Journal of Statistical Software

July 2019, Volume 90, Issue 3. doi: 10.18637/jss.v090.i03

corr2D: Implementation of Two-Dimensional Correlation Analysis in R

Robert Geitner
Friedrich Schiller University Jena

Robby Fritzsch
Friedrich Schiller University Jena

Jürgen Popp
Friedrich Schiller University Jena

Thomas W. Bocklitz
Friedrich Schiller University Jena

Abstract

In the package **corr2D** two-dimensional correlation analysis is implemented in R. This paper describes how two-dimensional correlation analysis is done in the package and how the mathematical equations are translated into R code. The paper features a simple tutorial with executable code for beginners, insight into the calculations done before the correlation analysis, a detailed look at the parallelization of the fast Fourier transformation based correlation analysis and a speed test of the calculation. The package **corr2D** offers the possibility to preprocess, correlate and postprocess spectroscopic data using exclusively the R language. Thus, **corr2D** is a welcome addition to the toolbox of spectroscopists and makes two-dimensional correlation analysis more accessible and transparent.

Keywords: correlation analysis, 2D correlation, spectroscopy, R, R package, **corr2D**.

1. Introduction to 2D correlation spectroscopy

Since their invention scientists used infrared (IR), Raman or nuclear magnetic resonance (NMR) spectroscopy to gain information on atoms and molecules. The usual way to extract information from IR, Raman or NMR spectra is to assign observed spectral signals to molecular structures and thus deducing molecular properties. When analyzing a series of spectra it is sometimes difficult to identify spectral changes of two overlapping signals making it impossible to assign these signals to specific molecular structures. To overcome these problems two-dimensional (2D) correlation analysis was invented (Noda 1989, 1993).

The basic idea of a correlation analysis is to analyze how similar (or dissimilar) two spectral

signals change. The correlation analysis describes in a quantitative manner how similar these two signals behave. 2D correlation spectroscopy is a pure mathematical processing of signals. 2D NMR or 2D IR experiments which are based on physical correlation processes during the respective spectroscopic measurements are related to 2D correlation spectroscopy. 2D correlation spectroscopy correlates spectroscopic data after the measurement while 2D NMR and 2D IR experiments generate the correlation during the data collection by special experimental setups.

2D correlation analysis (which is another term to describe 2D correlation spectroscopy) is used in spectroscopy to analyze spectral features more clearly and to extract additional information, which may be obscured in classical one-dimensional (1D) plots of spectra. To achieve this goal 2D correlation spectroscopy correlates a series of spectra collected under the influence of an external perturbation using the correlation integral. Isao Noda applied the correlation integral to a series of IR spectra of polymers collected under the influence of a sinusoidal tensile strain in 1986 (Noda 1986) and later generalized the approach in 1989 and 1993 (Noda 1989, 1993).

Today 2D correlation analysis is used in spectroscopy to analyze dynamic systems under a specific perturbation. In this context IR, Raman, NMR and UV/Vis spectroscopy as well as mass spectrometry have been used to study polymers, reaction solutions and pharmaceuticals under the influence of temperature, time and electro-magnetic radiation. For good reviews on spectroscopic methods, samples and perturbations used in 2D correlation spectroscopy the reader is referred to Noda (2014a,b) and Park, Noda, and Jung (2016).

Although 2D correlation spectroscopy is used by an ever growing community, there has – to the best of our knowledge – so far been no publicly available implementation of 2D correlation spectroscopy in R (R Core Team 2019). Furthermore there is only one standalone software available to do 2D correlation spectroscopy. It is called **2DShige**, can be downloaded for free and was developed by Shigeaki Morita (Morita 2005). Unfortunately, **2DShige** is a standalone program and thus it is difficult to use it in combination with other software, which may be used to preprocess the spectroscopic data accordingly. It is also not an open source software and thus lacks transparency. As an alternative to **2DShige** home-written MATLAB (The MathWorks Inc. 2016) scripts are often used to carry out 2D correlation analysis (López-Díez, Winder, Ashton, Currie, and Goodacre 2005; Barton, de Haseth, and Himmelsbach 2006; Spegazzini, Siesler, and Ozaki 2012) (see also MATLAB contribution **MIDAS 2010** by Ferenc 2011). These MATLAB scripts allow the user to preprocess and correlate the data within one program, but also lack transparency and comprehensibility. The spectroscopy and analysis software **OPUS** from Bruker Corporation (2016) also has an implemented 2D correlation spectroscopy algorithm. Unfortunately, **OPUS** is a commercial software and lacks some freedom as well as transparency, which other statistical software like R or MATLAB offer. Thus, **OPUS** is very rarely used to perform 2D correlation spectroscopy. The widespread analysis software **Origin** by OriginLab (2019) on the other hand offers the possibility to conduct homo as well as hetero 2D correlation analysis since 2018 via its twoDCorrSpec.opx extension. Unfortunately the use of the extension is (up to 2019) not free and only available for OriginPro users.

In this paper we present our R package **corr2D** (Geitner, Fritzsch, and Bocklitz 2019), which implements 2D correlation spectroscopy in R and is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=corr2D`. Package **corr2D** combines transparency, comprehensibility and the convenience to process and analyze

2D correlation spectra within one open source program. We already published some results (Geitner *et al.* 2015, 2016) utilizing the calculation and plotting properties of **corr2D**. For the calculation of the complex correlation matrix a parallelized fast Fourier transformation (FFT) approach is used. To illustrate the use of **corr2D** the package also features a set of preprocessed temperature-dependent experimental Raman spectra (Geitner *et al.* 2015) and a function to generate artificial data. We hope to enrich both the R community as well as the 2D correlation community with our package.

The paper is divided into three main sections. Section 2 deals with the mathematical background and the theoretical description of 2D correlation spectroscopy. The comprehensive mathematical description of 2D correlation spectroscopy is important because the package **corr2D** translates the 2D correlation theory into executable R code. For newcomers to the field of 2D correlation analysis we suggest reading Noda, Dowrey, Marcoli, Story, and Ozaki (2000) as it features a simplified introduction to the formal mathematical procedure and three application examples. Section 3 is meant as a tutorial for beginners. It describes the structure of the input data and how the resulting object containing the 2D correlation spectra can be visualized. In addition the arguments of the plotting functions `plot_corr2d()` and `plot_corr2din3d()` are presented. To round out the tutorial the section also gives a short introduction to the interpretation of 2D correlation spectra. Section 4 further dives into the technical details of **corr2D**. The section focuses on how the mathematical equations described in Section 2 are translated into R code, how special features of 2D correlation spectroscopy are implemented into `corr2d()`, how the 2D correlation analysis was parallelized and how fast the resulting R code is. The final section also explains the R code behind the plotting functions `plot_corr2d()` and `plot_corr2din3d()`.

2. Theoretical description of 2D correlation spectroscopy

The foundation of 2D correlation spectroscopy are the general auto- and cross-correlation integrals seen in Equations 1 and 2. The result of a general correlation analysis is the correlation coefficient C which describes how similar two signals $f(u)$ and $g(u)$ are depending on a lag τ between them. $f^*(u)$ denotes the complex conjugate of $f(u)$.

$$C_{\text{auto}}(\tau) = \int_{-\infty}^{\infty} f^*(u) \cdot f(u+\tau) du \tag{1}$$

$$C_{\text{cross}}(\tau) = \int_{-\infty}^{\infty} f^*(u) \cdot g(u+\tau) du \tag{2}$$

To use the general correlation integral on spectroscopic data the integral needs to be specified. This is accomplished by replacing the terms $f(u)$ and $g(u)$ in Equation 2 by the dynamic variations of two signals $y_1(\nu_1, t)$ and $y_2(\nu_2, t)$, e.g. spectra. Both spectra depend on their own spectral variables ν_1 and ν_2 as well as on an external perturbation variable t. The spectra are observed within a perturbation interval ranging from T_{min} to T_{max}. This interval is used together with the reference spectrum $\overline{y}(\nu)$ to formally define the dynamic spectrum $\widetilde{y}(\nu, t)$ as seen in Equation 3. The dynamic spectra represent the dynamic changes observed within the

signals $y_1(\nu_1, t)$ and $y_2(\nu_2, t)$ induced by the perturbation t.

$$\widetilde{y}(\nu, t) = \begin{cases} y(\nu, t) - \overline{y}(\nu) & \text{for } T_{\min} \leq t \leq T_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The reference spectrum $\overline{y}(\nu)$ can be chosen arbitrarily. Often the perturbation mean spectrum is used as the reference spectrum (see Equation 4). Other reference spectra could be spectra taken before or after the collection of the perturbation dependent spectra series.

$$\overline{y}(\nu) = \frac{1}{T_{\max} - T_{\min}} \int_{T_{\min}}^{T_{\max}} y(\nu, t) dt \tag{4}$$

There are in principle two ways to calculate 2D correlation spectra: The first approach is based on the Fourier transformation (FT; Noda 1993), while the second one uses the Hilbert transformation (HT; Noda *et al.* 2000). The results of both approaches are identical.

Following the FT approach the dynamic spectra need to be Fourier transformed to separate them into component waves as stated in Noda (1990) and as can be seen in Equation 5. The FTs of the dynamic spectra can then be used to obtain a complex cross correlation function (Equation 6). The real and imaginary part of the complex cross-correlation function are termed synchronous and asynchronous 2D correlation spectra $\Phi(\nu_1, \nu_2)$ and $\Psi(\nu_1, \nu_2)$.

$$\widetilde{Y}(\nu, \omega) = \mathcal{F}(\widetilde{y}(\nu, t)) = \int_{-\infty}^{\infty} \widetilde{y}(\nu, t) \cdot e^{-i\omega t} dt \tag{5}$$

$$\Phi(\nu_1, \nu_2) + i\Psi(\nu_1, \nu_2) = \frac{1}{2\pi(T_{\max} - T_{\min})} \int_{-\infty}^{\infty} \widetilde{Y}(\nu_1, \omega) \cdot \widetilde{Y}^*(\nu_2, \omega) d\omega \tag{6}$$

Figure 1 illustrates how two signals $y_1(t)$ and $y_2(t)$, which depend on the same perturbation t, can be correlated with each other. Both signals react to the external perturbation. This could be two specific wavenumber positions in a Raman spectrum reacting to a changing temperature. If both signals react in an identical way to the perturbation (Case (a) in Figure 1) the resulting complex correlation value has a non-zero real part while the imaginary part is zero. According to Equation 6 the real and imaginary part of the complex correlation value are called synchronous and asynchronous 2D correlation intensities Φ and Ψ. If both signals react exactly with a phase difference of $\pi/2$ to the perturbation (Case (b) in Figure 1) then the complex correlation value only consists of an imaginary part. This means that the two signals only show an asynchronous correlation behavior. The case that is most often encountered when analyzing real-world data is that the complex correlation coefficient is made up from both real and imaginary parts and that the two correlated signals show synchronous as well as asynchronous correlation behavior (Case (c) in Figure 1). During the process of a complete 2D correlation analysis not only two but all combinations of spectral signals are correlated with each other. To make the results accessible to humans the real and imaginary parts of the calculated complex correlation coefficients are presented as synchronous and asynchronous 2D correlation spectra.

When analyzing m discrete data values the FT has to change to the discrete Fourier transformation (DFT). Following this change Equations 5 and 6 transform into Equations 7 and 8.

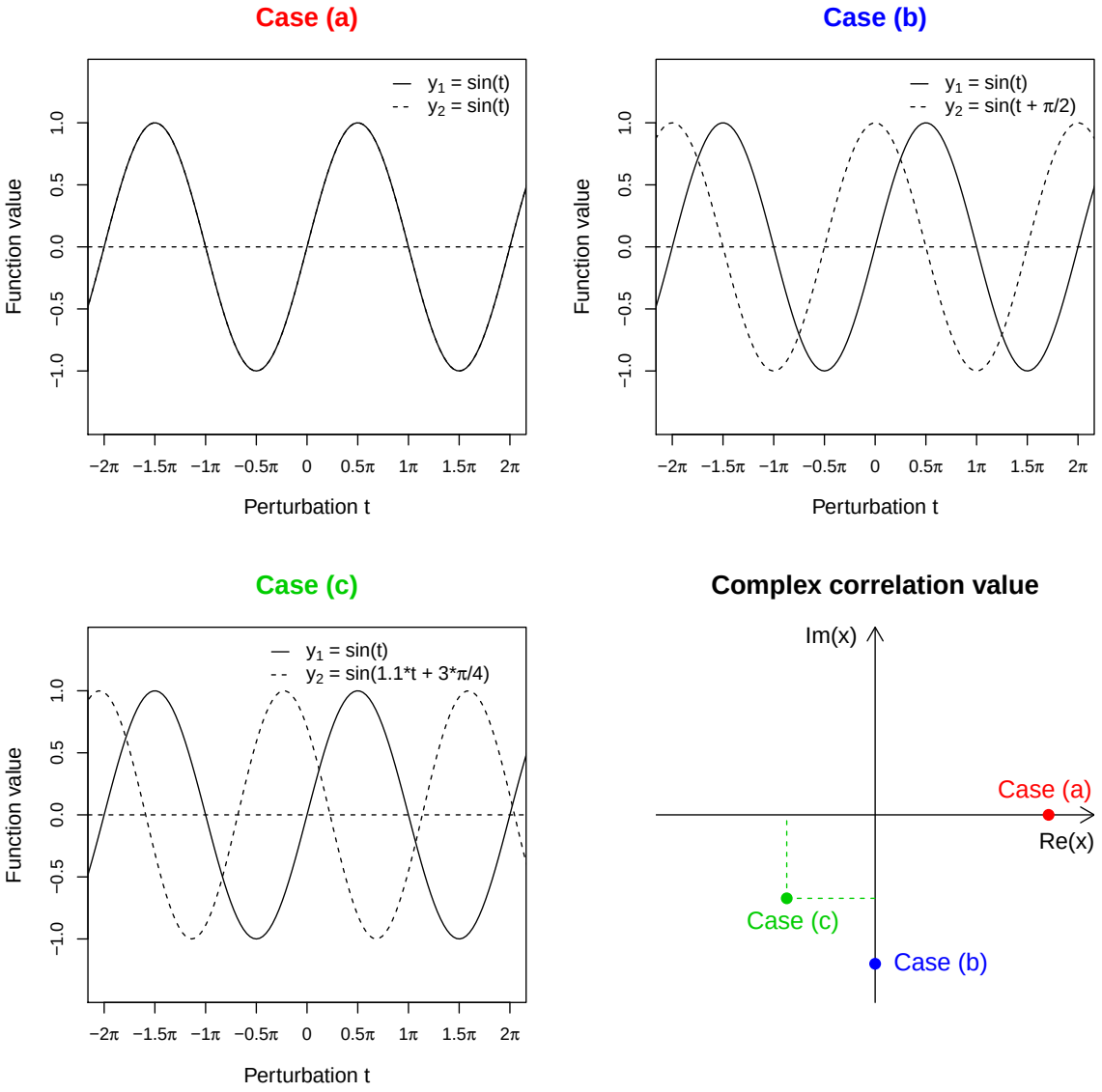

Figure 1: The figure illustrates three examples of a correlation analysis of two signals using Equations 7 and 8. The signals y_1 and y_2 react to a perturbation t. Case (a) (red; top left panel) shows a pure synchronous behavior, while case (b) (blue; top right panel) illustrates the pure asynchronous behavior. Case (c) (green; bottom left panel) showcases the ordinary correlation behavior where the complex correlation value (bottom right panel) shows synchronous and asynchronous contributions. For details see the text.

A fast implementation of the DFT is the fast Fourier transformation (FFT), which is often used to implement the DFT within computer algorithms. When using Equation 8 for the calculation of 2D correlation spectra of discrete data an important condition to be fulfilled is the even spacing of the discrete perturbation values T along the perturbation axis t. Otherwise the unevenly sampled data has to be interpolated to form evenly sampled data or the

correlation function needs to be modified (Noda 2003).

$$\widetilde{Y}(\nu,\omega) = \sum_{j=1}^{m} \widetilde{y}(\nu,t_j) \cdot e^{-i\frac{2\pi\cdot\omega}{m}(j-1)} \quad \text{for } \omega = 0,1,\ldots,m-1 \tag{7}$$

$$\Phi(\nu_1,\nu_2) + i\Psi(\nu_1,\nu_2) = \frac{1}{\pi(m-1)} \sum_{\omega=0}^{m-1} \widetilde{Y}(\nu_1,\omega) \cdot \widetilde{Y}^*(\nu_2,\omega) \tag{8}$$

If the input data is real (a condition that is often met when dealing with real world data) the DFT features Hermitian symmetry as seen in Equation 9, which can be used to further simplify the calculation process.

$$\widetilde{Y}(\nu,\omega) = \widetilde{Y}^*(\nu,m-\omega) \quad \text{for } \omega = 1,2,\ldots,m-1 \tag{9}$$

In summary the calculation of 2D correlation spectra following the FT approach consists of three main steps:

1. Calculation of the dynamic spectra $\widetilde{y}(\nu,t)$ from the preprocessed spectra $y(\nu,t)$ and a chosen reference spectrum $\overline{y}(\nu)$.

2. FT of the dynamic spectra $\widetilde{y}(\nu,t)$ to receive $\widetilde{Y}(\nu,\omega)$.

3. Correlation of the Fourier transformed dynamic spectra $\widetilde{Y}(\nu,\omega)$ for every spectral value pair using the correlation integral to calculate the synchronous and asynchronous correlation spectra $\Phi(\nu_1,\nu_2)$ and $\Psi(\nu_1,\nu_2)$.

Another approach to calculate 2D correlation spectra is based on the HT (Noda *et al.* 2000). The HT approach splits the calculation of the synchronous and asynchronous 2D correlation spectra. The synchronous 2D correlation spectrum $\Phi(\nu_1,\nu_2)$ can be directly calculated from the dynamic spectra (see Equation 10). For the calculation of the asynchronous 2D correlation spectrum $\Psi(\nu_1,\nu_2)$ the HT $\widetilde{z}(\nu_2,t)$ of one of the dynamic spectra is needed (see Equations 11 and 12). The HT is defined using the Cauchy principle value.

$$\Phi(\nu_1,\nu_2) = \frac{1}{T_{\max}-T_{\min}} \int_{T_{\min}}^{T_{\max}} \widetilde{y}(\nu_1,t) \cdot \widetilde{y}(\nu_2,t)dt \tag{10}$$

$$\widetilde{z}(\nu_2,t) = \mathcal{H}(\widetilde{y}(\nu_2,t)) = \frac{1}{\pi} \quad PV \int_{-\infty}^{\infty} \frac{\widetilde{y}(\nu_2,t')}{t'-t}dt \tag{11}$$

$$\Psi(\nu_1,\nu_2) = \frac{1}{T_{\max}-T_{\min}} \int_{T_{\min}}^{T_{\max}} \widetilde{y}(\nu_1,t) \cdot \widetilde{z}(\nu_2,t)dt \tag{12}$$

When dealing with m discrete data values Equations 10–12 can be transformed to their respective discrete forms as seen in Equations 13–16. The discrete HT can be done using the so called Hilbert-Noda matrix N_{jk}. The Hilbert-Noda matrix simplifies the discrete HT to a matrix vector multiplication. As discussed for the FT approach it is also important for the

HT approach that the discrete data is equidistant (Noda 2003).

$$\Phi(\nu_1, \nu_2) = \frac{1}{m-1} \sum_{j=1}^{m} \widetilde{y}(\nu_1, t_j) \cdot \widetilde{y}(\nu_2, t_j) \tag{13}$$

$$\widetilde{z}(\nu_2, t_j) = \sum_{k=1}^{m} N_{jk} \cdot \widetilde{y}(\nu_2, t_k) \tag{14}$$

$$N_{jk} = \begin{cases} 0 & \text{if } j = k \\ \frac{1}{\pi(k-j)} & \text{otherwise} \end{cases} \tag{15}$$

$$\Psi(\nu_1, \nu_2) = \frac{1}{m-1} \sum_{j=1}^{m} \widetilde{y}(\nu_1, t_j) \cdot \widetilde{z}(\nu_2, t_j) \tag{16}$$

In summary the calculation of 2D correlation spectra following the HT approach consists of four main steps:

1. Calculation of the dynamic spectra $\widetilde{y}(\nu, t)$ from the preprocessed spectra $y(\nu, t)$ and the reference spectrum $\overline{y}(\nu)$.

2. Calculation of the synchronous 2D correlation spectrum $\Phi(\nu_1, \nu_2)$ by multiplying $\widetilde{y}(\nu_1, t)$ and $\widetilde{y}(\nu_2, t)$.

3. HT of the dynamic spectra $\widetilde{y}(\nu_2, t)$ to get $\widetilde{z}(\nu_2, t)$.

4. Calculation of the asynchronous 2D correlation spectrum $\Psi(\nu_1, \nu_2)$ by multiplying $\widetilde{y}(\nu_1, t)$ and $\widetilde{z}(\nu_2, t)$.

As stated above the result of the FT and HT approach are identical. Thus the 2D correlation spectra display the same information independent of how they were calculated. The synchronous 2D correlation spectrum $\Phi(\nu_1, \nu_2)$ shows which spectral features change in-phase while the asynchronous 2D correlation spectrum $\Psi(\nu_1, \nu_2)$ shows which spectral features change out-of-phase. If a spectral dataset is correlated with itself (an autocorrelation in the general terminology) the resulting 2D spectra are called homo-correlation spectra. If two different spectral datasets are correlated with each other (a cross-correlation in the general terminology) the resulting 2D spectra are called hetero-correlation spectra.

In addition to the basic 2D correlation spectroscopy described above, modifications to the original technique have been developed. One of these modifications is the application of scaling techniques to 2D correlation spectra. The goal of scaling 2D correlation spectra is to enhance correlation signals with low intensity when compared to correlation signals with high intensity. High intensity signals sometimes dominate 2D correlation spectra and thus hamper the identification of smaller signals. The approach to solve this problem is described in Noda (2008). The basic idea is to scale the synchronous and asynchronous correlation spectra $\Phi(\nu_1, \nu_2)$ and $\Psi(\nu_1, \nu_2)$ using the standard deviation σ_ν, which is calculated from the original spectral dataset $y(\nu)$ and the reference spectrum $\overline{y}(\nu)$ as seen in Equation 17. Strictly this scaling is only defined when using the mean-spectrum (see Equation 4) as reference spectrum, but the scaling procedure can also be applied when using other reference spectra.

Care should be taken when interpreting scaled 2D correlation spectra which are not scaled using the perturbation mean-spectrum.

$$\sigma_{\nu_i} = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(y(\nu_i, t_j) - \overline{y}(\nu))^2} \tag{17}$$

For every spectral value ν_i (for $i = 1, 2, \ldots, n$) there is a standard deviation σ_{ν_i}. To scale the correlation intensity at the position (ν_1, ν_2) the correlation intensity at this position $\Phi(\nu_1, \nu_2)$ (or $\Psi(\nu_1, \nu_2)$) is divided by the product of the two standard deviations σ_{ν_1} and σ_{ν_2} sometimes referred to as total joint variance (Noda 2004). This conventional unit-variance scaling is called Pearson scaling (see Equations 18 and 19). Unfortunately, Pearson scaling strongly increases the influence of noise on 2D correlation spectra. To counteract this effect Noda suggested using Pareto scaling where the data is scaled by the square root of the standard deviations. The generalized approach – as seen in Equations 20 and 21 – is to introduce an exponent α which describes how the total joint variance is used to scale the correlation intensities.

$$\Phi(\nu_1, \nu_2)^{\text{Pearson}} = \frac{\Phi(\nu_1, \nu_2)}{(\sigma_{\nu_1} \cdot \sigma_{\nu_2})^{-1}} \tag{18}$$

$$\Psi(\nu_1, \nu_2)^{\text{Pearson}} = \frac{\Psi(\nu_1, \nu_2)}{(\sigma_{\nu_1} \cdot \sigma_{\nu_2})^{-1}} \tag{19}$$

$$\Phi(\nu_1, \nu_2)^{(\text{Scaled})} = \frac{\Phi(\nu_1, \nu_2)}{(\sigma_{\nu_1} \cdot \sigma_{\nu_2})^{-\alpha}} \tag{20}$$

$$\Psi(\nu_1, \nu_2)^{(\text{Scaled})} = \frac{\Psi(\nu_1, \nu_2)}{(\sigma_{\nu_1} \cdot \sigma_{\nu_2})^{-\alpha}} \tag{21}$$

For more information on the theory of 2D correlation spectroscopy, the calculations, scaling techniques and the influence of noise the reader is referred to the literature (Noda 1990, 2006, 2016a; Šašić, Muszynski, and Ozaki 2001; Yu, Liu, and Noda 2003; Noda 2008). For techniques derived from the basic 2D correlation spectroscopy like sample-sample 2D correlation spectroscopy, moving window 2D correlation spectroscopy, multiple-perturbation 2D correlation spectroscopy, double 2D correlation spectroscopy, 2D codistribution spectroscopy and quadrature 2D correlation spectroscopy the reader is also referred to the literature (Šašić, Muszynski, and Ozaki 2000a,b; Thomas and Richardson 2000; Shinzawa *et al.* 2009; Noda 2010, 2014a, 2016b,c).

3. corr2D tutorial and interpretation of 2D correlation spectra

The package **corr2D** contains a calculation function `corr2d()`, two plotting functions `plot_corr2d()` and `plot_corr2din3d()` as well as two S3 methods `summary()` and `plot()` for the resulting 2D correlation object.

To get started the user just has to supply an $[m \times n]$ matrix containing the data to `corr2d()`. The input matrix needs to contain the data as follows: m perturbation values T by rows and n spectral values ν by columns. The column names of the input matrix should contain the spectral value names. Alternatively, the spectral value names can be specified at the argument `Wave1`. The perturbation values can be included as row names. The `FuranMale`

dataset from **corr2D** can be viewed as an example. It contains temperature-dependent Raman spectra, therefore the perturbation variable t is the temperature and the spectral variable ν is the relative wavenumber.

```
R> library("corr2D")
R> data("FuranMale", package = "corr2D")
R> FuranMale[, 1:5]

       1550.26392    1550.74602   1551.22812   1551.71023  1552.19233
110  0.0058962811  5.506783e-03 0.0051347609 0.0047584418 0.004295668
120  0.0043970716  4.860985e-03 0.0052363316 0.0055872210 0.005806852
130  0.0055330645  4.916008e-03 0.0045517037 0.0043091089 0.004450099
140 -0.0008350893 -4.653592e-04 0.0003397819 0.0014053808 0.002903903
150 -0.0005203668  3.312193e-05 0.0003998590 0.0008810001 0.001332789
160  0.0060360763  6.776913e-03 0.0073772994 0.0076989039 0.007883910

R> twod <- corr2d(FuranMale)

HOMO-Correlation: 2 cores used for calculation
10:39:26 - using mean values as reference
10:39:26 - Fast Fourier Transformation and multiplication
 to obtain a 145 x 145 correlation matrix
10:39:26 - Done
```

`corr2d()` identifies that only one input matrix was given. Thus, `corr2d()` correlates the input data with itself which results in homo-correlation spectra. If a second input matrix would be given to `corr2d()` (at argument `Mat2`) the function would automatically calculate the hetero-correlation spectra from the two input matrices. Additionally the function also detects that no reference spectrum (at argument `Ref1`) was specified and thus builds the mean spectrum, which is then used as reference spectrum. The resulting correlation matrix has the dimensions $[n \times n]$, therefore the `FuranMale` matrix results in a $[145 \times 145]$ correlation matrix. The correlation function `corr2d()` will be discussed in detail in Section 4.2.

The correlation spectra can be plotted using the `plot()` command, which in turn calls `plot_corr2d()` via S3 method dispatch. The real part of the complex correlation spectrum is called synchronous correlation spectrum , while the imaginary part is called asynchronous correlation spectrum (see Figure 2). By default `plot_corr2d()` displays the synchronous spectrum.

```
R> plot_corr2d(twod)
R> plot(twod, Im(twod$FT))
```

The plot function `plot_corr2d()` offers a lot of features which can be used to alter the appearance of the 2D correlation plot. With the arguments `specx` and `specy` the data plotted at the top and on the left of the spectrum can be controlled. `xlim`, `ylim`, `zlim`, `axes`, `xlab` and `ylab` are inspired by the normal `plot()` (or better `image()`) function as they allow the user to control the displayed region as well as the x- and y-axis and their labels. By default `plot_corr2d()` uses the `contour()` function from package **graphics** to produce a contour

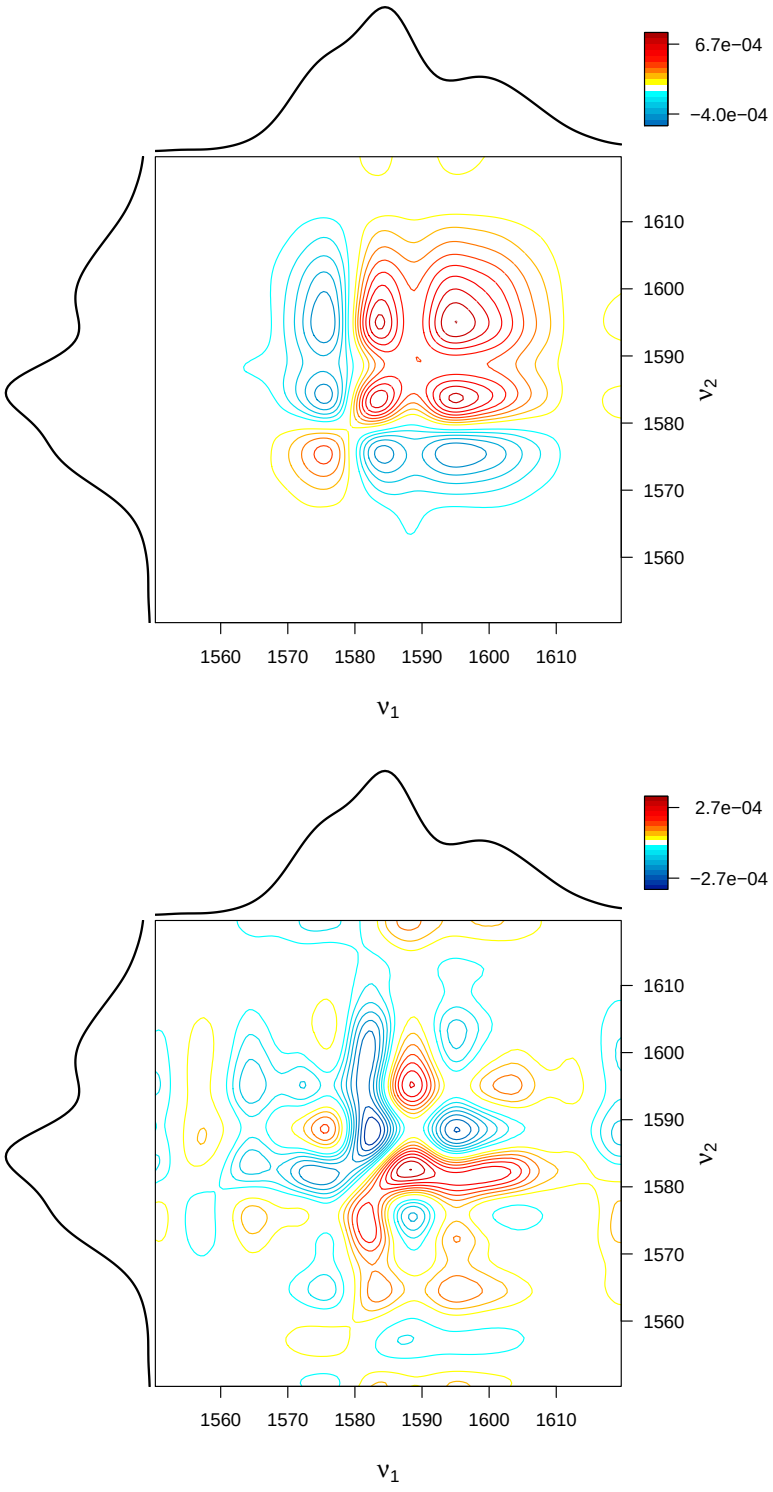

Figure 2: Synchronous (top) and asynchronous (bottom) 2D homo-correlation spectra of the dataset `FuranMale`. The reference spectrum is the mean spectrum.

plot. This can be changed via the `Contour` argument to produce an image (once again using the **graphics** package) representing the correlation spectrum. The number of equally spaced contour levels (or image layers) plotted can be adjusted with `N` and the `Cutout` argument. The `Cutout` argument can be used to define a number range which will not be plotted. This argument suppresses the plotting of smaller signals and allows to make the resulting plot clearer for the reader, but can also be used to remove unwanted signals and obscure results. Thus, it should always be used with care to simplify the plots without omitting important information. Finally the argument `Legend` controls the color legend in the top right corner. Usually the absolute values of correlation spectra are not relevant for the 2D correlation plot. Therefore, the legend can usually be omitted, because the graphical interpretation does not depend on it. For an example with most of the arguments in action see the following code snippet.

```
R> plot(twod, Re(twod$FT), xlim = c(1560, 1620), ylim = c(1560, 1620),
+    xlab = expression(paste("relative Wavenumber" / cm^-1)),
+    ylab = expression(paste("relative Wavenumber" / cm^-1)),
+    Contour = FALSE, N = 32, Cutout = c(-0.8 * 10^-4, 1.3 * 10^-4),
+    Legend = FALSE)
```

Besides `plot_corr2d()` **corr2D** features another plotting function: `plot_corr2din3d()`. This function can be used to draw a colored 3D surface representation of 2D correlation spectra. To achieve this `plot_corr2din3d()` makes use of `drape.plot()` and `drape.color()` from package **fields** (Nychka, Furrer, Paige, and Sain 2019). These two functions calculate the color values and graphical positions form the input 2D correlation data and hand these values over to `persp()` which then does the plotting of the 3D surface using the perspective angles `theta` and `phi`. To add information to the plot `plot_corr2din3d()` allows the user to add custom spectra (`specx` and `specy`) to the x- and y-axis of the `persp()` plot. These spectra can be scaled using the `scalex` and `scaley` arguments from `plot_corr2din3d()`. The sign of the scaling factor defines if the spectra are plotted inside (positive sign) or outside (negative sign) the `persp()` plot. In addition a 2D projection of the 3D surface can be added to the bottom of the plot. This is also done using `drape.plot()` and together with the x- and y-axis spectra the projection recalls the look of a flat 2D correlation plot. To reduce the computational demand of large 2D correlation matrices `plot_corr2din3d()` features the argument `reduce` which can be used to reduce the number of points used for drawing the 3D surface. Thus, the argument `reduce` allows to calculate a first draft of the 3D surface to adjust the plotting parameters without a high computational demand. Overall, `plot_corr2din3d()` is a useful addition to **corr2D** and allows to illustrate 2D correlation spectra with impressive colored 3D surface plots. Figure 3 shows a 3D surface plot of the synchronous 2D correlation spectrum from the `FuranMale` dataset.

```
R> plot_corr2din3d(Mat = Re(twod$FT), specx = twod$Ref1, specy = twod$Ref1,
+    reduce = 2, scalex = -150, scaley = -130, zlim = c(-0.7, 1) * 10^-3,
+    projection = TRUE, border = NULL, theta = 25, phi = 15,
+    add.legend = FALSE, Col = colorspace::diverge_hcl(129, h = c(240, 0),
+      c = 100, l = c(20, 100), power = 0.3))
```

The interpretation of 2D correlation spectra is based on the results of the correlation integral. Following the definition of the synchronous and asynchronous correlation spectra (see Figure 1

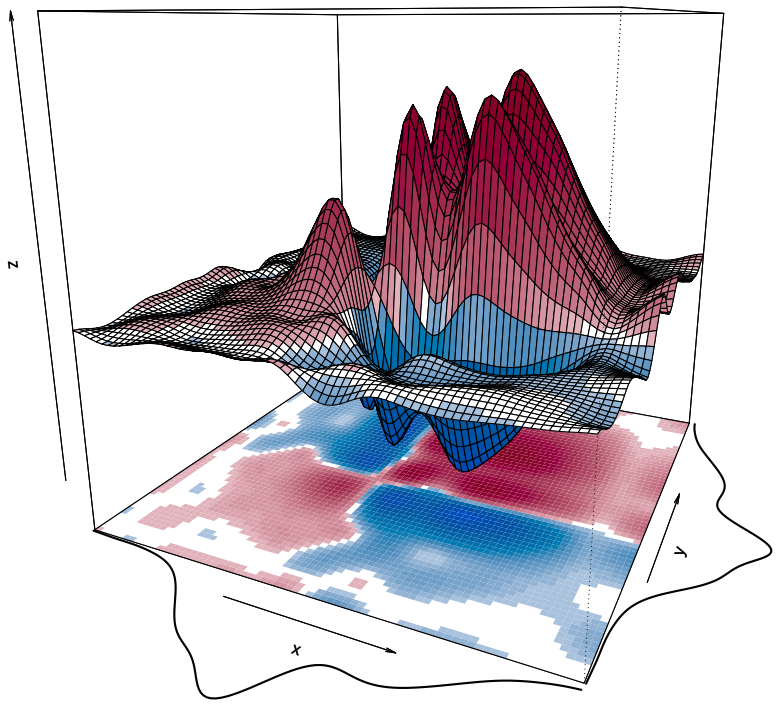

Figure 3: 3D surface plot of the synchronous 2D homo-correlation spectrum of the dataset `FuranMale` showcasing `plot_corr2din3d()`. The plot features two custom spectra on the x- and y-axes and a 2D projection of the 3D surface.

as well as Equations 6, 10 and 12), a set of rules can be derived, which can be used to understand the colorful 2D correlation spectra. These interpretation rules are called "Noda rules".

There are two types of signals in 2D correlation spectra: auto peaks and cross peaks. Auto peaks are located at the diagonal of 2D homo-correlation, therefore both spectral variables ν_1 and ν_2 have the same value at auto peaks (Case (a) in Figure 1). Auto peaks are always non-negative in the synchronous correlation spectrum and are always 0 in the asynchronous correlation spectrum (compare Case (a) in the bottom right panel in Figure 1). They indicate how strong the spectral intensity changes at a given spectral position ν_1. A strong synchronous auto peak signals characteristic high changes at the associated spectral position.

Often the information gathered from auto peaks is easy to interpret, but not too helpful when compared to the information gained from cross peaks. Cross peaks are not located at the diagonal and show the correlation between changes at two different spectral values ν_1 and ν_2 (Case (b) and Case (c) in Figure 1). Following Noda (1990, 2006) there are five Noda rules, which can be used to interpret synchronous and asynchronous 2D correlation cross peaks:

1. If the sign of a cross peak at a spectral coordinate pair (ν_1, ν_2) of a synchronous 2D correlation spectrum is positive, i.e., $\Phi(\nu_1, \nu_2) > 0$, the spectral intensities measured at ν_1 and ν_2 are changing in the same direction, i.e., both intensities are either increasing or decreasing simultaneously.

2. On the other hand, if the sign of a synchronous cross peak is negative, i.e., $\Phi(\nu_1, \nu_2) < 0$, the spectral intensities measured at ν_1 and ν_2 are changing in the different directions, i.e., one is increasing while the other is decreasing.

3. If the sign of a cross peak at a spectral coordinate pair (ν_1, ν_2) of an asynchronous 2D correlation spectrum is positive, i.e., $\Psi(\nu_1, \nu_2) > 0$, the spectral intensity measured at ν_1 varies before that measured at ν_2 with respect to the perturbation.

4. If the sign of an asynchronous cross peak is negative, i.e., $\Psi(\nu_1, \nu_2) < 0$, the spectral intensity measured at ν_1 varies after that measured at ν_2 with respect to the perturbation.

5. However, if the sign of a synchronous cross peak located at the same spectral coordinate (ν_1, ν_2) is negative, i.e., $\Phi(\nu_1, \nu_2) < 0$, the above two rules are reversed.

As a starting example we have a look at the synchronous 2D correlation spectrum of the `FuranMale` dataset (Figure 2): We see three (positive) auto peaks at (1575 cm^{-1}, 1575 cm^{-1}), (1585 cm^{-1}, 1585 cm^{-1}) and (1600 cm^{-1}, 1600 cm^{-1}) as well as two negative cross peaks at (1575 cm^{-1}, 1585 cm^{-1}) and (1575 cm^{-1}, 1600 cm^{-1}) as well as one positive cross peak at (1585 cm^{-1}, 1600 cm^{-1}). The other three cross peaks are redundant, because a synchronous 2D homo-correlation spectrum is always symmetric with respect to the diagonal. The auto peaks indicate that all three Raman bands are changing during the heating. The positive cross peak at (1585 cm^{-1}, 1600 cm^{-1}) tells us that the bands at 1585 cm^{-1} and 1600 cm^{-1} are changing in the same direction (Noda rule 1), while the two negative cross peaks at (1575 cm^{-1}, 1585 cm^{-1}) and (1575 cm^{-1}, 1600 cm^{-1}) unveil that the band at 1575 cm^{-1} is changing in a different direction when compared to the band at 1585 cm^{-1} and 1600 cm^{-1} (Noda rule 2). If this information about the three bands is combined with the information from the one-dimensional Raman spectrum that the band at 1585 cm^{-1} is increasing in intensity during the heating, it becomes clear that the band at 1600 cm^{-1} is also increasing in intensity while the band at 1575 cm^{-1} is falling in intensity during the heating.

For further conclusions from the Raman spectra the reader is referred to Geitner *et al.* (2015). Further examples and extended discussion on the interpretation of 2D correlation spectra can be found in Noda (1993); Czarnecki (1998); Noda (2006, 2012, 2014a).

4. Technical aspects

4.1. Software versions and hardware setup

R as a software language and its software packages are being actively developed. Thus, this manuscript can only be a snapshot regarding the ongoing development process. For information on the latest version of **corr2D** the user is referred to the online documentation of **corr2D** (Geitner *et al.* 2019).

The current R version is 3.4.1, the version of the R core packages **parallel**, **stats**, **graphics** and **grDevices** are also 3.4.1 (R Core Team 2019), the version of **doParallel** is 1.0.10 (Microsoft Corporation and Weston 2018), the version of **foreach** is 1.4.4 (Microsoft and Weston 2017), the version of **fields** is 9.0 (Nychka *et al.* 2019), the version of **mmand** is 1.5.1 (Clayden 2019),

the version of **rgl** is 0.98.1 (Adler and Murdoch 2019) and the version of **colorspace** is 1.3.2 (Zeileis *et al.* 2019). The version of **corr2D** discussed here is 0.1.12.

The calculation speed test of `corr2d()` depending on the input matrix dimensions and the number of processor cores used was done using the function `microbenchmark()` from package **microbenchmark** (Mersmann 2018) with 10 calculation cycles. The test system was a 64-bit Windows 7 (SP1) setup with 8 GB of random access memory (RAM) and a quadcore Intel Core i5-2450M processor.

The profiling of the two functions `corr2d()` and `plot_corr2d()` from **corr2D** was done using a 64-bit Windows 7 (SP1) system with Intel Core2 Duo CPU E7400 @ 2.80 GHz, 4 GB of RAM as well as an Intel G35 Express Chipset as graphical processing unit and the package **profr** (Wickham 2018). The 2D correlation software **2DShige** was tested with the same system.

4.2. `corr2d()`: From equations to R code

In the following two sections we will have a closer look at the correlation function `corr2d()`, how it handles the input perturbation variable, how it generates the dynamic spectra, how it applies scaling techniques to 2D correlation spectroscopy, how the parallel correlation process works and how the R script compares to other 2D correlation software in terms of speed and user-friendliness.

As described in the introduction it is important that the discrete perturbation values T are equidistant. This requirement is often difficult to fulfill when working with real world data, especially for big datasets with a lot of perturbation values or perturbation values which are hard to adjust like the pH value. To overcome this problem there are three approaches:

1. Ignore the requirement for equidistant perturbation values and use the data as it is.

2. Use modified correlation equations as described in Noda (2003) to account for the uneven sampling of the perturbation variable.

3. Interpolate the perturbation values and the associated spectral data to get an equidistant distribution of the perturbation values.

The first approach is the usual go-to solution, because it takes no complex interpolation and yields reasonable results if the perturbation value distribution is nearly equidistant. The approach fails if the perturbation values are distributed very unevenly over a large observation window. In this case it becomes necessary to interpolate the perturbation values to get an equidistant distribution to use the correlation equations for evenly sampled perturbation values (Equations 6, 10 and 12).

For the interpolation `corr2d()` can use a wide variety of interpolation algorithms, which can be specified at the `Int` argument. In a simple scenario this could be a linear function modeled by `approxfun()`. The default interpolation function for `corr2d()` is the cubic Hermite function `splinefun()` from package **stats**. The interpolation process consists of three steps: an interpolation to get m (specified by argument `N`) equidistant perturbation values, a value wise interpolation of the spectral dataset using the interpolation function given by `Int` and the calculation of the new spectral dataset using the interpolated spectral dataset and the interpolated perturbation values. The following lines of code show the interpolation process.

```
R> TIME <- seq(min(Time), max(Time), length.out = N)
R> tmp <- apply(Mat1, 2, function(y) Int(x = Time, y = y))
R> Mat1 <- sapply(tmp, function(x) x(TIME))
R> Time <- TIME
```

The old minimum and maximum perturbation values from `Time` are used to generate the new perturbation values `TIME`. The spectral dataset in `Mat1` is interpolated column wise (which is equal to spectral value wise) using the old perturbation values from `Time` and interpolation specified by `Int` to get a list of functions `tmp` describing the behavior of the spectral intensity at every spectral position along the perturbation axis. This function is then used together with the new perturbation values `TIME` to calculate the interpolated spectral dataset. In addition the new equidistant perturbation values are saved.

This approach to get an evenly sampled perturbation variable is very flexible and allows the use of a wide variety of interpolation functions. This flexibility can be used to get good results from an otherwise sub-optimal dataset.

The next step on the way to calculate 2D correlation spectra is the calculation of the dynamic spectra. The dynamic spectra are built from the original dataset by subtracting a reference spectrum (see Equation 4). Thus, the dynamic spectra show the changes with respect to the chosen reference spectrum.

The reference spectrum can be any spectrum as long as it has the same number of spectral values as the input data. When doing a 2D correlation analysis on a new dataset usually the perturbation mean spectrum is chosen as the reference spectrum. Other reference spectra could be the starting or the end spectrum as well as an external spectrum which is not part of the original dataset. The choice of the reference spectrum depends on the analytical problem and what spectral changes should be highlighted. The correlation function `corr2d()` takes one vector containing the reference spectrum at argument `Ref1` (and a second reference spectrum at `Ref2` when doing a hetero-correlation analysis) and builds the dynamic spectra through subtracting via the `sweep()` function following Equation 3. If no reference spectrum is specified the perturbation mean spectrum is calculated from the input matrix `Mat1` (or `Mat2`) according to Equation 4 and subtracted instead. In this case the dynamic spectra become the mean-centered spectra.

```
R> if (is.null(Ref1)) {
+    Ref1 <- colMeans(Mat1)
+  }
R> Mat1 <- sweep(Mat1, 2, Ref1, "-")
R> if (Het == FALSE) {
+    Mat2 <- Mat1
+    Ref2 <- Ref1
+  } else {
+    if (is.null(Ref2)) {
+      Ref2 <- colMeans(Mat2)
+    }
+    Mat2 <- sweep(Mat2, 2, Ref2, "-")
+  }
```

The final step before determining the actual correlation is applying scaling techniques. Unfortunately, Equations 20 and 21 discussed in Section 2 are not ideal to be directly incorporated

into R code. The number of calculations necessary to scale a synchronous (or asynchronous) homo-correlation spectrum of dimensions $[n \times n]$ is n^2, where n is the number of spectral values. Because homo-correlation spectra are symmetric regarding the diagonal the number of calculations necessary reduces to $(n^2 + n)/2$. Therefore, the time needed to scale a complex homo-correlation spectrum consisting of one synchronous and one asynchronous spectrum scales with $n^2 + n$.

```
R> if (scaling > 0) {
+    sd1 <- apply(Mat1, 1, sd)
+    sd2 <- apply(Mat2, 1, sd)
+    Mat1 <- Mat1 / (sd1^scaling)
+    Mat2 <- Mat2 / (sd2^scaling)
+  }
```

To circumvent the quadratic time scaling `corr2d()` scales the dynamic spectra before doing the correlation. The scaling of mean-centered dynamic spectra by the standard deviation is called auto-scaling. The number of calculations necessary to get auto-scaled dynamic spectra is $m \cdot n$, where m is the number of spectra in the dataset and n is the number of spectral values within each spectrum. Thus, the time needed to auto-scale a dataset is proportional to $m \cdot n$.

When comparing the two calculation times it becomes clear that applying the scaling before the correlation is faster than applying the scaling after the correlation as along m is smaller than n. In other words as long as the number of spectra within the dataset is smaller than the number of spectral values in each spectrum it is faster to apply the scaling before the correlation. This condition is often met because the usual 2D correlation analysis features 10–30 different perturbations values, while each spectrum consists of hundreds of spectral values, e.g., 1024. Therefore, the function `corr2d()` applies the scaling before the correlation.

The function `corr2d()` uses the FT approach described by Equations 5 and 6 to calculate 2D correlation spectra. To do the DFT `corr2d()` uses the FFT provided by the function `fft()` from package **stats**. This type of DFT is much faster than normal DFT when the number of perturbation values has a lot of factors. Thus, `corr2d()` tries to interpolate to 4, 8, 16, ..., 2^n perturbation values when interpolating the perturbation axis.

A problem when implementing the FFT in a linear `for` loop is the speed of the calculations, which drops dramatically when processing a dataset with a large number of spectral values. Therefore, we decided to parallelize the FFT calculations using the **foreach** package (Kane, Emerson, and Weston 2013). Parallel processing using the multi-core structure of modern computers can lead to significant reduced calculation times when doing a large number of similar operations.

```
R> ft1 <- foreach::foreach(i = 1:NCOL(Mat1), .combine = "cbind") %dopar% {
+    fft(Mat1[, i])[1:(NROW(Mat1) - 1) %/% 2 + 1]
+  }
R> if (Het == FALSE) {
+    ft2 <- ft1
+  } else {
+    ft2 <- foreach::foreach(i = 1:NCOL(Mat2), .combine = "cbind") %dopar% {
+      fft(Mat2[, i])[1:(NROW(Mat2) - 1) %/% 2 + 1]
```

```
+    }
+  }
```

The code snippet highlights the FFT implementation in `corr2d()`. With the functions `foreach()` and `%dopar%` the calculation process is split up according to the number of available cores so that the `fft()` calculations will be done in parallel. The result of the parallel calculations is a matrix containing the FTs for every spectral value over the whole perturbation range. This matrix is the discrete analogue to $\widetilde{Y}(\nu, \omega)$ from Equation 5. If a homo-correlation is done the matrix is duplicated, for a hetero-correlation the FFT is also done for the second matrix of dynamic spectra.

`corr2d()` discards half of the calculated FT values. This step can be explained by the symmetry of the FT of real input data. If a signal y gets Fourier transformed, the result is a complex signal $\mathcal{F}(y)$ consisting of a real and an imaginary part. If the input signal y is real, the real part of the FT is always even, while the imaginary part is always odd. In addition the FT values feature Hermitian symmetry $\mathcal{F}(y)(\omega) = \mathcal{F}^*(y)(m - \omega)$ as seen in Equation 9. Following this symmetry condition only half the values of a FT of a real signal are needed to fully describe the information in the corresponding FT. Thus, it is reasonable to only use one half of the FT values when doing a correlation analysis of real data. To extract the correct amplitudes of the Fourier frequencies, the corresponding amplitudes have to be doubled when utilizing the Hermitian symmetry to simplify the calculation process. To account for this, the function `corr2d()` uses all FT values were $\omega \neq 0$ twice.

```
R> cl <- parallel::makeCluster(corenumber)
R> doParallel::registerDoParallel(cl)
R> FT <- matrix(Norm * parallel::parCapply(cl, ft1, get("%*%"), Conj(ft2)),
+    NCOL(ft1), NCOL(ft2), byrow = TRUE)
```

The correlation procedure follows Equation 6 and is parallelized using the function `parCapply()` from package **parallel**. `parCapply()` directs the column wise matrix multiplication to the different cores which are registered before the calculation. The correlation is done using the Fourier transformed dynamic spectra for every spectral value pair (ν_1, ν_2). In addition the resulting correlation matrix gets normalized by factor `Norm` which gets specified in the input to `corr2d()`. The default normalization is the factor $1/(\pi \cdot (m-1))$ where m is the number of sampled perturbation values. The real part of the complex output matrix `FT` is the synchronous correlation spectrum $\Phi(\nu_1, \nu_2)$ while the imaginary part is the asynchronous correlation spectrum $\Psi(\nu_1, \nu_2)$ (see Equation 6).

4.3. Speed test of calculation

The parallelization speeds up the calculation process. To measure the influence of the parallelization and the influence of the input matrix dimensions we designed a small speed test. The speed of `corr2d()` was measured using `microbenchmark()` from package **microbenchmark** with 10 calculation cycles. The input matrix was simulated by `sim2ddata()` (from **corr2D**) with 200, 400, 600, 1000, 4000 and 8000 spectral values n and 5, 10, 20, 100 and 500 perturbation values m. The simulated spectral data of the consecutive first order reaction was also used in Noda (2014a). The calculations were done with 1, 2 or 4 cores, respectively. The calculation and plotting time used by **2DShige** was estimated 10 times using the Windows task manager.

			n						
		m	200	400	600	1000	4000	8000	
(a)	1 core	5	0.36	0.50	0.65	1.05	5.55	16.37	in s
		10	0.36	0.50	0.66	1.10	5.76	18.21	
		20	0.36	0.50	0.67	1.11	6.17	18.61	
		100	0.63	0.98	0.92	1.33	9.74	33.04	
		500	0.48	0.82	1.23	2.57	26.53	105.23	
(b)	2 cores	5	0.56	0.69	0.80	1.18	5.03	15.09	in s
		10	0.62	0.69	0.83	1.18	5.16	15.88	
		20	0.56	0.68	0.82	1.21	5.40	16.87	
		100	0.59	0.73	0.89	1.33	7.50	27.01	
		500	0.66	0.92	1.27	2.15	18.46	74.43	
(c)	4 cores	5	1.00	1.11	1.21	1.62	5.45	15.49	in s
		10	0.99	1.09	1.22	1.59	5.55	16.11	
		20	0.99	1.09	1.23	1.63	5.88	17.20	
		100	1.02	1.13	1.31	1.73	7.82	26.81	
		500	1.10	1.42	1.70	2.62	18.33	72.34	
(d)	$\text{ratio}_{4/1}$	5	2.81	2.21	1.85	1.54	0.98	0.95	/
		10	2.75	2.16	1.84	1.45	0.96	0.89	
		20	2.75	2.16	1.83	1.46	0.95	0.92	
		100	1.62	1.15	1.42	1.30	0.80	0.81	
		500	2.30	1.73	1.39	1.02	0.69	0.69	

Table 1: Mean calculation times (in seconds) needed by `corr2d()` to do a 2D homo-correlation analysis using 1 (a), 2 (b) or 4 (c) processor cores and a ratio (d) calculated from the mean calculation times using 4 and 1 processor cores. The table includes the number of perturbation values m by rows and the number of spectral values n by columns.

The result of the speed test can be seen in Table 1(a)–(c). As one can see from the table the parallelization speeds up the calculation of large 2D correlation spectra while it slows down the calculation for smaller 2D spectra. Table 1(d) illustrates this observation. This observation can be explained by how parallel computing works: For a parallelized calculation every core calculates only a small part of the problem. These small tasks need to be transferred to the calculating cores and later the results of all these small calculations need to be put together to get the result for the parallelized calculation. The time needed by this processes increases the more the original task is split up. Thus, a parallel computation is always a trade-off between speeding up the calculation process and increasing the amount of traffic needed to organize the parallel computation. This trade-off can be seen at the parallel calculations done by `corr2d()`: `corr2d()` needs more time calculating the correlation in parallel when the input matrix is small compared to a serialized calculation with only one processor core. For larger input matrices the effect tips and `corr2d()` is faster using more cores. For small input matrices the correlation speed differences are hardly noticeable (0.8 s vs. 1.4 s for a [500 $\times$ 400] matrix) while the speed differences are much more important if the input matrices get bigger and the calculation times get longer (26.5 s vs. 18.3 s for a [500 $\times$ 4000] matrix).

The number of computation steps necessary also depends on how the input for the correlation integral is calculated. The number of computation steps varies for the FT and the HT

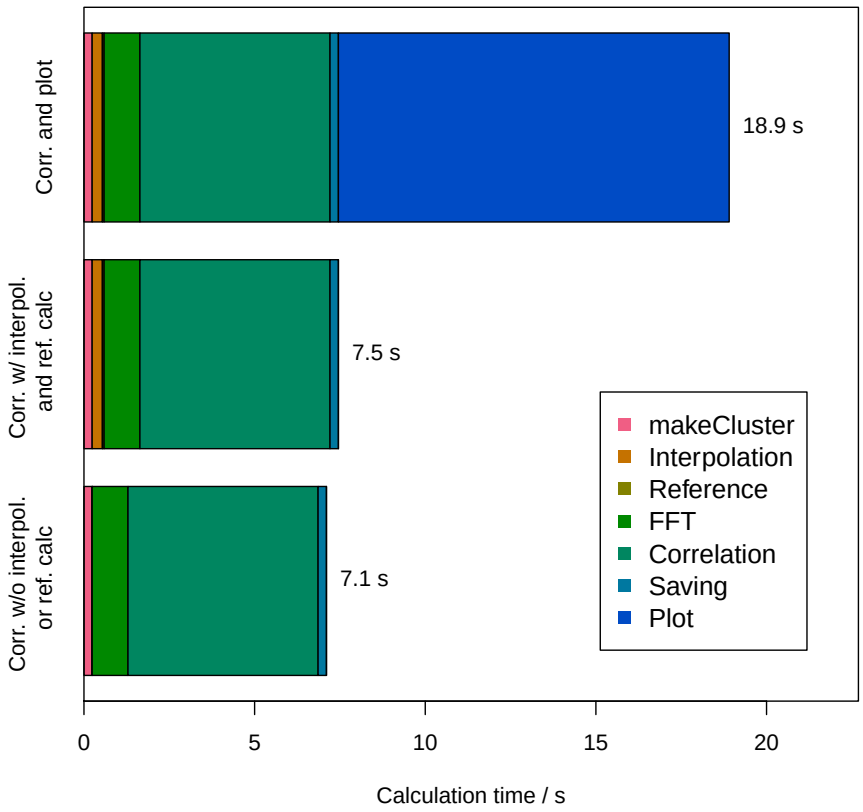

Figure 4: Calculation time used by `corr2d()` to calculate a [4000 × 4000] complex correlation spectrum from a [100 × 4000] input matrix. The colored bars show how much calculation time is used by each single calculation step.

approach. For m discrete input spectra the FFT needs $4 \cdot m \cdot \log_2(m)$ steps (using the Cooley-Tukey algorithm; Cooley and Tukey 1965) while the HT needs m^2 steps. Therefore, the HT is faster for smaller datasets while the FFT is faster for larger datasets. Because calculation speed matters more for larger datasets, `corr2d()` uses the FT approach. For details on the comparison between the FT and the HT approach the reader is referred to Noda *et al.* (2000).

The speed of the calculation drops off significantly if the RAM limit dedicated to R is reached. In this case `corr2d()` needs to save results outside the RAM which takes a large amount of time when compared to saving data inside the RAM. Under Windows there is a memory limit for the R process which might be necessary to increase using `memory.limit()` to overcome this problem.

```
R> simdata <- sim2ddata(4000, seq(0, 10, length.out = 100))
R> library("profr")
R> prof2d <- profr(c(speedtwod <- corr2d(simdata, Time =
+     as.numeric(rownames(simdata)), scaling = 0.5), plot_corr2d(speedtwod)),
+     interval = 0.005)
```

To illustrate how much time is spent in each step, the correlation function `corr2d()` and the plotting function `plot_corr2d()` were profiled using `profr()` from package **profr**. The

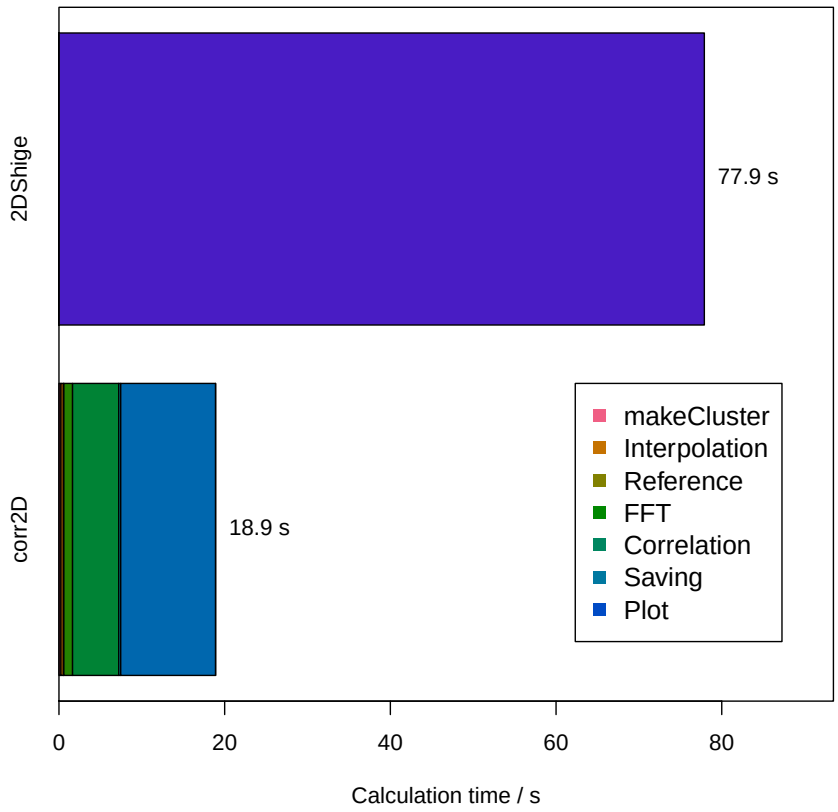

Figure 5: Comparison between **corr2D** and **2DShige**. Calculation and plotting time used to evaluate a [4000 × 4000] complex correlation spectrum from a [100 × 4000] input matrix. The colored bars show how much calculation time is used by each single calculation step.

results can be seen in Figure 4. Most of the calculation time used by `corr2d()` is needed for the parallel matrix multiplication (5.6 s) followed by the FFT (1.1 s). The interpolation process to get 128 perturbation values (0.3 s) and the application of scaling techniques (0.05 s) are faster. The plotting of the resulting synchronous 2D correlation spectrum also needs time (11.4 s) because the complete spectrum has 16 MPixel.

To evaluate the script in a larger scheme we tried to compare the R package **corr2D** to the software **2DShige** (Morita 2005) which is available free of charge from the internet. Because **2DShige** is a stand-alone program the timing of its calculations are not as precise as the timings from the profiling of `corr2d()` and `plot_corr2d()`. The calculation and plotting time used by **2DShige** was estimated 10 times using the Windows task manager. The input matrix was the same simulated [100 × 4000] data matrix used for the profiling.

2DShige took 77.9 s to calculate the [4000 × 4000] complex 2D correlation spectrum and plot the synchronous 2D correlation spectrum. Unfortunately, it is not possible to give calculation times for every calculation step. Comparing the calculation times needed by **2DShige** and **corr2D** it becomes clear that the R package is faster when calculating a large 2D correlation spectrum. Figure 5 illustrates the results. Overall, the profiling of `corr2d()` and `plot_corr2d()` and the speed test done for `corr2d()` prove that it is useful to parallelize the FFT and the matrix multiplication. Other calculation steps (interpolation, reference spec-

trum calculation and scaling) are much faster than the FFT and the matrix multiplication and thus do not need a parallelization. The parallel calculation approach is faster when compared to a serialized approach for large input matrices. The R code also compares favorably to the widespread 2D correlation software 2Dshige in terms of speed and transparency.

4.4. Visualizing 2D correlation spectra

The following section describes the programming and design details considered for the two plotting functions `plot_corr2d()` and `plot_corr2din3d()`. In general the representation of 2D correlation spectra is designed with 2D NMR plots in mind. 2D NMR plots also show 3D data with two frequency axes (the so called chemical shift δ) and contour levels to represent cross peaks. 2D NMR spectroscopy is a common technique in chemistry labs. Thus, the plotting function `plot_corr2d()` tries to mimic the appearance of 2D NMR plots to make the data accessible for beginners in the field of 2D correlation spectroscopy.

Most of the arguments in `plot_corr2d()` are inspired by the arguments in `plot()` or `image()` to allow users already experienced with R a smooth transition. The function `plot_corr2d()` takes an object `Obj` of class '`corr2d`' as produced by the correlation function `corr2d()` and extracts the different parameters needed for depicting the 2D correlation spectra contained in the given object.

Firstly, the function `plot_corr2d()` saves the prior user defined graphics parameters to restore them at the end. Secondly, `plot_corr2d()` also extracts the graphical parameters assigned by the user under the `...` argument. This procedure enables the function to later use these parameters in different plotting functions to adjust the appearance of the entire 2D correlation plot.

```
R> par_old <- par(no.readonly = TRUE)
R> on.exit(options(par(par_old)), add = TRUE)
R> getparm <- list(...)
R> graphparm <- utils::modifyList(par(), getparm)
```

After the extraction of the graphical parameters, the 2D plot function subsets the spectral variable axes `Obj$Wave1` (x-axis) and `Obj$Wave2` (y-axis) to the window range defined by the arguments `xlim` and `ylim`.

```
R> if (is.null(xlim)) {
+    Which1 <- 1:NROW(what)
+  } else {
+    Which1 <- which(xlim[1] < Obj$Wave1 & Obj$Wave1 < xlim[2])
+  }
R> if (is.null(ylim)) {
+    Which2 <- 1:NCOL(what)
+  } else {
+    Which2 <- which(ylim[1] < Obj$Wave2 & Obj$Wave2 < ylim[2])
+  }
```

The 2D plot function uses the `contour()` or `image()` function in a `split.screen()` environment (all from package **graphics**) to generate the 2D plot. The plot device is split into seven

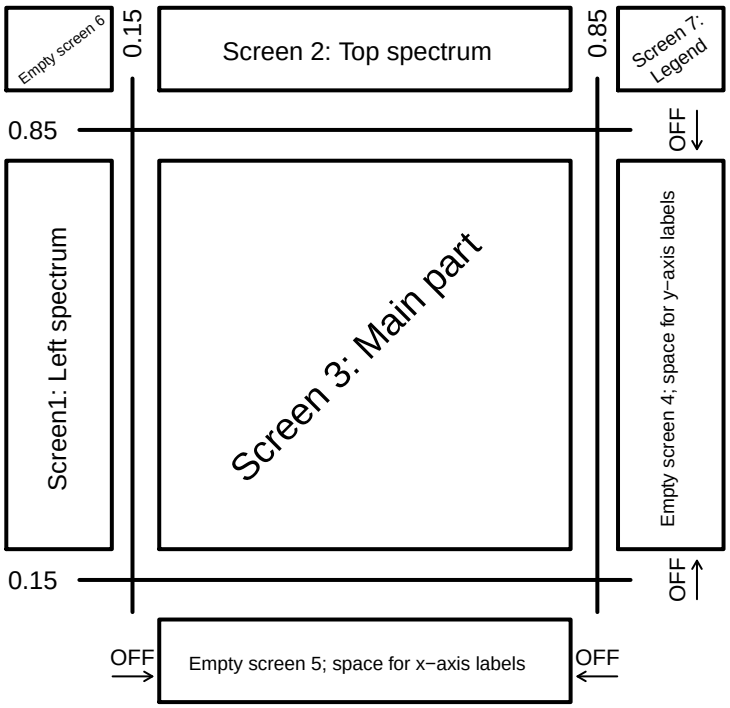

Figure 6: Schematic representation of the seven screens generated and used by `plot_corr2d()` to graphically represent 2D correlation spectra. The offset `OFF` is used during development to adjust the screen ratios.

screens (see Figure 6), which will be filled with two 1D spectra via a simple line plot (Screens 1 and 2), the central 2D plot (Screen 3), generated by the `contour()` or `image()` function with x- and y-axis as well as their labels and the color legend in the top right (Screen 7). The remaining three screens remain (up to now) empty. On exit `plot_corr2d()` closes all but Screen 3, thus further plots, lines and points can be added to the central plot or data can be read out interactively using the `locator()` function. The code snippet shows the creation of the `split.screen()` environment. The offset `OFF` is used during development to adjust the ratio between the main part (Screen 3) and the surrounding screens.

After the creation of the different screens the function `plot_corr2d()` switches to the different screens and fills them with plots, labels and legends. Screens 1 and 2 are used to plot the 1D reference spectra to the top (`specx`) and to the left (`specy`) of the main part in Screen 3. By default the reference spectra `Obj$Ref1` and `Obj$Ref2` are plotted, but the user can assign any data to be plotted on the axes. As x-axis for the two plots the previously subset spectral variable axes `Obj$Wave1[Which1]` and `Obj$Wave2[Which2]` are used to align the 1D spectra with the central 2D plot. Thus, the reference spectra (specified at `specx` and `specy`) must be of the same length as the spectral variable axes. The reference spectrum on the y-axis is rotated by switching the x- and y-axes around as well as by adjusting the new x-values accordingly. The plotting of the reference spectra can be suppressed by setting `specx` and/or `specy NULL`. The arguments defined in `par()` suppress the plotting of any axes or labels, which are not needed because they are added to the axes of the 2D correlation spectrum.

The line width of the 1D plots can be adjusted by assigning the `lwd` argument.

```
R> if (!is.null(specy)) {
+    screen(1)
+    par(xaxt = "n", yaxt = "n", mar = c(0, 0, 0, 0), bty = "n", yaxs = "i")
+    plot.default(x = max(specy[Which2]) - specy[Which2],
+      y = Obj$Wave2[Which2], type = "l", lwd = graphparm$lwd + 1,
+      ann = FALSE)
+  }
R> if (!is.null(specx)) {
+    screen(2)
+    par(xaxt = "n", yaxt = "n", mar = c(0, 0, 0, 0), bty = "n", xaxs = "i")
+    plot.default(x = Obj$Wave1[Which1], y = specx[Which1], type = "l",
+      lwd = graphparm$lwd + 1, ann = FALSE)
+  }
```

The centerpiece of the 2D plot is the 2D correlation spectrum depicted in Screen 3. To get a flexible plotting function `plot_corr2d()` features a lot of control arguments for the main part. In line with the underlying functions `contour()` and `image()` which are used for plotting the 2D spectrum `plot_corr2d()` has a `zlim` argument to define the range of the z-axis. Next the function `plot_corr2d()` builds `N` contour or image levels, which are evenly distributed along a modified z-axis. The levels are calculated using the maximum absolute value among all z-values. The color values for each level are calculated using the levels derived from the modified z-axis. The 2D plot uses the colors dark blue and cyan for negative z-values as well as yellow, red and dark red for positive z-values. The colors are taken from function `designer.colors()` from package **fields**. After the calculation of the color values the "real" contour or image levels are calculated using the borders defined by `zlim`.

This procedure looks unnecessary complicated at first, but the procedure ensures that the most extreme colors (dark blue for negative values and dark red for positive values) are used for the most extreme absolute z-values. The advantage of this approach is that positive and negative levels can be compared with each other by looking at the color code. When using just the number range provided by `zlim` this is not possible. As an example consider z-values ranging from -1 to 9. These values should be plotted with 9 levels. When using the original z-range (-1 to 9) for the calculation of the color code the color dark blue would be assigned to the level at -1 because it is the most extreme negative value and the color dark red would be assigned to the level at 9 because it is the most extreme positive value. By looking at the resulting 2D spectrum one would get the impression that "the dark blue level is as negative as the dark red level is positive" which is clearly not the case. By using the range defined by the most extreme absolute z-value (-9 to 9) for the calculation of the color code the color dark blue gets assigned to the (non-existent) level -9, whereas the color dark red still gets assigned to the level 9. The z-value -1 will now be plotted using the color cyan, which indicates a small negative value when compared to the level at 9 which is depicted in dark red.

In addition to the already discussed features, `plot_corr2d()` always uses an odd number of contour or image levels. The odd number of contour levels leads to a symmetric distribution of the contour and image levels in an asynchronous 2D homo-correlation spectrum. Asynchronous homo-correlation spectra are skew-symmetric regarding the diagonal and thus the absolute values of the positive and negative correlation intensities are always identical. The

odd number of contour or image levels also accomplishes that positive and negative values are represented by the same number of levels (in reference to the most extreme absolute z-value) and that the level around 0 can always be transparent to suppress the plotting of noise. Therefore, it can happen that no blue (or no yellow-red) levels are drawn, because the positive (or negative) values are much larger than the negative (or positive) values that the equal spacing of the levels leads to an agglomeration of all negative (or positive) values inside the level around 0, which will be transparent. To circumvent this problem the number of contour levels has to be increased.

The argument `Cutout` can be used to set more levels than the central level around 0 transparent. This argument can be used to simplify 2D correlation spectra when starting the interpretation of a new 2D correlation spectrum where one would like to have a look at the strongest correlation first and then gradually work through smaller correlations. Care should be taken when using the `Cutout` argument as it can also be used to erase unwanted signals and thus create unrealistic 2D correlation spectra. The code snippet illustrates how this is implemented in R code.

```
R> screen(3)
R> if (is.null(zlim)) {
+    zlim <- range(what[Which1, Which2])
+  }
R> if (N%%2 == 0) {
+    N <- N + 1
+  }
R> Where <- seq(-max(abs(zlim)), max(abs(zlim)), length.out = N)
R> if (is.null(Cutout)) {
+    OM <- which(Where < 0)
+    OP <- which(Where > 0)
+  } else {
+    OM <- which(Where <= Cutout[1])
+    OP <- which(Where >= Cutout[2])
+  }
R> COL <- rep("transparent", length(Where))
R> COL[OM] <- fields::designer.colors(col = c("darkblue", "cyan"),
+    n = length(OM))
R> COL[OP] <- fields::designer.colors(col = c("yellow", "red", "darkred"),
+    n = length(OP))
R> COL[(N + 1)/2] <- "transparent"
R> COL <- COL[which(zlim[1] < Where & Where < zlim[2])]
R> Where <- seq(zlim[1], zlim[2], length.out = length(COL))
```

After the calculation of the color code and the contour or image levels the actual 2D correlation spectrum is drawn either by `contour()` or by `image()` (both from package **graphics**). The preferred function can be selected from the logical argument `Contour` in `plot_corr2d()`. Both functions use the subset matrix `what[Which1, Which2]` as z-values and the subset vectors `Obj$Wave1[Which1]` and `Obj$Wave2[Which2]` as respective x- or y-values. By default `plot_corr2d()` plots the synchronous 2D correlation spectrum as specified by `Re(Obj$FT)`. The normal axes and axis labels are suppressed to allow for a flexible definition by the user.

After the plotting of the 2D plot, a white line gets drawn across the 2D spectrum which highlights the main diagonal in a 2D homo-correlation spectrum. Afterwards a box is drawn around the 2D plot, the x- and y-axis are added and the x- and y-axis labels get added as specified by arguments `xlab` and `ylab`. The axes of the 2D plot can be suppressed by using the argument `axes` in `plot_corr2d()`. The input arguments `lwd` and `cex` can be used to adjust the line width of the axes and the surrounding box as well as the size of the axes labels.

```
R> par(xaxt = "n", yaxt = "n", mar = c(0, 0, 0, 0), bty = "n",
+    xaxs = "i", yaxs = "i")
R> if (Contour == TRUE) {
+    graphics::contour(x = Obj$Wave1[Which1], y = Obj$Wave2[Which2],
+      z = what[Which1, Which2], col = COL, levels = Where, zlim = zlim,
+      drawlabels = FALSE, ...)
+  } else {
+    graphics::image(x = Obj$Wave1[Which1], y = Obj$Wave2[Which2],
+      z = what[Which1, Which2], col = COL, xlab = "", ylab = "",
+      zlim = zlim, ...)
+  }
R> abline(a = 0, b = 1, col = rgb(red = 1, green = 1, blue = 1,
+    alpha = 0.5), lwd = graphparm$lwd)
R> par(xpd = NA, xaxt = "s", yaxt = "s", xaxs = "i", yaxs = "i",
+    cex = graphparm$cex, mar = c(0, 0, 0, 0))
R> box(which = "figure", lwd = graphparm$lwd)
R> if ((axes == 1) | (axes == 3)) {
+    axis(side = 1, lwd = graphparm$lwd)
+  }
R> if ((axes == 2) | (axes == 3)) {
+    axis(side = 4, las = 2, lwd = graphparm$lwd)
+  }
R> mtext(side = 1, xlab, line = 3.5, cex = graphparm$cex * 1.3,
+    lwd = graphparm$lwd)
R> mtext(side = 4, ylab, line = 3.5, cex = graphparm$cex * 1.3,
+    lwd = graphparm$lwd)
```

The color code legend is plotted in Screen 7. For the legend the function `image.plot()` from package **fields** is used. Most arguments defined in `image.plot()` by `plot_corr2d()` are for setting the margins of the plot and arranging the number legend. The color code legend in `plot_corr2d()` has two number values written next to it specifying the 10% and 90% quantile of the plotted z-values. The legend can be turned off by setting the argument `Legend` in `plot_corr2d()` to `FALSE`. The specified `pin` parameter is a small hack to avoid an error produced by `image.plot()` in combination with the `split.screen()` environment. The argument `cex.axis` can be defined at the input to adjust the size of the legend labels. After finishing all plotting tasks `plot_corr2d()` changes back to the main 2D plot in Screen 3, closes all but Screen 3 and restores the old `par` parameters. By keeping Screen 3 active the user can add points or lines to the central screen and can read out data interactively by using the function `locator()`.

```
R> if (Legend ==  TRUE) {
+    screen(7)
+    par(pin = abs(par()$pin))
+    if (Contour == TRUE) {
+      fields::image.plot(z = what[Which1, Which2], legend.only = TRUE,
+        smallplot = c(0.15, 0.3, 0.2, 0.8), col = COL,
+        axis.args = list(at = quantile(Where, prob = c(0.1, 0.9)),
+          labels = format(x = quantile(Where, prob = c(0.1, 0.9)),
+          digit = 2, scientific = TRUE), cex.axis = graphparm$cex.axis),
+        zlim = zlim, graphics.reset = TRUE)
+    } else {
+      fields::image.plot(z = what[Which1, Which2], legend.only = TRUE,
+        smallplot = c(0.15, 0.3, 0.2, 0.8), col = COL,
+        axis.args = list(at = range(what[Which1, Which2]),
+          labels = format(x = range(what[Which1, Which2]),
+          digits = 2, scientific = TRUE), cex.axis = graphparm$cex.axis),
+        graphics.reset = TRUE)
+    }
+  }
R> screen(3, new = FALSE)
R> close.screen(c(1, 2, 4, 5, 6, 7))
```

The 3D plotting function `plot_corr2din3d()` works a little bit different than the 2D plotting function `plot_corr2d()` because it is meant for creating impressive 3D figures of 2D correlation spectra and not so much for scientific exact representation of 2D correlation spectra. Thus, `plot_corr2din3d()` takes a matrix `Mat` (for example the synchronous 2D correlation spectrum `Re(Obj$FT)` from an object of class '`corr2d`'), builds arbitrary x- and y-axes, plots the 3D surface using the function `drape.plot()` from package **fields** and adds user defined 1D spectra to the x- and y-axis.

The creation of arbitrary x- and y-axes is simply done by using the number of rows and columns in matrix `Mat`. To reduce the computational demand when plotting the 3D surface the function `plot_corr2din3d()` has the argument `reduce`. The argument `reduce` allows the user to resample the input matrix. The resampling is done using the function `resample()` from package **mmand**. When the matrix is resampled the x- and y-axes are also resampled to match the new matrix. The code snippet shows the axis generation and the resampling.

```
R> par_old <- par(no.readonly = TRUE)
R> on.exit(options(par(par_old)), add = TRUE)
R> x <- 1:NROW(Mat)
R> y <- 1:NCOL(Mat)
R> if (!is.null(reduce)) {
+    Which.x <- (1:length(x))[which(1:length(x)%%reduce == 0)]
+    Which.y <- (1:length(y))[which(1:length(y)%%reduce == 0)]
+    Mat <- mmand::resample(x = Mat, points =
+      list(x = x[Which.x], y = y[Which.y]), kernel = mmand::boxKernel())
+    x <- x[Which.x]
+    y <- y[Which.y]
+  }
```

If no color palette is specified at argument `Col` than `plot_corr2din3d()` builds the color specification from the z-values inside the matrix `Mat`. `Breaks` describes the numerical divisions of the color scale, which are used by `drape.plot()` for plotting the 3D surface. If no `zlim` argument, which describes the z-axis of the 3D plot, is specified than the `zlim` argument is also built from the input matrix. The HCL (hue-chroma-luminance) color space is superior to the widespread RGB (red-green-blue) color space (Zeileis, Hornik, and Murrell 2009; Stauffer, Mayr, Dabernig, and Zeileis 2015). Thus, `plot_corr2din3d()` uses the diverging HCL color palette from package **colorspace** (Zeileis *et al.* 2019) as default value.

```
R> N <- length(Col)
R> Zero <- 0
R> Max <- max(Mat)
R> Min <- min(Mat)
R> if (N%%2 == 0) {
+    Breaks <- c(seq(Min, Zero, length.out = round(N / 2, 0) + 1),
+      seq(Zero, Max, length.out =
+        round(N / 2, 0) + 1)[2:(round(N / 2, 0) + 1)])
+  } else {
+    Breaks <- c(seq(Min, Zero, length.out =
+      round(N / 2, 0) + 2)[1:(round(N / 2, 0) + 1)], seq(Zero, Max,
+        length.out = round(N / 2, 0) + 2)[2:(round(N / 2, 0) + 2)])
+  }
R> if (is.null(zlim)) {
+    zlim <- range(Mat, na.rm = TRUE)
+  }
```

All previously defined parameters are then fused together inside `drape.plot()` which plots the 3D surface for the first time. Arguments specified at `...` in `plot_corr2din3d()` are handed over to `drape.plot()`. The most important arguments are the two viewing angles `theta` (x-y rotation) and `phi` (z-rotation) as well as the argument `border` which takes a color and adds a grid in that color to the 3D surface.

`drape.plot()` returns a projection matrix which can be used to add a 2D projection of the 3D surface to the bottom of the plot. Unfortunately, `drape.plot()` has to be executed once to get the projection matrix. If the 2D surface is simply added to the 3D plot it may overlap with the 3D surface depending on the viewing angles. To circumvent this problem `plot_corr2din3d()` first executes `drape.plot()`, then adds the 2D surface and in the end executes `drape.plot()` once more to overlay the 2D projection with the 3D surface. The addition of a 2D projection can be specified at argument `projection`. The coordinates for the 2D projection are calculated by the function `trans3d()` from **grDevices** and the 2D plot is drawn by `polygon()`.

```
R> if (projection == TRUE) {
+    WW <- fields::drape.plot(x = x, y = y, z = Mat, col = Col,
+      breaks = Breaks, zlim = zlim, ...)
+    COL <- fields::drape.color(z = Mat, col = Col,
+      zlim = zlim, breaks = Breaks)$color.index
```

```
+    for (i in 2:NROW(Mat)) {
+      for (j in 2:NCOL(Mat)) {
+        Points <- grDevices::trans3d(
+          y = y[c(j - 1, j, j, j - 1, j - 1)],
+          x = x[c(i - 1, i - 1, i, i, i - 1)],
+          z = rep(zlim[1], length(5)), pmat = WW)
+        polygon(Points$x, Points$y, border = NA, col = COL[i - 1, j - 1])
+      }
+    }
+    par(new = TRUE)
+    fields::drape.plot(x = x, y = y, z = Mat, col = Col,
+      breaks = Breaks, zlim = zlim, ...)
+  } else {
+    WW <- fields::drape.plot(x = x, y = y, z = Mat, col = Col,
+      breaks = Breaks, zlim = zlim, ...)
+  }
```

After the 3D surface and its 2D projection are drawn `plot_corr2din3d()` can add custom spectra to the x- and y-axis. For this reason a new x- and/or y-axis gets calculated where the length is equal to the length of the spectra `specx` or `specy`. The `x`- and `y`-arguments for the function `lines()` are calculated using the aforementioned projection matrix `WW` and once again the function `trans3d()`. The scaling factors `scalex` and `scaley` are real numbers and are used to scale the spectra. Positive values at `scalex` and `scaley` ensure that the spectra are plotted inside the box of the 3D plot (where the 2D projection is also located) whereas negative values place the spectra on the outside of the box. After all plots are done the function `plot_corr2din3d()` restores the old `par` parameters.

```
R> if (!is.null(specx)) {
+    if (is.null(scalex)) {
+      scalex <- 1
+    }
+    X <- seq(min(x), max(x), length.out = length(specx))
+    Points.x <- grDevices::trans3d(x = X, y = min(y) + scalex * specx,
+      z = rep(zlim[1], length(X)), pmat = WW)
+    lines(x = Points.x$x, Points.x$y, lwd = 2)
+  }
R> if (!is.null(specy)) {
+    if (is.null(scaley)) {
+      scaley <- 1
+    }
+    Y <- seq(min(y), max(y), length.out = length(specy))
+    Points.y <- grDevices::trans3d(y = Y, x = max(x) - scaley * specy,
+      z = rep(zlim[1], length(Y)), pmat = WW)
+    lines(x = Points.y$x, Points.y$y, lwd = 2)
+  }
```

5. Conclusion and outlook

In this paper we demonstrated how to use our R package **corr2D** to calculate and visualize 2D correlation spectra from artificial and real data. The implementation into R offers the advantage to preprocess, correlate, visualize and further analyze data in one open-source program, which was not easily possible before and should help to make the technique of 2D correlation spectroscopy more accessible. For the calculation of the complex correlation spectra we use a parallel fast Fourier transformation approach, which can be adjusted by the user to best fit their needs. The paper also featured a detailed look on the inner workings of our package to help the user understand what is going on during the calculation.

In the future we plan on implementing the Hilbert transformation approach into the package as well as advanced 2D correlation spectroscopy techniques like moving-window 2D correlation spectroscopy, 2D codistribution spectroscopy or double 2D correlation spectroscopy. The implementation of advanced techniques into the package will help to make the package more interesting for seasoned spectroscopists and to make new 2D correlation approaches available to a broad user base. Furthermore we also aim to add a graphical user interface via **shiny** (Chang, Cheng, Allaire, Xie, and McPherson 2019) to the package, which will allow users without any programming experience to use the package as well.

Acknowledgments

We would like to thank the Deutsche Forschungsgemeinschaft (DFG) for funding the priority program SPP 1568 "Design and Generic Principles of Self-healing Materials" and the project PO563/25-2 therein. We would also like to thank Hadley Wickham for his very good tutorials on how to create R packages and for his helpful R package **devtools** (Wickham, Hester, and Chang 2019). Additionally we also appreciate the helpful comments of the three editors Achim Zeileis, Bettina Grün, and Edzer Pebesma.

References

Adler D, Murdoch D (2019). "**rgl**: 3D Visualization Using OpenGL." R package version 0.100.24, URL https://CRAN.R-project.org/package=rgl.

Barton FE, de Haseth JA, Himmelsbach DS (2006). "The Use of Two-Dimensional Correlation Spectroscopy to Characterize Instrumental Differences." *Journal of Molecular Structure*, **799**(1–3), 221–225. doi:10.1016/j.molstruc.2006.02.063.

Bruker Corporation (2016). "**OPUS**." Version 7.5, URL https://www.bruker.com/.

Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J (2019). "**shiny**: Web Application Framework for R." R package version 1.3.2, URL https://CRAN.R-project.org/package=shiny.

Clayden J (2019). "**mmand**: Mathematical Morphology in Any Number of Dimensions." R package version 1.5.4, URL https://CRAN.R-project.org/package=mmand.

Cooley JW, Tukey JW (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series." *Mathematics of Computation*, **19**(90), 297–301. doi:10.1090/s0025-5718-1965-0178586-1.

Czarnecki MA (1998). "Interpretation of Two-Dimensional Correlation Spectra: Science or Art?" *Applied Spectroscopy*, **52**(12), 1583–1590. doi:10.1366/0003702981943086.

Ferenc (2011). "**MIDAS 2010**." MATLAB Central File Exchange, URL http://www.mathworks.com/matlabcentral/fileexchange/32384-midas-2010.

Geitner R, Fritzsch R, Bocklitz T (2019). "**corr2D**: Implementation of 2D Correlation Analysis in R." R package version 0.4.0, URL https://CRAN.R-project.org/package=corr2D.

Geitner R, Kötteritzsch J, Siegmann M, Bocklitz T, Hager M, Schubert US, Gräfe S, Dietzek B, Schmitt M, Popp J (2015). "Two-Dimensional Raman Correlation Spectroscopy Reveals Molecular Structural Changes During Temperature-Induced Self-Healing in Polymers Based on the Diels-Alder Reaction." *Physical Chemistry Chemical Physics*, **17**(35), 22587–22595. doi:10.1039/c5cp02151k.

Geitner R, Kötteritzsch J, Siegmann M, Fritzsch R, Bocklitz T, Hager M, Schubert US, Gräfe S, Dietzek B, Schmitt M, Popp J (2016). "Molecular Self-Healing Mechanisms Between C60-Fullerene and Anthracene Unveiled by Raman and Two-Dimensional Correlation Spectroscopy." *Physical Chemistry Chemical Physics*, **18**(27), 17973–17982. doi:10.1039/c6cp03464k.

Kane M, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(14), 1–19. doi:10.18637/jss.v055.i14.

López-Díez EC, Winder CL, Ashton L, Currie F, Goodacre R (2005). "Monitoring the Mode of Action of Antibiotics Using Raman Spectroscopy: Investigating Subinhibitory Effects of Amikacin on Pseudomonas Aeruginosa." *Analytical Chemistry*, **77**(9), 2901–2906. doi:10.1021/ac048147m.

Mersmann O (2018). "**microbenchmark**: Accurate Timing Functions." R package version 1.4-6., URL https://CRAN.R-project.org/package=microbenchmark.

Microsoft, Weston S (2017). "**foreach**: Provides Foreach Looping Construct for R." R package version 1.4.4, URL https://CRAN.R-project.org/package=foreach.

Microsoft Corporation, Weston S (2018). "**doParallel**: Foreach Parallel Adaptor for the **parallel** Package." R package version 1.0.14, URL https://CRAN.R-project.org/package=doParallel.

Morita S (2005). "**2DShige**." URL https://sites.google.com/view/shigemorita/home/2dshige.

Noda I (1986). "Two-Dimensional Infrared (2D IR) Spectroscopy." *Bulletin of American Physical Society*, **31**, 520–524.

Noda I (1989). "Two-Dimensional Infrared Spectroscopy." *Journal of the American Chemical Society*, **111**(21), 8116–8118. doi:10.1021/ja00203a008.

Noda I (1990). "Two-Dimensional Infrared (2D IR) Spectroscopy: Theory and Applications." *Applied Spectroscopy*, **44**(4), 550–561. doi:10.1366/0003702904087398.

Noda I (1993). "Generalized Two-Dimensional Correlation Method Applicable to Infrared, Raman, and Other Types of Spectroscopy." *Applied Spectroscopy*, **47**(9), 1329–1336. doi:10.1366/0003702934067694.

Noda I (2003). "Two-Dimensional Correlation Analysis of Unevenly Spaced Spectral Data." *Applied Spectroscopy*, **57**(8), 1049–1051. doi:10.1366/000370203322259039.

Noda I (2004). "Graphical Representation of Two-Dimensional Correlation in Vector Space." *Vibrational Spectroscopy*, **36**(2), 261–266. doi:10.1016/j.vibspec.2004.01.005.

Noda I (2006). "Cyclical Asynchronicity in Two-Dimensional (2D) Correlation Spectroscopy." *Journal of Molecular Structure*, **799**(1–3), 41–47. doi:10.1016/j.molstruc.2005.12.060.

Noda I (2008). "Scaling Techniques to Enhance Two-Dimensional Correlation Spectra." *Journal of Molecular Structure*, **883-884**, 216–227. doi:10.1016/j.molstruc.2007.12.026.

Noda I (2010). "Double Two-Dimensional Correlation Analysis – 2D Correlation of 2D Spectra." *Journal of Molecular Structure*, **974**(1–3), 108–115. doi:10.1016/j.molstruc.2009.11.048.

Noda I (2012). "Close-up View on the Inner Workings of Two-Dimensional Correlation Spectroscopy." *Vibrational Spectroscopy*, **60**, 146–153. doi:10.1016/j.vibspec.2012.01.006.

Noda I (2014a). "Frontiers of Two-Dimensional Correlation Spectroscopy. Part 1. New Concepts and Noteworthy Developments." *Journal of Molecular Structure*, **1069**, 3–22. doi:10.1016/j.molstruc.2014.01.025.

Noda I (2014b). "Frontiers of Two-Dimensional Correlation Spectroscopy. Part 2. Perturbation Methods, Fields of Applications, and Types of Analytical Probes." *Journal of Molecular Structure*, **1069**, 23–49. doi:10.1016/j.molstruc.2014.01.016.

Noda I (2016a). "Quadrature Two-Dimensional Correlation Spectroscopy (Q-2DCOS)." *Journal of Molecular Structure*, **1124**, 42–52. doi:10.1016/j.molstruc.2016.01.090.

Noda I (2016b). "Techniques Useful in Two-Dimensional Correlation and Codistribution Spectroscopy (2DCOS and 2DCDS) Analyses." *Journal of Molecular Structure*, **1124**, 29–41. doi:10.1016/j.molstruc.2016.01.089.

Noda I (2016c). "Two-Dimensional Correlation Spectroscopy (2DCOS) Analysis of Polynomials." *Journal of Molecular Structure*, **1124**, 53–60. doi:10.1016/j.molstruc.2016.01.091.

Noda I, Dowrey AE, Marcoli C, Story GM, Ozaki Y (2000). "Generalized Two-Dimensional Correlation Spectroscopy." *Applied Spectroscopy*, **54**(7), 236–248. doi:10.1366/0003702001950454.

Nychka D, Furrer R, Paige J, Sain S (2019). "**fields**: Tools for Spatial Data." R package version 9.8-3, URL https://CRAN.R-project.org/package=fields.

OriginLab (2019). "**Origin**." Version 2019b, URL http://originlab.com/.

Park Y, Noda I, Jung YM (2016). "Novel Developments and Applications of Two-Dimensional Correlation Spectroscopy." *Journal of Molecular Structure*, **1124**, 11–28. doi:10.1016/j.molstruc.2016.01.028.

R Core Team (2019). "R: A Language and Environment for Statistical Computing." URL https://www.R-project.org/.

Šašić S, Muszynski A, Ozaki Y (2000a). "A New Possibility of the Generalized Two-Dimensional Correlation Spectroscopy. 1. Sample-Sample Correlation Spectroscopy." *The Journal of Physical Chemistry A*, **104**(27), 6380–6387. doi:10.1021/jp000510f.

Šašić S, Muszynski A, Ozaki Y (2000b). "A New Possibility of the Generalized Two-Dimensional Correlation Spectroscopy. 2. Sample-Sample and Wavenumber-Wavenumber Correlations of Temperature-Dependent Near-Infrared Spectra of Oleic Acid in the Pure Liquid State." *The Journal of Physical Chemistry A*, **104**(27), 6388–6394. doi:10.1021/jp0005118.

Šašić S, Muszynski A, Ozaki Y (2001). "New Insight into the Mathematical Background of Generalized Two-Dimensional Correlation Spectroscopy and the Influence of Mean Normalization Pretreatment on Two-Dimensional Correlation Spectra." *Applied Spectroscopy*, **55**(3), 343–349. doi:10.1366/0003702011951759.

Shinzawa H, Morita SI, Awa K, Okada M, Noda I, Ozaki Y, Sato H (2009). "Multiple Perturbation Two-Dimensional Correlation Analysis of Cellulose by Attenuated Total Reflection Infrared Spectroscopy." *Applied Spectroscopy*, **63**(5), 501–506. doi:10.1366/000370209788346977.

Spegazzini N, Siesler HW, Ozaki Y (2012). "Sequential Identification of Model Parameters by Derivative Double Two-Dimensional Correlation Spectroscopy and Calibration-Free Approach for Chemical Reaction Systems." *Analytical Chemistry*, **84**(19), 8330–8339. doi:10.1021/ac301867u.

Stauffer R, Mayr GJ, Dabernig M, Zeileis A (2015). "Somewhere Over the Rainbow: How to Make Effective Use of Colors in Meteorological Visualizations." *Bulletin of the American Meteorological Society*, **96**(2), 203–216. doi:10.1175/bams-d-13-00155.1.

The MathWorks Inc (2016). *MATLAB – The Language of Technical Computing, Version R2016a*. Natick. URL http://www.mathworks.com/products/matlab/.

Thomas M, Richardson HH (2000). "Two-Dimensional FT-IR Correlation Analysis of the Phase Transitions in a Liquid Crystal, 4'-n-Octyl-4-Cyanobiphenyl (8CB)." *Vibrational Spectroscopy*, **24**(1), 137–146. doi:10.1016/s0924-2031(00)00086-2.

Wickham H (2018). "**profr**: An Alternative Display for Profiling Information." R package version 0.3.3, URL https://CRAN.R-project.org/package=profr.

Wickham H, Hester J, Chang W (2019). "**devtools**: Tools to Make Developing R Packages Easier." R package version 2.0.2, URL https://CRAN.R-project.org/package=devtools.

Yu ZW, Liu J, Noda I (2003). "Effect of Noise on the Evaluation of Correlation Coefficients in Two-Dimensional Correlation Spectroscopy." *Applied Spectroscopy*, **57**(12), 1605–1609. doi:10.1366/000370203322640251.

Zeileis A, Fisher JC, Hornik K, Ihaka R, McWhite CD, Murrell P, Stauffer R, Wilke CO (2019). "**colorspace**: A Toolbox for Manipulating and Assessing Colors and Palettes." *arXiv 1903.06490*, arXiv.org E-Print Archive. URL http://arxiv.org/abs/1903.06490.

Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis*, **53**(9), 3259–3270. doi:10.1016/j.csda.2008.11.033.

Affiliation:

Robert Geitner, Robby Fritzsch
Institute for Physical Chemistry and Abbe Center of Photonics
Faculty of Chemistry and Geography
Friedrich Schiller University Jena
Helmholtzweg 4, 07743 Jena, Germany

Jürgen Popp, Thomas W. Bocklitz
Institute for Physical Chemistry and Abbe Center of Photonics
Faculty of Chemistry and Geography
Friedrich Schiller University Jena
Helmholtzweg 4, 07743 Jena, Germany
and
Leibniz Institute of Photonic Technology (IPHT) Jena
Albert-Einstein-Str. 9, 07745 Jena, Germany
E-mail: thomas.bocklitz@uni-jena.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2019, Volume 90, Issue 3 *Submitted:* 2016-10-27
doi:10.18637/jss.v090.i03 *Accepted:* 2018-03-08