



# Deriving principal channel metrics from bank and long-profile geometry with the R package cmgo

Antonius Golly and Jens M. Turowski

German Research Centre for Geosciences (GFZ), Telegrafenberg, 14473 Potsdam, Germany

*Correspondence to:* Antonius Golly (antonius.golly@gmail.com)

Received: 10 May 2017 – Discussion started: 16 May 2017

Revised: 14 August 2017 – Accepted: 21 August 2017 – Published: 14 September 2017

**Abstract.** Landscape patterns result from landscape forming processes. This link can be exploited in geomorphological research by reversely analyzing the geometrical content of landscapes to develop or confirm theories of the underlying processes. Since rivers represent a dominant control on landscape formation, there is a particular interest in examining channel metrics in a quantitative and objective manner. For example, river cross-section geometry is required to model local flow hydraulics, which in turn determine erosion and thus channel dynamics. Similarly, channel geometry is crucial for engineering purposes, water resource management, and ecological restoration efforts. These applications require a framework to capture and derive the data. In this paper we present an open-source software tool that performs the calculation of several channel metrics (length, slope, width, bank retreat, knickpoints, etc.) in an objective and reproducible way based on principal bank geometry that can be measured in the field or in a GIS. Furthermore, the software provides a framework to integrate spatial features, for example the abundance of species or the occurrence of knickpoints. The program is available at <https://github.com/AntoniusGolly/cmgo> and is free to use, modify, and redistribute under the terms of the GNU General Public License version 3 as published by the Free Software Foundation.

## 1 Introduction

Principal channel metrics, for example channel width or gradient, convey immanent information that can be exploited for geomorphological research (Wobus et al., 2006; Cook et al., 2014) or engineering purposes (Pizzuto, 2008). For example, a snapshot of the current local channel geometry can provide an integrated picture of the processes leading to its formation if examined in a statistically sound manner (Ferrer-Boix et al., 2016). Repeated surveys, as time series of channel gradients, can reveal local erosional characteristics that sharpen our understanding of the underlying processes and facilitate, inspire, and motivate further research (Milzow et al., 2006). However, these geometrical measures are not directly available. Typically, the measurable metrics are limited to the position of features, such as the channel bed or the water surface, and the water flow path or thalweg in two- or three-dimensional coordinates. The data can either be collected during field surveys with GPS or total stations or through remote sensing with the need for post-processing,

for example in a GIS (geographical information system). To effectively generate channel metrics such as channel width, an objective and reproducible processing of the geometric data is required, especially when analyzing the evolution of channel metrics over time. For river scientists and engineers, a convenient processing tool should incorporate a scale-free approach applicable to a broad spectrum of environments. It should be easy to access, use, and modify, and it should generate output data that can be integrated in further statistical analysis. Here, we present a new algorithm that meets these requirements and describe its implementation in the R package *cmgo* (<https://github.com/AntoniusGolly/cmgo>). The package derives a reference (centerline) of one or multiple given channel shapes and calculates channel length, local and average channel widths, local and average slopes, knickpoints based on a scale-free approach (Zimmermann et al., 2008), local and average bank retreats, and the distances from the centerline. It also allows for the projection of additional spatial metrics to the centerline.

**Table 1.** Overview of existing products; <sup>1</sup> the two values indicate free use of framework (first) and plugin (second value). <sup>2</sup> A product is considered free to modify if users can access and edit the source code and a license explicitly allows users to do so. <sup>3</sup> A product is considered a full-stack solution if it performs all steps from the bank geometry to the derived channel metrics. <sup>4</sup> This publication. <sup>5</sup> No information could be gathered.

Name of the tool	Platform	Data format	Last updated	Free to use <sup>1</sup>	Free to modify <sup>2</sup>	Configurable	Full-stack solution <sup>3</sup>	Scientific reference	Note
cmgo (this paper)	R	vector	Jul 2017	yes, yes	yes	yes	yes	yes <sup>4</sup>	
River Width Calculator	ArcMap	raster	Jun 2013	no, yes	no	no	no	yes	– single, average value for a stream
Perpendicular Transects	ArcMap	vector	Dec 2014	no, yes	yes	limited, no smoothing	no	no	– weak output on non-smooth centerlines
Channel Migration Toolbox	ArcMap	vector	Oct 2014	no, yes	no	limited	no	yes	– fails silently
RivEX	ArcMap	vector	Feb 2017	no, no	no	yes	no	no	– works only on demo data
HEC-GeoRAS	ArcMap	raster	Jul 2017	no, yes	no	yes	no	no	– only verified until ArcMap 10.2
Polygon to Centerline	ArcMap	vector	Nov 2016	no, yes	no	limited, no smoothing	no	no	– weak output for high-resolution bank geometry
Fluvial Corridor Toolbox	ArcMap	vector	Jan 2016	no, yes	no	yes	no	yes	– cannot be applied on the raw data, requires pre-vectorization of channel features
Stream Restoration Toolbox	ArcMap	vector	<sup>5</sup>	no, yes	no	very limited	no	no	– limited functionality
RivWidth	IDL	raster	May 2013	no, yes	yes	<sup>5</sup>	<sup>5</sup>	yes	– highly unstable
DSAS	ArcMap	vector	Dec 2012	no, yes	no	yes	no	yes	– limited access due to IDL license
AMBUR	R	vector	Jun 2014	yes, yes	yes	limited	no	yes	– primarily designed for coastlines
RivMAP	MATLAB	raster	Apr 2017	no, yes	yes	yes	limited	yes	– no multi-temporal analyses allowed
									– primarily for large-scale river systems
									– fails silently on errors

## 2 Literature review

Computer-aided products for studying rivers have a long tradition, and solutions for standardized assessments include many disciplines, for example for assessing the ecological status of rivers (Asterics, 2013) or characterizing heterogeneous reservoirs (Lopez et al., 2009). There are also numerous efforts to derive principal channel metrics from remote or in situ measurements of topography or directly from features such as channel banks. Available products, which we review in detail (Table 1), are helpful for many scientific applications and are used by a large community. However, they often do not provide the degree of independency, transparency or functionality that is necessary to fit the versatile requirements of academic or applied research, and thus the call for software solutions remains present (Amit, 2015). The currently available solutions can be separated into two groups: extensions for GIS applications and extensions for statistical programming languages. The first group incorporates programs that are published as extensions for the proprietary GIS software ArcMap (ESRI, 2017), which are generally not open

source and are thus lacking accessibility and often transparency and modifiability. Furthermore, the individual solutions lack functionality. For example, the River Width Calculator (Mir et al., 2013) calculates the average width of a given river (single value) without providing spatially resolved information. The toolbox Perpendicular Transects (Ferreira, 2014) is capable of deriving channel transects locally, which are generally suitable for calculating the width. However, the required centerline to which the orthogonals are computed is not generated within the tool itself. Thus, the tool does not represent a full-stack solution. Similarly, the Channel Migration Toolbox (Legg et al., 2014), RivEX (Hornby, 2017), and HEC-GeoRAS (Ackerman, 2011) require prerequisite products – a centerline – to compute transects and calculate the width. A centerline could be created with the toolbox Polygon to Centerline (Dilts, 2015), but manual post-processing is required to ensure that lines connect properly. Further, the details of the algorithm are poorly documented and intermediate results are not accessible, making it difficult to evaluate the data quality. Apart from this, all of these products are de-

pendent on commercial software, are bound to a graphical user interface (not scriptable), and cannot be parameterized to a high degree.

The second group of solutions represents extensions for statistical scripting languages. The full-stack solution RivWidth (Pavelsky and Smith, 2008) is written as a plugin for IDL, a data language with restricted usage. The program requires two binary raster masks, a channel mask, and a river mask, which need to be generated in a pre-processing step using, for example, a GIS. Bank geometry obtained from direct measurements, for example from GPS surveys, does not represent adequate input. As a result of the usage of pixel-based data, which in the first place does not properly represent the nature of the geometrical data, computationally intensive transformations are necessary and result in long computation times (the authors describe up to 1 h for their example). More importantly, the centerline position depends on the resolution of the input rasters and is thus scale dependent. Good results can only be obtained when the pixel size is at least an order of magnitude smaller than the channel width. The MATLAB toolbox RivMAP also works with raster data. It is well documented and has a scientific reference (Schwenk et al., 2017). However, intermediate results are not accessible. For example, the transects used for generating the local width are not accessible. Thus, the tool lacks an important mechanism to validate its results. However, since RivMAP represents the best documented and most versatile tool, we compare results from our package with this package in Sect. 8.

To quantify channel bank retreat for repeated surveys, tools designed for other purposes could potentially be used. Examples are DSAS (Thieler et al., 2009) and AMBUR (Jackson, 2009), designed for analyzing migrating shorelines. These tools also require a baseline that is not derived by the program. AMBUR, scripted in the open-source environment R (Jackson, 2009), could be adapted to channels. However, we judge its approach to derive transects to be unreliable and unsuitable for rivers, as the transects do not cross the channel orthogonally. This leads to implausible results, especially in regions with large curvature. A further correction step is included to alleviate this problem, but the resulting distances of the baselines seem arbitrary. Thus, although the tool is among the best documented and accessible solutions currently available, its algorithm is not suitable for generating channel metrics in an objective manner. We conclude that none of the available approaches combines the criteria of being a tool for objectively deriving channel metrics, being easy and free to use and modify, and allowing a high degree of parameterization and fine-tuning.

### 3 Description of the algorithm

Our aim with this package was to develop a program that does not have the shortcomings of previous approaches and offers a transparent and objective algorithm. The algorithm

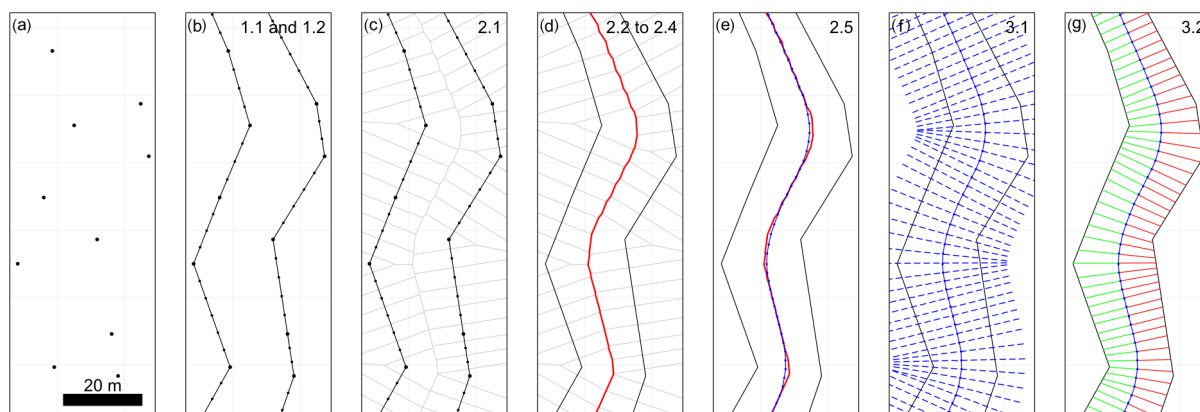
(full list of steps in Table 2 and visualization in Fig. 1) has two main parts. First, a centerline of the channel defined by the channel bank points is derived. Second, from this centerline the metrics channel length, width, and gradient (the latter only if elevation is provided) are calculated. This reference centerline allows for the projection of secondary metrics (for example, the occurrence of knickpoints) and the performing of temporal comparisons (more information on temporal analyses in Sect. 5).

A detailed description of all steps in the algorithm follows. In step 1.1, the algorithm creates a polygon feature from the bank points (Fig. 1b) in which the points are linearly interpolated (step 1.2) to increase their spatial resolution. This is a crucial step for improving the shape of the resulting centerline, even for straight channel beds (Fig. 2). From the interpolated points, Voronoi polygons (also called Dirichlet or Thiessen polygons) are calculated (2.1; Fig. 1c). In general, Voronoi polygons are calculated around center points (here the bank points) and denote the areas within which all points are closest to that center point. Next, the polygons are disassembled into single line segments. The segments in the center of the channel polygon form the desired centerline (see Fig. 1c). The algorithm then filters for these segments by first removing all segments that do not lie entirely within the channel banks (step 2.2; Fig. 3b). In a second step, dead ends are removed (step 2.3; Fig. 3c). Dead ends are segments that branch from the centerline but are not part of it, which are identified by the number of connections of each segment. All segments, other than the first and the last, must have exactly two connections. The filtering ends successfully if no further dead ends can be found. In step 2.4, the centerline segments are chained to one consistent line, the “original” centerline. In the final step 2.5 of the centerline calculation, the generated line is spatially smoothed (Fig. 1e) with a mean filter with definable width (see Sect. 4.2) to correct for sharp edges and to homogenize the resolution of the centerline points. This calculated centerline, the “smoothed” centerline, is the line feature representation of the channel; for example, it represents its length, which is calculated in step 2.6. If elevation data are provided with the bank point information (input data) the program also projects the elevation to the centerline points and calculates the slope of the centerline in step 2.7.

To calculate the channel metrics based on the centerline, channel transects are derived (step 3.1). Transects are lines perpendicular to a group of centerline points. In step 3.2, the intersections of the transects with the banks are calculated (Fig. 1g). When transects cross the banks multiple times, the crossing point closest to the centerline is used. The distance in the  $x$ – $y$  plane between the intersections represents the channel width at this transect. In addition to the width, the distances from the centerline points to banks are stored separately for the left and the right bank. The program also allows for the projecting of custom geospatial features to the centerline, such as the abundance of species or the occurrence of knickpoints (see Sect. 4.2). Projecting means here that ele-

**Table 2.** Full list of steps for the algorithm of the cmgo package and their functions.

Step	Description	Function
1.1	Generate polygon from bank points	CM.generatePolygon ()
1.2	Interpolate polygon points	
2.1	Create Voronoi polygons and convert to paths	CM.calculateCenterline ()
2.2	Filter out paths that do not lie within channel polygon entirely	
2.3	Filter out paths that are dead ends (have fewer than two connections)	
2.4	Sorting of the centerline segments to generate centerline	
2.5	Spatially smooth the centerline segments (mean filter)	
2.6	Measure the centerline length and slope	
2.7	Project elevation to the centerline points (optional)	
3.1	Derive transects of the centerline	CM.processCenterline ()
3.2	Calculate intersections of the centerline with the banks	
3.3	Project custom geospatial data onto centerline (optional)	
3.4	Calculate knickpoints based on scale-free approach (Zimmermann et al., 2008)	
4.1	Resample centerline and transect spacing (optional)	CM.resampleCenterline ()

**Figure 1.** Visualization of the work flow of the package. (a) The channel bank points represent the data input. (b) A polygon is generated where bank points are linearly interpolated, (c, d) the centerline is calculated via Voronoi polygons, (e) the centerline is spatially smoothed with a mean filter, (f) transects are calculated, (g) and the channel width is derived from the transects.

vation information or other spatial variables are assigned to the closest centerline points. Finally, a built-in function offers a way to set a homogenous spacing of the centerline in step 4.1, for example if transects should be spaced equally every  $x$  meters.

#### 4 Implementation and execution

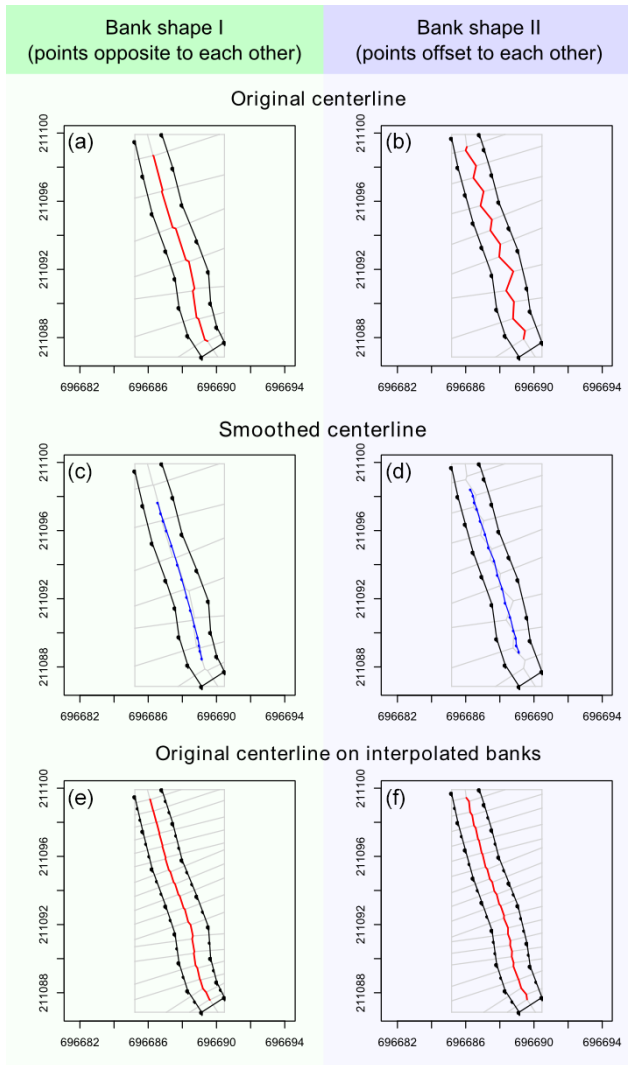
The program is written as a package for the statistical programming language R (R Development Core Team, 2011). The program can be divided into three main parts that are worked through during a project:

1. initialization (loading data and parameters; Sect. 4.1),
2. data processing (calculating centerline and channel metrics; Sect. 4.2), and

3. a review of the results (plotting or writing results to file; Sect. 4.3).

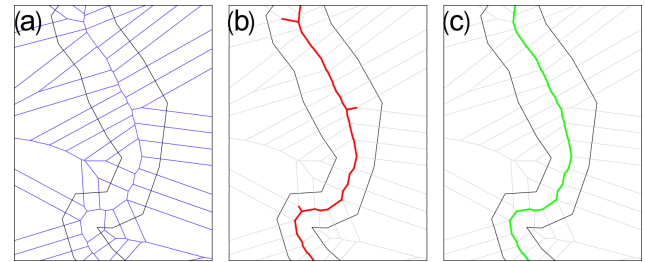
##### 4.1 Initialization: input data and parameters

The package cmgo requires basic geometrical information on the points that determine a channel shape: the bank points (Fig. 1a). In addition to the coordinates, the side of the channel must be specified for each point. In principle, a text file with the three columns “ $x$ ”, “ $y$ ”, and “side” represents the minimum input data required to run the program (Codebox 1). The coordinates  $x$  and  $y$  can be given in any number format representing Cartesian coordinates, and the column “side” must contain strings (e.g., left and right) as it represents information on the bank to which the given point is associated. Throughout this paper we refer to the left and right of the channel in regard to these attributes. Thus, the



**Figure 2.** The plot shows two digitizations (bank shapes I and II) of the same channel stretch. They differ only in the arrangement of bank points, which are mainly opposite (bank shape I; **a, c, e**) or offset (bank shape II; **b, d, f**) to each other. One can see how the offset negatively influences the shape of the centerline (**a, b**). The problem can be overcome by smoothing the centerline a posteriori (**c, d**) or interpolating between the bank points a priori (**e, f**). A combination of both methods is recommended and set as the default in *cmgo*.

user is generally free to choose which side to name “left”. However, we recommend sticking to the convention of naming the banks looking in the downstream direction. In addition, a fourth column *z* can be provided to specify the elevation of the points. This allows, for example, for the calculation of the channel gradient. Note that the order of the bank points matters. By default it is expected that the provided list contains all bank points in the upstream direction. If one, which can be the case when exporting the channel bed from a polygon shape, or both banks are reversed, the parameters `bank.reverse.left` and/or `bank.reverse.right`



**Figure 3.** The filtering of the centerline segments, **(a)** original Voronoi segments, **(b)** Voronoi segments filtered for segments that lie fully within the channel polygon, and **(c)** filtered for dead ends.

should be set `TRUE`. The units of the provided coordinates can be specified in the parameter `input.units` and defaults to `m` (meters).

The data can either be collected during field surveys with GPS or total stations or through remote sensing techniques with further digitizing, for example in a GIS. In the latter case, the data need to be exported accordingly. The input can be given in any ASCII table format. By default, the program expects a table with tab-delimited columns and one header line with the column names `POINT_X`, `POINT_Y`, and `POINT_Z` (the coordinates of the bank points), where the *z* component is optional, and `Name` (for the side). The tab delimiter and the expected column names can be changed in the parameters (see the Supplement for details). The input file(s) – for multiple files see also Sect. 5 – have to be placed in the input directory specified by the parameter `input.dir` (defaults to “./input”) and can have any file extension (`.txt`, `.csv`, etc.). The data-reading function iterates over all files in that directory and creates a data set for each file.

All the data and parameters used during runtime are stored in one variable of type list (see R documentation): the global data object. Throughout the following examples this variable is named `cmgo.obj` and its structure is shown in Code-box 2. The global data object also contains the parameter list, a list of more than 50 parameters specifying the generation and plotting of the model results. The full list of parameters with explanations can be found in the Supplement.

To create this object, the function `CM.ini(cmgo.obj, par)` is used. Initially, the function builds a parameter object based on the second argument `par`. If the `par` argument is left empty, the default configuration is loaded. Alternatively, a parameter file name can be specified (see the R documentation of `CM.par()` for further information). Once the parameter object is built, the function fills the data object by the following rules (if one rule was successful, the routine stops and returns the global data object).

1. If `cmgo.obj$par$workspace.read` is `TRUE` (default), the function looks for an `.RData` workspace file named `cmgo.obj$par$workspace.filename` (de-

```

Name POINT_X POINT_Y
right 401601.0819 3106437.335
right 401586.5327 3106406.896
right 401568.3238 3106383.586
right 401558.4961 3106364.129
...
left 401621.4337 3106431.134
left 401602.9913 3106405.991
left 401574.6073 3106352.232
left 401582.2671 3106323.134
...

```

**Codebox 1.** Example of input data table with columns for side and  $x$ ,  $y$  coordinates.

```

cmgo.obj = list(
  data = list(
    set1 = list(
      filename = "input.1.csv", # corresponding filename
      channel = list(), # input coordinates of banks
      polygon.bank.interpolate = TRUE,
      polygon = list(), # polygon object
      polygon.bank.interpolate.max.dist = 6,
      cl = list(), # centerlines (original and smoothed)
      metrics = list() # calculated metrics (width, etc.)
    ),
    set2 = list() # survey 2
    # ...
  ),
  par = list() # all model and plotting parameters
)

```

**Codebox 2.** Structure of the global data object containing data and parameters.

faults to “./user\_workspace.RData”). Note: there will be no such workspace file once a new project is started, since it needs to be saved by the user with `CM.writeData()`. If such a workspace file exists, the global data object is created from this source; otherwise the next source is tested.

2. If data input files are available in the directory `cmgo.obj$par$input.dir` (defaults to “./input”), the function iterates over all files in this directory and creates the data object from this source (see the “Input data” section above for further information on the data format). In this case the program starts with the bank geometry data set(s) found in the file(s). Otherwise the next source is tested.
3. If the `cmgo.obj` argument is a string or `NULL`, the function will check for a demo data set with the same name or “demo” if `NULL`. Available demo data sets are “demo”, “demo1”, “demo2”, and “demo3” (Sect. 7).

`CM.ini()` returns the global data object that must be assigned to a variable, for example `cmgo.obj=CM.ini()`. Once the object is created, the data processing can start.

#### 4.2 Controlling the data processing

The processing includes all steps from the input data (bank points) to the derivation of the channel metrics (Fig. 1). Next, we describe the parameters that are relevant during

the processing described in Sect. 3. When generating the channel polygon, the original bank points are linearly interpolated (Fig. 1b). The interpolation is controlled through the parameters `cmgo.obj$par$bank.interpolate` and `cmgo.obj$par$bank.interpolate.max.dist`. The first is a Boolean (TRUE/FALSE) that enables or disables the interpolation (default TRUE). The second determines the maximum distance of the interpolated points. The unit is the same as for the input coordinates, which means that if input coordinates are given in meters, a value of 6 (default) means that the points have a maximum distance of 6 m to each other. These parameters have to be determined by the user and are crucial for the centerline generation. Guidance on how to select and test these parameters can be found in Sect. 6.

During the filtering of the centerline paths, there is a routine that checks for dead ends. This routine is arranged in a loop that stops when there are no further paths to remove. In cases for which the centerline paths exhibit gaps (see Sect. 6), this loop runs indefinitely. To prevent this, there is a parameter `bank.filter2.max.it` (defaults to 12) that controls the maximum number of iterations used during the filtering.

In the final step of the centerline calculation, the generated line gets spatially smoothed with a mean filter (Fig. 1e) for which the width of smoothing in numbers of points can be adjusted through the parameter `cmgo.obj$par$centerline.smoothing.width` (by default equals 7). Note that the degree of smoothing has an effect on the centerline length (e.g., a higher degree of smoothing shortens the centerline). Similar to the coast-

```
cmgo.obj$data[[set]]$features = list(
  custom_feature_1 = list(
    x = c(),
    y = c()
  ),
  knickpoints = list(
    x = c(),
    y = c()
  )
)
```

**Codebox 3.** The format of secondary spatial features to be projected to the centerline.

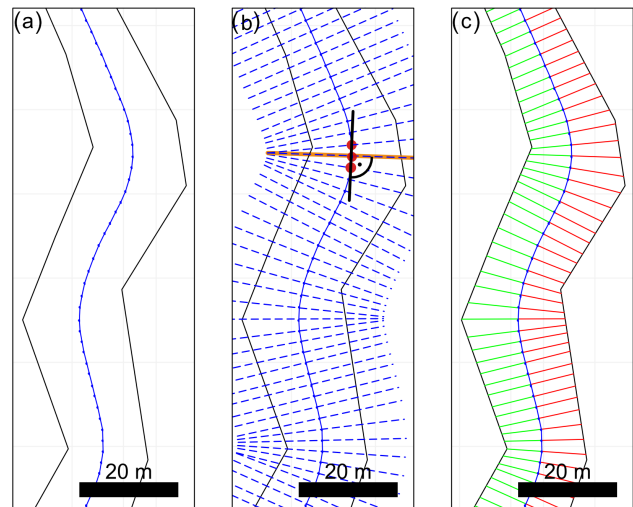
```
$metrics$tr      # linear equations of the transects
$metrics$cp.r    # coordinates of crossing points transects / right bank
$metrics$cp.l    # coordinates of crossing points transects / left bank
$metrics$d.r     # distance of reference centerline point / right bank
$metrics$d.l     # distance of reference centerline point / left bank
$metrics$w       # channel width
$metrics$r.r     # direction value: -1 for right, +1 for left to the centerline
$metrics$r.l     # direction value: -1 for right, +1 for left to the centerline
$metrics$diff.r  # difference between right bank point of actual time series and right bank
                 # point of reference series
$metrics$diff.l  # difference between left bank point of actual time series and left
                 # bank point of reference series
```

**Codebox 4.** The calculated metrics and their variable names (stored in the global data object under `cmgo.obj$data[[set]]`).

line paradox (Mandelbrot, 1967), the length of a channel depends on the scale of the observations. Technically, the length diverges to a maximum length at an infinitely high resolution of the bank points. However, practically there is an appropriate choice of a minimum feature size for which more detail in the bank geometry only increases the computational costs without adding meaningful information. The user has to determine this scale individually and should be aware of this choice. To check the consequences of this choice, the decrease in length due to smoothing is saved as `fraction` value in the global data object under `cmgo.obj$data[[set]]$cl$length.factor`. A value of 0.95 means that the length of the smoothed centerline is 95 % of the length of the original centerline paths. For the further calculations of transects and channel metrics, the smoothed version of the centerline is used by default.

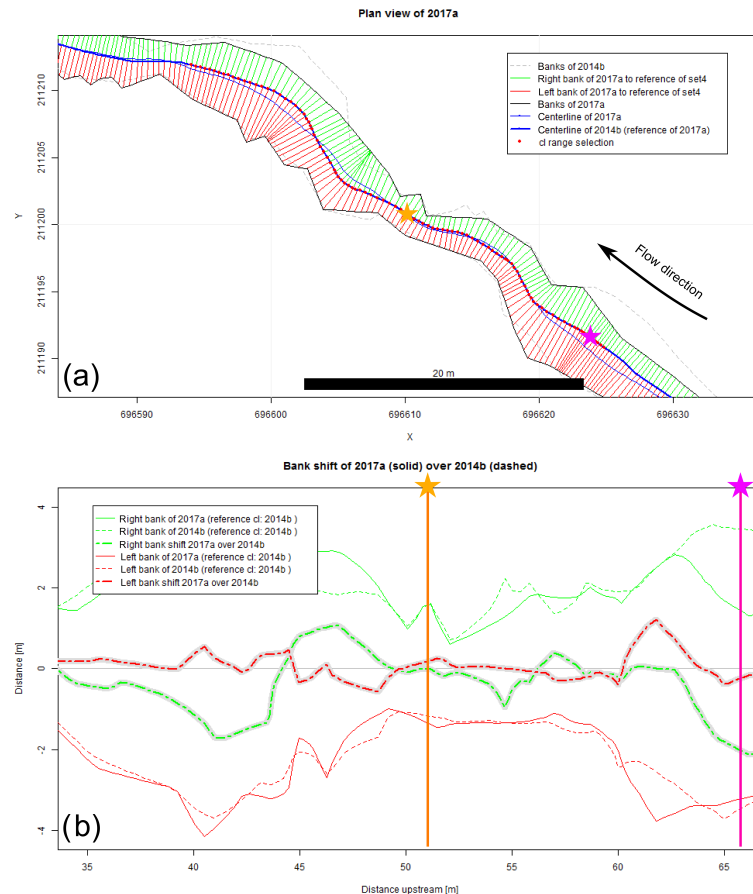
The program will automatically project the elevation of the bank points to the centerline if elevation information is provided in the input files ( $z$  component of bank points; see Sect. 4.1). Also, additional custom geospatial features, if available to the user, can be projected to the centerline, such as the abundance of species or the occurrence of knickpoints. Additional features are required to be stored in the global data object as lists with  $x$ ,  $y$  coordinates (Codebox 3) to be automatically projected to the centerline. Projecting here means that features with  $x$ ,  $y$  coordinates are assigned to the closest centerline point. The distance and the index of the corresponding centerline point are stored within the global data object.

To calculate the channel metrics based on the centerline, channel transects are derived. Transects are lines per-



**Figure 4.** (a) The smoothed centerline; (b) transects are calculated by taking a group of centerline points, creating a line through the outer points and calculating the perpendicular to that line, (c) and calculating the intersections of the transects with the channel banks.

pendicular to a group of  $n$  centerline points, where  $n$  – also called the transect span – is defined by the parameter `cmgo.obj$par$transects.span`. By default this span equals 3, which means that for each group of three centerline points a line is created through the outer points of that group to which the perpendicular – the transect – is calculated (see Fig. 4b). The number of resulting transects equals the number of centerline points, and for each centerline point the width  $w$  and further metrics are calculated (see Codebox 4). The distances of the centerline points to the banks



**Figure 5.** (a) Plan view of a short channel reach showing two channel surveys, 2014a (dashed channel outline) and 2017a (solid channel outline). A centerline is calculated for both, but due to an enabled reference mode, the centerline of 2014a is used for both surveys. This allows for the calculation of bank shift in panel (b). The two stars mark two random locations to compare the calculated metrics to each other.

are stored separately for the left and the right bank ( $d.r$  and  $d.l$ ) and a factor ( $r.r$  and  $r.l$ ) representing the side of the bank with regard to the centerline. Normally, looking downstream the right bank is always right to the centerline (value of  $-1$ ) and the left bank is always left to the centerline (value of  $+1$ ). However, when using a reference centerline to compare different channel surveys, the centerline can be outside the channel banks for which the metrics are calculated. To resolve the real position of the banks for tracing their long-term evolution (e.g., bank erosion and aggradation), the factors of  $r.r$  and  $r.l$  must be considered for further calculations (see also the “Reference centerline” section). A sample result for a reach of a natural channel is provided in Fig. 5.

#### 4.3 Review results: plotting and writing of the outputs

After the metrics are calculated and stored within the global data object, the results can be plotted or written to data files. The plotting functions include a map-like type plan view plot (`CM.plotPlanView()`), a plot of the spatial evolution of the channel width (`CM.plotWidth()`), and

a plot of the spatial and temporal evolution of the bank shift (`CM.plotMetrics()`). All plotting functions require a data set to be specified that is plotted (by default “set1”). Additionally, all plotting functions offer ways to specify the plot extent to zoom to a portion of the stream for detailed analyses. In the plan view plot, there are multiple ways to define the plot region (also called extent), which is determined by a center coordinate ( $x$ ,  $y$  coordinate) and the range on the  $x$  and  $y$  axes (zoom length). The zoom length is given via the function parameter `zoom.length` or, if left empty, is taken from the global parameter `cmgo.obj$par$plot.zoom.extent.length` (140 m by default). There are multiple ways to determine the center coordinate: via a predefined plot extent, via a centerline point index, or directly by  $x$ ,  $y$  coordinates. Predefined plot extents allow for quick access to frequently considered reaches of the stream and are stored in the parameter list (see Codebox 5). The list contains named vectors, each with one  $x$  and one  $y$  coordinate. To apply a predefined extent, the name of the vector has to be passed to the plot function as in `CM.plotPlanView(cmgo.obj,`

```

plot.zoom.extents = list(      # presets (customizable list) of plot regions
  e1 = c(400480, 3103130),    # plot region definition e1 with x/y center coordinate
  e2 = c(399445, 3096220),
  e3 = c(401623, 3105925),
  all = NULL
)

```

**Codebox 5.** Definition of predefined plot extents that allow for the quick plotting of frequently used map regions. The names, here “e1”, “e2”, and “e3”, contain a vector of two elements, the  $x$  and  $y$  coordinates at which the plot is centered. To plot a predefined region, call for example `CM.plotPlanView(cmgo.obj, extent = “e2”)`.

```

cmgo.obj$par$centerline.use.reference = TRUE
cmgo.obj$par$centerline.reference     = “set1”

```

**Codebox 6.** The parameters to enable the reference mode for channel metrics calculations (only necessary for time series analyses).

`extent = “extent_name”`). Another way of specifying the plot region is via a centerline point index, for example `CM.plotPlanView(cmgo.obj, cl = 268)`. This method guarantees that the plot gets centered on the channel. To find out the index of a desired centerline point, centerline text labels can be enabled with `cmgo.obj$par$plot.planview.cl.tx = TRUE`. Finally, the plot center coordinate can be given directly by specifying either an  $x$  or  $y$  coordinate or both. If either an  $x$  or  $y$  coordinate is provided, the plot centers at that coordinate and the corresponding coordinate will be determined automatically by checking where the centerline crosses this coordinate (if it crosses the coordinate multiple times, the minimum is taken). If both  $x$  and  $y$  coordinates are provided, the plot centers at these coordinates.

A plot of the width of the whole channel (default) or for a portion (via `cl` argument) can be created with `CM.plotWidth()`. Two data sets with the same reference centerline can also be compared. The `cl` argument accepts the range of centerline points to be plotted; if `NULL` (default), the full channel length is plotted. If a vector of two elements is provided (e.g., `c(200, 500)`), this `cl` range is plotted. If a string is provided (e.g., “cl1”), the range defined in `cmgo.obj$par$plot.cl.ranges$cl1` is plotted. Alternatively to the range of centerline indices, a range of centerline lengths can be provided with argument `d`. If a single value (e.g., 500) is given, 50 m around this distance is plotted. If a vector with two elements is given (e.g., `c(280, 620)`), this distance range is plotted.

The third plot function creates a plot of the bank shift (bank erosion and aggradation). This plot is only available when using multiple channel observations in the reference centerline mode (see the “Reference centerline” section). The arguments of the function regarding the definition of the plot region are the same as of the function `CM.plotWidth()`.

In addition to the plotting, the results can be written to output files and to an R workspace file with the function `CM.writeData()`. The outputs written by the function depend on the settings in the parameter ob-

ject. If `cmgo.obj$par$workspace.write = TRUE` (default is `FALSE`), a workspace file is written containing the global data object. The file name is defined in `cmgo.obj$par$workspace.filename`. Further, ASCII tables can be written containing the centerline geometry and the calculated metrics. If `cmgo.obj$par$output.write = TRUE` (default is `FALSE`), an output file for each data set is written to the output folder specified in `cmgo.obj$par$output.dir`. The file names are the same as the input file names with the prefixes `cl_*` and `metrics_*`. All parameters regarding the output generation can be accessed with `?CM.par` executed in the R console or can be found in the Supplement.

## 5 Temporal analysis of multiple surveys

The program can perform analyses on time series of channel shapes. To do this, multiple input files have to be stored in the input directory (see Sect. 4.1). A data set for each file will be created in global data object and mapped to the sub-lists “set1”, “set2”, etc. (see Codebox 1). The program automatically iterates over all data sets, processing each set separately. The order of the data sets is determined by the file names. Thus, the files need to be named according to their temporal progression, e.g., “channelsurvey\_2017.csv”, “channelsurvey\_2018.csv”, etc. The mapping of the file names to data sets is printed to the console and stored in each data set under `cmgo.obj$data[[set]]$filename`.

### Reference centerline

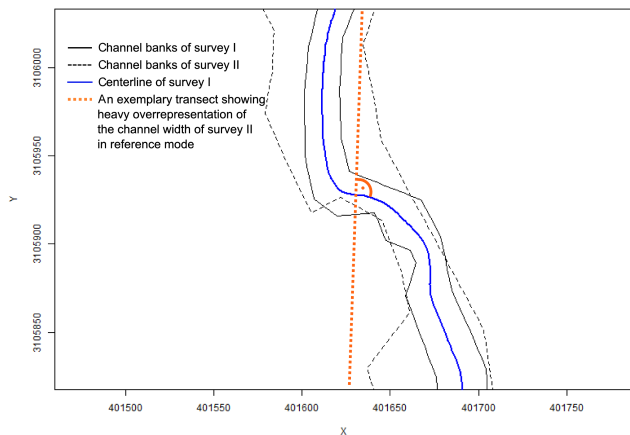
The channel metrics are calculated based on the centerline, which exists for every river bed geometry. When there are multiple temporal surveys of a river geometry, there is also a centerline for each data set. Multiple centerlines prevent a direct comparison of the channel metrics as they can be seen as individual channels. Thus, there are two modes for temporal comparisons of the channel metrics. Metrics can be calculated for each channel geometry individually. In this

```
# installation of dependencies (required only once)
install.packages(c("spatstat", "zoo", "sp", "stringr", "rgl", "shapefiles"))

# installation (required only once)
install.packages("cmgo", repos="https://raw.githubusercontent.com/AntoniusGolly/repos/master",
type="source")

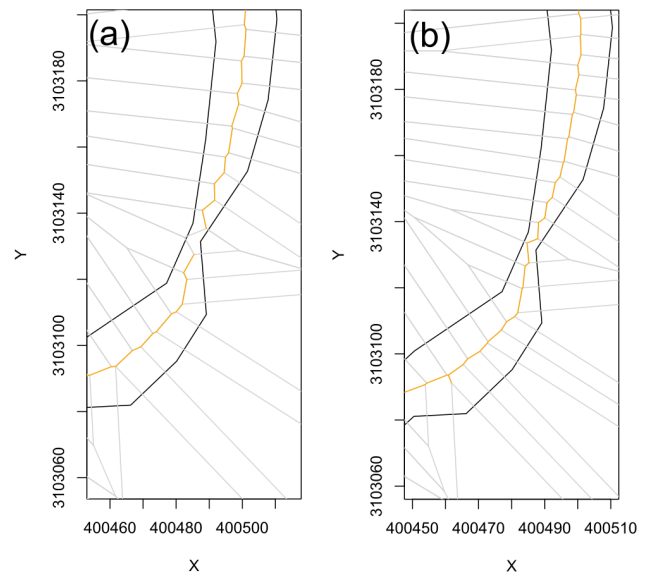
# include the package (required for every start of an R session)
library(cmgo)
```

**Codebox 7.** Installation and embedding of the package in R.



**Figure 6.** Two consecutive channel geometries (surveys I and II) with a profound reorganization of the channel bed. In the reference mode a centerline of one survey is used to build transects. Here, using the centerline of the first survey (blue line) as a reference is not suitable to capture the channel width correctly for the second survey (dashed line) as the exemplary transect (dashed orange line) suggests.

mode, the channel metrics are the most accurate representation for that channel observation; for example, channel width is most accurately measured, but does not allow for a direct comparison of consecutive surveys. In a second approach, a reference centerline for all metrics calculations can be determined. In this approach, all metrics for the various bank surveys are calculated based on the centerline of the data set defined in `cmgo.obj$par$centerline.reference` (default “set1”). This mode must be enabled manually (see Codebox 6) but should be used only if the bank surveys differ slightly. If there is profound channel migration or a fundamental change in the bed geometry, the calculated channel metrics might not be representative (shown in Fig. 6). To compare channel geometries for which the individual centerlines are not nearly parallel, we recommend calculating the metrics based on individual centerlines and developing a proper spatial projection for temporal comparisons.



**Figure 7.** (a) A gap in the centerline occurs when the spacing of the bank points is too large compared to the channel width; (b) the gap fixed by increasing the resolution of the bank points through the parameter `par$bank.interpolate.max.dist`.

## 6 Technical fails and how to prevent them

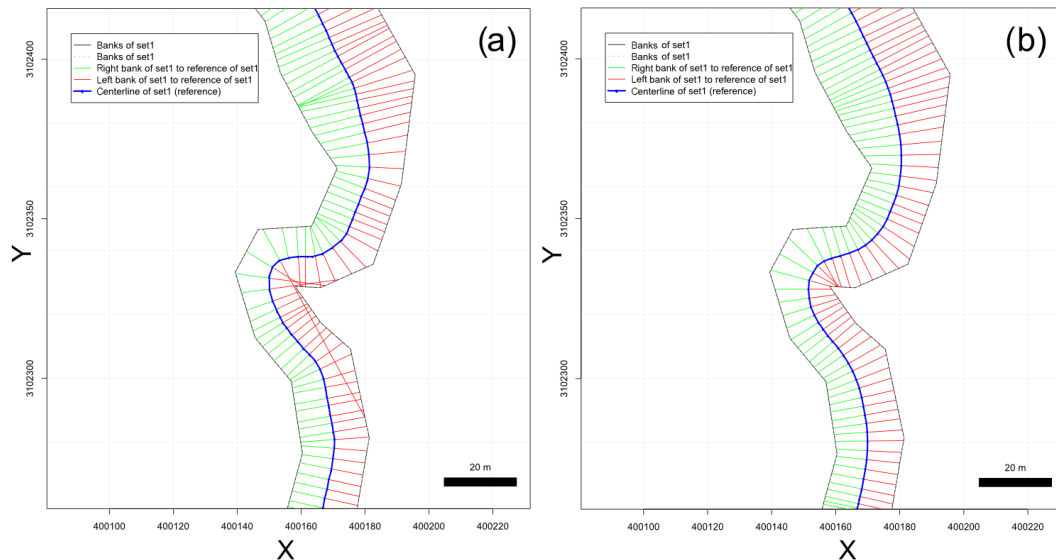
There are certain geometrical cases in which the algorithm can fail with the default parameterization. To prevent this, a customized parameterization of the model is required. The program prints notifications to the console during runtime if the generation of the centerline fails and offers solutions to overcome the issue. The main reason that failure occurs is that the resolution of channel bank points (controlled via `cmgo.obj$par$bank.interpolate.max.dist`) is relatively low compared to the channel width. In tests, a `cmgo.obj$par$bank.interpolate.max.dist` less than the average channel width was usually appropriate. Otherwise, the desired centerline segments produced by the Voronoi polygonization can protrude out of the bank polygon (Fig. 7a) and thus do not pass the initial filter of the centerline calculation (see Sect. 3), since this filter mechanism first checks for segments that lie fully within the channel polygon. This creates a gap in the centerline, which results in an endless loop during the filtering for dead ends.

```
# initialization: load data and parameters
cmgo.obj = CM.ini("demo") # check the data structure with str(cmgo.obj)

# processing
cmgo.obj = CM.generatePolygon(cmgo.obj)
cmgo.obj = CM.calculateCenterline(cmgo.obj)
cmgo.obj = CM.processCenterline(cmgo.obj)

# view results
CM.plotPlanView(cmgo.obj) # plot a map with pre-defined extent
CM.plotWidth(cmgo.obj)    # plot the channel width in downstream direction
CM.plotMetrics(cmgo.obj)  # plot a comparison of bank profiles
```

**Codebox 8.** Minimal example script to run cmgo with demo data set.



**Figure 8.** (a) The transects (perpendiculars to the centerline) do not intersect with banks properly, and thus the channel width is overrepresented; (b) an increased transect span fixes the problem and channel width is identified correctly.

Thus, if problems with the calculation of the centerline arise, an increase in the spatial resolution of bank points via `cmgo.obj$par$bank.interpolat.max.dist` is advised to naturally smooth the centerline segments (Fig. 7b).

Another problem can arise from an unsuitable setting during the calculation of transects. If the channel bed exhibits a sharp curvature, a misinterpretation of the channel width can result (see Fig. 8). In that case, one of the red transects does not touch the left bank of the channel properly, thus leading to an overestimated channel width at this location. To prevent this, the span of the transect calculation can be increased. The results have to be checked visually by using one of the plotting functions of the package.

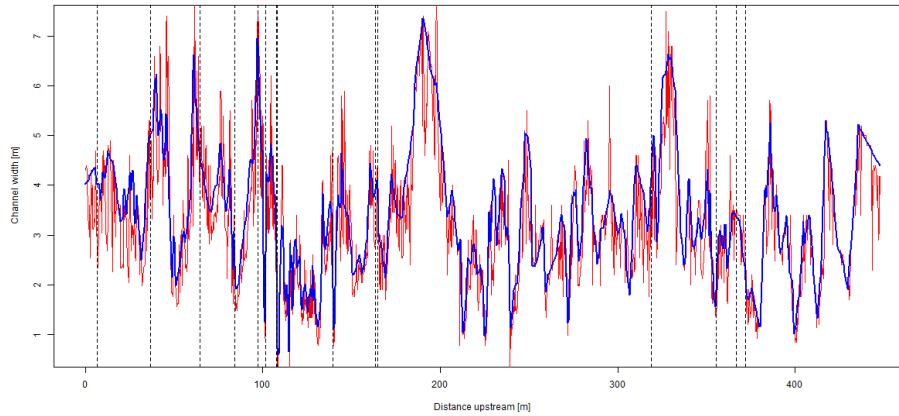
## 7 How to use the program: step by step instructions

The cmgo package can be used even without comprehensive R knowledge, and the following instructions do not require preparatory measures other than an installed R environment (R Development Core Team, 2011). Once the R console is

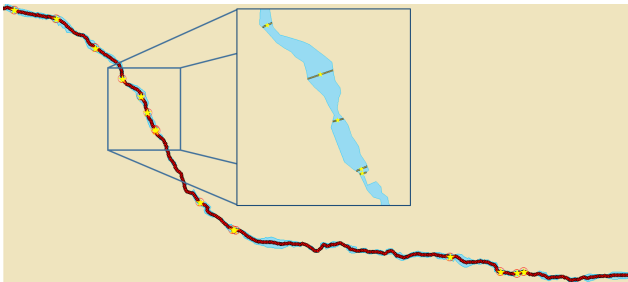
started, installation of the cmgo package is done with the `install.packages()` function (Codebox 7).

To quickly get started with cmgo, we provide four demo data sets. Using these data sets, the following examples demonstrate the main functions of the package but, more importantly, allow for the investigation of the proper data structure of the global data object. This is of particular importance when trouble shooting failures with custom input data.

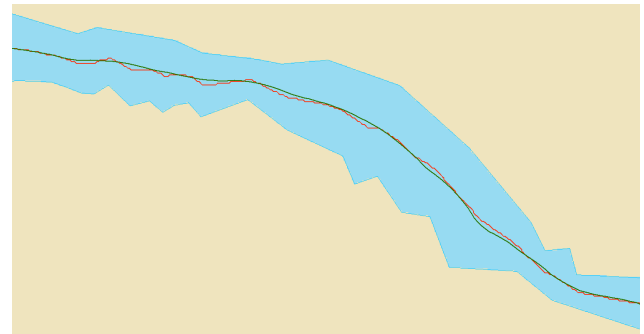
The general execution sequence includes initialization, processing, and reviewing the results with a standard execution sequence shown in Codebox 8. To switch from demo data to custom data, input files have to be placed in the specified input folder (“./input” by default) and `CM.ini()` has to be called without any arguments. Since the file format of the custom input files can differ from the expected default format, all program parameters regarding the data reading should be considered. A list of all parameters available can be accessed with `?CM.par` executed in the R console or can be found in the Supplement. To change a parameter, the new parameter value is assigned directly within the global data object (e.g., `cmgo.obj$par$input.dir = “./input”`).



**Figure 9.** Channel width as derived by cmgo (blue line) and RivMAP (red line) for 1506 locations along a 449 m reach of a natural channel (Fig. 10) in the upstream direction. The vertical dashed lines mark our points at which we investigated the width manually in a GIS.



**Figure 10.** Fifteen random locations (yellow stars) of the 1506 centerline points (red dots) at which we evaluated the width manually in a GIS (example in the inset) that are compared to the width of the automated products.



**Figure 11.** The two different centerlines of the products cmgo (green line) and RivMAP (red line) reveal differences in the shape that also influence the channel length.

The plotting functions include a map-like plan view plot (`CM.plotPlanView()`), a line chart with the channel width (`CM.plotWidth()`), and, if available, a plot of the bank retreat (`CM.plotMetrics()`). The latter is only available in the reference centerline mode (see the “Reference centerline” section).

## 8 Evaluation of the data quality

We evaluated the quality of the derived channel width by cmgo compared to manually measured data and to the best documented and versatile product of our literature review, RivMAP (Table 1). First, we compared the evolution of the channel width derived by the two automated products, showing that there is general agreement (Fig. 9). We then identified 15 locations randomly (vertical dashed lines in Fig. 9) at which we assessed the channel width manually in a GIS (Fig. 10).

The channel width at the transects is generally well captured by the automated products (Table 3) as the mean errors are relatively low compared to the absolute width. However, compared to the manually derived average width of 3.49 m,

the average width of all transects deviates only  $-0.07$  m for cmgo, while it deviates  $-0.42$  m for RivMAP. Thus, cmgo performs generally better in deriving the channel width for the test channel reach, and overall RivMAP underestimates the channel width. This is also expressed in the smaller standard deviation of the differences, which is 0.098 m for cmgo and 0.736 m for RivMAP. The large scatter can also be observed in Fig. 9. Compared to the error of the in situ measurements of the channel banks with a total station (1 cm), the precision of the channel width calculations by cmgo is within the same order of magnitude, while it is an order of magnitude larger for RivMAP.

The channel centerlines of the two products differ in length. While the centerline of cmgo has a length of 449 m along the river reach, the centerline of RivMAP has a length of 588 m (31 % longer). Looking at the shape of the centerlines (Fig. 11), we argue that the centerline of cmgo better represents the channel in terms of large-scale phenomena. It may, for example, be more accurate for reach-averaged calculations of bankfull flow. The centerline of RivMAP contains a stronger signal of the microtopography of the banks

**Table 3.** Channel width at 15 randomly selected locations along a natural channel. The width was identified manually in a GIS, by cmgo, and by RivMAP. Differences in the width from the automated products were compared to the manual approach.

Transect [no.]	Manual approach [m]	cmgo width [m]	cmgo difference to manual [m]	RivMAP width [m]	RivMAP difference to manual [m]
1	4.01	4.02	0.01	2.83	−1.18
2	5.01	5.02	0	3.75	−1.27
3	4.57	4.55	−0.01	4.03	−0.54
4	2.66	2.59	−0.07	2.6	−0.06
5	6.79	6.83	0.04	5.37	−1.41
6	2.82	2.66	−0.15	2.12	−0.7
7	3.02	2.97	−0.06	2.55	−0.48
8	1.76	1.67	−0.09	2.6	0.84
9	2.27	1.93	−0.34	2.6	0.33
10	3.9	3.91	0.01	2.83	−1.07
11	3.82	3.66	−0.17	4.4	0.58
12	4.19	4.14	−0.05	3.04	−1.15
13	2.04	1.89	−0.15	1.34	−0.7
14	3.37	3.37	0	3.5	0.13
15	2.14	2.11	−0.03	2.5	0.36
Avg.	3.49	3.42	−0.07	3.07	−0.42
SD	1.34	1.399	0.098	0.997	0.736

due to the way the centerline is created (eroding banks). The difference in length also has an influence on slope calculations, which will be lower for RivMAP.

## 9 Concluding remarks

The presented package cmgo offers a stand-alone solution to calculate channel metrics in an objective and reproducible manner; cmgo also allows for a close look into the interior of the processing. All intermediate results are accessible and comprehensible. Problems that arise for complex geometries can be overcome due to the high degree of parameterization. The cmgo package qualifies as a highly accurate tool suited to analyze especially complex channel geometries. However, if complex geometries should be compared to each other, for example when analyzing the evolution of meandering channels, our product does not offer the ideal solution due to the way that cmgo treats the reference of the channels. Thus, our product should be the tool of choice if precise measurements are required both in location and quantity and if geometrical and other spatial data should be statistically analyzed. However, when large time series of meandering rivers are the main purpose of the effort, other products, for example the Channel Migration Toolbox, are more suitable.

Since cmgo does not come with a graphical user interface, only static map views of the channel can be obtained by scripting them. The cmgo package offers various plotting functions to do this, which allow for a predictable and repro-

ducible plot. The downside of this approach is that plots are naturally not interactive, which is the case for GIS applications. For people who prefer this functionality, an export of the intermediate and end results to GIS is recommended.

The only requirement for running cmgo is an installed environment of the open-source framework R. Thus, the prerequisites are narrowed down to a minimum to facilitate easy integration and wide distribution for scientific or practical use. The license under which the package is provided allows for modifications to the source code. The nature of R packages determines the organization of the source code in functions. This encapsulation comes at the cost of a sometimes untransparent architecture, making it difficult to modify or understand the code. Thus, for advanced users who desire a more flexible way of interacting with the algorithm, we refer to the raw source codes at GitHub (<https://github.com/AntoniusGolly/cmgo>).

**Code and data availability.** All codes and demo data are available at <https://github.com/AntoniusGolly/cmgo>.

**The Supplement related to this article is available online at <https://doi.org/10.5194/esurf-5-557-2017-supplement>.**

**Author contributions.** AG (programming, paper preparation), JT (paper preparation).

**Competing interests.** The authors declare that they have no conflict of interests.

**Acknowledgements.** We thank Michael Dietze for giving the helpful R courses that facilitate the development of cmgo as an R package and for the support during the debugging, Kristin Cook for providing the sample data for the demo data sets, and Marisa Repasch-Elder for the guidance in MATLAB.

The article processing charges for this open-access publication were covered by a Research Centre of the Helmholtz Association.

Edited by: Arjen Stroeven

Reviewed by: two anonymous referees

## References

- Ackerman, P. E. C. T.: HEC-GeoRAS GIS Tools for Support of HEC-RAS using ArcGIS User's Manual, US Army Corps of Engineers, Davis, California, p. 244, 2011.
- Amit: Estimating river Channel Width using Python/ArcGIS/MATLAB/R?, available at: <http://gis.stackexchange.com/questions/164169/estimating-river-channel-width-using-python-arcgis-matlab-r> (last access: 14 March 2017), 2015.
- Asterics: Software-Handbuch ASTERICS, Version 4.1, 1–120, available at: [http://www.fliessgewaesserbewertung.de/downloads/ASTERICS\\_Softwarehandbuch\\_Version4.pdf](http://www.fliessgewaesserbewertung.de/downloads/ASTERICS_Softwarehandbuch_Version4.pdf) (last access: 13 September 2017), 2013.
- Cook, K. L., Turowski, J. M., and Hovius, N.: River gorge eradication by downstream sweep erosion, *Nat. Geosci.*, 7, 682–686, <https://doi.org/10.1038/ngeo2224>, 2014.
- Dilts, T. E.: Polygon to Centerline Tool for ArcGIS, University of Nevada, Reno, available at: <http://www.arcgis.com/home/item.html?id=bc642731870740aabf48134f90aa6165> (last access: 15 March 2017), 2015.
- ESRI: ESRI ArcMap Desktop, Environmental Systems Research Institute, Redlands, CA, available at: <http://desktop.arcgis.com/de/>, last access: 13 September 2017.
- Ferreira, M.: Perpendicular Transects, available at: <http://gis4geomorphology.com/stream-transects-partial/> (last access: 15 March 2017), 2014.
- Ferrer-Boix, C., Chartrand, S. M., Hassan, M. A., Martín-Vide, J. P., and Parker, G.: On how spatial variations of channel width influence river profile curvature, *Geophys. Res. Lett.*, 43, 6313–6323, <https://doi.org/10.1002/2016GL069824>, 2016.
- Hornby, D.: RivEX, available at: <http://www.rivex.co.uk/Online-Manual/RivEX-Online-Manual.html?Extractchannelwidths.html>, last access: 15 March 2017.
- Jackson, C. W.: The Ambur project: Analyzing Moving Boundaries Using R, Department of Geology & Geography Georgia Southern University, available at: <http://ambur.r-forge.r-project.org/> (last access: 27 March 2017), 2009.
- Legg, N., Heimburg, C. Collins, B. D., and Olson, P. L.: The Channel Migration Toolbox: ArcGIS Tools for Measuring Stream Channel Migration, Publication no. 14-06-032, available at: <https://fortress.wa.gov/ecy/publications/SummaryPages/1406032.html> (last access: 13 September 2017), 2014.
- Lopez, S., Cojan, I., Rivoirard, J., and Galli, A.: Process-Based Stochastic Modelling: Meandering Channelized Reservoirs, in: *Analogue and Numerical Modelling of Sedimentary Systems: From Understanding to Prediction*, Wiley, Oxford, UK, 139–144, <https://doi.org/10.1002/9781444303131.ch5>, 2009.
- Mandelbrot, B.: How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension, *Science*, 156, 636–638, <https://doi.org/10.1126/science.156.3775.636>, 1967.
- Milzow, C., Molnar, P., McArde, B. W., and Burlando, P.: Spatial organization in the step-pool structure of a steep mountain stream (Vogelbach, Switzerland), *Water Resour. Res.*, 42, 1–11, <https://doi.org/10.1029/2004WR003870>, 2006.
- Mir, K., Tariq, A., and Atif, S.: River Width Calculator, available at: <http://www.arcgis.com/home/item.html?id=4e7c9370e3e8455e8ff57d6b23baf760> (last access: 13 September 2017), 2013.
- Pavelsky, T. M. and Smith, L. C.: RivWidth: A Software Tool for the Calculation of River Widths From Remotely Sensed Imagery, *IEEE Geosci. Remote Sens. Lett.*, 5, 70–73, <https://doi.org/10.1109/LGRS.2007.908305>, 2008.
- Pizzuto, J. E.: Streambank Erosion and River Width Adjustment, in: *Sedimentation Engineering*, American Society of Civil Engineers, Reston, VA, 387–438, <https://doi.org/10.1061/9780784408148.ch07>, 2008.
- R Development Core Team: R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, <https://doi.org/10.1007/978-3-540-74686-7>, 2011.
- Schwenk, J., Khandelwal, A., Fratkin, M., Kumar, V., and Foufoula-Georgiou, E.: High spatiotemporal resolution of river planform dynamics from Landsat: The RivMAP toolbox and results from the Ucayali River, *Earth Space Sci.*, 4, 46–75, <https://doi.org/10.1002/2016EA000196>, 2017.
- Thieler, E. R., Himmelstoss, E. A., Zichichi, J. L., and Ergul, A.: Digital Shoreline Analysis System (DSAS) version 4.0 – An ArcGIS extension for calculating shoreline change, US Geological Survey Open-File Report 2008, p. 1278, available at: <http://woodshole.er.usgs.gov/project-pages/DSAS/> (last access: 13 September 2017), 2009.
- Wobus, C., Whipple, K. X., Kirby, E., Snyder, N., Johnson, J., Spyropoulou, K., Crosby, B., and Sheehan, D.: Tectonics from topography: procedures, promise, and pitfalls, *Geol. Soc. Am. Spec. Pap.*, 398, 5–74, [https://doi.org/10.1130/2006.2398\(04\)](https://doi.org/10.1130/2006.2398(04)), 2006.
- Zimmermann, A., Church, M., and Hassan, M.: Identification of steps and pools from stream longitudinal profile data, *Geomorphology*, 102, 395–406, <https://doi.org/10.1016/j.geomorph.2008.04.009>, 2008.