fourierin: An R package to compute Fourier integrals

by Guillermo Basulto-Elias, Alicia Carriquiry, Kris De Brabanter and Daniel J. Nordman

Abstract We present the R package **fourierin** (Basulto-Elias, 2017) for evaluating functions defined as Fourier-type integrals over a collection of argument values. The integrals are finitely supported with integrands involving continuous functions of one or two variables. As an important application, such Fourier integrals arise in so-called "inversion formulas", where one seeks to evaluate a probability density at a series of points from a given characteristic function (or vice versa) through Fourier transforms. This paper intends to fill a gap in current R software, where tools for repeated evaluation of functions as Fourier integrals are not directly available. We implement two approaches for such computations with numerical integration. In particular, if the argument collection for evaluation corresponds to a regular grid, then an algorithm from Inverarity (2002) may be employed based on a fast Fourier transform, which creates significant improvements in the speed over a second approach to numerical Fourier integration (where the latter also applies to cases where the points for evaluation are not on a grid). We illustrate the package with the computation of probability densities and characteristic functions through Fourier integrals/transforms, for both univariate and bivariate examples.

Introduction

Continuous Fourier transforms commonly appear in several subject areas, such as physics and statistics. In probability theory, for example, continuous Fourier transforms are related to the characteristic function of a distribution and play an important role in evaluating probability densities from characteristic functions (and vice versa) through inversion formulas (cf. Athreya and Lahiri (2006)). Similar Fourier-type integrations are also commonly required in statistical methods for density estimation, such as kernel deconvolution (cf. Meister (2009)).

At issue, the Fourier integrals of interest often cannot be solved in elementary terms and typically require numerical approximations. As a compounding issue, the oscillating nature of the integrands involved can cause numerical integration recipes to fail without careful consideration. However, Bailey and Swarztrauber (1994) present a mid-point integration rule in terms of appropriate discrete Fourier transforms, which can be efficiently computed using the Fast Fourier Transform (FFT). Inverarity (2002) extended this characterization to the multivariate integral case. These works consequently offer targeted approaches for numerically approximating types of Fourier integrals of interest (e.g., in the context of characteristic or density functions).

Because R is one of the most popular programming languages among statisticians, it seems worthwhile to have general tools available for computing such Fourier integrals in this software platform. However, we have not found any R package that specifically performs this type of integral in general, though this integration does intrinsically occur in some statistical procedures. See **?** for an application in kernel deconvolution where univariate Fourier integrals are required. Furthermore, beyond the integral form, the capacity to handle repeated calls for such integrals is another important consideration. This need arises when computing a function, that is itself defined by a Fourier integral, over a series of points. Note that this exact aspect occurs when determining a density function from characteristic function (or vice versa), so that the ability to efficiently compute Fourier integrals over a collection of arguments is crucial.

The intent of the package **fourierin** explained here is to help in computing such Fourier-type integrals within R. The main function of the package serves to calculate Fourier integrals over a range of potential arguments for evaluation and is also easily adaptable to several definitions of the continuous Fourier transform and its inverse (cf. Inverarity (2002)). (That is, the definition of a continuous Fourier transform may change slightly from one context to another, often up to normalizing constants, so that it becomes reasonable to provide a function that can accommodate any given definition through scaling adjustments.) If the points for evaluating Fourier integrals are specified on regular grid, then the package allows use of the FFT for particularly fast numerical integration. However, the package also allow the user to evaluate such integrals at arbitrary collections of points that need not constitute a regular grid (though, in this case, the FFT cannot be used and computations naturally become slower). The latter can be handy in some situations; for example, evaluations at zero can provide moments of a random variable when computing derivatives of a characteristic function from the probability density. The heavy computations in **fourierin** are performed in C++ via the **RcppArmadillo** package (cf. Eddelbuettel and Sanderson (2014)).

The rest of the paper has four sections. We first describe the Fourier integral for evaluation and its numerical approximation in "Fourier Integrals and Fast Fourier Transform (FFT)." We then illustrate how package **fourierin** may be used in univariate and bivariate cases of Fourier integration. In "Speed Comparison," we demonstrate the two approaches (FFT-based or not) for computing Fourier integrals, both in one and two dimensions and at several grid sizes. We provide evidence that substantial time savings occur when using the FFT-based method for evaluation at points on a grid. Finally, in "Summary," we present conclusions and upcoming extensions to the R package.

Fourier integrals and fast Fourier transform

For $\boldsymbol{w} = (w_1, \ldots, w_n), \boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$, define the vector dot product $\langle \boldsymbol{w}, \boldsymbol{t} \rangle = w_1 t_1 + \cdots + w_n t_n$ and recall the complex exponential function $\exp\{\imath x\} = \cos(x) + \imath \sin(x)$, $x \in \mathbb{R}$, where $\imath = \sqrt{-1}$.

This package aims to compute Fourier integrals at several points simultaneously, which namely involves computation of the integral

$$\left[\frac{|s|}{(2\pi)^{1-r}}\right]^{n/2} \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(\boldsymbol{t}) \exp\{\imath s \langle \boldsymbol{w}, \boldsymbol{t} \rangle\} \mathrm{d}\boldsymbol{t}, \tag{1}$$

at more than one potential argument $\boldsymbol{w} \in \mathbb{R}^n$, where f is a generic continuous n-variate function that takes real or complex values, for $n \in \{1, 2\}$, and the above limits of integration are defined by real values $a_j < b_j$ for $j = 1, \ldots, n$. Note that s and r in (1) denote real-valued constants, which are separately included to permit some flexibility in the definition of continuous Fourier transforms to be used (e.g., $s = 1$, $r = 1$). Hence, (1) represents a function of $\boldsymbol{w} \in \mathbb{R}^n$, defined by a Fourier integral, where the intention is to evaluate (1) over a discrete collection of $\boldsymbol{w}$-values, often defined by a grid in $\mathbb{R}^n$. For example, if $[c_1, d_1) \times \cdots \times [c_n, d_n) \subset \mathbb{R}^n$ denotes a rectangular region specified by some real constants $c_j < d_j$, $j = 1, \ldots, n$, one may consider evaluating (1) at points $\boldsymbol{w}$ lying on a regular grid of size $m_1 \times m_2 \times \cdots \times m_n$ within $[c_1, d_1) \times \cdots \times [c_n, d_n)$, say, at points $\boldsymbol{w}^{(j_1,\ldots,j_n)} = (w_{j_1}, \ldots, w_{j_n})$ for $w_{j_k} = c_k + j_k(d_k - c_k)/m_k$ with $j_k \in \{0, 1, \ldots, m_k - 1\}$, $k = 1, \ldots, n$ (where m_k denotes the number of grid points in each coordinate dimension). Argument points on a grid are especially effective for fast approximations of integrals (as in 1), as we discuss in the following.

At given argument $\boldsymbol{w} \in \mathbb{R}^n$, we numerically approximate the integral (1) with a discrete sum using the mid-point rule, whereby the approximation of the j-th slice of the multiple integral involves l_j partitioning rectangles (or equi-spaced subintervals) for $j = 1, \ldots, n$ and $n \in \{1, 2\}$; that is, for $l_1, \ldots, l_n$ representing a selection of the numbers of approximating nodes to be used in the coordinates of integration (i.e., a resolution size), the integral (1) is approximated as

$$\left(\prod_{j=1}^{n} \frac{b_j - a_j}{l_j}\right) \cdot \left[\frac{|s|}{(2\pi)^{1-r}}\right]^{n/2} \sum_{i_1=0}^{l_1-1} \sum_{i_2=0}^{l_2-1} \cdots \sum_{i_n=0}^{l_n-1} f(\boldsymbol{t}^{(i_1,\ldots,i_n)}) \exp\{\imath s \langle \boldsymbol{w}, \boldsymbol{t}^{(i_1,\ldots,i_n)} \rangle\}, \tag{2}$$

with nodes $\boldsymbol{t}^{(i_1,\ldots,i_n)} = (t_{i_1}, \ldots, t_{i_n})$ defined by coordinate midpoints $t_{i_k} = a_k + (2i_k + 1)/2 \cdot (b_k - a_k)/l_k$ for $i_k \in \{0, 1, \ldots, l_k - 1\}$ and $k = 1, \ldots, n$. Note that a large grid size $l_1 \times \cdots \times l_n$ results in higher resolution for the integral approximation (2), but at a cost of increased computational effort. On the other hand, observe that when a regular grid is used, the upper evaluation limits, $d_1, \ldots, d_n$ are not included in such grid, however, the higher the resolution, the closer we get to these bounds.

To reiterate, the goal is then to evaluate the Fourier integral (1) over some set of argument points $\boldsymbol{w} \in \mathbb{R}^n$ by employing the midpoint approximation (2), where the latter involves a $l_1 \times \cdots \times l_n$ resolution grid (of midpoint nodes) for $n \in \{1, 2\}$. It turns out that when the argument points $\boldsymbol{w}$ fall on a $m_1 \times \cdots \times m_n$-sized regular grid and this grid size matches the size of the approximating node grid from (2), namely $l_j = m_j$ for each dimension $j = 1, \ldots, n$, then the sum (2) may be written in terms of certain discrete Fourier transforms and inverses of discrete Fourier transforms that can be conveniently computed with a FFT operation. Details of this derivation can be found in Inverarity (2002). It is well known that using FFT greatly reduces the computational complexity of discrete Fourier transforms from $O(m^2)$ to $O(m \log m)$ in the univariate case, where m is the resolution or grid size. The complexity of computing the multivariate discrete Fourier transform of an n-dimensional array using the FFT is $O(M \log M)$, where $M = m_1 \cdots m_n$ and m_j is the grid/resolution size in the j-th coordinate direction, $j = 1, \ldots, n$.

The R package **fourierin** can take advantage of such FFT representations for the fast computation of Fourier integrals evaluated on a regular grid. The package can also be used to evaluate Fourier integrals at arbitrary discrete sets of points. The latter becomes important when one wishes to evaluate the a continuous Fourier transform at only a few specific points (that may not necessarily constitute a regular grid). We later compare evaluation time of Fourier integrals on a regular grid, both using the

FFT and without using it in **fourierin**.

Examples

In this section we present examples to illustrate use of the **fourierin** package. We begin with a univariate example which considers how to compute continuous Fourier transforms evaluated on a regular grid (therefore using the FFT operation) as well as how the computations proceed at three specified points not on a regular grid (where the FFT is not be used). The second example considers a two dimensional, or bivariate, case of Fourier integration.

The code that follows shows how the package can be used in univariate cases. The example we consider is to recover a χ^2 density f with five degrees of freedom from its characteristic function ϕ, where the underlying functions are given by

$$f(x) = \frac{1}{2^{5/2}\Gamma\left(\frac{5}{2}\right)} x^{\frac{5}{2}-1} e^{-\frac{x}{2}} \qquad \text{and} \qquad \phi(t) = (1-2\imath t)^{-5/2}, \tag{3}$$

for all $x > 0$ and $t \in \mathbb{R}$. We also show how to use the package on non-regular grids. Specifically, we generate sample of three points from a χ^2 distribution with five degrees of freedom and evaluate the density in Formula 3 at these three points where the density has been computed using the Fourier inversion formula approximated at four different resolutions. Results are presented in Table 1.

For illustration, the limits of integration are set from -10 to 10 and we compare several resolutions (64, 256 or 512) or grid node sizes for numerically performing integration (cf. (2)), recalling that the higher the resolution, the better the integral approximation. To evaluate the integrals at argument points on a regular grid, we choose $[-3, 20]$ as an interval for specifying a collection of equi-spaced points, where the number of such points equals the resolution specified (as needed when using FFT).

```
## ---------------------------------------------------------------------
## Univariate example
## ---------------------------------------------------------------------

## Load packages
library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)

## Set functions
df <- 5
cf <- function(t) (1 - 2i*t)^(-df/2)
dens <- function(x) dchisq(x, df)


## Set resolutions
resolutions <- 2^(6:8)

## Compute integral given the resoltion
recover_f <- function(resol){
    ## Get grid and density values
    out <- fourierin(f = cf, lower_int = -10, upper_int = 10,
                     lower_eval = -3, upper_eval = 20,
                     const_adj = -1, freq_adj = -1,
                     resolution = resol)
    ## Return in dataframe format
    out %>%
        as_data_frame() %>%
        transmute(
            x = w,
            values = Re(values),
            resolution = resol)
}

## Density approximations
vals <- map_df(resolutions, recover_f)
```

```
## True values
true <- data_frame(x = seq(min(vals$x), max(vals$x), length = 150),
                   values = dens(x))

univ_plot <-
    vals %>%
    mutate(resolution = as.character(resolution),
           resolution = gsub("64", "064", resolution)) %>%
    ggplot(aes(x, values)) +
    geom_line(aes(color = resolution)) +
    geom_line(data = true, aes(color = "true values"))

univ_plot

## Evaluate in a nonregular grid
set.seed(666)
new_grid <- rchisq(n = 3, df = df)
resolutions <- 2^(6:9)

fourierin(f = cf, lower_int = -10, upper_int = 10,
          eval_grid = new_grid,
              const_adj = -1, freq_adj = -1,
          resolution = 128) %>%
    c () %>% Re() %>%
    data_frame(x = new_grid, fx = .)


## Function that evaluates the log-density on new_grid at different
## resolutions (i.e., number of points to approximate the integral in
## the Fourier inversion formula).
approximated_fx <- function (resol) {
    fourierin(f = cf, lower_int = -10, upper_int = 12,
              eval_grid = new_grid,
              const_adj = -1, freq_adj = -1,
              resolution = resol) %>%
        c() %>% Re() %>%
        {data_frame(x = new_grid,
                    fx = dens(new_grid),
                    diffs = abs(. - fx),
                    resolution = resol)}
}

## Generate table
tab <-
    map_df(resolutions, approximated_fx) %>%
    arrange(x) %>%
    mutate(diffs = round(diffs, 7))  %>%
    rename('f(x)' = fx,
           'absolute difference' = diffs)

tab
```

Observe that the first call of the `fourierin` function above has the default argument `use_fft = TRUE`. Therefore, this computation uses the the FFT representation described in Inverarity (2002) for regular grids, which is substantially fast (Figure 5, as described later, provides timing comparisons without the FFT for contrast). Also note that, when a regular evaluation grid is used, `fourierin` returns a list with both the Fourier integral values and the evaluation grid. Figure 1 shows the resulting plot generated. A low resolution (64) for numerical integration has been included in order to observe differences between the true density and its recovered version using Fourier integrals.

At the bottom of the code above, we also show how `fourierin()` works when a non-regular "evaluation grid" is provided. Observe that, in this case, one directly specifies separate points for evaluation of the integral in addition to separately specifying a resolution level for integration. This aspect is unlike the evaluation case on a regular grid. Consequently, only the Fourier integral values

x	$f(x)$	absolute difference	resolution
3.0585883	0.1541368	0.0002072	64
		0.0000476	128
		0.0000472	256
		0.0000471	512
6.4144242	0.0874281	0.0000780	64
		0.0000155	128
		0.0000148	256
		0.0000147	512
11.7262677	0.0151776	0.0000097	64
		0.0000025	128
		0.0000022	256
		0.0000022	512

Table 1: Absolute differences of true density values at three random points and density values at these same three points obtained using the Fourier inversion formula approximated at different resolutions.

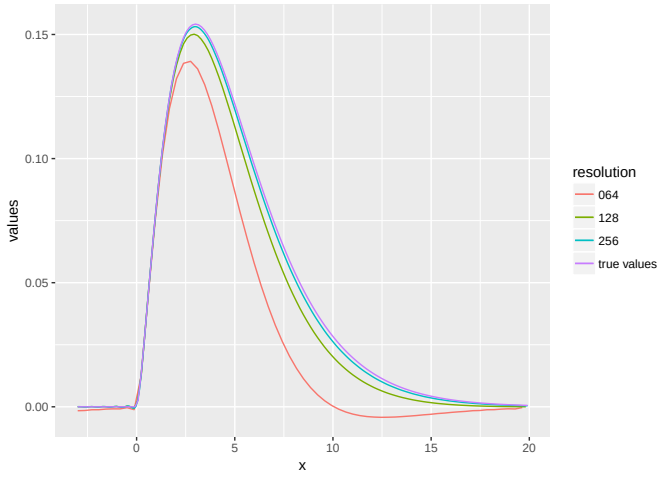

Figure 1: Example of `fourierin()` function for univariate function at resolution 64. Recovering a χ^2 density from its characteristic function. See Equation 3.

are returned, which is also unlike the regular grid case (where the evaluation grid is returned with corresponding integrals in a list). Note that the function f from (1), when having a real-valued argument, should be able to be evaluated at vectors in R.

In a second example, to illustrate how the `fourierin()` function works for bivariate functions, we use a bivariate normal density f and find its characteristic function ϕ. In particular, we have these underlying functions as

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})'\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \qquad \text{and} \qquad \phi(\boldsymbol{t}) = \exp\left(\imath \boldsymbol{x}'\boldsymbol{\mu} - \frac{1}{2}\boldsymbol{t}'\Sigma\boldsymbol{t}\right), \quad (4)$$

for all $\boldsymbol{t}, \boldsymbol{x} \in \mathbb{R}^2$, with $\boldsymbol{\mu} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$.

Below is the code for this bivariate case using a regular evaluation grid, where the output is a complex matrix whose components are Fourier integrals corresponding to the gridded set of bivariate arguments. As illustration, the limits of integration are set from $(-8, -6)$ to $(6, 8)$ (a square) and we consider a resolution 128, where the range $[-4, 4] \times [-4, 4]$ is also chosen to define a collection of evaluation points on a grid, where the number of such points again equals the resolution specified (i.e., for applying FFT).

```
## ------------------------------------------------------------------
## Bivariate example
## ------------------------------------------------------------------

## Load packages
```

```
library(fourierin)
library(tidyr)
library(dplyr)
library(purrr)
library(lattice)
library(ggplot2)

## Set functions to be tested with their corresponding parameters.
mu <- c(-1, 1)
sig <- matrix(c(3, -1, -1, 2), 2, 2)

## Multivariate normal density, x is n x d
f <- function(x) {
    ## Auxiliar values
    d <- ncol(x)
    z <- sweep(x, 2, mu, "-")
    ## Get numerator and denominator of normal density
    num <- exp(-0.5*rowSums(z * (z %*% solve(sig))))
    denom <- sqrt((2*pi)^d*det(sig))
    return(num/denom)
}

## Characteristic function, s is n x d
phi <- function (s) {
    complex(modulus = exp(-0.5*rowSums(s*(s %*% sig))),
            argument = s %*% mu)
}

## Evaluate characteristic function for a given resolution.
eval <- fourierin(f,
                  lower_int = c(-8, -6), upper_int = c(6, 8),
                  lower_eval = c(-4, -4), upper_eval = c(4, 4),
                  const_adj = 1, freq_adj = 1,
                  resolution = 2*c(64, 64),
                  use_fft = T)

## Evaluate true and approximated values of Fourier integral
dat <- eval %>%
    with(crossing(y = w2, x = w1) %>%
         mutate(approximated = c(values))) %>%
    mutate(true = phi(matrix(c(x, y), ncol = 2)),
           difference = approximated - true) %>%
    gather(value, z, -x, -y) %>%
    mutate(real = Re(z), imaginary = Im(z)) %>%
    select(-z) %>%
    gather(part, z, -x, -y, -value)


## Surface plot
wireframe(z ~ x*y | value*part, data = dat,
          scales =
              list(arrows=FALSE, cex= 0.45,
                   col = "black", font = 3, tck = 1),
          screen = list(z = 90, x = -74),
          colorkey = FALSE,
          shade=TRUE,
          light.source= c(0,10,10),
          shade.colors = function(irr, ref,
                                  height, w = 0.4)
              grey(w*irr + (1 - w)*(1 - (1 - ref)^0.4)),
          aspect = c(1, 0.65))

## Contours of values
biv_example1 <-
```

```
    dat %>%
    filter(value != "difference") %>%
    ggplot(aes(x, y, z = z)) +
    geom_tile(aes(fill = z)) +
    facet_grid(part ~ value) +
    scale_fill_distiller(palette = "Reds")

biv_example1

## Contour of differences
biv_example2 <-
    dat %>%
    filter(value == "difference") %>%
    ggplot(aes(x, y, z = z)) +
    geom_tile(aes(fill = z)) +
    facet_grid(part ~ value) +
    scale_fill_distiller(palette = "Spectral")

biv_example2
```

The result of `fourierin()` was stored above in `eval`, which is a list with three elements: two vectors with the corresponding evaluation grid values in each coordinate direction and a complex matrix containing the Fourier integral values. If we do not wish to evaluate the Fourier integrals on a regular grid and instead wish to evaluate these at, say l bivariate points, then we must pass a $l \times 2$ matrix in the argument `w` and the function will return a vector of size l with the Fourier integrals, rather than a list. In the bivariate situation here, the function f must be able to receive a two-column matrix with m rows, where m is the number of points where the Fourier integral will be evaluated.

Corresponding to this bivariate example, we have generated three plots to compare the approximation from Fourier integrals to the underlying truth (i.e., compare the approximated and true characteristic functions of the bivariate normal distribution). In Figure 2, we present the surface plots of the approximated and the true values, as well as their differences for both the real and imaginary parts. One observes that differences are small, indicating the adequacy of the numerical integration.

For a different perspective of the resulting Fourier integration, Figure 3 presents a contour plot showing the approximated and true values of the bivariate normal characteristic function, for both real and imaginary parts. We show a tile plot of the differences in Figure 4. Observe that the range of differences in Figure 4 is relatively much smaller than the values in Figure 3.

Speed comparison

Through a small numerical study, here we compare the differences in execution times using `fourierin()` for integration at points on a regular grid, both with or without FFT steps, considering univariate and bivariate Fourier integrals. Figure 5 shows timing results for a univariate example of the integral in (1) evaluated on a grid, while Figure 6 presents timing results for a bivariate example. Note that the reported time units differ between these figures, as the bivariate case naturally requires more time. These figures provide evidence that, for evaluating integrals on a regular grid, the FFT option in `fourierin()` creates large advantages in time.

The code that was used to generate Figure 5 and 6 is below.

```
## ------------------------------------------------------------------
## Univariate speed test
## ------------------------------------------------------------------

library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)
library(microbenchmark)

## Test speed at several resolutions
resolution <- 2^(3:8)

## Function to be tested
```

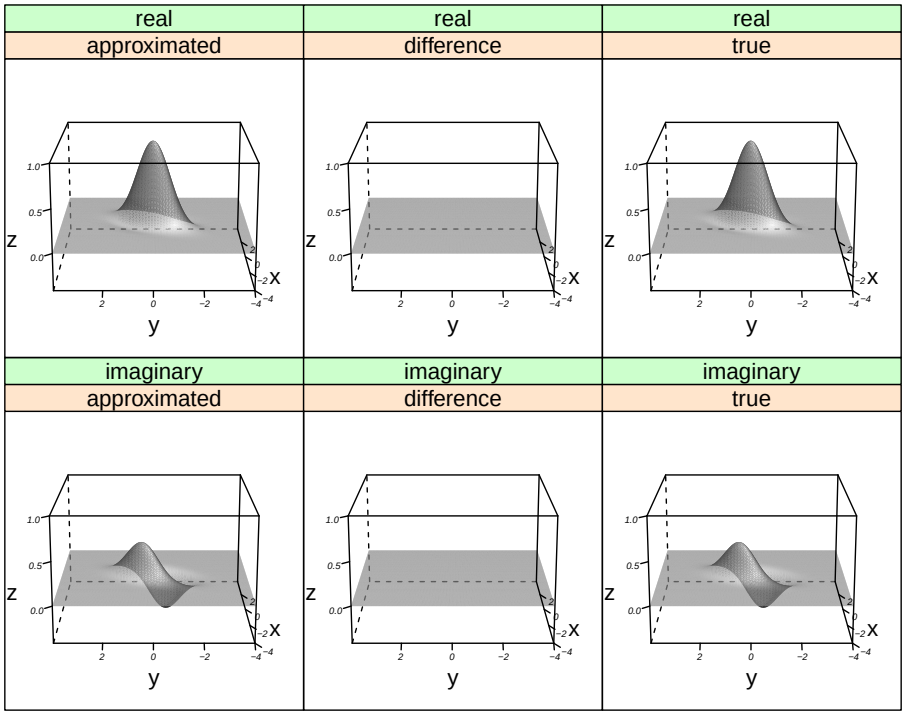

Figure 2: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. This panel contains every combination of approximation-true-difference with real-imaginary parts.

```
myfnc <- function(t) exp(-t^2/2)

## Aux. function
compute_times <- function(resol){
    out <-
        microbenchmark(
            fourierin_1d(f = myfnc, -5, 5, -3, 3, -1, -1, resol),
            fourierin_1d(f = myfnc, -5, 5, -3, 3, -1, -1, resol,
                         use_fft = FALSE),
            times = 5) %>%
        as.data.frame()
    ## Rename levels
    levels(out$expr) <- c("yes", "no")
    ## Obtain median of time.
    out %>%
        group_by(expr) %>%
        summarize(time = median(time*1e-6),
                  resolution = resol) %>%
        rename(FFT = expr)
}

speed1 <- resolution %>%
    map_df(compute_times) %>%
    mutate(resolution = as.factor(resolution)) %>%
    ggplot(aes(resolution, log(time), color = FFT)) +
    geom_point(size = 2, aes(shape = FFT)) +
```

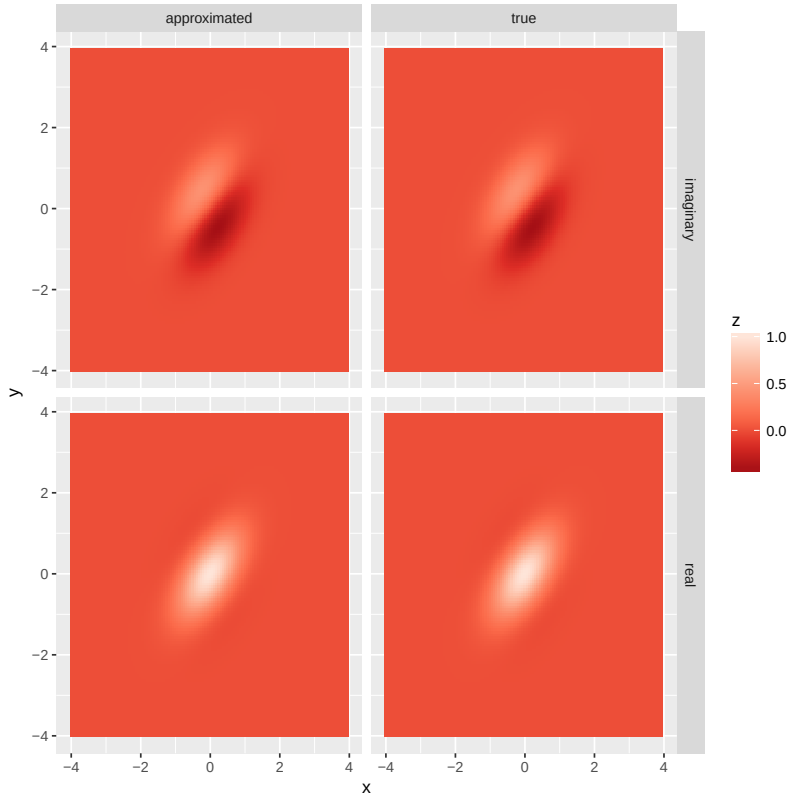

Figure 3: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. Each combination the approximated and true values are shown for both the real and imaginary parts.

```
    geom_line(aes(linetype = FFT, group = FFT)) +
    ylab("time (in log-milliseconds)")

speed1

## ------------------------------------------------------------------
## Bivariate test
## ------------------------------------------------------------------

## Load packages
library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)
library(microbenchmark)

## Test speed at several resolutions
resolution <- 2^(3:7)

## Bivariate function to be tested
myfnc <- function(x) dnorm(x[, 1])*dnorm(x[, 2])

## Aux. function
compute_times <- function(resol){
    resol <- rep(resol, 2)
    out <-
        microbenchmark(
            fourierin(myfnc,
                      lower_int = c(-8, -6), upper_int = c(6, 8),
                      lower_eval = c(-4, -4), upper_eval = c(4, 4),
                      const_adj = 1, freq_adj = 1,
```

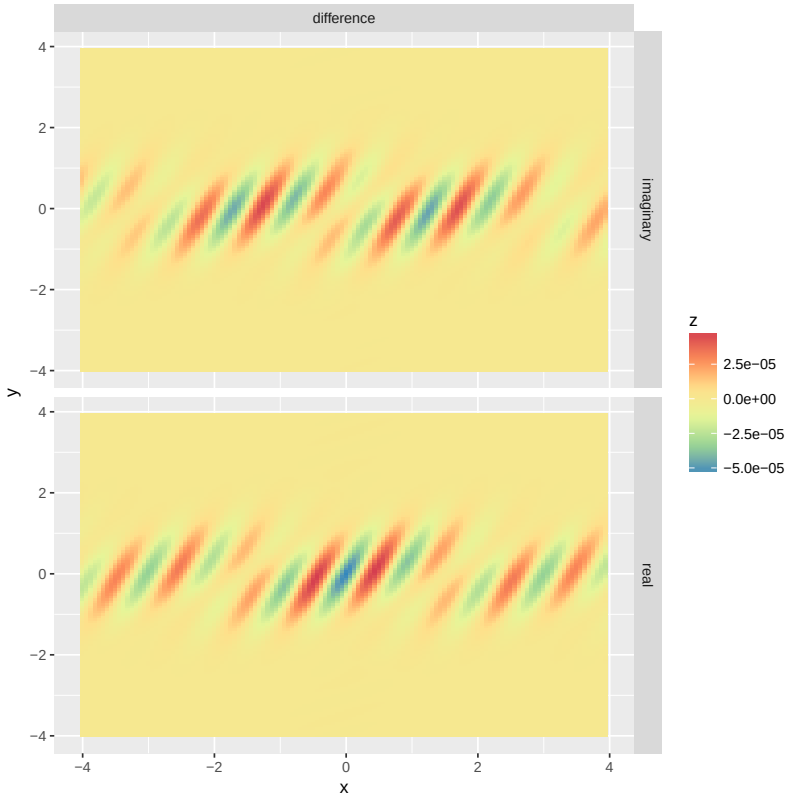

Figure 4: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. This plot show the difference between the approximated and true values for the real and imaginary parts.

```
                  resolution = resol),
        fourierin(myfnc,
                  lower_int = c(-8, -6), upper_int = c(6, 8),
                  lower_eval = c(-4, -4), upper_eval = c(4, 4),
                  const_adj = 1, freq_adj = 1,
                  resolution = resol, use_fft = FALSE),
        times = 3) %>%
      as.data.frame()

    ## Rename levels
    levels(out$expr) <- c("yes", "no")
    ## Obtain median of time.
    out %>%
        group_by(expr) %>%
        summarize(time = median(time*1e-9),
                  resolution = resol[1]) %>%
        rename(FFT = expr)
}

## Values
comparison <-
    resolution %>%
    map_df(compute_times)

fctr_order <-
    unique(comparison$resolution) %>%
    paste(., ., sep = "x")

## Plot
speed2 <- comparison %>%
    mutate(resolution = paste(resolution, resolution, sep = "x"),
```

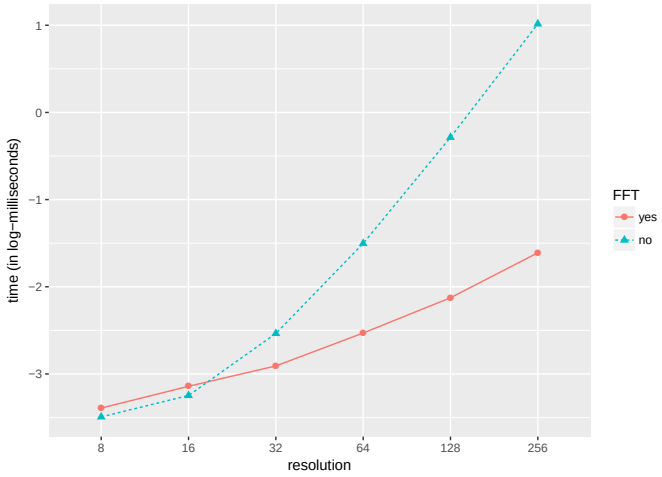

Figure 5: Example of a univariate Fourier integral over grids of several (power of two) sizes. Specifically, the standard normal density is being recovered using the Fourier inversion formula. Time is in log-milliseconds. The Fourier integral has been applied five times for every resolution and each dot represents the mean for the corresponding grid size and method. Observe that both, x and y axis are in logarithmic scale.

```
        resolution = ordered(resolution, levels = fctr_order)) %>%
    ggplot(aes(resolution, log(time), color = FFT)) +
    geom_point(size = 2, aes(shape = FFT)) +
    geom_line(aes(linetype = FFT, group = FFT)) +
    ylab("time (in log-seconds)")

speed2
```

Summary

Continuous Fourier integrals/transforms are useful in statistics for computation of probability densities from characteristic functions, as well as the reverse, when describing probability structure; see the "Examples" section for some demonstrations. The usefulness and potential application of Fourier integrals, however, also extends to other contexts of physics and mathematics, as well as to statistical inference (e.g., types of density estimation). For this reason, we have developed the **fourierin** package as a tool for computing Fourier integrals over collections of evaluation points, where repeat evaluation steps and often complicated numerical integrations are involved. When evaluation points fall on a regular grid, **fourierin** allows use of a Fast Fourier Transform as a key ingredient for rapid numerical approximation of Fourier-type integrals.

In "Speed Comparison," we presented evidence of the gain in time when using this fast implementation of `fourierin()` on regular grids, while we also illustrated the versatility of **fourierin** in "Examples" section. At present (version 0.2.1), the **fourierin** package performs univariate and bivariate Fourier integration. An extension of the package to address higher dimensional integration will be included in future versions.

Bibliography

K. B. Athreya and S. N. Lahiri. *Measure theory and probability theory*. Springer Science & Business Media, 2006. [p72]

D. H. Bailey and P. N. Swarztrauber. A fast method for the numerical evaluation of continuous Fourier and Laplace transforms. *SIAM Journal on Scientific Computing*, 15(5):1105–1110, 1994. [p72]

G. Basulto-Elias. *fourierin: Computes Numeric Fourier Integrals*, 2017. URL https://CRAN.R-project.org/package=fourierin. R package version 0.2.2. [p72]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p72]

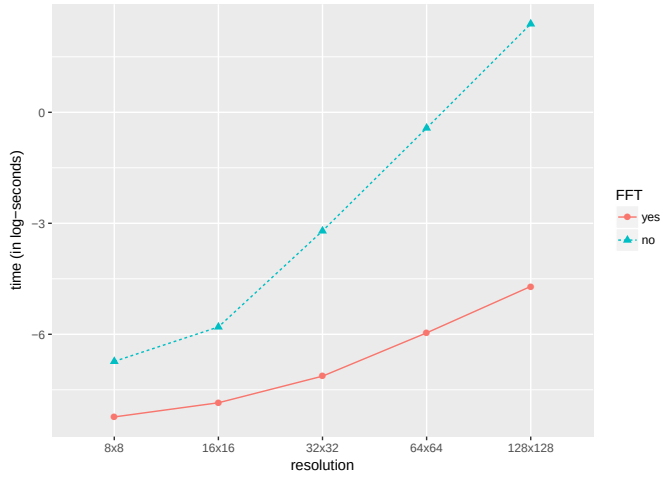

Figure 6: Example of a bivariate Fourier integral over grids of several (power of two) sizes. Both axis have the same resolution. Specifically, the characteristic function of a bivariate normal distribution is being computed. Time is in log-seconds (unlike 5). The Fourier integral has been applied five times for every resolution and each dot represents the mean for the corresponding grid size and method. Observe that both, x and y axis are in logarithmic scale.

G. Inverarity. Fast computation of multidimensional Fourier integrals. *SIAM Journal on Scientific Computing*, 24(2):645–651, 2002. [p72, 73, 75]

A. Meister. *Deconvolution problems in nonparametric statistics*, volume 193. Springer, 2009. [p72]

Guillermo Basulto-Elias
Iowa State University
Ames, IA
United States
basulto@iastate.edu

Alicia Carriquiry
Iowa State University
Ames, IA
United States
alicia@iastate.edu

Kris De Brabanter
Iowa State University
Ames, IA
United States
kbrabant@iastate.edu

Daniel J. Nordman
Iowa State University
Ames, IA
United States
dnordman@iastate.edu