Reply to comments and queries of the Editor for the manuscript “Deriving principle channel metrics from bank and long-profile geometry with the R-package cmgo”

Original manuscript title:	Deriving principle channel metrics from bank and long-profile geometry with the R-package cmgo
Date of first submission:	May 10 2017
Manuscript No:	esurf-2017-32
Date of first review:	June 04 2017
Resubmission:	July 19 2017
Date of second review:	August 4 2017

Please note:

- This document contains the comments of the Editor and our replies in black text color.
- Line numbers of the Editor’s comments refer to the version edited by the Editor (comments-to-author.pdf)

Reply to comments of the Editor

We thank the Editor for suggesting minor revisions on the manuscript. Below, we answer the comments one by one in detail.

- Line [27].

Agreed. Deleted.

- Line [30].

Agreed, added.

- Line [40].

Agreed, added.

- Line [42].

We have not added a hyperlink here and have nothing changed.

- Line [44]

Agreed.

- Line [45]

Agreed.

- Line [46a]

Agreed.

- Line [46b]

Agreed.

- Line [46c]

Agreed.

- Line [51]

Agreed.

- Between line [53] and [54]

Agreed.

- Line [71]

Agreed, very nice suggestion.

- Line [75]

Agreed.

- Line [89]

Agreed, good solution.

- Line [92]

Agreed.

- Line [101]

Agreed.

- Table from line [102 ff]

Both suggestions agreed.

- Line [103]

Agreed. I changed the font of the footnotes. Other suggestions how to clarify this?

- Line [106]

Agreed.

- Line [109]

Agreed, rephrased.

- Line [110]

Currently, the Supplementary contains only the list of parameters, which is not mentioned here. Thus, we did no changes here.

- Caption Figure 2

Thanks for the correction of the typo.

- Line [122]

We argue the introductory sentence is useful but it can be deleted if necessary.

- Line [125]

Agreed.

- Line [143]

Agreed. Good input.

- Line [147]

Agreed.

- Line [163a]

Agreed.

- Line [163b]

Agreed.

- Line [179]

Agreed, changed in the caption.

- Line [184]

Agreed. Thanks for the hint.

- Line [186]

We are open to any suggestions.

- Line [215]

Agreed. Thanks.

- Line [221a]

Corrected. This read fine for us but apparently the automatic reference broke.

- Line [221b]

Agreed, shortened.

- Line [232]

Agreed

- Line [233a]

Agreed.

- Line [233b]

Agreed.

- Line [252]

Agreed.

- Line [262]

Agreed, changed.

- Line [267]

Agreed.

- Line [276]

Agreed, caption updated.

- Line [288]

Agreed.

- Caption Figure 5

Agreed. Thanks for finding the typo.

- Line [292]

Agreed.

- Line [301a] and [301b]

Agreed.

- Line [304]

Agreed. Done! Found four more locations in the manuscript to change. If there is both coordinates mentioned, it is now it is always “x,y-coordinates” or “x- and y-coordinates” if there is an “and” or “or” between the coordinates. If there is only one coordinate mentioned it is “x-coordinate”.

- Line [316]

Agreed.

- Line [355]

Agreed, that was cryptic. We revised the description.

- Caption Figure 6

Agreed. Updated.

- Line [366]

Agreed. Sorry, again the automatic reference broke somewhere. Now hard-coded.

- Caption Figure 7

Agreed, fixed.

- Caption Figure 8

Agreed, updated.

- Line [389]

Agreed.

- Line [406]

Agreed.

- Line [408]

Agreed.

- Line [409a]

Agreed. Changed.

- Line [409b]

Agreed. Thanks for finding the typo.

- Line [409c]

Agreed.

- Line [409d]

Agreed.

- Line [412]

Agreed. Changed to “derived”.

- Line [413]

Agreed.

- Line [414]

Agreed.

- Line [416a]

Agreed.

- Line [416b]

Agreed. Clarified in caption Figure 10.

- Line [412]

Agreed. Rephrased.

- Line [423]

Agreed.

- Header Table 3

Agreed.

- Line [428]

Agreed.

- Line [430]

Agreed.

- Caption Table 3

Agreed. Updated.

- Caption Figure 11

Agreed.

Please note: changes in the References are not tracked by Word since the information are coming out of a reference manger. We apologize for the inconvenience.

- Line [475]

Agreed.

- Line [479a]

Agreed.

- Line [479b]

Agreed. Updated link.

- Line [483]

Agreed.

- Line [484]

Agreed.

- Line [487]

Agreed.

- Line [488]

Agreed.

- Line [493]

Agreed.

- Line [495]

Agreed.

- Line [497]

Agreed.

- Line [500]

Agreed. Updated reference style for all references.

- Line [505]

Agreed.

- Line [526]

Agreed.

- Line [528]

Agreed. Fixed.

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

Antonius Golly[1], Jens M. Turowski[1]

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

Correspondence to: Antonius Golly (golly@gfz-potsdam.de)

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 restauration 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 principle 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 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

Principle 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 snap-shot of the current local channel geometry can provide an integrated picture of the processes leading to its formation, if ~~interpreted correctly and~~ 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 water surface, or the water flow path or thalweg in two- or three-dimensional coordinates. The data can be either collected during field surveys with GPS or total stations or through remote sensing, with the need of 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 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, ~~or~~ and the distances from the centerline ~~respectively~~, as well as allows to project additional spatial metrics to the centerline.

2. Literature review

Computer-aided products for studying rivers have a long tradition, and solutions for standardized assessments include many disciplines, as for example for assessing the ecological status of rivers (Asterics, 2013) or for characterizing heterogeneous reservoirs (Lopez *et al.*, 2009). There are also numerous efforts to derive principle channel metrics from remote or in-situ measurements of topography or directly of features such as channel banks. Available products, which we review in detail ~~next~~ (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 ~~understand~~ evaluate the data quality. Apart from this, all of these products are dependent on commercial software, are bound to a graphical user interface (not scriptable) and cannot be parametrized to a high degree.

The second group of solutions represent 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 ~~marginal use (Tiobe 2017), which recently became member~~ 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, do 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 – computational intensive transformations are necessary, resulting in long computation times (the authors describe up to an hour for their example). More importantly, the centerline position depends on the resolution of the input rasters, and thus is 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 ~~choose it to~~ compare ~~our~~ results from our package with this package ~~to~~ in ~~the~~ section *8. Evaluation of the data quality~~Evaluation of the data quality~~*.

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 shore-lines. 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, leading 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 parametrization and fine-tuning.

Name of the tool	Platform	Data format	Last updated	Free to use [1]	Free to modify [2]	Configurable	Full-stack solution [3]	Scientific reference	Note
cmgo (this paper)	R	Vector	July 2017	yes, yes	yes	yes	yes	yes [4]	
RiverWidthCalculator	ArcMap	Raster	June 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	July 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	[5]	no, yes	no	very limited	no	no	• limited functionality • ~~very~~ highly unstable
RivWidth	IDL	Raster	May 2013	no, yes	yes	[5]	[5]	yes	• limited access due to IDL license
DSAS	ArcMap	Vector	Dec 2012	no, yes	no	yes	no	yes	• primarily designed for coast-lines
AMBUR	R	Vector	June 2014	yes, yes	yes	limited	no	yes	• no multi-temporal analyses allowed
RivMap	MATLAB	Raster	Apr 2017	no, yes	yes	yes	limited	yes	• primarily for large scale river systems • fails silent on errors

Table 1: overview of existing products, [1] the two values indicate free use of framework (first) and plugin (second value), [2] 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, [3] a product is considered a full-stack solution if it performs all steps from the bank geometry to the derived channel metrics, [4] relies on the publication of this manuscript, [5] gray cells indicate that no information could be gathered ~~by the time of writing this~~.

Formatiert: Schriftart: Kursiv

Formatiert: Hochgestellt

Formatiert: Hochgestellt

Formatiert: Hochgestellt

Formatiert: Hochgestellt

Formatiert: Hochgestellt

3. Description of the algorithm

Our aim ~~in this paper~~with this package ~~is~~ 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~~Table 2~~ and visualization in Figure 1~~Figure 1~~) has two main parts. First, a centerline of the channel – defined by the channel bank points – is derived and second, from this centerline the metrics – channel length, width and gradient (the latter only if elevation is provided) – are calculated. Furthermore, this reference centerline allows for projecting secondary metrics (as for example the occurrence of knickpoints) and performing temporal comparisons (more information on temporal analyses in section 5).

Formatiert: Interner Link Zchn

Formatiert: Interner Link Zchn

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 less than 2 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's 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)	

Table 2: full list of steps of the algorithm of the package *cmgo* and their functions

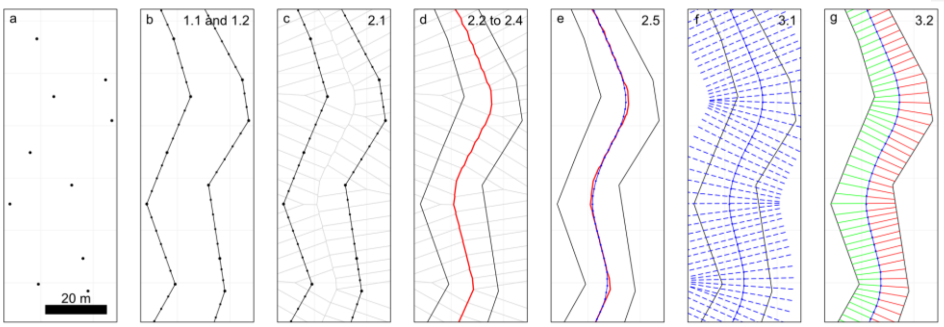

Figure 1: visualization of the work flow of the package, a) the channel bank points represent the data input, b) a polygon is generated where bank points are linearly interpolated, c-d) the centerline is calculated via Voronoi polygons, e) the centerline is spatially smoothed with a mean filter, f) transects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

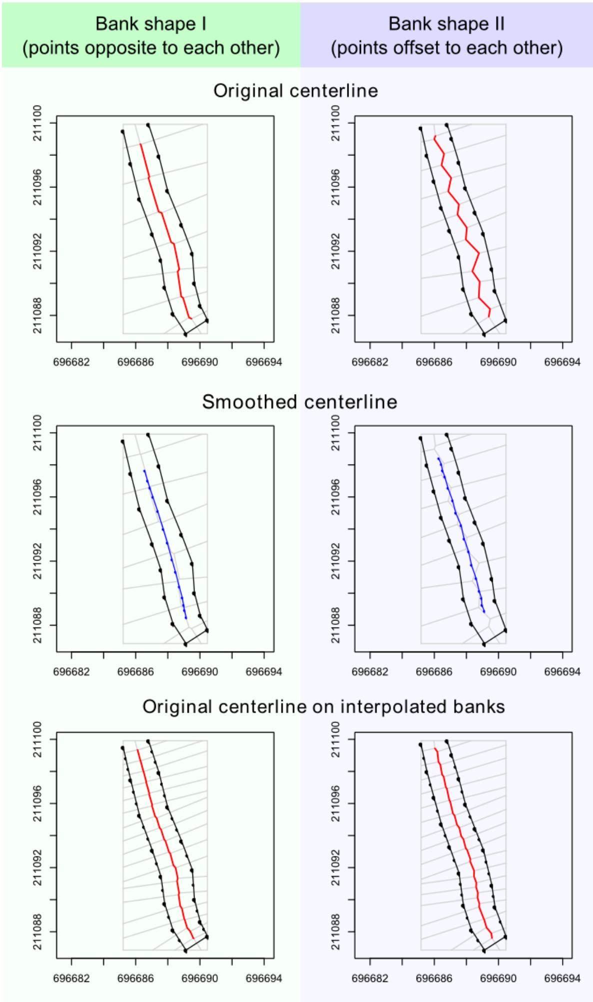

Figure 2: the plot shows two digitizations (Bank shape I and II) of the same channel stretch. They differ only in the arrangement of bank points which are mainly opposite (Bank shape I, left column) or offset (Bank shape II, right column) to each other. One can see how the offset negatively influences the shape of the centerline (top row). The problem can be overcome by smoothing the centerline a-posteriori (middle row) or interpolating between the bank points a-priori (bottom row). A combination of both methods is recommended and set as the default in cmgo.

It follows a detailed description of all steps of the algorithm. In step 1.1~~1.1~~, the algorithm creates a polygon feature from the bank points (Figure 1b), where 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 (~~see~~ Fig. 2). From the interpolated points, Voronoi polygons (also called Dirichlet or Thiessen polygons) are calculated (2.1~~2.1~~, Figure 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

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

single line segments. The segments in the center of the channel polygon form the desired centerline (see Figure 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~~2.2~~, Figure 3b). In a second step, dead ends are removed (step 2.3~~2.3~~, Figure 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~~2.4~~, the centerline segments are chained to one consistent line, the "original" centerline. In the final step 2.5~~2.5~~ of the centerline calculation, the generated line is spatially smoothed (Figure 1e) with a mean filter with definable width (see section 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~~2.6~~. If elevation data is 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~~2.7~~. The program also allows projecting custom geospatial features to the centerline – ~~for example~~such as the abundance of species~~,~~ or the occurrence of knickpoints~~, etc. – if in hand~~ (see section 4.2). Projecting means here that elevation information or other spatial variables are assigned to the closest centerline points.

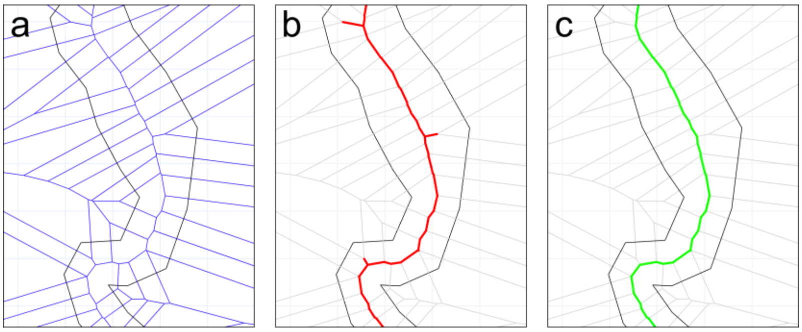

Figure 3: the filtering of the ~~Voronoi~~ centerline segments, ~~(~~a) original Voronoi segments, b) ~~the~~ Voronoi segments filtered for ~~final centerline by first taking only~~ segments that lie fully within the channel polygon, and c ~~(b)~~ ~~and then~~ filtered ~~out~~ for dead ends ~~(c)~~.

To calculate the channel metrics based on the centerline, channel transects are derived (step 3.1~~3.1~~). Transects are lines perpendicular to a group of centerline points. In step 3.2~~3.2~~, the intersections of the transects with the banks are calculated (Figure 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 represent 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.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

Formatiert: Interner Link Zchn, Schriftart: (Standard) Times New Roman, 10 Pt.

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 which are worked through during a project: 1. initialization (loading data and parameters, section 4.1), 2. data processing (calculating centerline and channel metrics, section 4.2), and 3. review of results (plotting or writing results to file, section 4.3).

4.1. INITIALIZATION: INPUT DATA AND PARAMETERS

The package *cmgo* requires basic geometrical information of the points that determine a channel shape – the bank points (Figure 1a) – while in addition ~~of~~ 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” represent 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 to which of the banks the given point is associated. Throughout this paper we refer to left and right of the channel always in regard to these attributes. Thus, the user is generally free to choose which side to name “left”. However, we recommend to stick to the convention to name the banks looking in 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 matter. By default it is expected that the provided list are all bank points in upstream direction. If one – this 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` should be set `TRUE`. The units of the provided coordinates can be specified in the parameter `input.units` and defaults to `m` (meters).

```
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 side and x,y~~,z~~ coordinates.

The data can be either 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 needs 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 `Names` (for the side). The tab delimiter and the expected column names can be changed in the parameters (see SM I for details). The input file(s) – for multiple files see also section 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 Codebox 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 SM I.

```
cmgo.obj = list(
data = list(                # the data set(s), different surveys of the channel
    set1 = list(            # survey 1
      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.

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 filename 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` (defaults 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 section "Input data" 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"~~“demo3”~~ (section 7).

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

4.2. CONTROLLING THE DATA PROCESSING

The processing includes all steps from the input data (bank points) to the derivation of the channel metrics (Figure 1). Next, we describe the parameters that are relevant during the processing described in section ~~3~~3. When generating the channel polygon the original bank points are ~~the spatial resolution of the bank points is increased by~~ linearly interpolated~~ion~~ (Figure 1b)~~ in order to increase the resulting resolution of the channel centerline.~~. 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 of the input coordinates, which means, if input coordinates are given in meters, a value of 6 (default) means that the points have a maximum distance of 6 meters to each other. These parameters have to be determined by the user and are crucial for the centerline generation. Guidance of how to select and test these parameters can be found in paragraph *6. Technical fails and how to prevent them*.

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 ~~is~~ are no further paths to remove. In cases, where the centerline paths exhibit gaps (see section 6), this loop would run 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 (Figure 1e) where 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 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 where 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% the length of the original centerline paths. For the further calculations of transects and channel metrics by default the smoothed version of the centerline is used.

The program will project automatically the elevation of the bank points to the centerline if elevation information is provided in the input files (z component of bank points, see paragraph 4.1). Also additional custom geospatial features – if available to the user – can be projected to the centerline, such as ~~for example~~ the abundance of species or~~,~~ the occurrence of knickpoints~~, etc~~. 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.

```
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.

To calculate the channel metrics based on the centerline channel transects are derived. Transects are lines perpendicular to a group of n centerline points, where ~~the size~~n – also called the transect span ~~– of that group~~ is defined by the parameter `cmgo.obj$par$transects.span`. By default this span equals three, which means 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 Figure 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 is stored separately for the left and the right bank (`d.r`. and `d.l`), as well as a factor (`r.r` and `r.l`) ~~of +/- 1~~ representing the side of the bank with regard to the centerline. Normally, looking downstream the right bank is ~~also~~ 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 section 5.1). A sample result for a reach of a natural channel is provided in Figure 5.

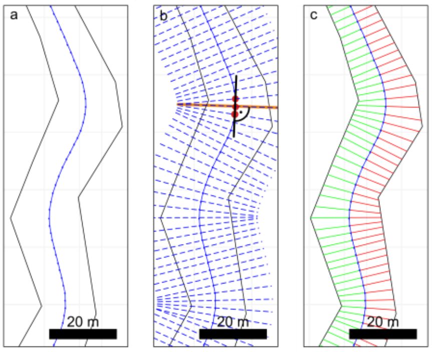

Figure 4: a) the ~~from the~~ smoothed centerline, b) ~~(a)~~ transects are calculated ~~(b)~~ by taking a group of centerline points ~~and~~ , creatinge a line through the outer points and calculate ~~. T~~the perpendicular to that line, c) ~~is the transect. The algorithm now~~ calculating the ~~checks for the~~ intersections of the transects with the channel banks ~~(c)~~.

```
$metrics$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]]).

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

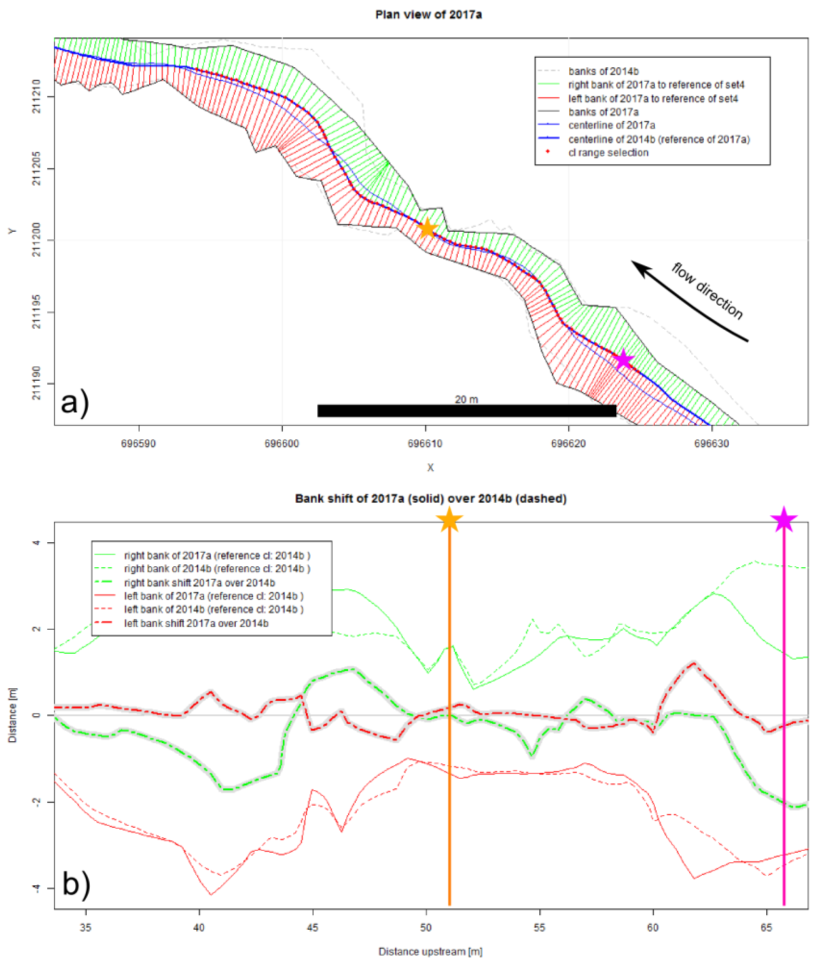

Figure 5: a) plan view of a short channel reach showing two channel surveys, 2014a (dashed channel outline) and 2017a (solid channel outline. A centerline is calculated for both, but due to an enabled reference mode, the centerline of 2014a is used for both surveys. This allows for the calculation of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

analyses. In the plan view plot, multiple ways exists 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). Multiple ways exists to determine the center coordinate: via pre-defined plot extent, via centerline point index, or directly by x,/y–coordinates. Pre-defined plot extents allow for quickly accessing 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 pre-defined extent the name of the vector has to be passed to the plot function as in `CM.plotPlanView(cmgo.obj, 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.

```
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 pre-defined plot extents that allow to quickly plot frequently used map regions. The names, here “e1”, “e2”, “e3”, contain a vector of two elements, the x and y-coordinates where the plot is centered at. To plot a pre-defined region call for example CM.plotPlanView(cmgo.obj, extent=”e2”).

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 section 5.1). The arguments of the function regarding the definition of the plot region is 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 object. If `cmgo.obj$par$workspace.write = TRUE` (default is FALSE) a workspace file is written containing the global data object. The filename 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 filenames 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 SM I.

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 section 4.1). A data set for each file will be created in global data object, 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 filenames. 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 filenames to data sets is printed to the console and stored in each data set under `cmgo.obj$data[[set]]$filename`.

5.1. 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, a centerline for each data set exists. Multiple centerlines prevent a direct comparison of the channel metrics as they can be seen as individual channels. Thus, for temporal comparisons of the channel metrics, two modes exist. Metrics are either calculated for each channel geometry individually. In this mode, the channel metrics are the most accurate representation for that channel observation, for example channel width is most accurately measured, but do 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)~~. This option~~ but should ~~only~~ be used only if the bank surveys differ ~~only~~ 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 Figure 6). To compare channel geometries of which the individual centerlines are not nearly parallel ~~differing like that~~ we recommend to calculate the metrics based on individual centerlines and develop a proper spatial projection for temporal comparisons.

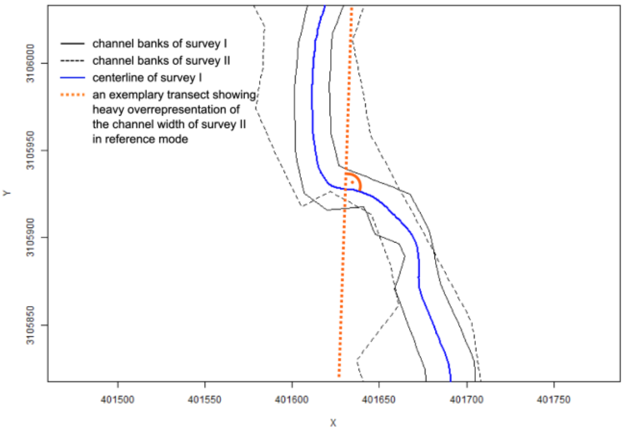

Figure 6: two consecutive channel geometries (surveys I and II) with a profound reorganization of the channel bed. In the reference mode a centerline of one survey is used to build transects. Here, using the centerline of the first survey (blue line) as a reference is not suitable to capture the channel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

```
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).

6. Technical fails and how to prevent them

There are certain geometrical cases in which the algorithm can fail with the default parametrization. To prevent this, a customized parametrization 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 for failure occurs if 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.interpolat.max.dist` less than the average channel width was usually appropriate. Otherwise, the desired centerline segments produced by the Voronoi polygonization can protrude the bank polygon (Figure 7a) and thus do not pass the initial filter of the centerline calculation (see section 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. Thus, if problems with the calculation of the centerline arise, an increase of the spatial resolution of bank points via `cmgo.obj$par$bank.interpolat.max.dist` is advised to naturally smooth the centerline segments (see Figure 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 Figure 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.

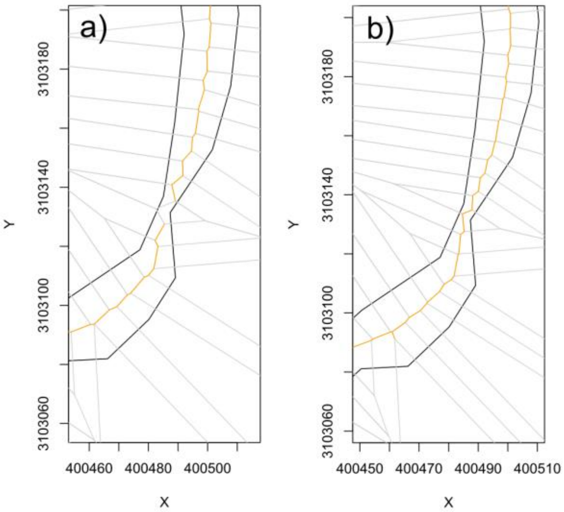

Figure 7: a) a gap in the centerline occurs when the spacing of the bank points is too ~~high~~ large compared to the channel width ~~(left)~~ , b) ~~which can~~ the gap ~~be~~ fixed by ~~previously~~ increasing the resolution of the bank points ~~(right)~~through the parameter par$bank.interpolate.max.dist.

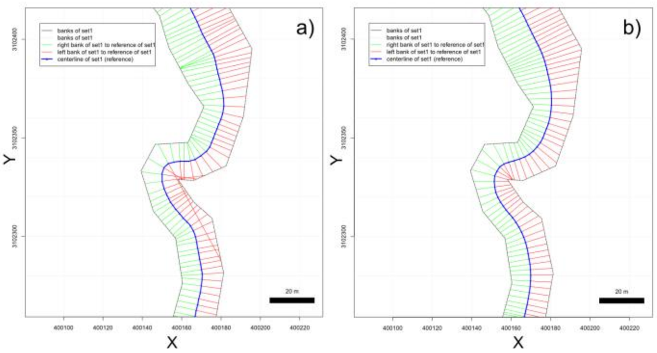

Figure 8: ~~left:~~ a) the transects (perpendiculars to the centerline) do not intersect with banks properly, thus the channel width is overrepresented~~. Right:~~ b) an increased transect span fixes the problem and channel width is ~~now~~ identified correctly.

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

cmgo 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 to investigate 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 SM I. To change a parameter, the new parameter value is assigned directly within the global data object (e.g. `cmgo.obj$par$input.dir = "./input"`).

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 section 5.1).

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

# installation (required only once)
install.packages("cmgo", repos="http://code.backtosquareone.de", type="source")

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

Codebox 78: installation and embedding of the package in R

```
# 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 89: minimal example script to run cmgo with demo data set.

8. Evaluation of the data quality

We evaluated the quality of the derived channel width by cmgo 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 a general agreement (Figure 9). We then identified ~~picked~~ 15 locations randomly (vertical dashed lines Figure 9) ~~, marked with the dashed vertical lines,~~ where we assessed the channel width manually in a GIS (Figure 10). ~~In a GIS we measured channel width manually at these 15 locations on a "best guess" approach.~~

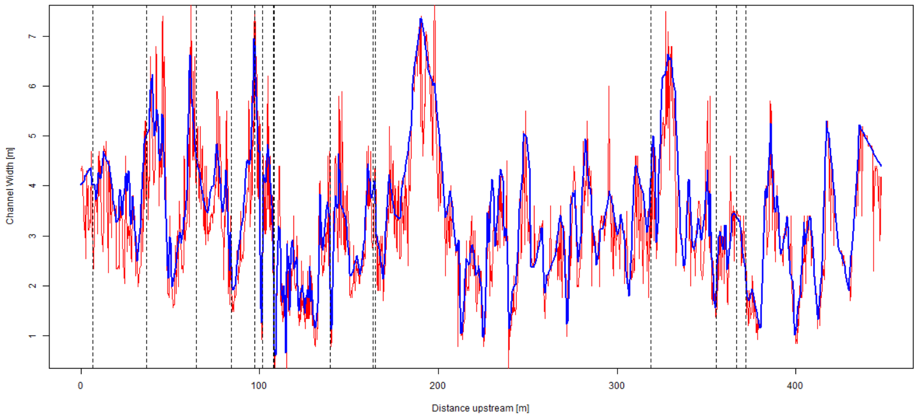

Figure 9: channel width as ~~observed~~ derived by cmgo (blue line) and RivMap (red line) for 1506 locations along a 449 m reach of a natural channel (Figure 10) in upstream direction. The vertical dashed lines mark our points where we investigated the width manually ~~next~~in a GIS.

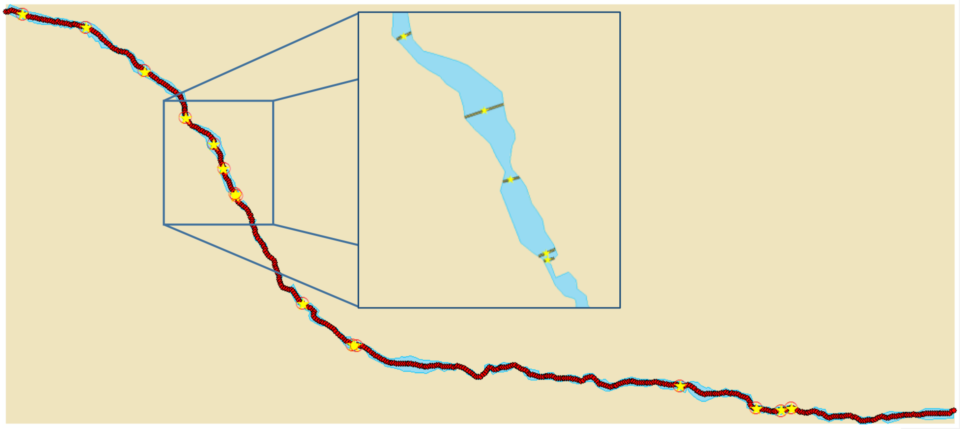

Figure 10: ~~15~~ Fifteen random locations (yellow stars) of the 1506 centerline points (red dots) where we evaluated the width manually in a GIS (example in the inlet) ~~and~~ that are compared to the width of the automated products.

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* ~~seems to consistently~~ 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 Figure~~.~~ 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 (~~30~~31% longer). Looking at the shape of the centerlines (Figure 11) we argue that the centerline of *cmgo* better represents the channel in terms of large scale phenomena. It may for example be more ~~useful~~ accurate for reach-averaged calculations of bankfull flow. The centerline of *RivMap* contains a stronger signal of the micro topography of the banks 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*.

Transect [No.]	Manual approach [m]	cm~~CM~~go width [m]	~~CMgo~~ 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	0.00	3.75	-1.27

3	4.57	4.55	-0.01	4.03	-0.54
4	2.66	2.59	-0.07	2.60	-0.06
5	6.79	6.83	0.04	5.37	-1.41
6	2.82	2.66	-0.15	2.12	-0.70
7	3.02	2.97	-0.06	2.55	-0.48
8	1.76	1.67	-0.09	2.60	0.84
9	2.27	1.93	-0.34	2.60	0.33
10	3.90	3.91	0.01	2.83	-1.07
11	3.82	3.66	-0.17	4.40	0.58
12	4.19	4.14	-0.05	3.04	-1.15
13	2.04	1.89	-0.15	1.34	-0.70
14	3.37	3.37	0.00	3.50	0.13
15	2.14	2.11	-0.03	2.50	0.36
avg.	3.49	3.42	-0.07	3.07	-0.42
st. dev.	1.340	1.399	**0.098**	0.997	**0.736**

Table 3: channel width at 15 randomly selected locations along a natural channel. The width was identified manually in a GIS, by ~~CMgo~~ cmgo, and by RivMap. Differences of the width from the automated products were compared to the manual

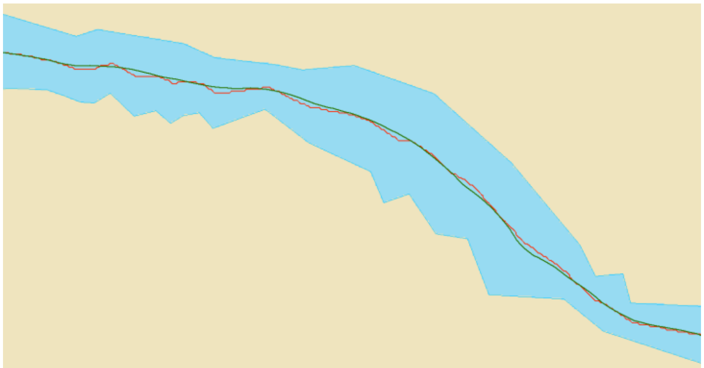

Figure 11: the two different centerlines of the products cmgo (green line) and RivMap (red line) reveal differences in the shape that influence also the channel length.

9. Concluding remarks

The presented package *cmgo* offers a stand-alone solution to calculate channel metrics in an objective and reproducible manner. At this, *cmgo* allows for 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 parametrization. *cmgo* qualifies for 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 style *cmgo* treats the reference of the channels. Thus, our product should be the tool of choice if precise measurements – both in location and quantity – are required 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, as for example the Channel Migration Toolbox, are more suitable.

Since *cmgo* does not come with graphical user interface only static map views of the channel can be obtained by scripting them. *cmgo* offers various plotting functions to do this which allow for predictable and reproducible 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 an easy integration and wide a distribution for scientific or practical use. The license under which the package is provided allows 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).

10. Code and Data availability

All codes and demo data are available at https://github.com/AntoniusGolly/cmgo.

11. Team list

Antonius Golly (Programming, Manuscript), Jens Turowski (Manuscript)

12. Competing interests

The authors declare that they have no conflict of interests.

13. Acknowledgments

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.

References

Ackerman, P. E. C. T. (2011) 'HEC-GeoRAS GIS Tools for Support of HEC-RAS using ArcGIS User's Manual', (February 2011), p. 244.

Amit (2015) *Estimating river Channel Width using Python/ArcGIS/MATLAB/R?, Sep 24, 2015*. Available at: http://gis.stackexchange.com/questions/164169/estimating-river-channel-width-using-python-arcgis-matlab-r (Accessed: 14 March 2017).

Asterics, S. (2013) 'Software-Handbuch ASTERICS, Version 4.1', pp. 1–120. Available at: http://www.fliessgewaesserbewertung.de/downloads/ASTERICS_Softwarehandbuch_Version4.pdf.

Cook, K. L., Turowski, J. M. and Hovius, N. (2014) 'River gorge eradication by downstream sweep erosion', *Nature Geoscience*, 7(9), pp. 682–686. , https://doi.org/10.1038/ngeo2224.

Dilts, T. E. (2015) *Polygon to Centerline Tool for ArcGIS, University of Nevada Reno*. Available at: http://www.arcgis.com/home/item.html?id=bc642731870740aabf48134f90aa6165 (Accessed: 15 March 2017).

ESRI (2017) 'ESRI ArcMap Desktop'. Environmental Systems Research Institute: Redlands, CA: Environmental Systems Research Institute. Available at: http://desktop.arcgis.com/de/.

Ferreira, M. (2014) *Perpendicular Transects.* Available at: http://gis4geomorphology.com/stream-transects-partial/ (Accessed: 15 March 2017).

Ferrer-Boix, C. *et al.* (2016) 'On how spatial variations of channel width influence river profile curvature', *Geophysical Research Letters.* , https://doi.org/10.1002/2016GL069824.

Hornby, D. (2017) *RivEX*. Available at: http://www.rivex.co.uk/Online-Manual/RivEX-Online-Manual.html?Extractchannelwidths.html (Accessed: 15 March 2017).

Jackson, C. W. (2009) *The Ambur project: Analyzing Moving Boundaries Using R, Department of Geology & Geography Georgia Southern University*. Available at: http://ambur.r-forge.r-project.org/ (Accessed: 27 March 2017).

Legg, N. *et al.* (2014) 'The Channel Migration Toolbox: ArcGIS Tools for Measuring Stream Channel Migration', (Publication no. 14-no. 06-no. 032). Available at: https://fortress.wa.gov/ecy/publications/SummaryPages/1406032.html.

Lopez, S. *et al.* (2009) 'Process-Based Stochastic Modelling: Meandering Channelized Reservoirs', in *Analogue and Numerical Modelling of Sedimentary Systems: From Understanding to Prediction*, pp. 139–144. , https://doi.org/10.1002/9781444303131.ch5.

Mandelbrot, B. (1967) 'How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension', *Science*, 156(3775), pp. 636–638. , https://doi.org/10.1126/science.156.3775.636.

Milzow, C. *et al.* (2006) 'Spatial organization in the step-pool structure of a steep mountain stream (Vogelbach, Switzerland)', *Water Resources Research*, 42(4), pp. 1–11. , https://doi.org/10.1029/2004WR003870.

Mir, K., Tariq, A. and Atif, S. (2013) 'River Width Calculator'. Available at: http://www.arcgis.com/home/item.html?id=4e7c9370e3e8455e8ff57d6b23baf760.

Pavelsky, T. M. and Smith, L. C. (2008) ‘RivWidth: A Software Tool for the Calculation of River Widths From Remotely Sensed Imagery’, *IEEE Geoscience and Remote Sensing Letters*, 5(1), pp. 70–73. , https://doi.org/10.1109/LGRS.2007.908305.

Pizzuto, J. E. (2008) ‘Streambank Erosion and River Width Adjustment’, in *Sedimentation Engineering*. Reston, VA: American Society of Civil Engineers, pp. 387–438. , https://doi.org/10.1061/9780784408148.ch07.

R Development Core Team (2011) *R: A Language and Environment for Statistical Computing*, *R Foundation for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. , https://doi.org/10.1007/978-3-540-74686-7.

Schwenk, J. *et al.* (2017) ‘High spatiotemporal resolution of river planform dynamics from Landsat: The RivMAP toolbox and results from the Ucayali River’, *Earth and Space Science*. , https://doi.org/10.1002/2016EA000196.

Thieler, E. R. *et al.* (2009) ‘Digital Shoreline Analysis System (DSAS) version 4.0— An ArcGIS extension for calculating shoreline change’, *U.S. Geological Survey Open-File Report 2008*, p. 1278. Available at: http://woodshole.er.usgs.gov/project-pages/DSAS/.

Wobus, C. *et al.* (2006) ‘Tectonics from topography: procedurses, promise, and pitfalls’, *Geological Society of America Special Paper*, 398(4), pp. 55–74. , https://doi.org/10.1130/2006.2398(04).

Zimmermann, A., Church, M. and Hassan, M. (2008) ‘Identification of steps and pools from stream longitudinal profile data’, *Geomorphology*, 102(3–4), pp. 395–406. , https://doi.org/10.1016/j.geomorph.2008.04.009.