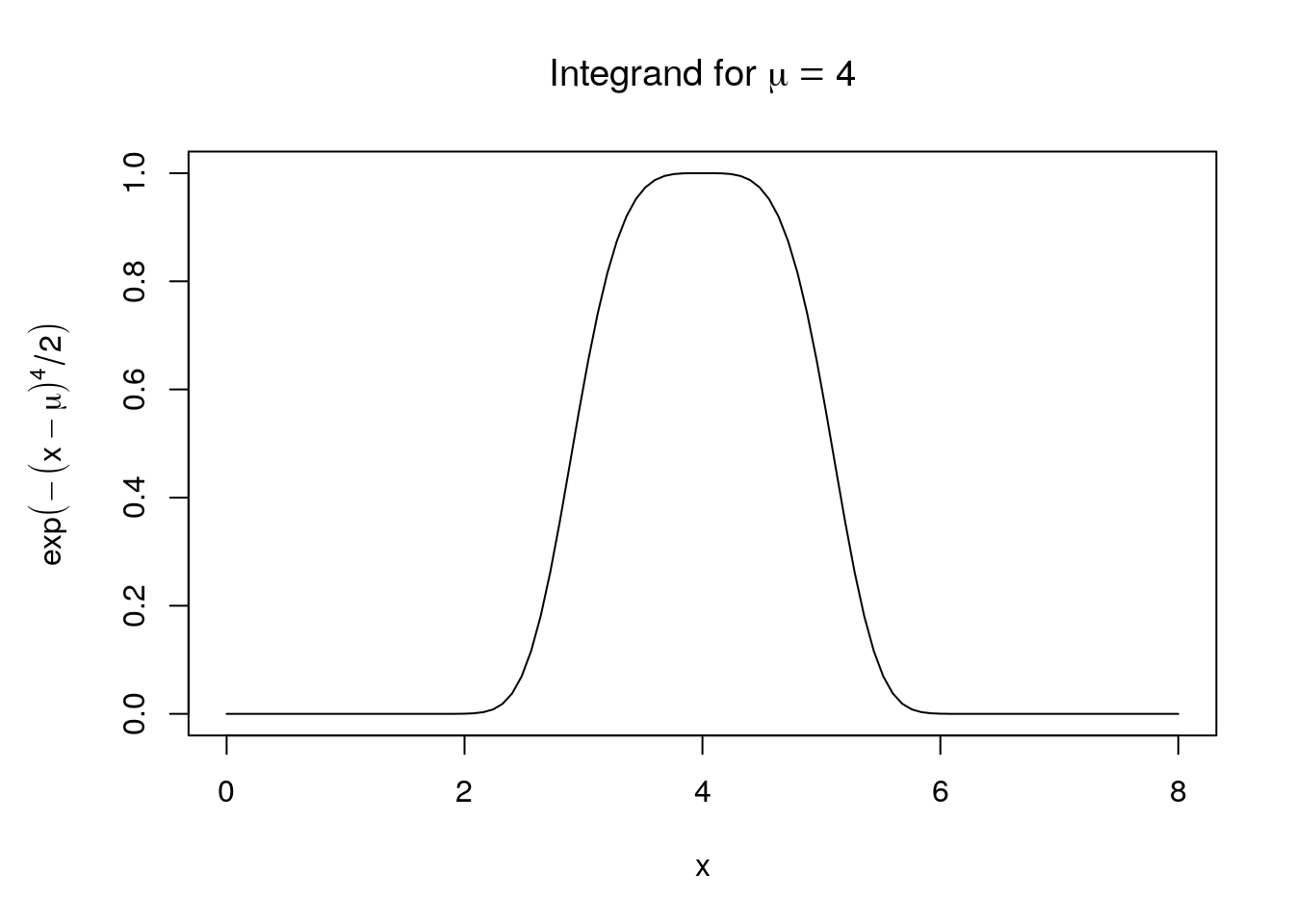
integrate(exp4, -Inf, Inf, mean = 4)

**## 2.155801 with absolute error < 7.9e-06**

In this particular case the problem can be easily solved in two different ways. First, the integral can be expressed in terms of the Gamma function:

∫∞−∞dxexp(−(x−μ)42)=2−34Γ(14)≈2.155801∫−∞∞dxexp⁡(−(x−μ)42)=2−34Γ(14)≈2.155801

Second, the integrand is almost zero almost everywhere:



It is therefore sufficient to integrate over a small region around mean to get a reasonable approximation for the integral over the infinite range:

integrate\_exp4(4, 1, 7)

**#***# result error*

**#***# 2.155801e+00 9.926448e-13*

However, the trick to approximate the integral over an infinite range with an integral over a (possibly large) finite range does not work for functions that approach zero more slowly. The help page for integrate() has a nice example for this effect:

*## a slowly-convergent integral*

integrand <- **function**(x) {1/((x+1)\*sqrt(x))}

integrate(integrand, lower = 0, upper = Inf)

**## 3.141593 with absolute error < 2.7e-05**

*## don't do this if you really want the integral from 0 to Inf*

integrate(integrand, lower = 0, upper = 10)

**## 2.529038 with absolute error < 3e-04**

integrate(integrand, lower = 0, upper = 100000)

**## 3.135268 with absolute error < 4.2e-07**

integrate(integrand, lower = 0, upper = 1000000, stop.on.error = FALSE)

**#***# failed with message 'the integral is probably divergent'*

How does integrate() handle the infinite range and can we replicate this in Rcpp? The help page states:

If one or both limits are infinite, the infinite range is mapped onto a finite interval.

This is in fact done by a different function from R’s C-API: Rdqagi() instead of Rdqags(). In principle one could call Rdqagi() via Rcpp, but this is not straightforward. Fortunately, there are at least two other solutions.

The GNU Scientific Library provides a function to integrate over the infinte interval (−∞,∞)(−∞,∞), which can be used via the RcppGSL package:

*// [[Rcpp::depends(RcppGSL)]]*

**#include <RcppGSL.h>**

**#include <gsl/gsl\_integration.h>**

**double** **exp4** (**double** x, **void** \* params) {

**double** mean = \*(**double** \*) params;

**return** exp(-pow(x-mean, 4) / 2);

}

*// [[Rcpp::export]]*

Rcpp::NumericVector **normalize\_exp4\_gsl**(**double** &mean) {

gsl\_integration\_workspace \*w = gsl\_integration\_workspace\_alloc (1000);

**double** result, error;

gsl\_function F;

F.function = &exp4;

F.params = &mean;

gsl\_integration\_qagi(&F, 0, 1e-7, 1000, w, &result, &error);

gsl\_integration\_workspace\_free (w);

**return** Rcpp::NumericVector::create(Rcpp::Named("result") = result,

Rcpp::Named("error") = error);

}

normalize\_exp4\_gsl(4)

**#***# result error*

**#***# 2.155801e+00 3.718126e-08*

Alternatively, one can apply the transformation used by GSL (and probably R) also in conjunction with RcppNumerical. To do so, one has to substitute x=(1−t)/tx=(1−t)/t resulting in

∫∞−∞dxf(x)=∫10dtf((1−t)/t)+f(−(1−t)/t)t2∫−∞∞dxf(x)=∫01dtf((1−t)/t)+f(−(1−t)/t)t2

Now one could write the code for the transformed function directly, but it is of course nicer to have a general solution, i.e. use a class template that can transform *any function* in the desired fashion

*// [[Rcpp::depends(RcppEigen)]]*

*// [[Rcpp::depends(RcppNumerical)]]*

**#include <RcppNumerical.h>**

**class** **exp4**: **public** Numer::Func {

**private**:

**double** mean;

**public**:

exp4(**double** mean\_) : mean(mean\_) {}

**double** **operator**()(**const** **double**& x) **const** {

**return** exp(-pow(x-mean, 4) / 2);

}

};

*// [[Rcpp::plugins(cpp11)]]*

**template**<**class** **T**> **class** **trans\_func**: **public** T {

**public**:

**using** T::T;

**double** **operator**()(**const** **double**& t) **const** {

**double** x = (1-t)/t;

**return** (T::**operator**()(x) + T::**operator**()(-x))/pow(t, 2);

}

};

*// [[Rcpp::export]]*

Rcpp::NumericVector **normalize\_exp4**(**const** **double** &mean) {

trans\_func<exp4> f(mean);

**double** err\_est;

**int** err\_code;

**const** **double** result = Numer::integrate(f, 0, 1, err\_est, err\_code);

**return** Rcpp::NumericVector::create(Rcpp::Named("result") = result,

Rcpp::Named("error") = err\_est);

}

normalize\_exp4(4)

**#***# result error*

**#***# 2.155801e+00 1.439771e-06*

Note that the exp4 class is identical to the one from the initial example. This means one can use the same class to calculate integrals over a finite range and after transformation over an infinite range.

In this post I want to generalize the method to integrals where only one of the limits is infinite. In addition, I want to make it more user friendly, since the below code is written something like

// [[Rcpp::depends(RcppEigen)]]

// [[Rcpp::depends(RcppNumerical)]]

#include

namespace rstub {

// [...]

}

class exp4: public Numer::Func {

private:

double mean;

public:

exp4(double mean\_) : mean(mean\_) {}

double operator()(const double& x) const {

return exp(-pow(x - mean, 4) / 2);

}

};

// [[Rcpp::export]]

Rcpp::NumericVector integrate\_exp4(const double &mean, double lower, double upper) {

exp4 function(mean);

double err\_est;

int err\_code;

double result = rstub::integrate(function, lower, upper, err\_est, err\_code);

return Rcpp::NumericVector::create(Rcpp::Named("result") = result,

Rcpp::Named("error") = err\_est);

}

and have it correctly handle different input:

rbind(

integrate\_exp4(4, 0, 4),

integrate\_exp4(4, -Inf, Inf),

integrate\_exp4(4, 3, Inf),

integrate\_exp4(4, -Inf, 3)

)

## result error

## [1,] 1.0779003 9.252237e-08

## [2,] 2.1558005 1.439771e-06

## [3,] 1.9903282 4.250105e-11

## [4,] 0.1654723 6.251315e-14

The only differences in the above code to the sample code is the usage or rstub::integrate instead of Numer:integrate and the as yet unspecified rstub namespace. What is needed in that namespace? First, we will need a template class that does the necessary variable substitutions. In the case where both limits are infinite, we use as before \(x = (1-t)/t\) resulting in

\[  
\int\_{-\infty}^{\infty} \mathrm{d}x f(x) = \int\_0^1 \mathrm{d}t \frac{f((1-t)/t) + f(-(1-t)/t)}{t^2}  
\]

If only one of the limits is infinite, we use the substitutions \(x = a + (1-t)/t\) and \(x = b – (1-t)/t\) resulting in

\[  
\int\_{a}^{\infty} \mathrm{d}x f(x) = \int\_0^1 \mathrm{d}t \frac{f(a+(1-t)/t)}{t^2}  
\]

and

\[  
\int\_{-\infty}^{b} \mathrm{d}x f(x) = \int\_0^1 \mathrm{d}t \frac{f(b-(1-t)/t)}{t^2}  
\]

For the C++ template class aggregation is used instead of inheritance, allowing to easily specify the limits:

template

class transform\_infinite: public Numer::Func {

private:

T func;

double lower;

double upper;

public:

transform\_infinite(T \_func, double \_lower, double \_upper) :

func(\_func), lower(\_lower), upper(\_upper) {}

double operator() (const double& t) const {

double x = (1 - t) / t;

bool upper\_finite = (upper < std::numeric\_limits::infinity());

bool lower\_finite = (lower > -std::numeric\_limits::infinity());

if (upper\_finite && lower\_finite) {

Rcpp::stop("At least on limit must be infinite.");

} else if (lower\_finite) {

return func(lower + x) / pow(t, 2);

} else if (upper\_finite) {

return func(upper - x) / pow(t, 2);

} else {

return (func(x) + func(-x)) / pow(t, 2);

}

}

};

Finally we need a wrapper function for Numer::integrate which checks if both limits are finite or not:

using Numer::Integrator;

template

double integrate(const T& f, double lower, double upper,

double& err\_est, int& err\_code,

const int subdiv = 100, const double& eps\_abs = 1e-8, const double& eps\_rel = 1e-6,

const Integrator::QuadratureRule rule = Integrator::GaussKronrod41) {

if (upper == lower) {

err\_est = 0.0;

err\_code = 0;

return 0.0;

}

if (std::abs(upper) < std::numeric\_limits::infinity() &&

std::abs(lower) < std::numeric\_limits::infinity()) {

return Numer::integrate(f, lower, upper, err\_est, err\_code, subdiv, eps\_abs, eps\_rel, rule);

} else {

double sign = 1.0;

if (upper < lower) {

std::swap(upper, lower);

sign = -1.0;

}

transform\_infinite g(f, lower, upper);

return sign \* Numer::integrate(g, 0.0, 1.0, err\_est, err\_code, subdiv, eps\_abs, eps\_rel, rule);

}

}

If both limits are finite, Numer::integrate is used directly. Otherwise the function is transformed and Numer::integrate is used with adjusted range. In addition, it is first checked that the upper limit is actually larger than the lower limit. If this is not the case, one of the properties of integration is used to swap the limits and change the sign:

\[  
\int\_{a}^{b} \mathrm{d}x f(x) = -\int\_{b}^{a} \mathrm{d}x f(x)  
\]

Thereby we get the correct result even when the limits have been exchanged:

rbind(

integrate\_exp4(4, 3, Inf),

integrate\_exp4(4, Inf, 3)

)

## result error

## [1,] 1.990328 4.250105e-11

## [2,] -1.990328 4.250105e-11

In the end we needed on template class and one template function, which could be put into a separate header file, to generalize Numer::integrate for integration over an infinite interval.