Watershed-scale runoff routing and solute transport in a spatially aggregated hydrological framework

Jairo Arturo Torres Matallana

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

MASTERS PROGRAM IN GEOSPATIAL TECHNOLOGIES

Watershed-scale runoff routing and solute transport in a spatially aggregated hydrological framework

Master thesis

by

Jairo Arturo Torres Matallana

Institute for Geoinformatics, ifgi,
Westfälische Wilhelms-Universität, Münster, Germany

Instituto Superior de Estatística e Gestão de Informação, ISEGI,
Universidade NOVA de Lisboa, Portugal

Deptartamento de Lenguajes y Sistemas Informaticos, LSI,
Universitat Jaume I, Castellón, Spain

February 2014

Watershed-scale runoff routing and solute transport in a spatially aggregated hydrological framework

Dissertation

Supervised by
Prof. Dr. Edzer Pebesma
ifgi - Institute for Geoinformatics
Westfälische Wilhelms-Universität
Münster, Germany

Co-supervised by
Dra. Ana Cristina Costa
ISEGI - Instituto Superior de Estatística e Gestão de Informação
Universidade NOVA de Lisboa
Lisboa, Portugal

Co-supervised by
Dr. Jorge Mateu
LSI - Departamento Lenguajes y Sistemas Informaticos
Universitat Jaume I
Castellón, Spain

Disclaimer

Author's Declaration

I hereby declare that this Master thesis has been written independently by me, solely based on the specified literature and resources. All ideas that have been adopted directly or indirectly from other works are denoted appropriately. The thesis has not been submitted for any other examination purposes in its present or a similar form and was not yet published in any other way.

Signature: ....

Date and Place: 28 February 2014, Münster, Germany

Abstract

This document presents the implementation of several methods for geo-spatial analysis of river networks and watersheds for runoff routing and solute transport in R in order to contribute in a comprehensive hydrological modelling to the current framework of the R package "hydromad".

The main aim of the study is to develop R routines to coupled the outputs of the hydrological framework of the R package "hydromad" to the selected solute transport model looking forward better simulation of water-quality determinants transport at watershed scale.

Following the research scheme presented in this proposal it is possible to prove the hypothesis behind the study. The simulation of solute transport at specific places of the river network was improved by implementing a runoff routing method at watershed-scale, the "hydromad" package, and by coupling it into a suitable modeling framework for representing solute transport processes.

The developed package, "watersheds", allows geo-spatial river network analysis and makes use of the Catchments and Rivers Network System (ECRINS) version 1.1, which constitutes the hydrographical system currently in use at the European Environment Agency as well as widely serving as the reference system for the Water Information System for Europe (WISE). The versatility of the code generated lets to implement geo-spatial analysis in any watershed included into the ECRINS, as a consequence, watersheds along entire Europe could be analyzed. This constitutes an important fact because several institutions or scientific community related with the WISE system could take advantage of the package and this document.

Keywords

Geospatial river networks, hydrological modelling, solute transport, R

Contents

List of Figures

List of Tables

List of Acronyms

AWBM:	Australian Water Balance Model
BfG:	Federal Institute of Hydrology, Germany
CCM:	Catchment Characterisation and Modelling
CMD:	Catchment Moisture Deficit
CORINE:	Coordination of Information on the Environment
CWI:	Catchment Wetness Index
ECRINS:	Catchments and Rivers Network System
ECA&D:	European Climate Assessment & Dataset
EEA:	European Environment Agency
EOBS:	Observational dataset for precipitation, temperature and sea level pressure in Europe
FEC:	Functional Elementary Catchment
HEC-HMS:	Hydrologic Engineering Center - Hydrological Modeling System
IHACRES:	Identification of unit Hydrographs And Component flows from Rainfall, Evaporation and Streamflow
LISFLOOD:	GIS-based hydrological rainfall-runoff-routing model
NetCDF:	network Common Data Form
SASHI:	Sistema de Análise de Simulação Hidrológica

SMA: Soil Moisture Accounting

SWAT: Soil Water Assessment Tool Model

USACE: United States Army Corps of Engineers.

USGS: United States Geological Survey

WATFLOOD: Integrated set of computer programs to forecast flood flows

WaSiM: Water balance Simulation Model

WFD: Water Framework Directive

WISE: Water Information System for Europe

zhyd: ECRINS' primary hydrological unit

1 Introduction

In a comprehensive environmental modelling in the water resources domain, is of paramount importance to understand the interaction between the material flows to coastal waters that are constrained by catchment boundaries and the human activities therein, and those materials that are tied to trade and other trans-boundary processes (e.g. residence time, transport and fate of physical, chemical and microbiological water-quality determinants) and their global implications on preserving the quality of the natural environment.

In a similar sense, applications in river and environmental engineering, specifically related with hydrological modelling, are related to the analysis of rainfall and hydrometric time series in order to implement rainfall-runoff models as a conceptual mathematical basis in flood risk management e.g. the study of the probable maximum precipitation and probable maximum flood for basin water resources management. In this case, the modelling framework is for creating the conceptual basis to simulate flood events in probable scenarios of storm events.

The present document has the purpose of illustrating the implementation of the spatial analysis and the runoff routing and solute transport in the framework of the ECRINS river network a reference system for hydrological and climate change modelling in order to contribute in a comprehensive modelling framework by means of the understanding and representation of the flow celerities dynamics and spatial distribution in the river network at the watershed scale.

In the following sections are presented the justification of the proposal (Section 1.1), the hypothesis behind the study (Section 1.2) and the objectives (Section 1.3). Also, in the Section 2, the preliminary review of literature is introduced; the software and datasets required, and the methods to follow are presented in Section 3. The Section 4 presents the results of the study. The conclusions (Section 5) and some considerations for further work are presented (Section 6). Finally, references and appendices (Section 7) are presented.

1.1 Justification

The ultimate aim of flow prediction using models must be to improve decision making e.g. in water resources planning, flood protection and mitigation of contamination (K. J. Beven, 2012). From the Millenium Development Goals (MDGs) point of view (United Nations - UN, 2012), to secure water-quality and predict floods have an impact on reducing child mortality (Goal 4) and ensure environmental sustainability (Goal 7). Currently, UN (2012) also recognizes that improving monitoring systems is paramount due to *reliable, timely and internationally comparable data on the MDG indicators are crucial for devising appropriate policies and interventions needed to achieve the MDGs and for holding the international community to account.* In this sense, rainfall-runoff models are a primary component in the monitoring system for real-time flow and water-quality prediction.

In the literature exists several hydrological models for rainfall-runoff modelling. However, there are few studies that attempt to model both flow and water-quality in a totally consistent way because before is required to represent adequately the complexity of the system e.g. the dynamics of celerities and the complete distribution of water pore velocities (K. J. Beven, 2012). Some studies that points toward this direction are provided by Botter et al. (2009, 2010) and Duffy (2010).

Moreover, for addressing a comprehensive hydrological modelling, specifically looking forward on the validation of the hydrological cycle, and the transport and fate of sediments and solutes in surface water resources, it is paramount to recognize that in environmental modelling all model structures, regardless of their complexity, are to some extent in error (K. J. Beven, 1989; Grayson et al., 1992; Freer et al., 2004).

Therefore, model comparison in structure, calibration methods and simulation events, is essential for choosing objectively the suitable configuration of the model for addressing a specific task related with hydrological modelling. To accomplish this, a novel, versatile, and open source application is provided with the R Project for Statistical Computing (Ihaka & Gentleman, 1996; R Development Core Team, 2013). R more than a statistical software is a modeling framework that provides for standardised tests and comparisons of models. Also, the R environment allows the reproducibility of methods and results, as is often required by science and research.

The implementation of a method for runoff routing in the river network contributes to the existing hydrological modelling framework and intends for the representation of the solute transport in the river network at the watershed-scale domain.

1.2 Hypothesis

The simulation of solute transport at specific places of the river network (measurement stations) could be improved by implementing a runoff routing method at watershed-scale and by coupling it into a suitable modeling framework for representing transport processes.

1.3 Objectives

- Primary objective
  - To coupled the selected runoff routing model to R for representing solute transport in the river network at the watershed-scale domain.
- Secondary objectives
  - To implement several methods for geo-spatial analysis of river networks in R.
  - To implement the selected runoff routing in R.
  - To define a simulation configuration for testing.

2 Literature review

2.1 Hydrological modelling

Several applications in river and environmental engineering and science, related with hydrological modelling are related to the analysis of rainfall and hydrometric time series in order to implement rainfall-runoff and water-quality models as a conceptual mathematical basis for solute transport and fate assessment. Similarly, such applications and models are common in basin water resources and flood risk management e.g. in the study of the probable maximum precipitation and probable maximum flood. In this case, the application

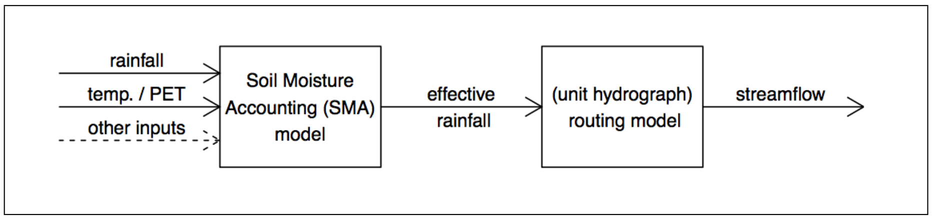
rainfall
temp. / PET
other inputs
Soil Moisture Accounting (SMA) model
effective rainfall
(unit hydrograph) routing model
streamflow

Figure 1: The modelling framework in the hydromad package.

From F. Andrews (2011).

of hydrological models is done as the conceptual basis to simulate flood events in probable scenarios of storm events (Torres & Pebesma, 2013).

Regarding to the challenges for modern hydrological and environmental research as it is depicted in current researches –e.g. McDonnell et al. (2010); Swaney et al. (2011)– is essential to understand and develop a comprehensive modelling framework that includes as an important step the uncertainty analysis in order to identify primary physical controls, and henceforth for coupling, in a most suitable way, inland hydrological models with the coastal system at regional, transboundary and global scales.

Henceforth, in hydrological applications a main aim is to consider a suitable and reproducible modelling framework that takes into account data input, spatial interpolation, calibration and simulation, and includes geospatial capabilities for querying, updating, sharing and visualization of data, methods and results (Torres & Pebesma, 2013). This focus is addressed in the following subsections where is presented in a succinct manner the existing open source software `hydromad` a R package for hydrological modelling.

`hydromad` is an interesting hydrological modelling framework presented by F. T. Andrews, Croke, and Jakeman (2011). The framework is based loosely on the unit hydrograph theory of rainfall-runoff modelling. The documentation of the methods in `hydromad` could be found in the web page of the project (`http://hydromad.catchment.org`). The Figure 1 illustrates the work-flow of the modelling framework in the hydromad package and the following subsections present a description of the three main components of the hydromad package: the Soil Moisture Accounting (Section 2.1.1), the routing models (Section 2.1.2) and the calibration methods (Section 2.1.3). The Figure 2 resumes the components and structure of the package `hydromad` (F. Andrews, 2011).

2.1.1 Soil Moisture Accounting models

`hydromad` counts with 11 Soil Moisture Accounting (SMA) models (F. Andrews, 2014): 1) the Catchment Moisture Deficit (CMD) an effective rainfall model for IHACRES. It is a conceptual-type model, where input rainfall is partitioned explicitly into drainage, evapo-transpiration, and changes in catchment moisture; 2) the Catchment Wetness Index (CWI) effective rainfall model for IHACRES. This is the classic model of Jakeman and Hornberger (1993), with the extensions to ephemeral catchments of Ye et al. (1997); 3) the GR4J model (mode'le du Ge´nie Rural a' 4 parame'tres Journalier); 4) the Australian Water Balance Model (AWBM): simple 3 bucket model; 5) bucket: the Single-bucket Soil Moisture Accounting models with saturated/unsaturated zones and interception; 6)

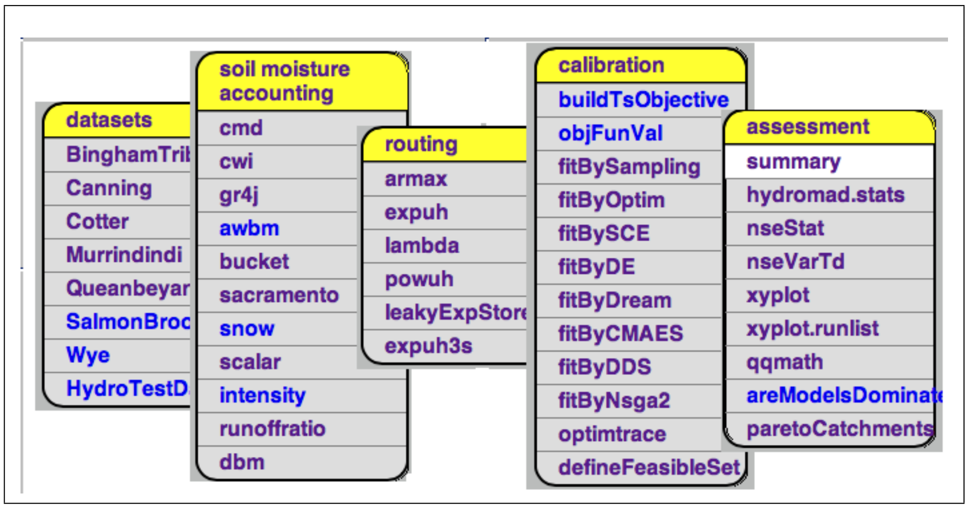

Figure 2: Overview of the `hydromad` package.

Adapted from F. Andrews (2014).

the Sacramento Soil Moisture Accounting model. Developed by the US National Weather Service; 7) snow: a simple degree day factor snow model coupled with IHACRES CMD soil moisture model; 8) scalar: a simple constant runoff proportion: a constant fraction of rainfall reaches the stream; 9) intensity: Runoff as rainfall to a power. This allows an increasing fraction of runoff to be generated by increasingly intense/large rainfall events (for power > 0). The fraction increases up to a full runoff level at maxP; 10) runoffratio: simple time-varying runoff proportion. Rainfall is scaled by the runoff coefficient estimated in a moving window; and 11) dbm: Typical initial model used in Data-Based Mechanistic modelling. Rainfall is scaled by corresponding streamflow values raised to a power.

The models 10 and 11 use streamflow data, so can not be used for prediction.

2.1.2 Routing models

`hydromad` counts with 6 routing models (F. Andrews, 2014): 1) armax: the ARMAX linear transfer functions with a single input and single output series. Can be used as a general Unit Hydrograph transfer function, defined by Auto-Regressive and Moving Average coefficients; 2) expuh: A unit hydrograph (linear transfer function) defined as a system of exponentially receding components. Each component is defined by its time constant and fractional volume, and if there are multiple (up to 3) such components they may be in a parallel and/or series configuration; 3) lambda: Lambda unit hydrograph. Transfer function with two exponential components and variable partitioning; 4) powuh: a power-law form of unit hydrograph (transfer function); 5) leakyExpStore: an exponential store (linear transfer function) which has a loss term, produces no flow when the store drops below a level, and can therefore model longer-term disconnection of a store from streamflow.; and 6) expuh3s: a unit hydrograph with a quickflow pathway and two layered slow-flow pathways modelling recharge to groundwater in order to allow modelling of long-term disconnection of slow-flow stores from streamflow.

2.1.3 Calibration methods

Accordingly with F. Andrews (2011), currently implemented calibration methods in `hydromad` include simple sampling schemes (fitBySampling), general optimisation methods with multistart or presampling (fitByOptim) and the more sophisticated Shuffled Complex Evolution (fitBySCE) and Differential Evolution (fitByDE) methods. All attempt to maximise a given objective function. Other 8 calibration algorithms are available (see F. Andrews (2014)).

Accordingly to F. Andrews (2014) the ”fitBySampling” method fit a hydromad model by sampling the parameter space. Returns best result from sampling in parameter ranges using random, latin hypercube sampling, or a uniform grid (all combinations). The function also retains the parameter sets and objective function values, which can be used to define a feasible parameter set. The ”fitByOptim” method fits a hydromad model using R’s optim or nlminb functions. Has multi-start and pre-sampling options. The ”fitBySCE” fit a hydromad model using the SCE (Shuffled Complex Evolution) algorithm, and finally, the ”fitByDE” fit a hydromad model using the DE (Differential Evolution) algorithm.

2.2 Geo-spatial and Geo-temporal capabilities

Several packages developed in the programming language R Project for Statistical Computing (Ihaka & Gentleman, 1996; R Development Core Team, 2013) are available for Geo-spatial and Geo-temporal analysis. The packages `sp` (Pebesma & Bivand, 2005-2012), `rgeos` (Bivand & Rundel, 2012), `rgdal` (Bivand et al., 2003-2013), `maptools` (Lewin-Koh & Bivand, 2012), `raster` (Hijmans, 2014), `Lattice` (Sarkar, 2012), `multicore` (Urbanek, 2013) and `Watersheds` was used in the present study and are described in the following subsections.

The present subsection has the purpose of introducing the packages developed in the programming language R available for Geo-spatial and Geo-temporal analysis and used and implemented in the present work.

2.2.1 `sp`

The package `sp` (Pebesma & Bivand, 2005-2012), provides classes and methods for spatial data. The classes document where the spatial location information resides, for 2D or 3D data. Utility functions are provided, e.g. for plotting data as maps, spatial selection, as well as methods for retrieving coordinates, for subsetting, print, and summary.

The package has i.a. the class `SpatialPolygons` which is a data object equivalent to an ESRI polygon shapefile containing information for polygons, and additional similar definitions for spatial points and lines are defined through the objects `SpatialPoints` and `SpatialLines`, respectively.

2.2.2 `rgeos`

This package developed by Bivand and Rundel (2012), is an interface to Geometry Engine - Open Source (GEOS) using the C API for topology operations on geometries. The packages provides methods and functions for geospatial analysis i.a. garea, gBoundary, gBuffer, gCentroid, gContains, gConvexHull, gCrosses, gDifference, gDistance, gEnvelope, gEquals, gIntersection, gIntersects, gRelate, gSimplify, gSymdifference, gTouches, gUnion, SpatialCollections, SpatialRings.

2.2.3 rgdal

A binding package for the Frank Warmerdam's Geospatial Data Abstraction Library (GDAL, http://www.gdal.org) is available in R through the package rgdal (Bivand et al., 2003-2013). It allows to deploy multiple classes defined in the sp package and access to the projection/transformation operations from the PROJ.4 library (https://trac.osgeo.org/proj/) and to the OGR library. The OGR Simple Features Library is a C++ open source library for reading, and in some cases writing, a variety of vector file formats including ESRI Shapefiles and PGDBs .mdb files via ODBC (Warmerdam, 2013). Therefore, using rgdal both GDAL raster and OGR vector map data can be imported into R and exported, and the ECRINS database could be handled properly.

2.2.4 maptools

The maptools package (Lewin-Koh & Bivand, 2012) is a set of tools for manipulating and reading geographic data, in particular ESRI shape- files; C code used from shapelib. It includes binary access to GSHHS shoreline files. The package also provides interface wrappers for exchanging spatial objects with packages such as PBSmap- ping, spatstat, maps, RArcInfo, Stata tmap, WinBUGS, Mondrian, and others.

2.2.5 raster

The raster package (Hijmans, 2014) has capabilities for reading, writing, manipulating, analyzing and modeling of gridded spatial data. The package implements basic and high-level functions and processing of very large files is supported.

2.2.6 lattice

Lattice (Sarkar, 2012), is a powerful and elegant high-level data visualization system, with an emphasis on multivariate data, that is sufficient for typical graphics needs, and is also flexible enough to handle most nonstandard requirements.

2.2.7 multicore

This package (Urbanek, 2013) provides a way of running parallel computations in R on machines with multiple cores or CPUs. Jobs can share the entire initial workspace and it provides methods for results collection.

2.2.8 `Watersheds`

The package `Watersheds` developed by the author for the present work, allows spatial analysis for watersheds aggregation and ordering accordingly to an outlet point and size of tributary watershed of the current watershed. Also, enables spatial drainage networks analysis inside the aggregated watersheds. It makes use of the functionalities of the spatial classes, functions and methods of the R package `sp` (Pebesma & Bivand, 2005-2012). Also is build on the capabilities of the R packages `rgeos`, `maptools`, `lattice`, `splancs`, and `multicore`.

The `Watersheds` package allows creation and handling of objects class `Watershed` for identifying the subbasin that contains the current `station` (class `Spatial- Points`) and subsets the `zhyd` object to subbasin and extract the current `zhy` object that contains `station` via the S4 method `Watershed.Order`. Identification of the inlet and outlet stretches and inlet and outlet nodes of the `zhyd`. Implementation of the functions `Watershed.IOR1, .IOR2, .IOR3`, and `.IOR4` for determining the actual inlet and outlet nodes. S4 methods `Watershed.Order2` and `Watershed.Tributary` for defining tributary nodes and tributary catchments of the current `zhyd` watershed.

2.3 Runoff routing and solute transport

A large-scale runoff routing with an aggregated network-response function is presented by Gong et al. (2009). A scale dependency of routing dynamics is evaluated, as well as the flow velocities and the routing performance at different spatial resolutions. Also, some limitations of aggregated networks are evaluated.

An example of runoff routing at large scales that involves development of low-resolution flow networks, with spatial resolutions of which range from 1 Km is presented with the model HYDRO1k (USGS, 1996).

Regarding to identify different schemes of runoff routing in the river network, some distributed and semi-distributed models could be evaluated i. a. WATFLOOD (Kouwen, 1988); TOPMODEL (K. Beven et al., 1995; K. J. Beven, 1997; Buytaert, 2012); SASHI (Sistema de Análise de Simulação Hidrológica, INPE, Rennó (2003)); SCS-TerraMe (INPE, Pereira (2009)); LISFLOOD (van der Knijff et al., 2010); SWAT (Soil Water Assessment Tool Model, Neitsch et al. (2011)); WaSiM (Water balance Simulation Model, Schulla (2012)); and aggregated models as the Hydrological Modeling System HEC-HMS (USACE, HEC, 2006).

2.3.1 General reaction transport equation in 1-Dimension

Accordingly with Soetaert and Meysman (2012), the general 1-D reaction-transport equation in multi-phase environments and for shapes with variable geometry is:

$$\frac{\partial \xi C}{\partial t} = -\frac{1}{A} \cdot \frac{\partial (A \cdot J)}{\partial x} + reac \qquad (1)$$

where [1]

- t is time

[1] here the units are M for Mass, L for Length and t for time

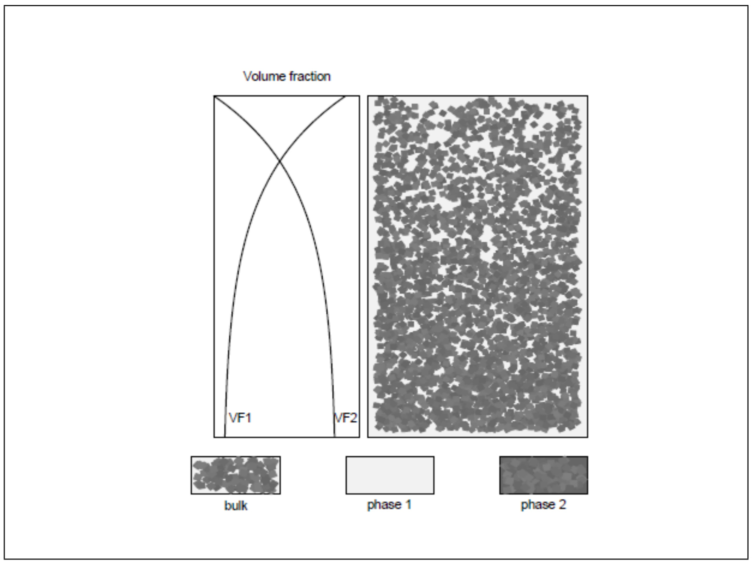

Figure 3: An illustration of multiple phases in ReacTran. From Soetaert and Meysman (2012), Figure 1

- x is space
- C is concentration of a substance in its respective phase
- ξ is the volume fraction (-), i.e. the fraction of a phase in the bulk volume (see Figure 3). In most of cases, when one phase is considered $\xi = 1$. For sediments, ξ would be porosity (solutes), or 1-porosity (solids)
- A is the total surface area (L^2)
- J are fluxes ($ML^{-2}t^{-1}$)

The Fluxes, J, are estimated per unit of total surface, and represents a dispersive and a advective component:

$$J = -\xi D \cdot \frac{\partial C}{\partial x} + \xi u \cdot C \tag{2}$$

where:

- D is the diffusion (or dispersion) coefficient (L^2t^{-1})
- u is the advection velocity (Lt^{-1})

2.3.2 Boundary conditions in 1-D models

Accordingly with Soetaert and Meysman (2012), the boundaries at the extremes of the model domain e.g. at $x = 0$ could be one of the following options:

- A concentration boundary, $C|_{x=0} = C_0$
- A diffusive + advective flux boundary, $J_{x=0} = J_0$
- A boundary layer convective exchange flux boundary $J_{x=0} = a_{bl} \cdot (C_{bl} - C_0)$

2.3.3 Numerical approximation of the Advection Dispersion Equation

Following Soetaert and Meysman (2012), the reaction-transport formula is a partial differential equation (PDE), as a consequence it is solve by approximating the spatial gradients using the numerical differences by the method-of-lines, MOL, approach. This converts the PDE into ordinary differential equations (ODE).

Thus, the model is divided into a number of grid cells, and for each grid cell i is writen:

$$\frac{d\xi_i C_i}{\partial t} = -\frac{1}{A} \cdot \frac{\Delta_i \left(A \cdot J\right)}{\Delta x_i} + reac_i \tag{3}$$

where Δ_i denotes that the flux gradient is to be taken over box i, and Δx_i is the thickness of the box i:

$$\Delta_i \left(A \cdot J\right) = A_{i,i+1} \cdot J_{i,i+1} - \left(A_{i-1,i} \cdot J_{i-1,i}\right) \tag{4}$$

where $i, i+1$ denotes the interface between box i and $i+1$.

The fluxes at the box interfaces are discretized as:

$$J_{i-1,i} = -\xi_{i-1,i} D_{i-1,i} \cdot \frac{C_i - C_{i-1}}{\Delta x_{i-1,i}} + \xi_{i-1,i} u_{i-1,i} \cdot \left(\vartheta_{i-1,i} \cdot C_{i-1} + \left(1 - \vartheta_{i-1,i}\right) \cdot C_i\right) \tag{5}$$

where $\Delta x_{i-1,i}$ is the distance between the centre of the grid cells $i-1$ and i, and ϑ the upstream weighing coefficients for the advective term.

2.3.4 1-D finite difference grids and properties in ReacTran

the spatial discretization grid could be generated with the function setup.grid.1D of the package ReacTran. The generated grid comprises several zones:

```
setup.grid.1D = function(x.up = 0,x.down = NULL, L = NULL, N = NULL,
    dx.1 = NULL, p.dx.1 = rep(1,length(L)), max.dx.1 = L,
    dx.N = NULL, p.dx.N = rep(1,length(L)), max.dx.N = L)
```

with the following arguments:

- x.up, the position of the upstream boundary
- x.down, the position of the downstream boundaries in each zone
- L, N, the thickness and the number of grid cells in each zone.
- dx.1, p.dx.1, max.dx.1, the size of the first grid cell, the factor of increase near upstream boundary, and maximal grid cell size in the upstream half of each zone
- dx.N, p.dx.N, max.dx.N, the size of the last grid cell, the factor of increase near the downstream boundary, and maximal grid cell size in the downstream half of each zone

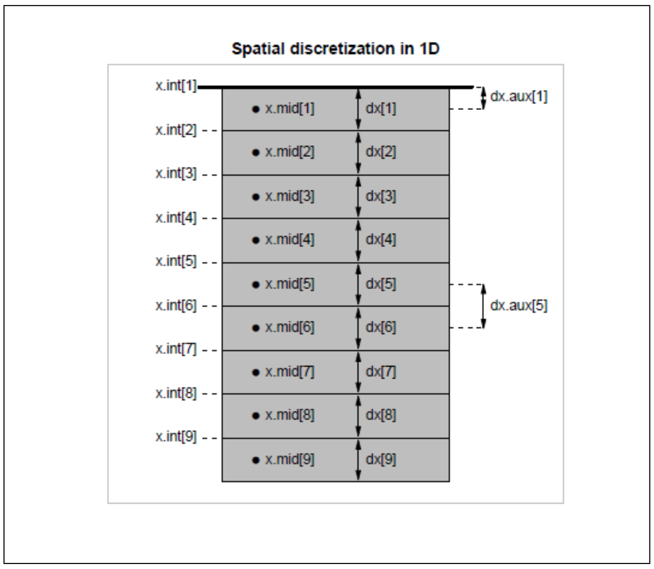

Figure 4: Spatial 1-D discretization in ReacTran. From Soetaert and Meysman (2012), Figure 2

The function returns an element of class grid.1D that contains the following elements (units L) (see Figure 4):

- x.up, x.down, the position of the upstream and downstream boundary
- x.int, the position of the grid cell interfaces, where the fluxes are specified, a vector of length N+1
- x.mid, the position of the grid cell centres, where the concentrations are specified, a vector of length N. This is equivalent to $\Delta x_{i-1,i}$
- dx, the thickness of boxes, i.e. the distance between the grid cell interfaces, a vector of length N. Equivalent to Δx_i
- dx.aux, the distance between the points where the concentrations are specified, a vector of length N+1. This is equivalent to $\Delta x_{i-1,i}$

For example, to represent a subdivision of a river streach of 100 Km long into 50 boxes, with the first box size of 1 Km, is established by:

```
grid = setup.grid.1D(L=90, dx.1=1, N=50)
```

and the grid is plotted with the command:

```
plot(grid)
```

2.3.5 Stability

The stability criteria followed for determining the relation between the temporal interval and the residence time in a finite volume is given by the Courant number (Chaudhry, 2008, p. 375):

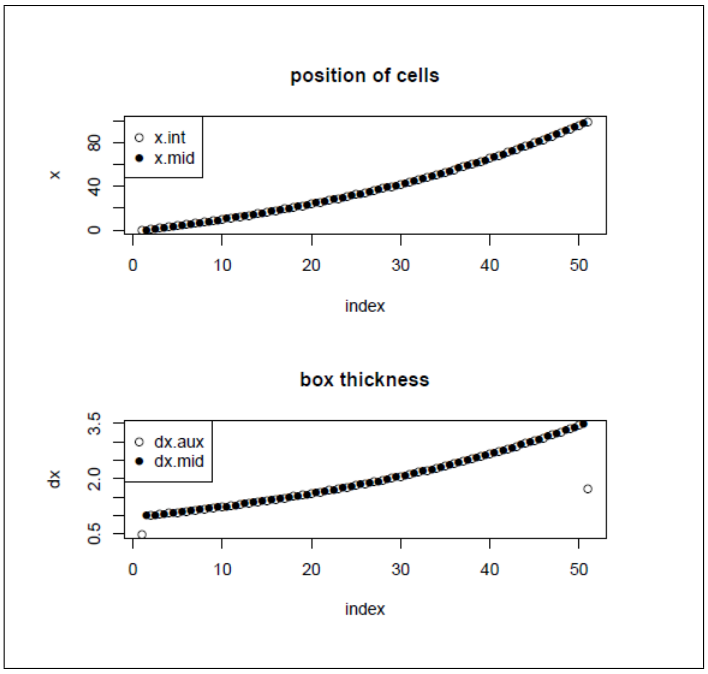

Figure 5: Exponential grid cell size in ReacTran. From Soetaert and Meysman (2012), Figure 3

$$C = \frac{\Delta t}{\frac{\Delta x}{u}} = u\frac{\Delta t}{\Delta x} \tag{6}$$

where:

- C is the Courant number
- Δt is the time interval
- Δx is the space interval
- u is velocity

2.4 R Packages for routing and solute transport modelling

2.4.1 `ReacTran`

The R package `ReacTran` contains routines that enable the development of reactive transport models in aquatic systems (rivers, lakes, oceans), porous media (floc aggregates, sediments,...) and even idealized organisms (spherical cells, cylindrical worms,...) (Soetaert & Meysman, 2012).

The geometry of the model domain is either one-dimensional, two-dimensional or three-dimensional. The package contains (Soetaert & Meysman, 2012):

- Functions to setup a finite-difference grid (1D or 2D)
- Functions to attach parameters and properties to this grid (1D or 2D)

- Functions to calculate the advective-diffusive transport term over the grid (1D, 2D, 3D)

2.4.2 `deSolve`

Citing Soetaert, Petzoldt, and Setzer (2013) from the users manual, the package `deSolve` provides "Functions that solve initial value problems of a system of first-order ordinary differential equations (ODE), of partial differential equations (PDE), of differential algebraic equations (DAE), and of delay differential equations. The functions provide an interface to the FORTRAN functions lsoda, lsodar, lsode, lsodes of the ODEPACK collection, to the FORTRAN functions dvode and daspk and a C-implementation of solvers of the Runge-Kutta family with fixed or variable time steps. The package contains routines designed for solving ODEs resulting from 1-D, 2-D and 3-D partial differential equations (PDE) that have been converted to ODEs by numerical differencing".

2.4.3 `rootSolve`

Accordingly Soetaert (2014) from the users manual, the package `rootSolve` provides "routines to find the root of nonlinear functions, and to perform steady-state and equilibrium analysis of ordinary differential equations (ODE). Includes routines that: (1) generate gradient and Jacobian matrices (full and banded),(2) find roots of non-linear equations by the Newton-Raphson method,(3) estimate steady-state conditions of a system of (differential) equations in full, banded or sparse form, using the Newton-Raphson method, or by dynamically running, (4) solve the steady-state conditions for uni-and multicomponent 1-D, 2-D, and 3-D partial differential equations, that have been converted to ODEs by numerical differencing (using the method-of-lines approach)."

3 Materials and methods

This section presents a summary of the specific techniques used in the study, procedures, statistical design, and data collection and analysis.

3.1 Datasets

Primary datasets for the present study are defined in the following subsections.

3.1.1 The ECRINS dataset

The European Environment Agency (EAA) has been developed the Catchments and Rivers Network System (ECRINS) version 1.1 (EAA, 2012). The ECRINS is the hydrographical system currently in use at the EEA as well as widely serving as the reference system for the Water Information System for Europe (WISE)(EAA, 2012, p. 49).

3.1.2 Water quality determinants

The following physical and water quality determinants are available at Department of Hydrometry and Hydrological Survey of the Federal Institute of Hy-

Table 1: Flow and level measurement stations, river Weser

GRDC Number	National ID	River	Station name	Latitude	Longitude	Area	Altitude
6337400	43100109	WESER	HANN.-MUENDEN	51.426	9.641	12442	114.95
6337519	43900105	WESER	WAHMBECK	51.625	9.52	12996	98
6337516	45100100	WESER	KARLSHAFEN	51.648	9.438	14794	94.05
6337514	45300200	WESER	BODENWERDER	51.973	9.516	15924	69.39
6337100	45900208	WESER	VLOTHO	52.176	8.862	17618	41.66
6337518	47100100	WESER	PORTA	52.249	8.922	19162	37.04
6337517	47500200	WESER	LIEBENAU	52.594	9.113	19910	20
6337515	47900209	WESER	DOERVERDEN	52.852	9.211	22110	7.99
6337200	49100101	WESER	INTSCHEDE	52.964	9.125	37720	4.79

drology (BfG):

- water level, cm
- water temperature, degree Celsius
- conductivity, μS/cm
- pH, pH units
- oxygen content, mg/l
- turbidity, TE/F

3.1.3 River discharge stations

The data for flow level and discharge are also available at Department of Hydrometry and Hydrological Survey of the German Federal Institute of Hydrology. The Table 1 presents the details of the nine stations analyzed in this study and Figure 6 shows their location.

More stations in Germany could be used for implementing the methods for runoff routing and solute transport analysis. Figure 7 presents the available measurement stations in Germany, and Figures 8 and 9 show, as an example, the river Rhein level retrieved at Köeln station and Düsseldorf, respectively.

3.1.4 Further datasets available

- The world-wide repository of river discharge data and associated metadata of the Global Runoff Data Centre - GRDC (2013).
- Water levels data at selected gauging stations on German federal waterways from the German Federal Institute of Hydrology - BfG (2013).
- The land cover dataset from European Environment Agency - EAA (1995). This data project is part of the CORINE programme and is intended to provide consistent localized geographical information on the land cover of the Member States of the European Community.
- Climate data for Germany from the Federal Ministry of Transport, Building and Urban Development `http://www.dwd.de/`. From this is created the layer "dwd_PrecipitationStations" and the ODS spreadsheet "DWD_precipitation_stations.csv-ODS"

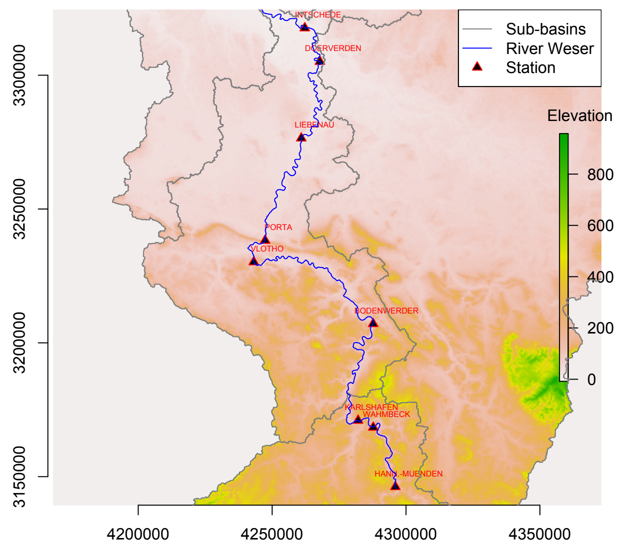

Figure 6: Flow and level stations on river Weser

3.2 Methodology

The Catchments and Rivers Network System (ECRINS) version 1.1. from the EAA (2012) is the hydrographical system currently in use at the European level as well as widely serving as the reference system for the Water Information System (WISE).

According with the European Water Framework Directive (WFD), the smallest unitary catchment suggested is 10 Km^2. The overall aim of ECRINS, however, is to centre the watersheds between 50 km^2 and 100 km^2, since such a small area is not compatible with production constraints and the source data available. The FEC, or Functional Elementary Catchment, stands as the central element of ECRINS. FEC refers to the smallest catchment identified as an ECRINS elementary catchment. A FEC is built via aggregating elementary CCM (Catchment Characterisation and Modelling) catchments. It could be either a 'continental FEC' when built by aggregating elementary CCM catchments from a non-coastal basin, or a 'coastal FEC' when elementary CCM catchments belong to a coastal basin (EAA, 2012, p. 49).

The average area of possible FEC building from basins at Strahler 3 level is 39 Km^2, which is compatible with both this WFD threshold and specific requirements; using basins at level 4 would not allow small enough FECs (EAA, 2012, p. 49). The Figure 10 presents an illustration of the Stralher stream order 1 to 4.

The *database access and manipulation* of the ECRINS dataset, which basically has been delivery in different layers and ancillary tables in MS Access®

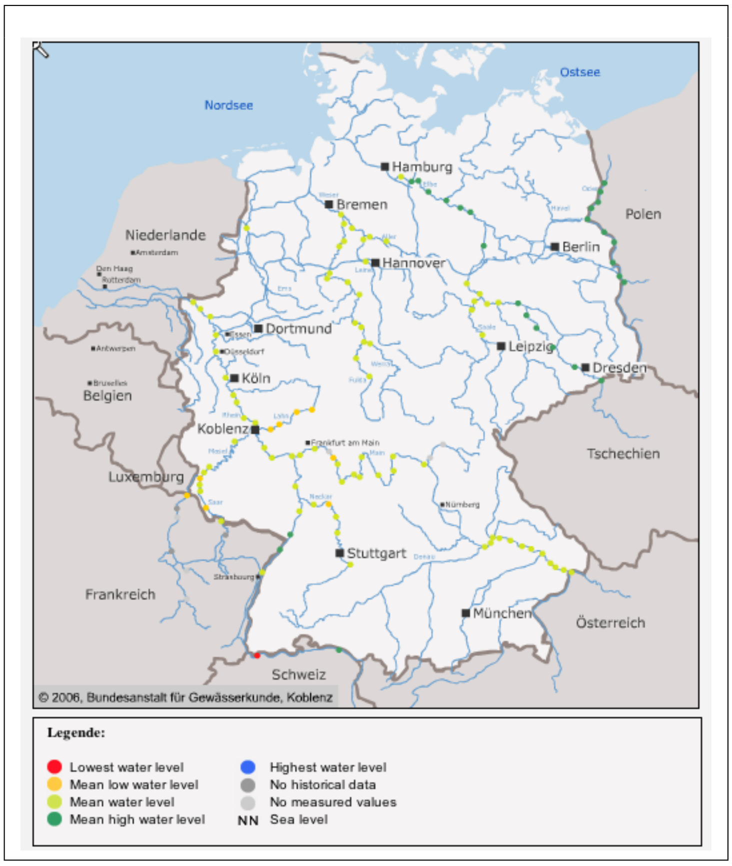

Figure 7: Flow and level stations at Germany available in the BfG portal

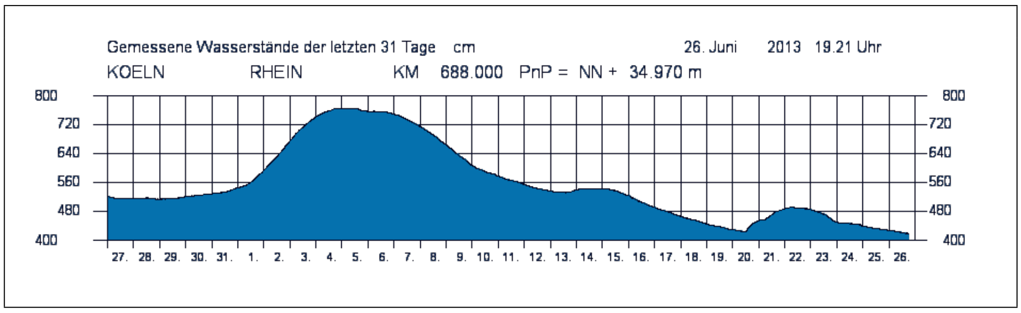

Figure 8: River Rhein level at Köeln station

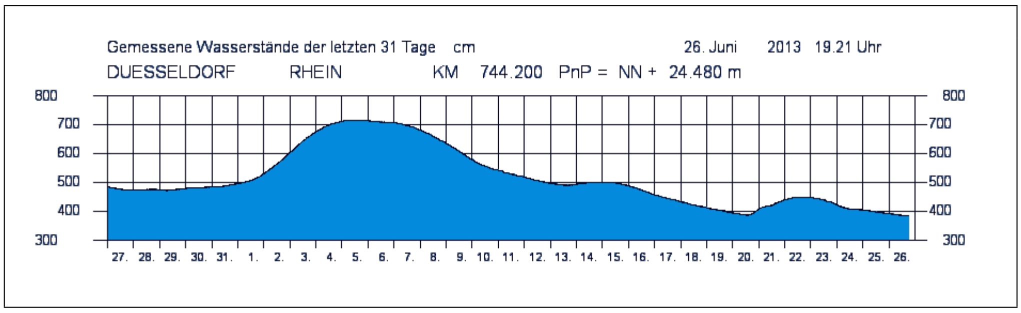

Figure 9: River Rhein level at Düsseldorf station

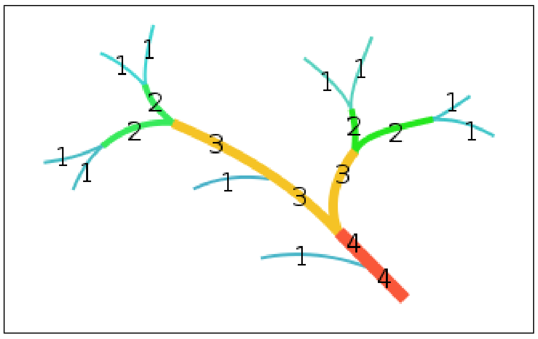

Figure 10: Strahler stream order. Illustration.

Personal GeoDatabases (PGDBs) format (a Microsoft® proprietary format, handed with both MS Access® and ArcGIS ®), is done by using open source GIS methods and database managers. In this sense, R packages as `foreign` (R Core Team et al., 1999-2013) for importing a .dbf file into a R dataframe, and the S4 methods for manipulating spatial data provided by `sp` (Pebesma & Bivand, 2005-2012) was applied.

The followed methodology was to create a R package ("Watersheds") for geospatial analysis of the ECRINS river network for runoff routing and watershed aggregation based on the order of contribution of tributaries watersheds (accordingly Strahler order) in the basin of the river Weser, Germany. Although the site of study is defined in the package, it is possible to implement similar analysis for other places contained into the ECRINS dataset (European level).

After implementing the geospatial analysis a method for runoff routing and solute transport is developed based on the solution of the advection-dispersion equation in one dimension and steady state for routing and trace a water quality determinant (e.g. organic carbon, OC) between two stations along the river Weser. Beyond the focus of this work, numerical homologous frameworks could be developed for the cases of bi- and three-dimensional frameworks, for example for simulating pollutants dispersion in a estuary, a lake or the ocean. Also, the case of unsteady state could be implemented.

3.3 Site study: river Weser basin, Germany

The site study is presented along with the package "Watersheds". The package has an example dataset of the ECRINS database for the river Weser basin, Germany. The European Environment Agency (EEA) has been developed the Catchments and Rivers Network System (ECRINS) version 1.1. The ECRINS is the hydrographical system currently in use at the European level as well as widely serving as the reference system for the Water Information System (WISE). The current version of ECRINS is based on previous work carried out by the Joint Research Centre (JRC) Catchment Characterisation and Modelling (CCM) and the EEA (European Lakes, Dams and Reservoirs Database (Eldred2), European Rivers and Catchments (ERICA), (EAA, 2012).

3.3.1 Subsets

The dataset contains the following subsets:

- `basin`: an object `SpatialPolygonsDataFrame` as is defined in package `sp` that represents the river Weser basin. The `data` slot contains 6 variables as attributes of 1 observation.

- `ctry`: an object `SpatialPolygonsDataFrame` as is defined in package `sp` that represents the administrative boundary of Germany. The `data` slot contains 6 variables as attributes of 1 observation.

- `node:` an object `SpatialPointsDataFrame` as is defined in package `sp` that represents the nodes of the ECRINS river network of the river Weser basin. The `data` slot contains 13 variables as attributes of 3882 observations.

- `rAller` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Aller, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 88 observations.

- `rDiemel` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Diemel, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 39 observations.

- `rFulda` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Fulda, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 82 observations.

- `rHunte` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Hunte, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 34 observations.

- `river` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the ECRINS river network of the river Weser basin. The `data` slot contains 52 variables as attributes of 3874 observations.

- `rWerra` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Werra, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 120 observations.

- `rWeser` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Weser. The `data` slot contains 74 variables as attributes of 104 observations.

- `rWiumme` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Wiumme, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 18 observations.

- `station` an object `SpatialPoints` as is defined in package `sp` that represents a point of interest for which the watershed will be aggregated an ordered. Could be a point with the coordinates of a measurement station.

- `subbasin` an object `SpatialPolygonsDataFrame` as is defined in package `sp` that represents the subbasins of the tributaries of the river Weser. The `data` slot contains 4 variables as attributes of 4 observations.

- `zhyd` an object `SpatialPolygonsDataFrame` as is defined in package `sp` that contains the primary hydrological units of the river Weser basin accordingly with ECRINS. The `data` slot contains 50 variables as attributes and 915 observations.

Some examples for visualising the dataset are presented in the following Figures. Figure 11 illustrates the River Weser basin location into the German territory. The river Weser is formed after the confluence of the rivers Werra and Fulda. The Figure 12 presents the River Weser subbasin and its main tributaries: the rivers Wümme, Aller, Hunte and Diemel, and its former rivers Werra and Fulda. The Figure 14 shows the River Weser and its intersecting zhyd subbasins along its trajectory, which represent the primary hydrological units that contribute with the runoff toward the main course.

The Figure 14 presents all the zhyd subbasins in the entire basin of the river Weser and the Figure 15 shows the entire river network of the river Weser basin. From these last two Figures it is possible to understand the necessity for developing a spatial analysis of tributary zhyd (primary hydrological units in the terminology of the ECRINS dataset) units, identifying the current zhyd under analysis and its subsequent zhyd tributaries and defining the inlet and

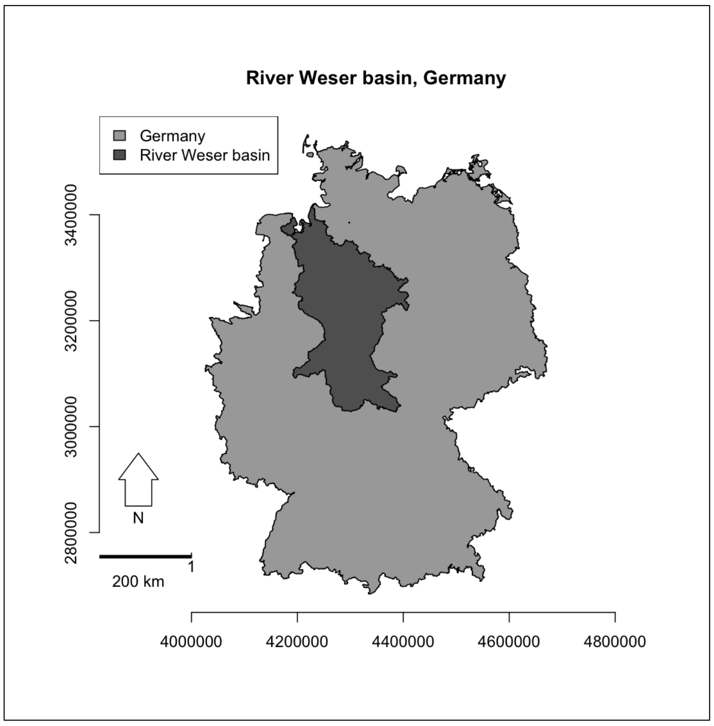

Figure 11: River Weser basin

outlet nodes of each zhyd and the river network inside them. This is the main effort for developing the package ”Watersheds” a contribution for geo-spatial analysis of the river network of one zhyd unit.

4 Results

This section presents the data acquired for the research and their meaning and analysis. The Section 4.1 include some examples for illustrating the capability of geo-spatial analysis in the river network before applying the technique of solute transport in the desired stretch of river. Here are presented the functionality of the `Watersheds` object and the `Watersheds.Order` method, the `Watersheds.Order2` method and the functions `Watershed.  ,IOR1, IOR2,` `IOR3`, `IOR4`.

Posteriorly, in Section 4.2 is presented the result of the precipitation time series management; in Section 4.3 is presented the flow time series management; in Section 4.4 the Runoff routing and hydrological modelling setup is presented; and finally in Section 4.5 are presented the result of applying the numerical approximation of the Advection-Dispersion Equation as the main component of the solute transport modelling.

4.1 Geo-spatial analysis of zhyd subbasins

The FEC or Functional Elementary Catchment Stands as the central element of ECRINS. FEC refers to the smallest catchment identified as an ECRINS el-

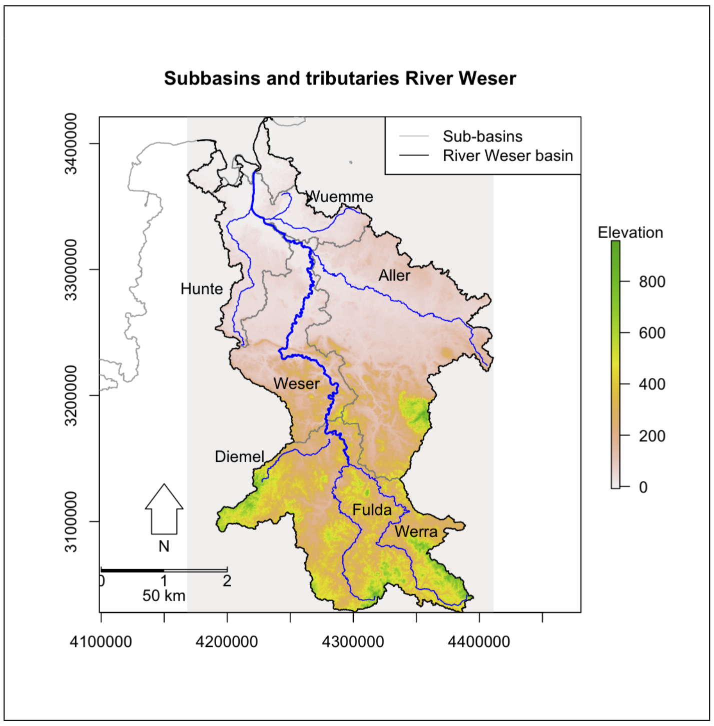

Figure 12: River Weser subbasin and tributaries

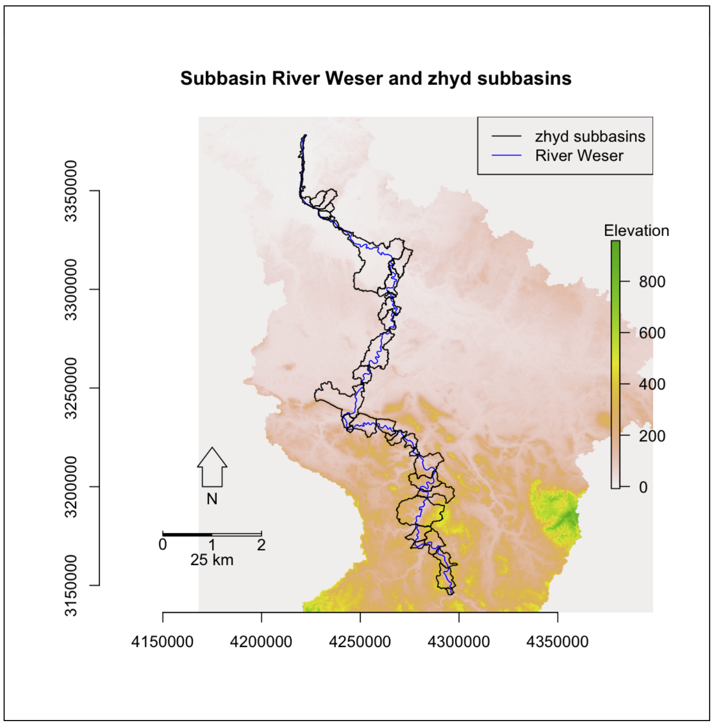

Figure 13: River Weser and intersecting zhyd subbasins

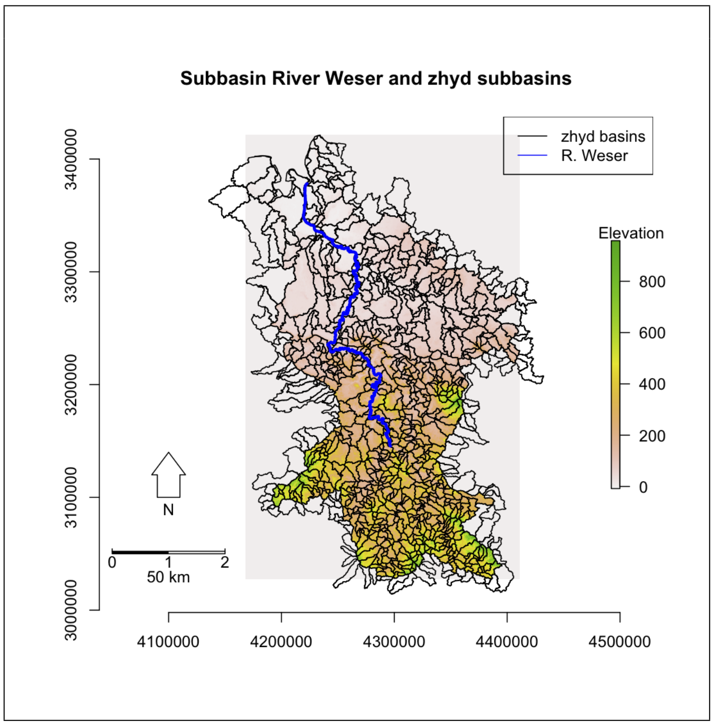

Figure 14: River Weser and all zhyd subbasins

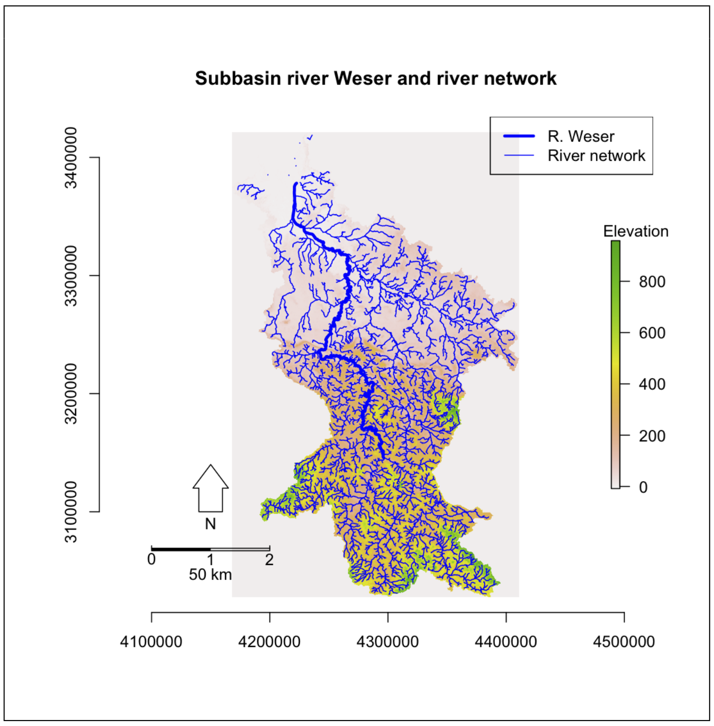

Figure 15: River Weser and river network

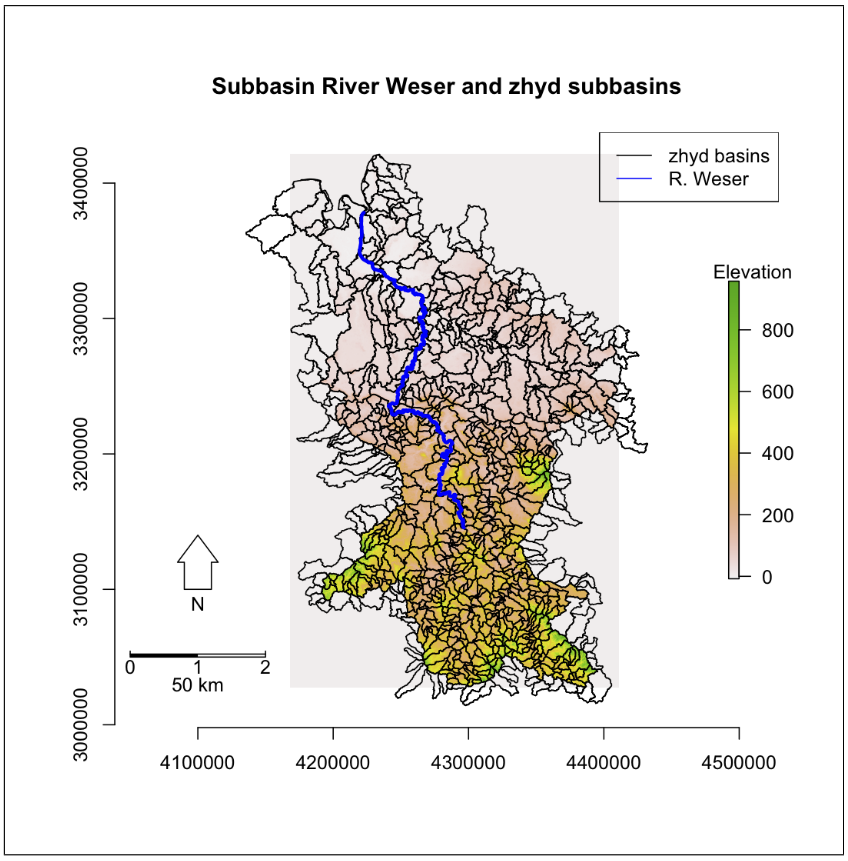

Figure 16: River Weser and all zhyd subbasins

ementary catchment. A FEC is built via aggregating elementary CCM (Catchment Characterisation and Modelling) catchments. It could be either a 'continental FEC' when built by aggregating elementary CCM catchments from a non-coastal basin, or a 'coastal FEC' when elementary CCM catchments belong to a coastal basin (EAA, 2012). The FECs database contains feature class C_Zhyd, hereinafter zhyd, which is the most important data set in ECRINS because it constitutes the primary hydrological unit. The structure of zhyd is reported in EAA (2012), Annex 1. This table sets out the FEC IDs (field ZHYD) and all the required IDs of the useful data sets: aggregation watersheds and reference watersheds, the connection between FECs and sources of information.

Some examples done via the package "Watersheds" with the application of the method `Watershed.Order` and the functions `Watershed.  ,IOR1, IOR2, IOR3`, `IOR4` are presented in the next subsections.

As a guide to follow the process followed by the package `Watersheds` the Figure 17 presents a flow chart of the package. In this flow chart the input actors are illustrated as ellipsoidal red boxes, the operations are presented as blue boxes and the functions or methods are presented as green boxes. The decision nodes are shown as blue diamond boxes. The flow chart is composed by 11 levels: the first one the "ECRINS dataset" as input from the European

Environmental Agency (EAA) and the last one the evaluation of the "Watersheds.IOR4" method and the "stop" node.

After the definition of the "ECRINS dataset" there are three important task to be developed. These task are the "identification of the measurement station", "the identification of the current zhyd object" and the "identification of the probable inlet and outlet nodes", and corresponds to the levels 2 to 4 of the flow chart, respectively.

In the level 5 of the flow chart is created the object "Watersheds" as is illustrated in the Section 4.1.1. Subsequently, in the level 6 is executed the method "Watershed.Order" which constitutes the core of the algorithm because through this method are invoked the "Watersheds. IOR1, IOR2, IOR3 or IOR4" functions (level 8 to 11). Each one of these functions constitute a decision node where is checked if the inlet and outlet stretches of the river network inside the current zhyd are of length 1 to 4, respectively. Each one of these functions performs the different spatial operations for identifying the inlet and outlet nodes and stretches of river inside and around the current zhyd. Subsequently after the definition of the inlet and outlet object (nodes and stretches of river) in the level 8 right side of the flow chart, is executed the method "Watershed.tributary", which performs the spatial operations for identifying the tributary nodes and subsequently the tributary zhyd watersheds of the current zhyd.

Finally after being applied the "Watershed.Tributary" method, is checked in the level 9, right side of the flow chart, if the object "Station" from "Watershed.Tributary" is of length equal to 2, which means that two tributary catchments contributes to the current zhyd watershed. In this positive case is executed the method "Watershed.Order2" which internally calls the method "Watershed.Order" for identifying the structure of the inlet and outlet objects (nodes and stretches of river) in each one of the tributary zhyd watersheds. In the negative case, the object "Station" from "Watershed.Tributary" is length equal to 1, which mean that just one zhyd watershed is tributary to the current zhyd and as a consequence the method "Watershed.Order" is invoked again (see right side of the level 9 of the flow chart that return to level 6) for determining the inlet and outlet nodes and stretches of river and the river network of this tributary watershed.

In Appendix 7.10 is presented the user manual of the developed package "Watersheds".

4.1.1 The `Watersheds` object

The package Watersheds contains a class `"Watershed"` for representing `"Watershed"` objects. In the following lines is presented an example for the definition of a `"Watershed"` object.

```
# definition of the current station point
station1 = WatershedsData["station"][[1]]

# definition of the current subbasin of study. IN thi case the river Weser
# basin
subbasin1 = WatershedsData["subbasin"][[1]]

# definition of the zhyd1 object which contains all the zhyds inside the
```

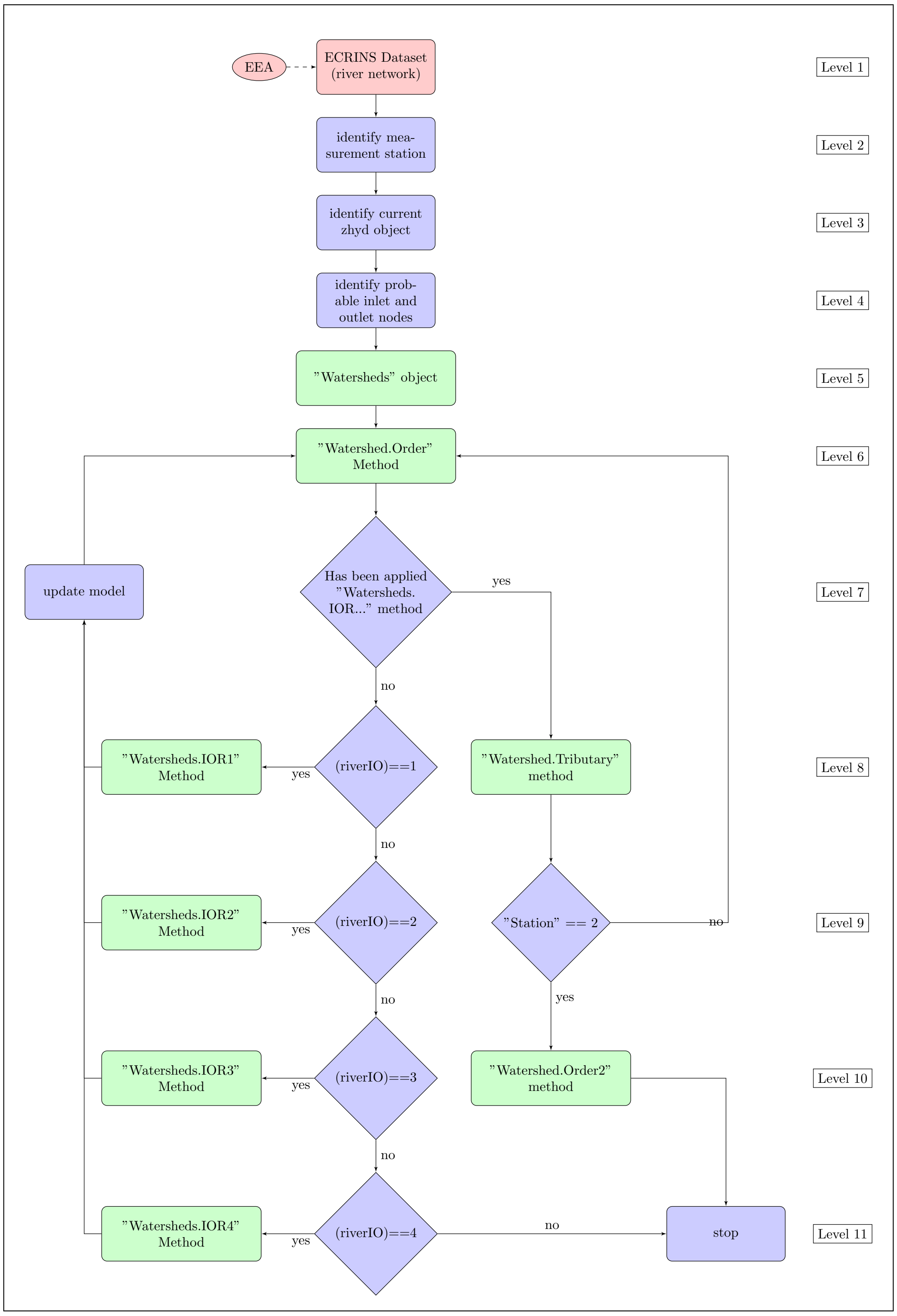

Figure 17: Flow chart of the `Watersheds` package. Red=data input; blue rectangle = process; green rectangle = algorithm; diamond shape = decision.

```r
# subbasin
zhyd1 = WatershedsData["zhyd"][[1]]

# definition of the river network inside the subbasin
river1 = WatershedsData["river"][[1]]

# definition of the nodes of the river network
node1 = WatershedsData["node"][[1]]

# definition of the 'Watersheds' object:
station1 = SpatialPoints(station1, proj4string = slot(subbasin1, "proj4string"))
watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)
class(watershed)
```

4.1.2 The `Watersheds.Order` method

The Method for function `Watershed.Order` allows definition of the properties of the current `zhyd` watershed over `Watershed` objects.

The function takes the object of class `Watershed` and identifies the subbasin that contains the current `station` (class `SpatialPoints`) and subsets the `zhyd` object to subbasin and extract the current `zhy` object that contains `station`. Posteriorly, identifies the inlet and outlet stretches and probable inlet and outlet nodes of the `zhyd`. Then, runs the functions `Watershed .IOR1, .IOR2, .IOR3`, or `.IOR4` for determining the actual inlet and outlet nodes. Finally, the method executes the S4 method `Watershed.Tributary` for defining tributary nodes and tributary catchments of the current `zhyd` watershed. As orientation, the method is located in the level 6 of the flow chart presented in Figure 17.

An example of the application of the method `Watershed.Order` is presented in Figure 18. In this Figure it is possible to see the primary zhyd object numbered as 1. The inlet node (green dot) and the outlet node (red dot) are presented. The tributary watersheds to 1 are labeled as 1.1 and 1.2. Finally the river network is presented in blue stretches. The R-code for reproducing the Figure and illustrating the method is presented in Appendix 7.1.

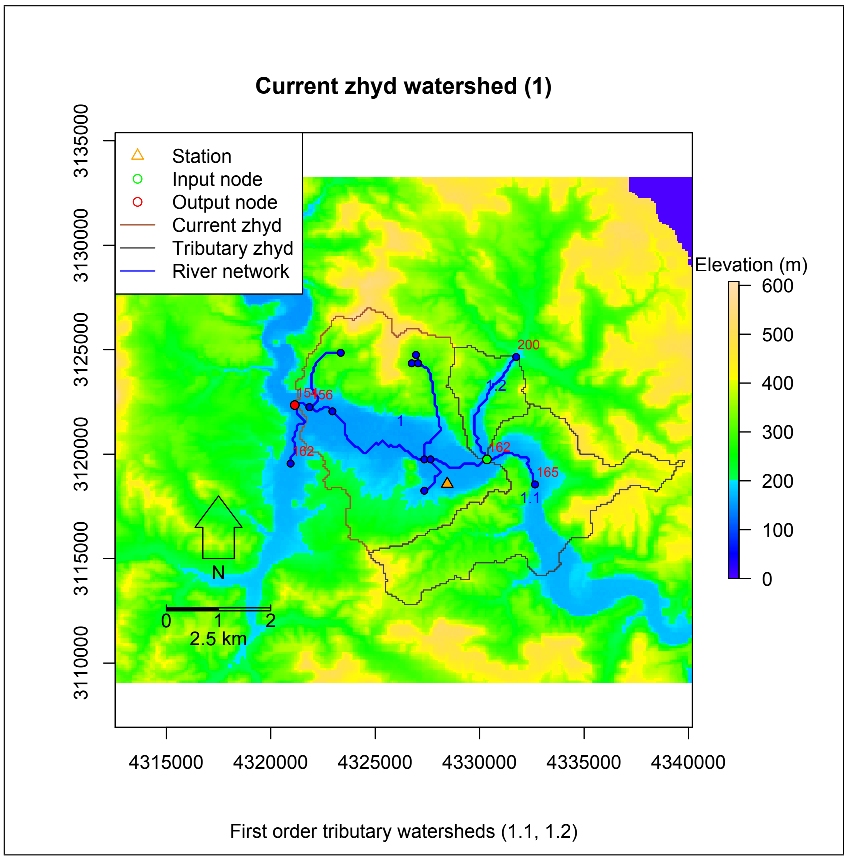

Figure 18: Current zhyd watershed (1) and first order tributary watersheds (1.1 , 1.2)

4.1.3 The `Watersheds.Order2` method

The S4 Method for function `Watershed.Order2` is a definition of the tributary `zhyd` watersheds of the current `zhyd` watershed.

The method takes the object of class `Watershed` when object `node_trib` is length 2. The method identifies the `zhyd` watershed that contains the current `station` (class `SpatialPoints`) and apply the method `Watershed.Order` on each point of `node_trib` returning a list of objects `Watershed.Order`. The computation is done via parallel processes for optimizing and take advance of multicore functionalities.

The Figure 19 is an illustration of the method `Watershed.Order2` and the corresponding code is presented in Appendix 7.2. Also as orientation, the method is located in the right side of the level 10 of the flow chart presented in Figure 17.

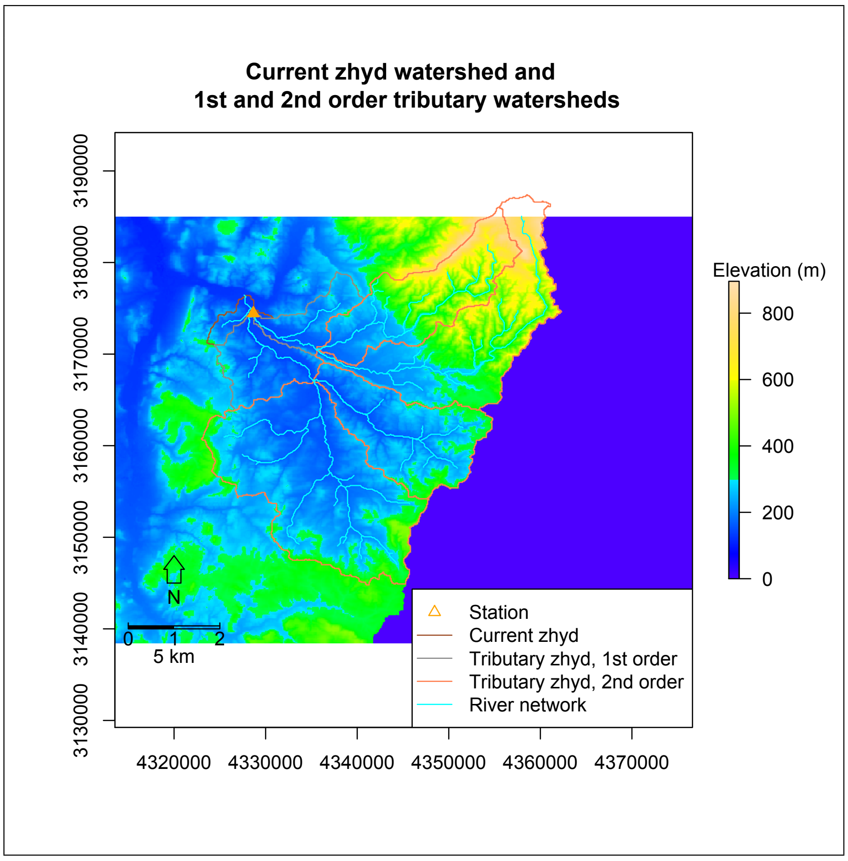

Figure 19: Current zhyd watershed and 1st and 2nd order tributary watersheds

4.1.4 The `Watersheds.IOR1` function

The `Watersheds.IOR1` function means Watershed inlet and outlet nodes: case 1. This function determines the inlet and outlet nodes for `zhyd` watershed objects. This case 1 is for those watersheds that its river inlet and outlet object is length 1 (length(riverIO)==1). The Figure 20 is an illustration of the `Watersheds.IOR1` function and the R-code for producing it is presented in Appendix 7.3. As an orientation, the function is located in the left side of the level 8 of the flow chart presented in Figure 17.

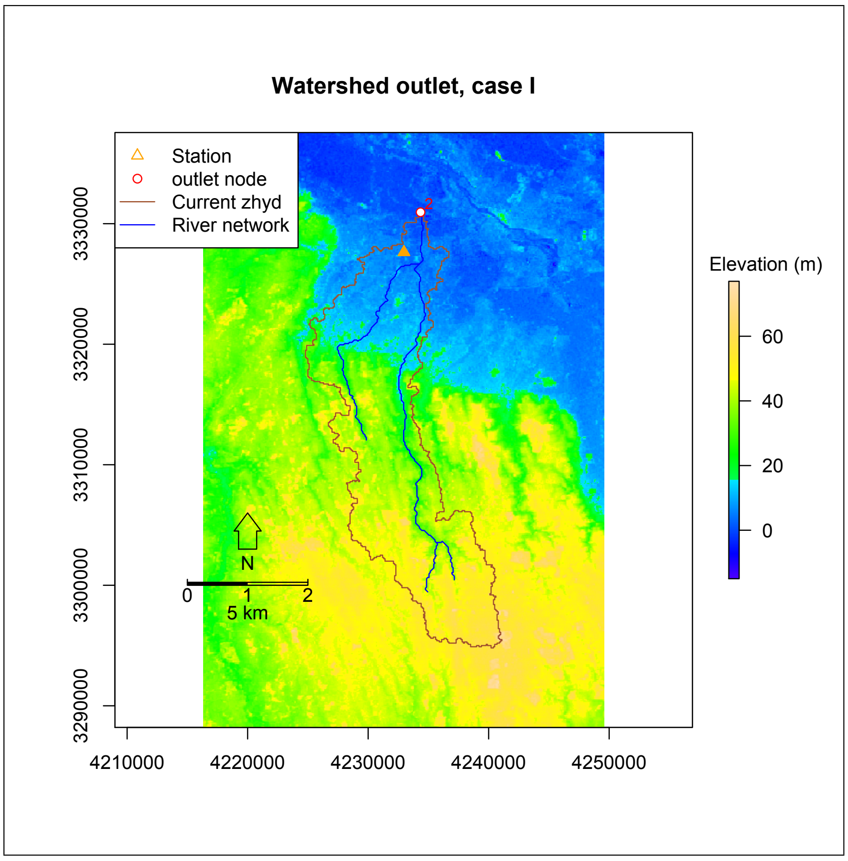

Figure 20: Spatial analysis of watershed outlet, case I

4.1.5 The `Watersheds.IOR2` function

The `Watersheds.IOR2` function means Watershed inlet and outlet nodes: case 2. The function determines the inlet and outlet nodes for `zhyd` watershed objects. This case 2 is for those watersheds that its river inlet and outlet object is length 2 (length(riverIO)=2). The Figure 21 is an illustration of the method and the R-code for reproducing it is presented in the Appendix 7.4. As an orientation, the function is located in the left side of the level 9 of the flow chart presented in Figure 17.

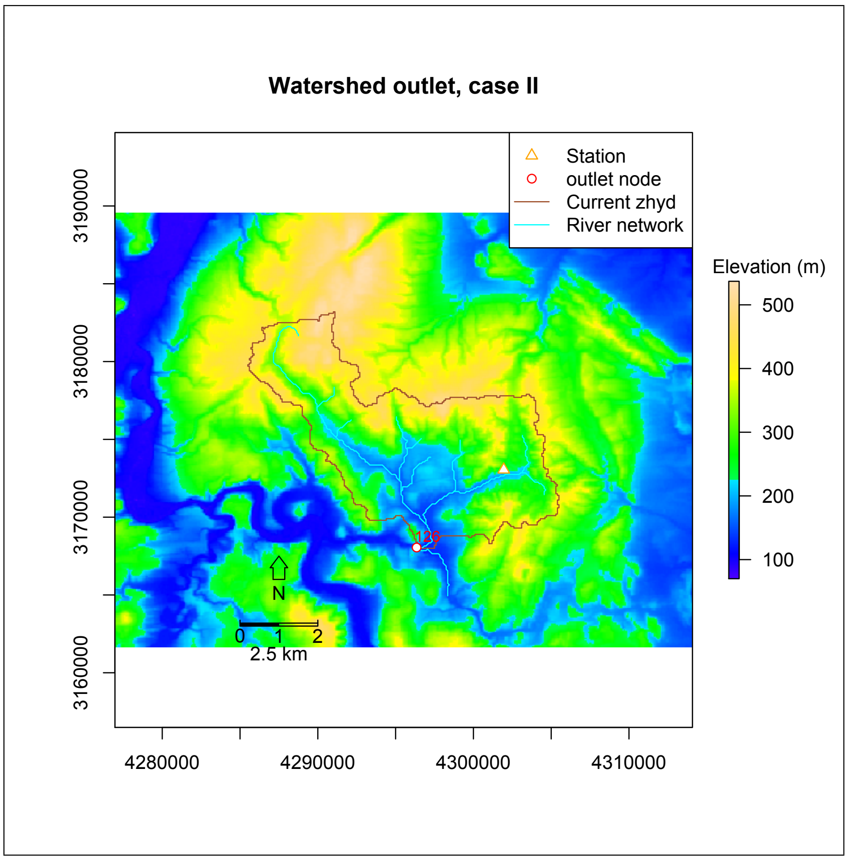

Figure 21: Spatial analysis of watershed outlet, case II

4.1.6 The `Watersheds.IOR3` function

The `Watersheds.IOR3` function means: Watershed inlet and outlet nodes: case 3. The function determines the inlet and outlet nodes for `zhyd` watershed objects. This case 3 is for those watersheds that its river inlet and outlet object is length 3 (length(riverIO)=3). An illustration of the `Watersheds.IOR3` function is presented in the Figure 22 and the R-code for reproducing it is presented in Appendix 7.5. As orientation, the function is located in the left side of the level 10 of the flow chart presented in Figure 17.

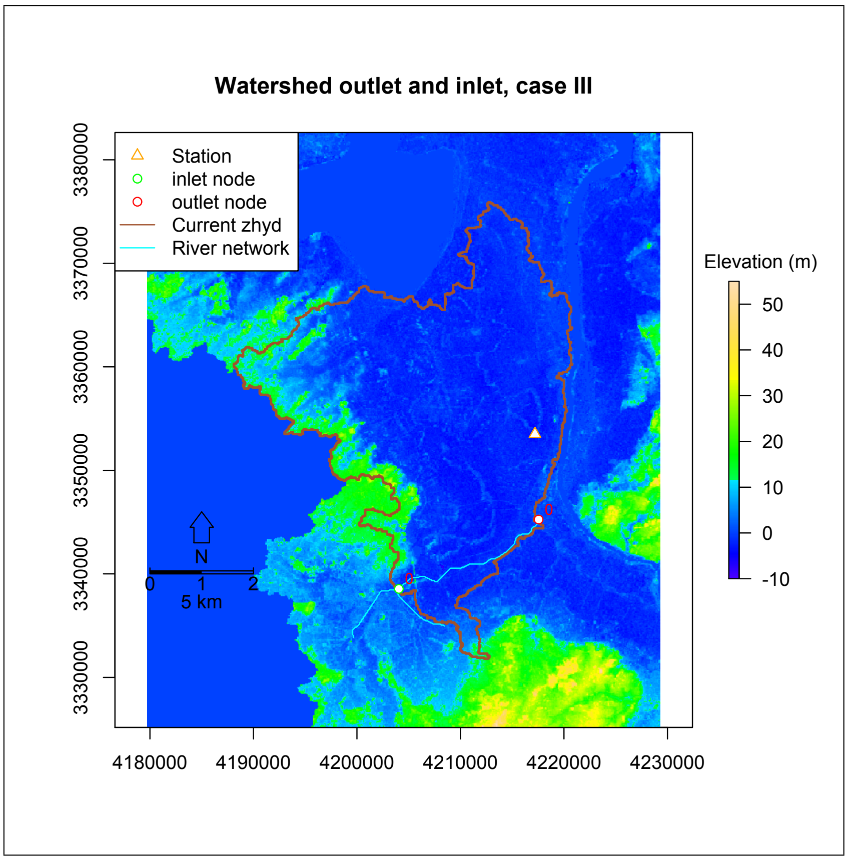

Figure 22: Spatial analysis of watershed inlet and outlet, case III

4.1.7 The `Watersheds.IOR4` function

The `Watersheds.IOR4` function means Watershed inlet and outlet nodes: case 4. The function determines the inlet and outlet nodes for `zhyd` watershed objects. This case 4 is for those watersheds that its river inlet and outlet object is length 4 (length(riverIO)=4). The Figure 23 is an illustration of the method and the corresponding R-code is presented in Appendix 7.6. As an orientation, the function is located in the left side of the level 11 of the flow chart presented in Figure 17.

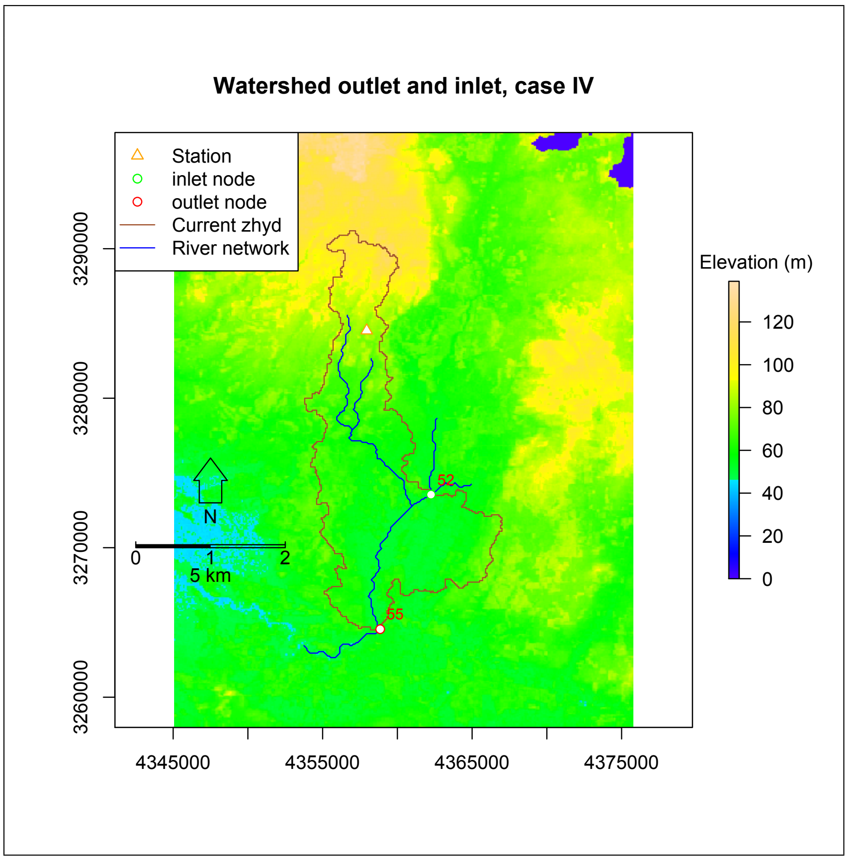

Figure 23: Spatial analysis of watershed inlet and outlet, case IV

4.1.8 The Karlshafen and Wahmbeck Stations watersheds

Previously in Section 3.1.3 and Figure 6, the details and location of the nine stations analyzed in this study for retrieving data for flow level and discharge was presented and also summarized in Table 1. From this data, the major attention is on two main stations: the Karlshafen station, which its tributary watershed is the watershed of the station Wahmbeck. Figure 24 presents the location and the river network of this two stations after applied the Watersheds flow chart presented in Figure 17 and the corresponding code in R for performing the spatial analysis and reproducing the Figure is presented in Appendix 7.7.

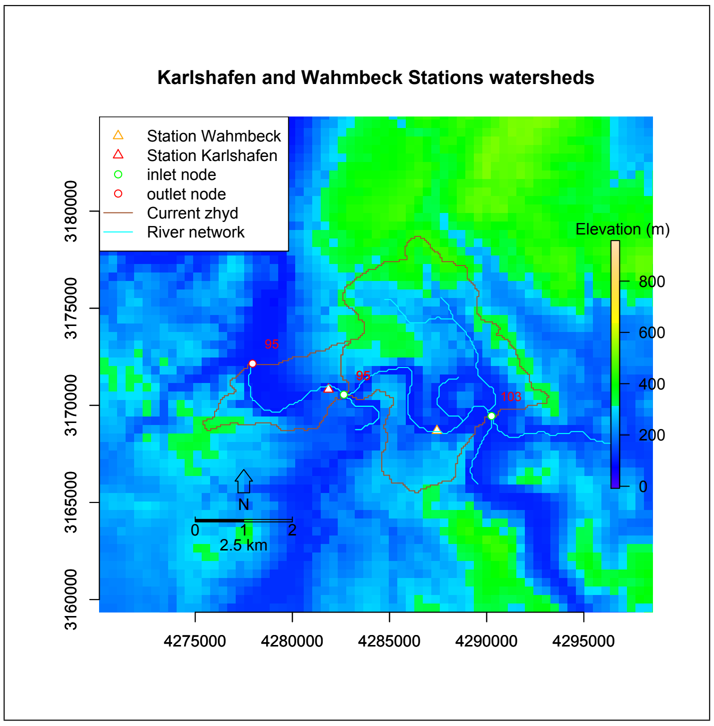

Figure 24: The Karlshafen and Wahmbeck Stations watersheds

4.2 Precipitation time series management

The primary data source of precipitation data are gridded daily precipitation time series obtained from the gridded dataset from ENSEMBLES (E-OBS) dataset for precipitation, temperature and sea level pressure in Europe and provided by the European Climate Assessment & Dataset (ECA&D) project (Haylock et al., 2008). This project presents information on changes in weather and climate extremes, as well as the daily dataset needed to monitor and analyse these extremes. A resolution of 0.25° (21 Kilometres east, 28 Kilometres north) precipitation gridded data is used. The gridded datasets is available for downloading from the web page of the ENSEMBLES project (ECA&D (2012), `http://eca.knmi.nl`).

The format of the gridded data is NetCDF (network Common Data Form) which is a set of interfaces for array-oriented data access and a freely-distributed collection of data access libraries for C, Fortran, C++, Java, and other languages. The netCDF libraries support a machine-independent format for representing scientific data (Unidata, 2012). The Open Geospatial Consortium membership has approved the Enhanced Data Model Extension to the OGC Network Common Data Form (netCDF) Core Encoding Standard `http://www.unidata.ucar.edu/blogs/news/entry/ogc_adopts_netcdf_enhanced_data`.

In addition, the NetCDF format is the current precipitation data format of the ECRINS which is a remarkable climate project of the European Comunity.

The access to NetCDF files for reading data into R and for creating new netCDF dimensions, variables, and files, or manipulating existing netCDF files from R, was possible through the `ncdf` package (Pierce, 2013). The data downloaded was 1.29 GB as a consequence for reading and working whit the file is necessary to subset the retrieval of information, in this case were retrieved 1096 attributes (precipitation) which represents the precipitation time series from netCDF for Wahmbeck station (9.875°E , 51.625°N) between the dates 01.01.1995 and 31.12.1997. The original file was downloaded directly from the European Climate Assessment & Dataset repository and comprises data from 01.01.1995 to 12.31.2013. The Figure 25 presents the time series extracted from the netCDF for the Wahmbeck station in the time window 01.01.1995 and 31.12.1997.

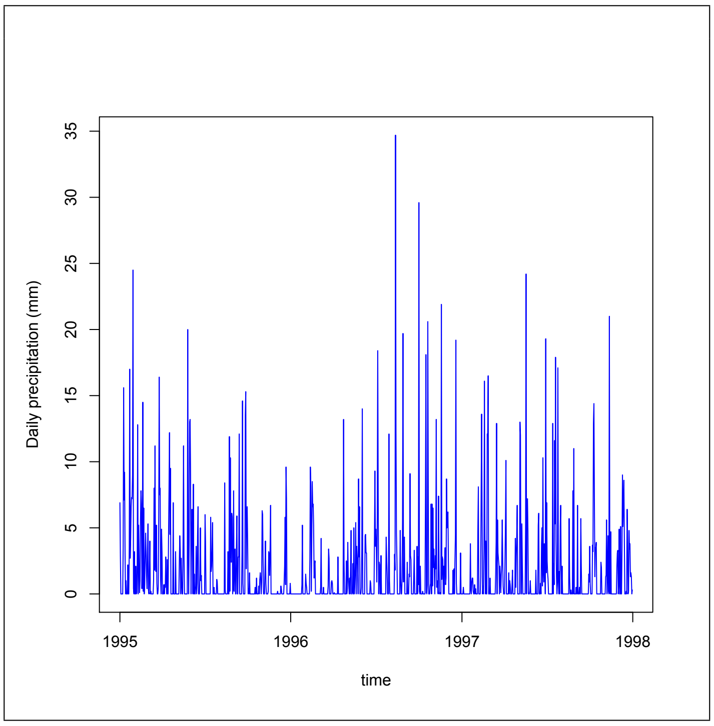

Figure 25: Precipitation time series at Wahmbeck Station

4.3 Flow time series management

Data for flow level and discharge are also available at Department of Hydrometry and Hydrological Survey of the German Federal Institute of Hydrology was used. In the Table 1 was presented the details of the nine stations analyzed in this study and in Figure 6 was showed their location.

In order to compose the hydrological framework for simulation the daily precipitation from 01.01.1995 to 31.12.1997 we use the time series of flow at Wahmbeck station. The Figure 26 presents the time series for the flow in cubic meter per second in the specified time interval.

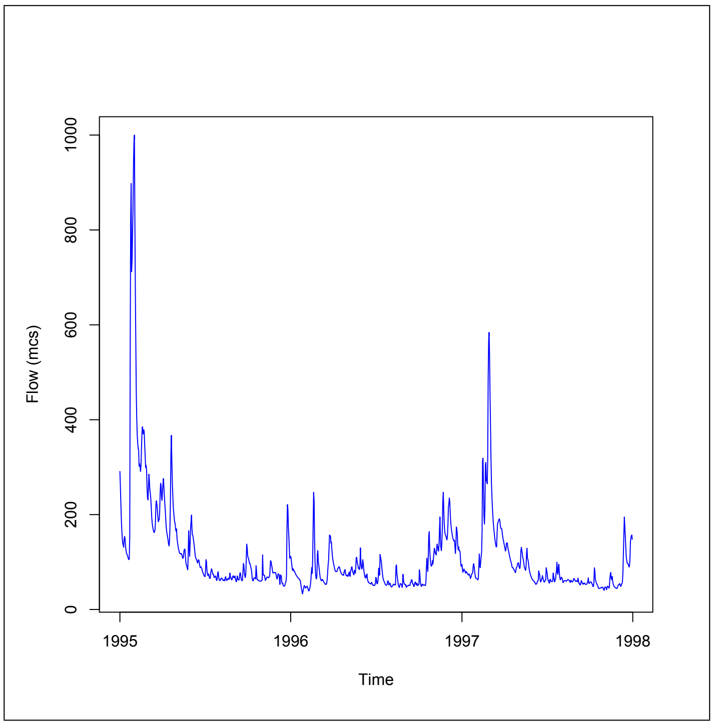

Figure 26: Precipitation time series at Wahmbeck Station

4.4 Runoff routing and hydrological modelling

An important contribution in hydrological modelling is done by F. T. Andrews et al. (2011) with the R package `hydromad` (`http://hydromad.catchment.org`). It is based loosely on the unit hydrograph theory of rainfall-runoff modelling. More than a single hydrological model `hydromad` is a framework with several options of configurations that includes different Soil Moisture Accounting (SMA) models and objective calibration methodologies. In consequence, it can be used cohesively with workflows based on R. Two areas of focus for the package are discrete event separation and the design of fit statistics, and how event-based data analysis can be useful in a modelling context (F. T. Andrews et al., 2011).

For this case, the model will be calibrated using the fitBy- Optim function, which accordingly to F. Andrews (2011) performs parameter sampling over the pre-specified ranges, selecting the best of these, and then runs an optimisation algorithm from that starting point.

After the calibration process two parameters are returned for the SMA (Soil Moisture Accounting) component: 1) rrthresh, a theshold value of the runoff ratio, below which there is no effective rainfall; and 2) scale, a constant multiplier of the result, for mass balance. If this parameter is set to NA (as it is by default) in hydromad it will be set by mass balance calculation.

Also, 2 parameters are returned regarding the routing method, in this case the exponential components transfer function models "expuh". The parameters calibrated are 1) tau_s that represents time constants (*tau*) for the exponential components; and 2) v_s that represents fractional volumes (v) for the exponential

Table 2: Parameter definition

SMA Parameters:		
	lower	upper
rrthresh	0	0.2
scale	NA	NA

Routing Parameters:		
	lower	upper
tau_s	2	100
v_s	0	1

Table 3: Parameter calibration results

Hydromad model with "runoffratio" SMA and "expuh" routing: Start = 1995-01-01, End = 1998-01-01	
SMA Parameters:	
rrthresh	scale
0.1152	1.1191
Routing Parameters	
tau_s	v_s
14.77	1.00
TF Structure: single store: Poles:0.9345 Fit: ($fit.result) fitByOptim(MODEL = modx) 128 function evaluations in 28.09 seconds	

components.

A quick way to view the modelled and observed streamflow time series together is to call xyplot() on the model object, as in Figure 27.

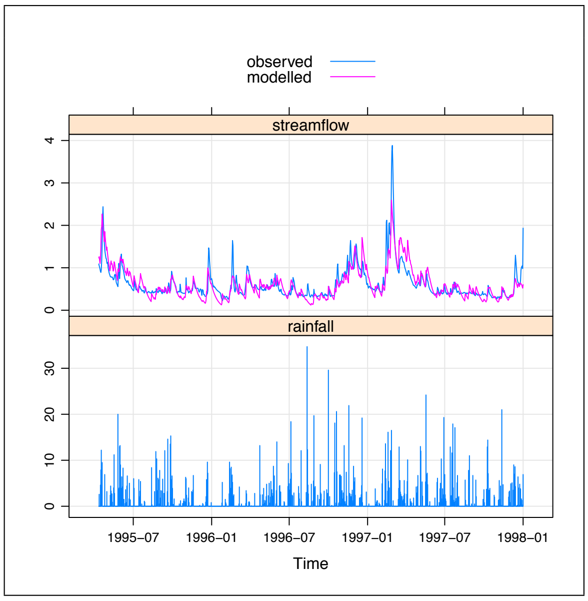

Figure 27: Modelled and observed streamflow time series

In summary, the Figure 28 presents the flow chart of the process of hydrological modelling discussed before. The flow chart presents in the initial level the input data that is the ECRINS dataset, in the first and second level are represented the geospatial analysis performed by the package "Watersheds" (see flow chart in Figure 17). In the third and fourth level is represented the hydrological modelling framework executed within the package "hydromad" with input data from the EOBS dataset (time series of precipitation) and the BfG (flow time series).

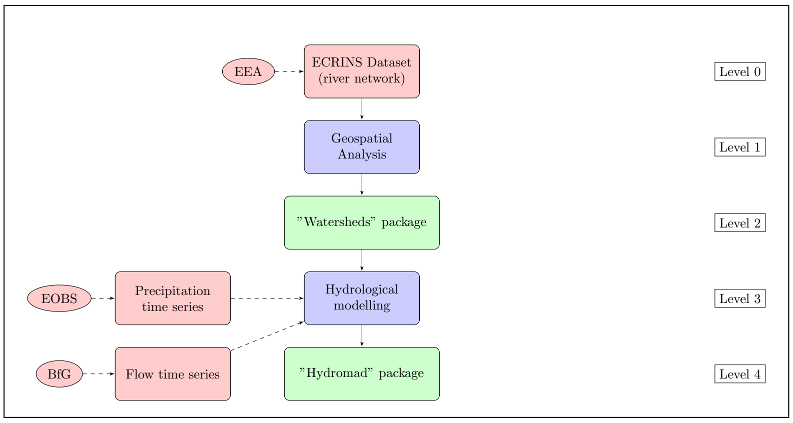

Figure 28: Flow chart of the hydrological modelling. Red=data input; blue rectangle = process; green rectangle = algorithm

4.5 Routing and solute transport modelling

After have been done the hydrological routing, is applied the numerical approximation of the Advection Dispersion Equation, an application is performed for transporting and decaying of organic carbon (OC) in the river Weser, in a widening stretch at Wahmbeck station as upstream boundary and Karlshafen station as downstream boundary. Two scenarios are simulated: the baseline includes only input of organic matter upstream. The second scenario simulates the input of an important side river half way the river. The theoretical description of the numerical approximation of the Advection Dispersion Equation was presented in Section 2.3.

The Table 4 presents the boundaries conditions for simulating the solute transport (organic carbon) for the first five days of the year 1995, where "flow.up" is the flow in upper boundary, "factor" is a scalar for internal computations, "flow.lat.0" is the inflow in the stretch, "F.OC" is the concentration of organic carbon in the upper boundary, "F.lat.0" is the concentration of organic carbon in the inflow and "k" is the reaction rate of organic carbon.

The resulting code in R is adapted from Soetaert and Meysman (2012) and is presented in the Appendix 7.9 for the conditions on river Weser between the stations Wahmbeck and Kalshafen on 01.01.1995, similar code is generated for each one of the fifth first days of 1995 year as illustrated in the Figures 29 to 33.

Table 4: Initializing parameter and boundary conditions

	flow.up [mcs]	factor	flow.lat.0 [mcs]	F.OC [mol s^{-1}]	F.lat.0 [mol s^{-1}]	k [s^{-1}]
Q1	63	2.85	63	63	63	3.17E-007
Q3	59	3.06	59	59	59	3.17E-007
Q4	55	3.28	55	55	55	3.17E-007
Q5	51	3.51	51	51	51	3.17E-007

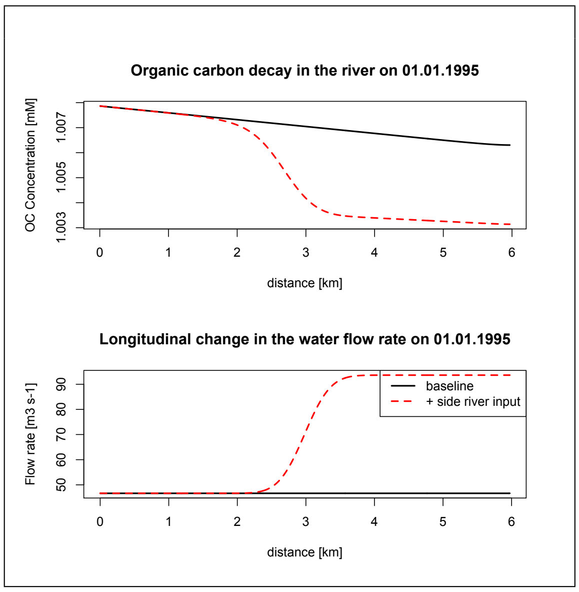

Figure 29: Simulation time series of OC and flow between the Wahmbeck and Kalshafen stations on 01.01.1995

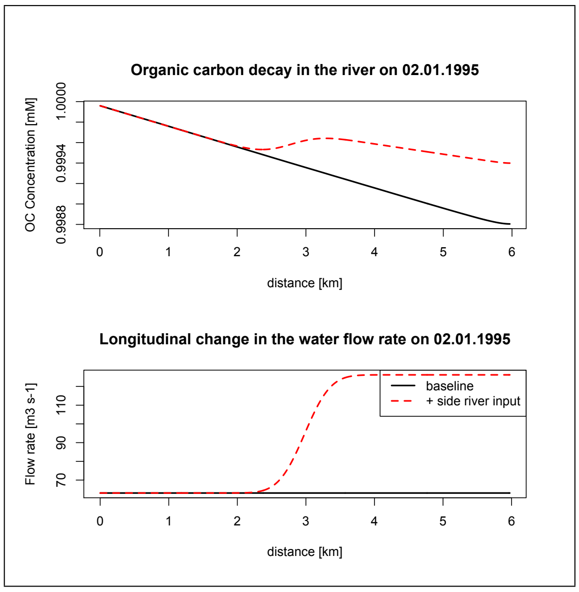

Figure 30: Simulation time series of OC and flow between the Wahmbeck and Kalshafen stations on 02.01.1995

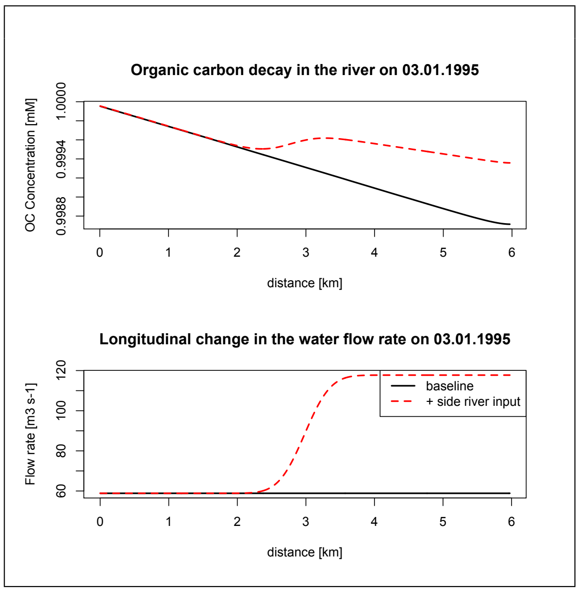

Figure 31: Simulation time series of OC and flow between the Wahmbeck and Kalshafen stations on 03.01.1995

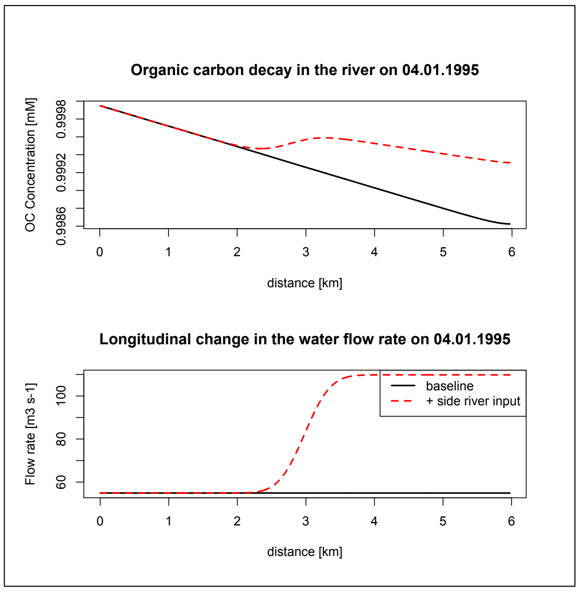

Figure 32: Simulation time series of OC and flow between the Wahmbeck and Kalshafen stations on 04.01.1995

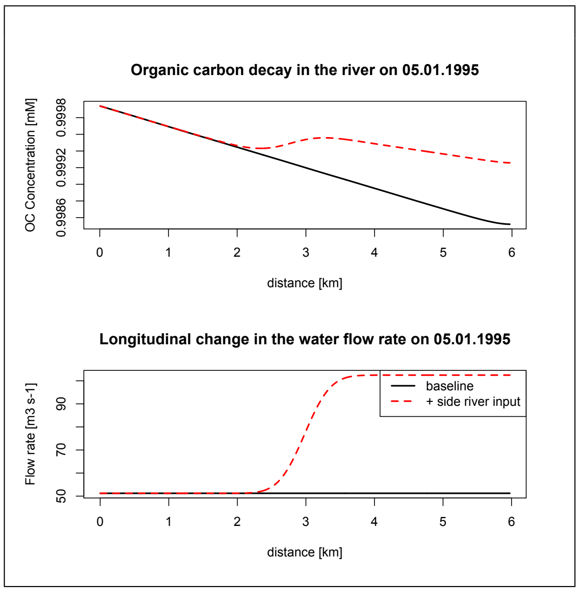

Figure 33: Simulation time series of OC and flow between the Wahmbeck and Kalshafen stations on 05.01.1995

In summary, the Figure 34 presents the flow chart of the process of hydrological modelling plus routing for solute transport simulation discussed before. Until level 4, the flow chart is the same than that presented previously in Figure 28) and described at the end of Section 4.4. Additionally, this flow chart includes the decision of performing solute transport simulation represented as

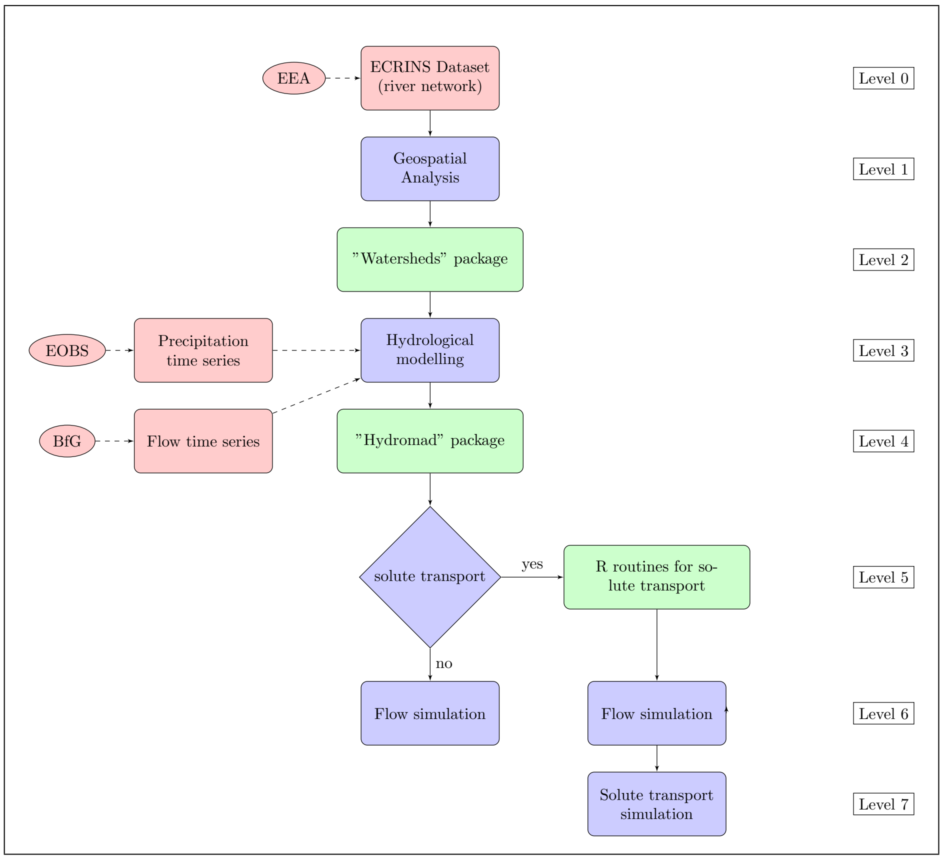

Figure 34: Flow chart of the solute transport simulation. Red=data input; blue rectangle = process; green rectangle = algorithm; diamond shape = decision.

the decision diamond in the level 5, the "no" route concludes with only flow simulation (level 6) as the result of the process of hydrological modelling, the "yes" route enables an additional processes (level 7) related with the routing for solute transport via the R routines implemented through the R packages discussed in Section 2.4. This additional process deploys new functionality of a typical model in hydromad by adding the capabilities for solute transport simulation which constitutes one of the major aims of the present work.

5 Conclusions

Has been presented the implementation of several methods for geo-spatial analysis of river networks and watersheds for runoff routing and solute transport in R in order to contribute in a comprehensive hydrological modelling to the current framework of the R package "hydromad".

The main aim of the study is fulfilled because the versatile code developed lets to coupled the outputs of the hydrological framework of the R package "hydromad" to the selected solute transport model looking forward better simulation of water-quality determinants transport at watershed scale.

Following the research scheme presented in this proposal it is possible to prove the hypothesis behind the study. The simulation of solute transport at specific places of the river network was improved by implementing a runoff routing method at watershed-scale, the "hydromad" package, and by coupling it into a suitable modeling framework for representing solute transport processes.

The developed package, "watersheds", allows geo-spatial river network analysis and makes use of the Catchments and Rivers Network System (ECRINS) version 1.1, which constitutes the hydrographical system currently in use at the European Environment Agency as well as widely serving as the reference system for the Water Information System for Europe (WISE). The versatility of the code generated lets to implement geo-spatial analysis in any watershed included into the ECRINS. As a consequence, watersheds along entire Europe could be analyzed, this constitutes an important fact because several institutions or scientific community related with the WISE system could take advantage of the package and this document.

6 Further work

Further development for hydrograph and solute transport calibration may be done through a versatile open source, multiple platform programming language as R. A useful tool for model calibration and sensitivity analysis processes in R is the `hydroPSO` package (Zambrano-Bigiarini & Rojas, 2013). This package implements several state-of-the-art enhancements and fine-tuning options to the Particle Swarm Optimisation (PSO) algorithm. `hydroPSO` interfaces the calibration engine to different model codes through ASCII files and/or R wrapper functions for exchanging information on the calibration parameters. The optimisation is based on evaluating the goodness-of-fit functions until a maximum number of iterations or a convergence criterion are met. The evaluation of the calibration process is supported by plotting functions that facilitate the interpretation of results (Zambrano-Bigiarini & Rojas, 2013).

In the near future the methodology presented could be upgraded substantially by integrating Sensor Observations Services for retrieving the hydrological data for the modelling processes, therefore real-time simulation and prediction could be done. In this sense, applications developed in the framework of the 52 °North SOS implementation are advisable (`http://52north.org/communities/sensorweb/sos/`).

References

Andrews, F. (2011, June). Hydromad tutorial [Computer software manual].

Andrews, F. (2014, February). *Hydromad: Hydrological Model Assessment and Development.* Internet. Retrieved from http://hydromad.catchment.org/

Andrews, F. T., Croke, B. F. W., & Jakeman, A. J. (2011). An open software environment for hydrological model assessment and development. *Environmental Modelling and Software*, *26*. Retrieved from http://hydromad.catchment.org/ doi: doi:10.1016/j.envsoft.2011.04.006

Beven, K., Lamb, R., Quinn, P., Romanowicz, R., & Freer, J. (1995). TOPMODEL. In V. P. Sing (Ed.), *Computer models of watershed hydrology. Water Resources Publications, Colorado. pp. 627-668.*

Beven, K. J. (1989). Changing ideas in hydrology. the case of physically-based models. *Journal of Hydrology*, *105 (1-2)*, 157-172.

Beven, K. J. (1997). *Distributed hydrological modelling: Applications of the TOPMODEL Concept.* Wiley.

Beven, K. J. (2012). *Rainfall-runoff modelling: The Primer* (Second ed.). Wiley-Blackwell. (Lancaster University, UK)

Bivand, R., Keitt, T., Rowlingson, B., Pebesma, E., Sumner, M., & Hijmans, R. (2003-2013, May). Package "rgdal": Bindings for the Geospatial Data Abstraction Library (.8-9 ed.) [Computer software manual].

Bivand, R., & Rundel, C. (2012, June). Package "rgeos": Interface to Geometry Engine - Open Source (GEOS) (.2-7 ed.) [Computer software manual].

Botter, G., Bertuzzo, E., & Rinaldo, A. (2010). Transport in the hydrologic response: Travel time distributions, soil moisture dynamics, and the old water paradox. *Water Resources Research*, *46*, 1-18. doi: 10.1029/2009WR008371

Botter, G., Milan, E., Bertuzzo, E., Zanardo, S., Marani, M., & Rinaldo, A. (2009). Inferences from catchment-scale tracer circulation experiments. *Journal of Hydrology*, *369*, 368-380. doi: 10.1016/j.jhydrol.2009.02.012

Buytaert, W. (2012, February). Package 'topmodel' [Computer software manual].

Chaudhry, M. H. (2008). *Open-channel flow.* Springer.

Duffy, C. J. (2010). Dynamical modelling of concentration-age-discharge in watersheds. *Hydrological Processes*, *24*, 1711-1718. doi: 10.1002/hyp.7691

EAA. (2012). *EEA catchments and rivers network system, ECRINS v1.1. rationales, building and improving for widening uses to Water Accounts and WISE applications* (EEA Technical report No. 7/2012). Copenhagen: European Environment Agency - EAA. (Luxembourg: Publications Office of the European Union)

ECA&D. (2012). *E-obs datafiles 1950-01-01 until 2012-06-30.* Retrieved from http://eca.knmi.nl/download/ensembles/download.php

European Environment Agency - EAA. (1995). *CORINE land cover* (Tech. Rep.). Commission of the European Communities: Commission of the European Communities.

Freer, J. E., McMillan, H., McDonnell, J. J., & Beven, K. J. (2004). Constraining dynamic TOPMODEL responses for imprecise water table information using fuzzy rule based performance measures. *Journal of Hydrology*, *291*, 254-277.

German Federal Institute of Hydrology - BfG. (2013). *Water levels - data from selected gauging stations on german federal waterways.* Retrieved from http://www.bafg.de/ (Retrieved on 20.06.2013)

Global Runoff Data Centre - GRDC. (2013). *The GRDC world-wide repository of river discharge data and associated metadata.* Retrieved from http://www.bafg.de/GRDC/ (Retrieved on 20.06.2013)

Gong, L., Widén-Nilsson, E., Halldin, S., & Xu, C.-Y. (2009). Large-scale runoff routing with an aggregated network-response function. *Journal of Hydrology*, *368*, 237 - 250. Retrieved from http://www.sciencedirect.com/science/article/pii/S0022169409000857 doi: http://dx.doi.org/10.1016/j.jhydrol.2009.02.007

Grayson, R. B., Moore, I. D., & McMahon, T. A. (1992). Physically based hydrologic modelling 2. is the concept realistic. *Water Resources Research*, *28 (10)*, 2659-2666.

Haylock, M. R., Hofstra, N., Tank, A. M. G. K., Klok, E. J., Jones, P. D., & New, M. (2008). A European daily high-resolution gridded data set of surface temperature and precipitation for 1950-2006. *Journal of Geophysical Research*, *113*. doi: 10.1029/2008JD010201

Hijmans, R. J. (2014, January). Package "raster": Geographic data analysis and modeling (2.2-5 ed.) [Computer software manual].

Ihaka, R., & Gentleman, R. (1996). R: a languague for data analysis and graphics. *Journal of Computational and Graphical Statistics*, *5*, 299-314.

Kouwen, N. (1988). WATFLOOD: A micro-computer based flood forecasting system based on real-time weather radar. *Canadian Water Resources Journal*, *13(1)*, 62-77. Retrieved from http://www.civil.uwaterloo.ca/watflood/Manual/01_1.htm (User manual online)

Lewin-Koh, N. J., & Bivand, R. (2012). Package "maptools": Tools for reading and handling spatial objects (.8-16 ed.) [Computer software manual].

McDonnell, J. J., McGuire, K., Aggarwal, P., Beven, K. J., Biondi, D., Destouni, G., ... Wrede, S. (2010). How old is streamwater? Open questions in catchment transit time conceptualization, modelling and analysis. *Hydrological Processes*, *24*, 1745-1754.

Neitsch, S., Arnold, J., Kiniry, J., & J.R., W. (2011). Soil and Water Assessment Tool, theoretical documentation, version 2009 [Computer software manual].

Pebesma, E., & Bivand, R. (2005-2012, May). Package "sp": classes and methods for spatial data (.9-99 ed.) [Computer software manual].

Pereira, L. M. (2009). *Modelagem hidrologica dinamica distribuida para estimativa do escoamento superficiail em uma microbacia urbana.* Master dissertation, Instituto Nacional de Pesquisas Espaciais - INPE.

Pierce, D. (2013, August). Package "ncdf" (1.6.6 ed.) [Computer software manual].

R Core Team, Bivand, R., Carey, V. J., DebRoy, S., Eglen, S., Guha, R., ... Free Software Foundation, Inc. (1999-2013, May). Package "foreign": Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase,... (.8-54 ed.) [Computer software manual].

R Development Core Team. (2013). *R: A Language and Environment for Statistical Computing.* Vienna, Austria. Retrieved from http://www.R-project.org/

Rennó, C. D. (2003). *Construção de um sistema de análise e simulação hidrológica: Aplicação a bacias hidrográficas.* Tese de doutorado do curso da pós-graduação e sensoriamento remoto, Instituto Nacional de Pesquisas Espaciais (INPE).

Sarkar, D. (2012, March). Package lattice: Lattice graphics [Computer software manual].

Schulla, J. (2012). Model description wasim (water balance simulation model) [Computer software manual]. Retrieved from http://www.wasim.ch/the_model.html

Soetaert, K. (2014, January). Package "rootsolve" [Computer software manual].

Soetaert, K., & Meysman, F. (2012). R-package ReacTran : Reactive Transport Modelling in R [Computer software manual]. Royal Netherlands Institute of Sea Research (NIOZ).

Soetaert, K., Petzoldt, T., & Setzer, R. W. (2013, September). Package "desolve" [Computer software manual].

Swaney, D., Humborg, C., Emeis, K., Kannen, A., Silvert, W., Tett, P., ... Nicholls, R. (2011). Five critical questions of scale for the coastal zone. *Estuarine, Coastal and Shelf Science*, *xxx(2011)*, 1-13.

Torres, J. A., & Pebesma, E. J. (2013, September 16-17). State of R in hydrological modeling. In *2nd open water symposium.* Brussels, Belgium.

Unidata. (2012). *NetCDF FAQ.* Retrieved from http://www.unidata.ucar.edu/software/netcdf/docs/faq.html#whatisit

United Nations - UN. (2012). *The Millennium Development Goals Report* (Technical Report). New York: Author. Retrieved from http://www.un.org/millenniumgoals/reports.shtml

Urbanek, S. (2013, February). Package "multicore": Parallel processing of R code on machines with multiple cores or CPUs (.1-7 ed.) [Computer software manual].

USACE, HEC. (2006). Hydrologic modelling system HEC-HMS: User's manual [Computer software manual].

van der Knijff, J. M., Younis, J., & Roo, A. P. J. D. (2010, February). LISFLOOD: a GIS-based distributed model for river basin scale water balance and flood simulation. *International Journal of Geographical Information Science*, *24*(2), 189-212. doi: 10.1080/13658810802549154

Warmerdam, F. (2013). *GDAL - Geospatial Data Abstraction Library.* Internet. Retrieved from http://www.gdal.org/index.html (Consulted in 24.05.2013)

Zambrano-Bigiarini, M., & Rojas, R. (2013). hydroPSO: A Model-independent Particle Swarm Optimization Software for Model Calibration. *Environmental Modelling & Software*, *43*, 5-25.

7 Appendices

7.1 The Watersheds.Order method

```r
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData["station"][[1]]
subbasin1 = WatershedsData["subbasin"][[1]]
zhyd1 = WatershedsData["zhyd"][[1]]
river1 = WatershedsData["river"][[1]]
node1 = WatershedsData["node"][[1]]

station1 = SpatialPoints(coords = cbind(4328448.74, 3118576.86), proj4string = slot(subbas
    "proj4string"))
watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
c1_inlet = a[[2]]
c1_outlet = a[[3]]
c2 = a[[4]]
c3 = a[[5]]
node_trib = a[[6]]
sb1 = a[[7]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]
c1_node = a[[11]]

bbox1 = slot(c1, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.998
bbox[, 2] = bbox1[, 2] * 1.002

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])
# plot(r1, col=(topo.colors(255)), xlim=bbox[1,], ylim=bbox[2,], add=T)

plot(c1, xlim = bbox[1, ], ylim = bbox[2, ], border = "Sienna", col = "transparent",
    add = TRUE)
plot(c2, border = "gray25", col = "transparent", add = TRUE)
plot(c3, border = "gray25", col = "transparent", add = TRUE)

plot(slot(watershed, "station"), pch = 24, bg = "orange", add = TRUE)
plot.PolyLineAttribute(c1, "order", 450, 0.8)
plot.PolyLineAttribute(c2, "order", 450, 0.8)
plot.PolyLineAttribute(c3, "order", 450, 0.8)
```

```r
plot(c1_river, col = "blue", lw = 2, add = TRUE)
plot(c1_node, pch = 21, bg = "blue", cex = 0.8, add = TRUE)
plot(nodeIO, pch = 21, bg = "blue", cex = 0.8, add = TRUE)
plot(c1_inlet, pch = 21, bg = "green", add = TRUE)
plot(c1_outlet, pch = 21, bg = "red", add = TRUE)
plot.PointAttribute(nodeIO, "ELEV", 600, 0.7)
title(main = "Current zhyd watershed (1)", sub = "First order tributary watersheds (1.1, 1

# legend
legend("topleft", legend = c("Station", "Input node", "Output node", "Current zhyd",
    "Tributary zhyd", "River network"), pch = c(24, 21, 21, NA, NA, NA), lty = c(NA,
    NA, NA, 1, 1, 1), col = c("orange", "green", "red", "Sienna", "gray25",
    "blue"), bg = "white")

# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4343000, y = 3129000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4317500, yc = 3112500, len = 5000, units = "2.5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4317500, yb = 3115000, len = 750, lab = "N")
```

7.2 The `Watersheds.Order2` method

```r
library(Watersheds)
data(WatershedsData)

station1 = SpatialPoints(coords = cbind(4328650, 3174450), proj4string = slot(subbasin1,
    "proj4string"))
watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
node_trib = a[[6]]
c1_river = a[[10]]
class(c1_river)

watershed2 = new("Watershed", station = node_trib, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = c1, node = node1)
c23 = Watershed.Order2(watershed2)
c2 = c23[[1]]
c3 = c23[[2]]

c2.0 = c2[[1]]
c2_inlet = c2[[2]]
c2_outlet = c2[[3]]
c2.1 = c2[[4]]
c2.2 = c2[[5]]
c2_node_trib = c2[[6]]
c2_sb1 = c2[[7]]
c2_riverIO = c2[[8]]
c2_nodeIO = c2[[9]]
c2_river = c2[[10]]
c2_node = c2[[11]]

c3.0 = c3[[1]]
c3_inlet = c3[[2]]
c3_outlet = c3[[3]]
c3.1 = c3[[4]]
c3.2 = c3[[5]]
c3_node_trib = c3[[6]]
c3_sb1 = c3[[7]]
c3_riverIO = c3[[8]]
c3_nodeIO = c3[[9]]
c3_river = c3[[10]]
c3_node = c3[[11]]

# subsetting river networks
id = list(gIntersects(c2.1, WatershedsData$river, byid = TRUE))
c21_river = SpDF_Subset(id, WatershedsData$river)
```

```r
id = list(gIntersects(c2.2, WatershedsData$river, byid = TRUE))
c22_river = SpDF_Subset(id, WatershedsData$river)

id = list(gIntersects(c3.1, WatershedsData$river, byid = TRUE))
c31_river = SpDF_Subset(id, WatershedsData$river)

id = list(gIntersects(c3.2, WatershedsData$river, byid = TRUE))
c32_river = SpDF_Subset(id, WatershedsData$river)

# plots
bbox1 = slot(c3.2, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.995
bbox[, 2] = bbox1[, 2] * 1.005

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])

plot(c1, border = "Sienna", lw = 2, xlim = bbox[1, ], ylim = bbox[2, ], add = T)
plot(c2.0, border = "gray55", lw = 2, add = TRUE)
plot(c3.0, border = "gray55", lw = 2, add = TRUE)
plot(c2.1, border = "coral", lw = 2, add = TRUE)
plot(c2.2, border = "coral", lw = 2, add = TRUE)
plot(c3.1, border = "coral", lw = 2, add = TRUE)
plot(c3.2, border = "coral", lw = 2, add = TRUE)

plot(c1_river, col = "cyan", lw = 1, add = TRUE)
plot(c2_river, col = "cyan", add = TRUE)
plot(c3_river, col = "cyan", add = TRUE)
plot(c21_river, col = "cyan", add = TRUE)
plot(c22_river, col = "cyan", add = TRUE)
plot(c31_river, col = "cyan", add = TRUE)
plot(c32_river, col = "cyan", add = TRUE)

plot(station1, col = "orange", pch = 24, bg = "orange", add = T)

title(main = "Current zhyd watershed and \n 1st and 2nd order tributary watersheds")

# legend
legend("bottomright", legend = c("Station", "Current zhyd", "Tributary zhyd, 1st order",
    "Tributary zhyd, 2nd order", "River network"), pch = c(24, NA, NA, NA, NA,
    NA), lty = c(NA, 1, 1, 1, 1), col = c("orange", "Sienna", "gray55", "coral",
    "cyan"), bg = "white")

# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
```

```r
text(x = 4385000, y = 3179000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4320000, yc = 3140000, len = 10000, units = "5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4320000, yb = 3145000, len = 750, lab = "N")
```

7.3 The `Watersheds.IOR1` function

```r
library(Watersheds)
data(WatershedsData)

station1 = SpatialPoints(coords = cbind(4232972, 3327634), proj4string = slot(subbasin1,
    "proj4string"))

watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid = TRUE)
a = Watershed.IOR1(x = nodeIO, dist = dist)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet

bbox1 = slot(c1, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.998
bbox[, 2] = bbox1[, 2] * 1.002



library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])

plot(c1, border = "Sienna", add = T)
plot(station1, pch = 24, col = "orange", bg = "orange", add = TRUE)
plot(c1_river, col = "blue", add = TRUE)
plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)
plot.PointAttribute(c1_outlet, "ELEV", 700, 0.8)

title(main = "Watershed outlet, case I")

# legend
legend("topleft", legend = c("Station", "outlet node", "Current zhyd", "River network"),
    pch = c(24, 21, NA, NA), lty = c(NA, NA, 1, 1), col = c("orange", "red",
        "Sienna", "blue"), bg = "white")
```

```r
# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4263000, y = 3326500, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4220000, yc = 3300000, len = 10000, units = "5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4220000, yb = 3303000, len = 750, lab = "N")
```

7.4 The Watersheds.IOR2 function

```r
library(Watersheds)
data(WatershedsData)

station1 = SpatialPoints(coords = cbind(4301949, 3173053), proj4string = slot(subbasin1,
    "proj4string"))

watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
nodeIO = a[[9]]
c1_river = a[[10]]
c1_node = a[[11]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid = TRUE)
a = Watershed.IOR2(x = nodeIO, dist = dist, node = c1_node)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet

bbox1 = slot(c1, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.998
bbox[, 2] = bbox1[, 2] * 1.002

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])

plot(c1, border = "Sienna", add = T)
plot(station1, pch = 24, col = "orange", bg = "white", add = TRUE)
plot(c1_river, col = "cyan", add = TRUE)
plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)
plot.PointAttribute(c1_outlet, "ELEV", 700, 0.8)

title(main = "Watershed outlet, case II")

# legend
legend("topright", legend = c("Station", "outlet node", "Current zhyd", "River network"),
    pch = c(24, 21, NA, NA), lty = c(NA, NA, 1, 1), col = c("orange", "red",
        "Sienna", "cyan"), bg = "white")
```

```
# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4319000, y = 3186000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4287500, yc = 3163000, len = 5000, units = "2.5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4287500, yb = 3166000, len = 375, lab = "N")
```

7.5 The Watersheds.IOR3 function

```
library(Watersheds)
data(WatershedsData)

station1 = SpatialPoints(coords = cbind(4217199.42, 3353511.83), proj4string = slot(subbas
    "proj4string"))

watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid = TRUE)
a = Watershed.IOR3(x = nodeIO, y = riverIO, dist = dist)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet

bbox1 = slot(c1, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.998
bbox[, 2] = bbox1[, 2] * 1.002

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])

plot(c1, border = "Sienna", lwd = 2, add = T)

plot(station1, pch = 24, col = "orange", bg = "white", add = TRUE)
plot(c1_river, col = "cyan", add = TRUE)
plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)
plot(c1_inlet, pch = 21, col = "green", bg = "white", add = TRUE)
plot.PointAttribute(c1_outlet, "ELEV", 1000, 0.8)
plot.PointAttribute(c1_inlet, "ELEV", 1000, 0.8)
title(main = "Watershed outlet and inlet, case III")

# legend
legend("topleft", legend = c("Station", "inlet node", "outlet node", "Current zhyd",
    "River network"), pch = c(24, 21, 21, NA, NA), lty = c(NA, NA, NA, 1, 1),
```

```r
    col = c("orange", "green", "red", "Sienna", "cyan"), bg = "white")

# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4239000, y = 3370000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4185000, yc = 3340000, len = 10000, units = "5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4185000, yb = 3343000, len = 750, lab = "N")
```

7.6 The Watersheds.IOR4 function

```
library(Watersheds)
data(WatershedsData)

station1 = SpatialPoints(coords = cbind(4357947, 3284525), proj4string = slot(subbasin1,
    "proj4string"))
watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid = TRUE)
a = Watershed.IOR4(x = nodeIO, y = riverIO, dist = dist)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet

bbox1 = slot(c1, "bbox")
bbox = matrix(0, 2, 2)
bbox[, 1] = bbox1[, 1] * 0.998
bbox[, 2] = bbox1[, 2] * 1.002

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), xlim = bbox[1, ], ylim = bbox[2, ])

plot(c1, border = "sienna", add = T)

plot(station1, pch = 24, col = "orange", bg = "white", add = TRUE)
plot(c1_river, col = "blue", add = TRUE)
plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)
plot(c1_inlet, pch = 21, col = "green", bg = "white", add = TRUE)
plot.PointAttribute(c1_outlet, "ELEV", 1000, 0.8)
plot.PointAttribute(c1_inlet, "ELEV", 1000, 0.8)

title(main = "Watershed outlet and inlet, case IV")

# legend
legend("topleft", legend = c("Station", "inlet node", "outlet node", "Current zhyd",
    "River network"), pch = c(24, 21, 21, NA, NA), lty = c(NA, NA, NA, 1, 1),
    col = c("orange", "green", "red", "Sienna", "blue"), bg = "white")

# axis axis(1); axis(2)

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4384000, y = 3289000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)

# map scale
library(maps)
map.scale(xc = 4347500, yc = 3270000, len = 10000, units = "5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4347500, yb = 3273000, len = 750, lab = "N")
```

7.7 The Karlshafen and Whambeck Stations watersheds

```
## the Wahmbeck Station watershed
station1 = SpatialPoints(coords = cbind(4287441, 3168693), proj4string = slot(subbasin1,
    "proj4string"))

watershed = new("Watershed", station = station1, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

a = Watershed.Order(watershed)
c1 = a[[1]]

## the Karlshafen Station watershed

station2 = SpatialPoints(coords = cbind(4281860, 3170824), proj4string = slot(subbasin1,
    "proj4string"))

watershed = new("Watershed", station = station2, subbasin = subbasin1, zhyd = zhyd1,
    river = river1, c1 = subbasin1, node = node1)

aa = Watershed.Order(watershed)
c1a = aa[[1]]

## defining common axis

(bbox = slot(c1, "bbox"))
(bboxa = slot(c1a, "bbox"))
(xmin = 0.999 * min(c(bbox[1], bboxa[1])))
(xmax = 1.001 * max(c(bbox[1, 2], bboxa[1, 2])))
(ymin = 0.999 * min(c(bbox[2, 1], bboxa[2, 1])))
(ymax = 1.001 * max(c(bbox[2, 2], bboxa[2, 2])))

# plotting both stations
plot(station1, xlim = c(xmin, xmax), ylim = c(ymin, ymax))
plot(station2, add = T)
axis(1)
axis(2)

# further parameters of the Wahmbeck Station watershed

riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1

boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid = TRUE)
```

```r
a = Watershed.IOR4(x = nodeIO, y = riverIO, dist = dist)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet
plot(c1, border = "sienna", add = T)

library(raster)
r1 = brick("dem_grid_todo1_Weser.tif")
plot(r1, col = (topo.colors(255)), add = T)

plot(c1, border = "sienna", add = T)
plot(station1, pch = 24, col = "orange", bg = "white", add = TRUE)

plot(c1_river, col = "cyan", add = TRUE)
plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)
plot(c1_inlet, pch = 21, col = "green", bg = "white", add = TRUE)
plot.PointAttribute(c1_outlet, "ELEV", 1000, 0.8)
plot.PointAttribute(c1_inlet, "ELEV", 1000, 0.8)

# further parameters of the Karlshafen Station watershed

riverIOa = aa[[8]]
nodeIOa = aa[[9]]
c1_rivera = aa[[10]]

# determining inlet and outlet watershed nodes determining distances of
# nodeIO to c1

boundary = gBoundary(c1a)
dist = gDistance(nodeIOa, boundary, byid = TRUE)
a = Watershed.IOR3(x = nodeIOa, y = riverIOa, dist = dist)
c1_inlet = a["inlet"][[1]]
c1_inlet
c1_outlet = a["outlet"][[1]]
c1_outlet
plot(c1a, border = "sienna", add = T)

plot(station2, pch = 24, col = "orange", bg = "white", add = TRUE)

plot(station2, pch = 24, col = "red", bg = "white", add = TRUE)

plot(c1_rivera, col = "cyan", add = TRUE)

plot(c1_outlet, pch = 21, col = "red", bg = "white", add = TRUE)

plot(c1_inlet, pch = 21, col = "green", bg = "white", add = TRUE)

plot.PointAttribute(c1_outlet, "ELEV", 1000, 0.8)
plot.PointAttribute(c1_inlet, "ELEV", 1000, 0.8)

title(main = "Karlshafen and Wahmbeck Stations watersheds")

# legend
legend("topleft", legend = c("Station Wahmbeck", "Station Karlshafen", "inlet node",
    "outlet node", "Current zhyd", "River network"), pch = c(24, 24, 21, 21,
    NA, NA), lty = c(NA, NA, NA, NA, 1, 1), col = c("orange", "red", "green",
    "red", "Sienna", "cyan"), bg = "white")

# axis
axis(1)
axis(2)

# map scale
library(maps)
map.scale(xc = 4277500, yc = 3164000, len = 5000, units = "2.5 km", ndivs = 2)

# north arrow
library(GISTools)
north.arrow(xb = 4277500, yb = 3165500, len = 300, lab = "N")

# labeling the color bar
par(xpd = TRUE)  #allow for plotting outside the plot
text(x = 4297000, y = 3179000, labels = "Elevation (m)", srt = 0)
par(xpd = FALSE)
```

7.8 Precipitation time series management

```
setwd("/Users/j_mata01/Documents/timeCapsule/documents/03_III_ifgi_Muenster/03_thesis/03_d

# accesing ncdf file
library(ncdf)
nc = open.ncdf("rr_0.25deg_reg_1995-2013_v9.0.nc")

# Converting the numeric vector to a 'Date' class object representing
# calendar dates using as.Date.

# First of all we need to know the units:
nc$dim$time$units

# extract the variable time
ti = get.var.ncdf(nc, varid = "time")
str(ti)

# convert the time vector to dates
dates = as.Date(ti, origin = c("1950-01-01"))
str(dates)
range(dates)
class(dates)
length(dates)

# recalling the Lon/Lat variables
lon = get.var.ncdf(nc, varid = "longitude")
lat = get.var.ncdf(nc, varid = "latitude")

# Now the precip. data can be represented as a time series: tx =
# get.var.ncdf(nc, varid='rr')
tx = get.var.ncdf(nc, varid = "rr", count = c(-1, -1, 1096))  #cut from 01.01.1995 to 31.1
# tx = get.var.ncdf(nc, varid='rr', start=c(1,1,1000), count=c(-1,-1,1096))

str(tx)
dim(tx)

# locating the coordinates for the current station
which(lon == 9.875)
nc$dim$longitude$vals[202]
lon[202]

which(lat == 51.625)
nc$dim$latitude$vals[106]
lat[106]

# time serie definition ts = as.data.frame(matrix(0,365,2))
# ts[,1]=dates[1:365] ts = as.data.frame(matrix(0,45,2)) ts[,1]=dates[1:45]
ts = as.data.frame(matrix(0, 1096, 2))
```

```r
ts[, 1] = dates[1:1096]

ts[, 2] = tx[202, 106, ]
plot(ts, ty = "l", col = "blue")
write.table(ts, "out_ts.csv", sep = "\t")
```

7.9 Routing and solute transport modelling

```r
library(ReacTran)

# Model formulation

river.model <- function(t = 0, OC, pars = NULL) {
    tran <- tran.volume.1D(C = OC, F.up = F.OC, F.lat = F.lat, Disp = Disp,
        flow = flow.up, flow.lat = flow.lat, V = Volume, full.output = TRUE)
    reac <- -k * OC
    return(list(dCdt = tran$dC + reac, Flow = tran$flow))
}

# Parameter definition

# Initialising river morphology:

nbox <- 500  # number of grid cells
lengthEstuary <- 5974  # length of estuary [m]
BoxLength <- lengthEstuary/nbox  # [m]
Distance <- seq(BoxLength/2, by = BoxLength, len = nbox)  # [m]
Int.Distance <- seq(0, by = BoxLength, len = (nbox + 1))  # [m]

# Cross sectional area: wide river [m2]
CrossArea = 40
# Volume of boxes (m3)
(Volume <- CrossArea * BoxLength)

# Transport coefficients
Disp <- 1000  # m3/s, bulk dispersion coefficient

# flow.up <- 180/3.83 # m3/s, main river upstream inflow
flow.up <- as.numeric(xx_fit[, 1][1])  # m3/s, main river upstream inflow
class(flow.up)

flow.lat.0 <- 180/3.83  # m3/s, side river inflow
F.OC <- 180/3.83  # input organic carbon [mol s-1]
F.lat.0 <- 180/3.83  # lateral input organic carbon [mol s-1]
k <- 10/(365 * 24 * 3600)  # decay constant organic carbon [s-1]

# Model solution

# scenario 1: without lateral input
F.lat <- rep(0, length.out = nbox)
length(F.lat)
flow.lat <- rep(0, length.out = nbox)
Conc1 <- steady.1D(runif(nbox), fun = river.model, nspec = 1, name = "OC")
str(Conc1)
```

```
# scenario 2: with lateral input
F.lat <- F.lat.0 * dnorm(x = Distance/lengthEstuary, mean = Distance[nbox/2]/lengthEstuary
    sd = 1/20, log = FALSE)/nbox

flow.lat <- flow.lat.0 * dnorm(x = Distance/lengthEstuary, mean = Distance[nbox/2]/lengthE
    sd = 1/20, log = FALSE)/nbox

Conc2 <- steady.1D(runif(nbox), fun = river.model, nspec = 1, name = "OC")
str(Conc2)

# Plotting output

# use S3 plot method
plot(Conc1, Conc2, grid = Distance/1000, which = "OC", mfrow = c(2, 1), lwd = 2,
    xlab = "distance [km]", main = "Organic carbon decay in the river on 01.01.1995",
    ylab = "OC Concentration [mM]")

plot(Conc1, Conc2, grid = Int.Distance/1000, which = "Flow", mfrow = NULL, lwd = 2,
    xlab = "distance [km]", main = "Longitudinal change in the water flow rate on 01.01.19
    ylab = "Flow rate [m3 s-1]")

legend("topright", lty = 1:2, col = 1:2, lwd = 2, c("baseline", "+ side river input"))
```

7.10 The ”Watersheds” package: user manual

Package ‘Watersheds’

February 25, 2014

Type Package

Title Spatial watershed aggregation and spatial drainage network analysis

Version 1.0

Date 2013-08-10

Author J.A. Torres

Maintainer J. A. Torres <arturo.torres@uni-muenster.de>

Description A package for watersheds aggregation and spatial drainage network analysis.

License GPL (>= 2)

Depends R (>= 2.10), methods, sp, maptools, rgeos, lattice, splancs

R topics documented:

`Watersheds-package` *Spatial watershed aggregation and spatial drainage network analysis*

Description

Spatial analysis for watersheds aggregation and ordering accordingly to an outlet point and size of tributary watershed of the current watershed. Spatial drainage networks analysis inside the aggregated watersheds.

Details

Package:	Watersheds
Type:	Package
Version:	1.0
Date:	2013-08-10
License:	GPL (>= 2)
Depends:	R (>= 2.10), methods, sp, maptools, rgeos, lattice, splancs, multicore

Creation and handling of objects class `Watershed` for identifying the subbasin that contains the current `station` (class `SpatialPoints`) and subsets the `zhyd` object to subbasin and extract the current `zhy` object that contains `station` via the S4 method `Watershed.Order`. Identification of the inlet and outlet stretches and inlet and outlet nodes of the `zhyd`. Implementation of functions `Watershed. ,IOR1, IOR2, IOR3`, and `IOR4` for determining the actual inlet and outlet nodes. S4 methods `Watershed.Order2` and `Watershed.Tributary` for defining tributary nodes and tributary catchments of the current `zhyd` watershed.

Author(s)

J.A. Torres

Maintainer: J.A. Torres <arturo.torres@uni-muenster.de>

See Also

See Also the class `Watershed` and the methods `Watershed.Order`, `Watershed.Order2` and `Watershed.Tributary`.

`plot.PointAttribute-methods` *Plotting attributes of* `SpatialPointsDataFrame` *objects*

Description

S4 Method for plotting attributes of `SpatialPointsDataFrame` objects.

Methods

`signature(x = "SpatialPointsDataFrame", y = "character", dist = "numeric", cex = "numeric")`

`x` A `"SpatialPointsDataFrame"` object from where the coordinates of the attribute will be retrieved.

`y` A `"character"` with the name of the attribute.

`dist` A `"numeric"` with the distance from the coordinate to plot the attribute text.

`cex` A `"numeric"` with the relative size to plot the attribute text.

`plot.PolyLineAttribute-methods` *Plot attributes of* `Spatial-Lines,Polygons-DataFrame` *objects.*

Description

S4 Method for plotting attributes of `SpatialLinesDataFrame` and `SpatialPolygonsDataFrame` objects.

Methods

`signature(x = "SpatialPolygonsDataFrame", y = "character", dist = "numeric", cex = "numeric")`

`x` `"SpatialPointsDataFrame"` or `"SpatialPointsDataFrame"` object from where the coordinates of the attribute will be retrieved.

`y` `"character"` with the name of the attribute.

`dist` `"numeric"` with the distance from the coordinate to plot the attribute text.

`cex` `"numeric"` with the relative size to plot the attribute text.

`RiverStation` *Intersection of* `SpatialPoints` *and* `SpatialLinesDataFrame`

Description

The function intersects objects `SpatialPoints` and `SpatialLinesDataFrame`. Identyfies the closer stretch(es) to a station. The `SpatialPoints` must be length 1.

Usage

```
RiverStation(x, y, window = 100)
```

Arguments

`x`	An object of class `SpatialPoints` as is defined in package `sp` and length 1.
`y`	An object of class `SpatialLinesDataFrame` as is defined in package `sp`.
`window`	A numeric value that represents the size of the square (window) around the `x` object.

Details

window value magnifies the object x in order to certainly secure the intersection with the object y. The greater value the more intersection area is defined.

Value

An object SpatialLinesDataFrame that is a subsect of th object x that represents the current intersection withe object x.

Author(s)

J.A. Torres

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
river1 = WatershedsData$river

tributary = RiverStation(station1, river1)
plot(tributary, col="blue")
plot(station1,pch=21,bg="red",cex=.8,add=TRUE)
plot.PolyLineAttribute(x=tributary, y="OBJECTID", dist=100, cex=.8)
title(main="Point station and tributary rivers")
```

SpDF_Subset *Subsetting spatial dataframe objects*

Description

Given and list x of logical values, the function subsets the object z accordingly the TRUE values of x.

Usage

```
SpDF_Subset(x, y)
```

Arguments

x	A list of logical values where TRUE values indicates the index of the subset.
y	A spatial object as is defined in package sp from extracting the subset.

Value

A spatial object of the same class of y.

Author(s)

J.A. Torres

Examples

```
library(Watersheds)
data(WatershedsData)

# subsetting the river Werra subbasin
id = list(gIntersects(WatershedsData$rWerra, WatershedsData$subbasin,byid=TRUE))
subbasin_rWerra = SpDF_Subset(id,WatershedsData$subbasin)
plot(subbasin_rWerra)

# subsetting the river Werra zhyd watersheds
id = list(gIntersects(WatershedsData$rWerra, WatershedsData$zhyd,byid=TRUE))
zhyd_rWerra = SpDF_Subset(id,WatershedsData$zhyd)
plot(WatershedsData$rWerra,col="blue",lwd=1,add=TRUE)
plot(zhyd_rWerra,col="green3",add=TRUE)
title("Subbasin River Weser and primary zhyd watersheds")

# subsetting the river Werra river drainage watersheds
id = list(gIntersects(subbasin_rWerra, WatershedsData$river,byid=TRUE))
river_rWerra = SpDF_Subset(id,WatershedsData$river)
plot(subbasin_rWerra)
plot(WatershedsData$rWerra,col="blue",lwd=3,add=TRUE)
plot(river_rWerra,col="blue1",add=TRUE)
title("Subbasin River Weser and drainage network")
```

`SpDF_Touch` *Touch function for spatial objects*

Description

The SpatialDataFrame Touch function. Identifies which nodes has touching lines and retrives a list with two elements.

Usage

```
SpDF_Touch(x, y)
```

Arguments

`x`	An spatial object as is described in package sp.
`y`	An spatial object as is described in package sp.

Value

A list with two elements:

`comp1`	A matrix with the `OBJECTID` of the node (column 1), the maximum number of lines that are touching the node (column 2), and the elevation of that node (column 3).
`comp2`	A matrix with the `OBJECTID` of the lines that are touching the node.

Author(s)

J.A. Torres

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4328448.74, 3118576.86),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
riverIO = a[[8]]
nodeIO = a[[9]]

touch = SpDF_Touch(nodeIO, riverIO)
touch1 = touch[[1]]; touch1
```

`Watershed` *Class* `"Watershed"`

Description

A S4 class `"Watershed"` for representing `"Watershed"` objects.

Objects from the Class

Objects can be created by calls of the form `new("Watershed", ...)`.

Slots

`station`**:** Object of class `"SpatialPoints"` of length 1. Represents a point from which aggregation fo watersheds will occur.

`subbasin`**:** Object of class `"SpatialPolygonsDataFrame"` of length 1. Represents the current boundary of the hydrological units or `zhyd` objects.

`zhyd`**:** Object of class `"SpatialPolygonsDataFrame"`. Represents the current hydrological units (zhyd accordingly to ECRINS (EAA, 2012)) to be analized inside the subbasin boundary.

`river`**:** Object of class `"SpatialLinesDataFrame"` that represents the current river network to be analised inside the subbasin boundary.

`c1`**:** Object of class `"SpatialPolygonsDataFrame"` of lentgh 1. Represents the curren `zhyd` object of analysis.

`node`**:** Object of class `"SpatialPointsDataFrame"`. Represents the current nodes of the river network to be analised inside the subbasin boundary.

Methods

Watershed.Order `signature(x = "Watershed")`: ...

Watershed.Order2 `signature(watershed = "Watershed")`: ...

Watershed.Tributary `signature(x = "SpatialPointsDataFrame", xo = "SpatialPointsDataFrame", y = "S`

Author(s)

J.A. Torres

References

European Environment Agency - EAA. (2012). EEA catchments and rivers network system, ECRINS v1.1. rationales, building and improving for widening uses to Water Accounts and WISE applications (EEA Technical report No. 7/2012). (Luxembourg: Publications Office of the European Union).

See Also

See Also as the functions `Watershed.IOR1`, `Watershed.IOR2`, `Watershed.IOR3`, `Watershed.IOR4`, or the S4 methods `Watershed.Order`, `Watershed.Order2`, `Watershed.Tributary`

Examples

```
data(WatershedsData)
station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=slot(station1,"coords"),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)
```

`Watershed.IOR1` *Watershed inlet and outlet nodes: case 1*

Description

The function determines the inlet and outlet nodes for zhyd watershed objects. This case 1 is for those watersheds that its river inlet and outlet object is length 1 (length(riverIO)=1).

Usage

```
Watershed.IOR1(x, dist)
```

Arguments

`x`	An object `"SpatialPointsDataFrame"` as is described in package sp over the function will search the inlet and outlet nodes of the watershed.
`dist`	A vector with the distances of each point in x to the current zhyd boundary.

Value

A list of length 2:

`inlet`	A `"SpatialPointsDataFrame"` that represents the inlet node of the current zhyd.
`outlet`	A `"SpatialPointsDataFrame"` that represents the outlet node of the current zhyd.

Note

If there are not inlet or outlet node of the current zhyd, 0 is returned.

Author(s)

J.A. Torres

See Also

See Also the functions `Watershed.IOR2`, `Watershed.IOR3`, `Watershed.IOR4`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4232972,3327634),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes
# determining distances of nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid =TRUE)
a = Watershed.IOR1(x=nodeIO, dist=dist)
c1_inlet = a$inlet; c1_inlet
c1_outlet = a$outlet; c1_outlet

plot(c1,col="gray50")
plot(station1,pch=24, bg="blue",add= TRUE)
plot(c1_river, col="blue", add=TRUE)
plot(c1_outlet,pch=21, bg="red",add= TRUE)
plot.PointAttribute(c1_outlet,"ELEV",700,0.8)
title(main="Watershed outlet, case I")
```

`Watershed.IOR2` *Watershed inlet and outlet nodes: case 2*

Description

The function determines the inlet and outlet nodes for zhyd watershed objects. This case 2 is for those watersheds that its river inlet and outlet object is length 2 (length(riverIO)=2).

Usage

```
Watershed.IOR2(x, dist, node)
```

Arguments

`x`	An object `"SpatialPointsDataFrame"` or `"SpatialPoints"` as are described in package `sp` over the function will search the inlet and outlet nodes of the watershed.
`dist`	A vector with the distances of each point in `x` to the current zhyd boundary.
`node`	An object `"SpatialPointsDataFrame"` as are described in package `sp` over the function will search the inlet and outlet nodes of the watershed. It must be the entire node search object.

Value

A list of length 2:

`inlet`	A `"SpatialPointsDataFrame"` that represents the inlet node of the current `zhyd`.
`outlet`	A `"SpatialPointsDataFrame"` that represents the outlet node of the current `zhyd`.

Note

If there are not inlet or outlet node of the current zhyd is returned 0.

Author(s)

J.A. Torres

See Also

See Also the functions `Watershed.IOR1`, `Watershed.IOR3`, `Watershed.IOR4`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
```

```
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4330341.36,3284797.06),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
nodeIO = a[[9]]
c1_river = a[[10]]
c1_node = a[[11]]

# determining inlet and outlet watershed nodes
# determining distances of nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid =TRUE)
a = Watershed.IOR2(x=nodeIO, dist=dist, node=c1_node)
str(a)
c1_inlet = a$inlet; c1_inlet
c1_outlet = a$outlet; c1_outlet

plot(c1,col="gray60")
plot(station1,pch=24, bg="blue",add= TRUE)
plot(c1_river, col="blue", add=TRUE)
plot(c1_outlet,pch=21, bg="red",add= TRUE)
plot.PointAttribute(c1_outlet,"ELEV",700,0.8)
title(main="Watershed outlet, case II")
```

`Watershed.IOR3` *Watershed inlet and outlet nodes: case 3*

Description

The function determines the inlet and outlet nodes for zhyd watershed objects. This case 3 is for those watersheds that its river inlet and outlet object is length 3 (length(riverIO)=3).

Usage

```
Watershed.IOR3(x, y, dist)
```

Arguments

x	An object "SpatialPointsDataFrame" as is described in package sp over them the function will search the inlet and outlet nodes of the watershed.
y	An object "SpatialLinesDataFrame" as is described in package sp that represents the inlet and outlet rivers of the watershed.
dist	A vector with the distances of each point in x to the current zhyd boundary.

Value

inlet	A "SpatialPointsDataFrame" that represents the inlet node of the current zhyd.
outlet	A "SpatialPointsDataFrame" that represents the outlet node of the current zhyd.

Note

If there are not inlet or outlet node of the current zhyd is returned 0.

Author(s)

J.A. Torres

See Also

See Also the functions `Watershed.IOR1`, `Watershed.IOR2`, `Watershed.IOR4`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4217199.42,3353511.83),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes
# determining distances of nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid =TRUE)
a = Watershed.IOR3(x=nodeIO, y=riverIO, dist=dist)
c1_inlet = a$inlet; c1_inlet
c1_outlet = a$outlet; c1_outlet

plot(c1,col="gray60")
plot(station1,pch=24, bg="blue",add= TRUE)
plot(c1_river, col="blue", add=TRUE)
plot(c1_outlet,pch=21, bg="red",add= TRUE)
plot(c1_inlet,pch=21, bg="green",add= TRUE)
plot.PointAttribute(c1_outlet,"ELEV",1000,0.8)
plot.PointAttribute(c1_inlet,"ELEV",1000,0.8)
title(main="Watershed outlet and inlet, case III")
```

`Watershed.IOR4` *Watershed inlet and outlet nodes: case 4*

Description

The function determines the inlet and outlet nodes for zhyd watershed objects. This case 4 is for those watersheds that its river inlet and outlet object is length 4 (length(riverIO)=4).

Usage

```
Watershed.IOR4(x, y, dist)
```

Arguments

`x`	An object `"SpatialPointsDataFrame"` as is described in package `sp` over them the function will search the inlet and outlet nodes of the watershed.
`y`	An object `"SpatialLinesDataFrame"` as is described in package `sp` that represents the inlet and outlet rivers of the watershed.
`dist`	A vector with the distances of each point in `x` to the current zhyd boundary.

Value

`inlet`	A `"SpatialPointsDataFrame"` that represents the inlet node of the current `zhyd`.
`outlet`	A `"SpatialPointsDataFrame"` that represents the outlet node of the current `zhyd`.

Note

If there are not inlet or outlet node of the current `zhyd` is returned 0.

Author(s)

J.A. Torres

See Also

See Also the functions `Watershed.IOR1`, `Watershed.IOR2`, `Watershed.IOR3`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4357947,3284525),
proj4string=slot(subbasin1,"proj4string"))
```

```
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]

# determining inlet and outlet watershed nodes
# determining distances of nodeIO to c1
boundary = gBoundary(c1)
dist = gDistance(nodeIO, boundary, byid =TRUE)
a = Watershed.IOR4(x=nodeIO, y=riverIO, dist=dist)
c1_inlet = a$inlet; c1_inlet
c1_outlet = a$outlet; c1_outlet

plot(c1,col="gray60")
plot(station1,pch=24, bg="blue",add= TRUE)
plot(c1_river, col="blue", add=TRUE)
plot(c1_outlet,pch=21, bg="red",add= TRUE)
plot(c1_inlet,pch=21, bg="green",add= TRUE)
plot.PointAttribute(c1_outlet,"ELEV",1000,0.8)
plot.PointAttribute(c1_inlet,"ELEV",1000,0.8)
title(main="Watershed outlet and inlet, case IV")
```

`Watershed.Order-methods`
S4 Method for Function `Watershed.Order`

Description

S4 Method for function `Watershed.Order`. Definition of the properties of the current zhyd watershed.

Value

The method returns a list of 11 objects:

`c1`	An object `SpatialPolygonsDataFrame` of length 1 that represents the current zhyd watershed object.
`c1_inlet`	An object `SpatialPointsDataFrame` of length 1 that represents the current inlet node of the zhyd watershed object.
`c1_outlet`	An object `SpatialPointsDataFrame` of length 1 that represents the current outlet node of the zhyd watershed object.
`c2`	An object `SpatialPolygonsDataFrame` of length 1 that represents the greater watershed tributary of the current zhyd watershed object.
`c3`	An object `SpatialPolygonsDataFrame` of length 1 that represents the second watershed tributary of the current zhyd watershed object.
`node_trib`	An object `SpatialPointsDataFrame` of length 2 that represents the station points of the tributary watershed objects.

`sb1`	An object `SpatialPointsDataFrame` of length 1 that represents the subbasin that contains the current zhyd watershed object.
`riverIO`	An object `SpatialLinesDataFrame` that represents the inlet (I) and outlet (O) rivers that crosses the boundary of the current zhyd watershed object.
`nodeIO`	An object `SpatialPointsDataFrame` that represents the nodes of the inlet (I) and outlet (O) rivers that crosses the boundary of the current zhyd watershed object.
`c1_river`	An object `SpatialLinesDataFrame` that represents the river network inside the current zhyd watershed object.
`c1_node`	An object `SpatialPointsDataFrame` that represents the node network inside the current zhyd watershed object.

Methods

`signature(x = "Watershed")` The function takes the object of class `Watershed` and identifies the subbasin that contains the current `station` (class `SpatialPoints`) and subsets the `zhyd` object to subbasin and extract the current `zhy` object that contains `station`. Posteriorly, identifies the inlet and outlet stretches and probable inlet and outlet nodes of the `zhyd`. Then, runs the functions `Watershed. ,IOR1, IOR2, IOR3`, or `IOR4` for determining the actual inlet and outlet nodes. Finally, the method executes the S4 method `Watershed.Tributary` for defining tributary nodes and tributary catchments of the current `zhyd` watershed.

See Also

See Also the class `Watershed` and the methods `Watershed.Order2` and `Watershed.Tributary`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node


station1 = SpatialPoints(coords=cbind(4328448.74, 3118576.86),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
c1_inlet = a[[2]]
c1_outlet = a[[3]]
c2 = a[[4]]
c3 = a[[5]]
node_trib = a[[6]]
sb1 = a[[7]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]
```

```
c1_node = a[[11]]

bbox1 = slot(c1, "bbox")
bbox = matrix(0,2,2)
bbox[,1] = bbox1[,1]*.998
bbox[,2] = bbox1[,2]*1.002

plot(c1, xlim=bbox[1,], ylim=bbox[2,],col="gray50")
plot(c2, col="gray75", add=TRUE)
plot(c3, col="gray85", add=TRUE)
plot(slot(watershed,"station"),pch=24, bg="blue",add= TRUE)
plot.PolyLineAttribute(c1, "order", 450, 0.8)
plot.PolyLineAttribute(c2, "order", 450, 0.8)
plot.PolyLineAttribute(c3, "order", 450, 0.8)
plot(c1_river, col="blue", add=TRUE)
plot(c1_node,pch=21,bg="blue",cex=.5,add=TRUE)
plot(nodeIO,pch=21,bg="blue",cex=.5,add=TRUE)
plot(c1_inlet, pch=21, bg="green",add= TRUE)
plot(c1_outlet,pch=21, bg="red",add= TRUE)
plot.PointAttribute(nodeIO,"ELEV",600,0.7)
title(main="Current zhyd watershed (1)",
sub="First order tributary watersheds (1.1, 1.2)")
```

Watershed.Order2-methods
S4 Method for Function `Watershed.Order2`

Description

S4 Method for function `Watershed.Order2`. Definition of the tributary zhyd watersheds of the current zhyd watershed.

Value

The method returns a list of 2 objects:

`c2`	An object with the output of the method `Watershed.Order` of length 11 for one of the points of `node_trib`. The properties of the greater tributary watershed of the current zhyd watershed.
`c3`	An object with the output of the method `Watershed.Order` of length 11 for the other points of `node_trib`. The properties of the second tributary watershed of the current zhyd watershed.

Methods

`signature(watershed = "Watershed")` The method takes the objec of class `Watershed` when object `node_trib` is length 2. The method identifies the zhyd watershed that contaions the current `station` (class `SpatialPoints`) and apply the method `Watershed.Order` on each point of `node_trib` returning a list of objects `Watershed.Order`. The computation is done via parallel processes for optimizing and take advance of multicore functionalities.

See Also

See Also the class `Watershed` and the methods `Watershed.Order` and `Watershed.Tributary`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node


station1 = SpatialPoints(coords=cbind(4328650,3174450),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
node_trib = a[[6]]
c1_river = a[[10]]

watershed2 = new("Watershed", station=node_trib, subbasin=subbasin1, zhyd=zhyd1, river=river1, c1=c1,node=
c23 = Watershed.Order2(watershed2)
c2 = c23[[1]]
c3 = c23[[2]]

c2.0 = c2[[1]]
c2_inlet = c2[[2]]
c2_outlet = c2[[3]]
c2.1 = c2[[4]]
c2.2 = c2[[5]]
c2_node_trib = c2[[6]]
c2_sb1 = c2[[7]]
c2_riverIO = c2[[8]]
c2_nodeIO = c2[[9]]
c2_river = c2[[10]]
c2_node = c2[[11]]

c3.0 = c3[[1]]
c3_inlet = c3[[2]]
c3_outlet = c3[[3]]
c3.1 = c3[[4]]
c3.2 = c3[[5]]
c3_node_trib = c3[[6]]
c3_sb1 = c3[[7]]
c3_riverIO = c3[[8]]
c3_nodeIO = c3[[9]]
c3_river = c3[[10]]
c3_node = c3[[11]]

# subsetting river networks
id = list(gIntersects(c2.1, WatershedsData$river,byid=TRUE))
c21_river = SpDF_Subset(id,WatershedsData$river)

id = list(gIntersects(c2.2, WatershedsData$river,byid=TRUE))
c22_river = SpDF_Subset(id,WatershedsData$river)
```

```
id = list(gIntersects(c3.1, WatershedsData$river,byid=TRUE))
c31_river = SpDF_Subset(id,WatershedsData$river)

id = list(gIntersects(c3.2, WatershedsData$river,byid=TRUE))
c32_river = SpDF_Subset(id,WatershedsData$river)


# plots
bbox1 = slot(c3.2, "bbox")
bbox = matrix(0,2,2)
bbox[,1] = bbox1[,1]*.995
bbox[,2] = bbox1[,2]*1.005

plot(c1, col="gray50", xlim=bbox[1,], ylim=bbox[2,])
plot(c2.0, col = "gray95", add=TRUE)
plot(c3.0, col="gray79", add=TRUE)
plot(c2.1, col="gray78", add=TRUE)
plot(c2.2, col="gray85", add=TRUE)
plot(c3.1, col="gray53", add=TRUE)
plot(c3.2, col="gray63", add=TRUE)

plot(c1_river, col="blue",add=TRUE)
plot(c2_river, col="blue",add=TRUE)
plot(c3_river, col="blue",add=TRUE)
plot(c21_river, col="blue",add=TRUE)
plot(c22_river, col="blue",add=TRUE)
plot(c31_river, col="blue",add=TRUE)
plot(c32_river, col="blue",add=TRUE)
```

Watershed.Tributary-methods

S4 Method for Function `Watershed.Tributary`

Description

S4 Method for function `Watershed.Tributary`. Definition of the order of tributary `zhyd` watersheds of the current `zhyd` watershed.

Value

The method returns a list of 4 objects:

`c2c3`	A list of length 2 with objects `SpatialPolygonsDataFrame` of length 1 ordered that represents the greater watershed and second tributary of the current `zhyd` watershed object.
`c2`	An object `SpatialPolygonsDataFrame` of length 1 ordered that represents the greater watershed tributary of the current `zhyd` watershed object.
`c3`	An object `SpatialPolygonsDataFrame` of length 1 ordered that represents the second watershed tributary of the current `zhyd` watershed object.
`node_trib`	An object `SpatialPointsDataFrame` of length 2 that represents the station points of the tributary watershed objects.

Methods

```
signature(x = "SpatialPointsDataFrame", xo = "SpatialPointsDataFrame", y = "SpatialLinesDataFram
```

See Also

See Also the class `Watershed` and the methods `Watershed.Order` and `Watershed.Order2`.

Examples

```
library(Watersheds)
data(WatershedsData)

station1 = WatershedsData$station
subbasin1 = WatershedsData$subbasin
zhyd1 = WatershedsData$zhyd
river1 = WatershedsData$river
node1 = WatershedsData$node

station1 = SpatialPoints(coords=cbind(4328448.74, 3118576.86),
proj4string=slot(subbasin1,"proj4string"))
watershed = new("Watershed",station=station1,subbasin=subbasin1,
zhyd=zhyd1,river=river1,c1=subbasin1,node=node1)

a = Watershed.Order(watershed)
c1 = a[[1]]
c1_inlet = a[[2]]
c1_outlet = a[[3]]
sb1 = a[[7]]
riverIO = a[[8]]
nodeIO = a[[9]]
c1_river = a[[10]]
c1_node = a[[11]]

a = Watershed.Tributary(x=c1_inlet,xo= c1_outlet,y=riverIO,z=nodeIO,zhyd=zhyd1, c1=c1)
c2c3 = a[[1]]
c2 = a[[2]]
c3 = a[[3]]
node_trib = a[[4]]

bbox1 = slot(c2c3, "bbox")
bbox = matrix(0,2,2)
bbox[,1] = bbox1[,1]*.998
bbox[,2] = bbox1[,2]*1.002

plot(c1, xlim=bbox[1,], ylim=bbox[2,],col="gray50")
plot(c2, col="gray75", add=TRUE)
plot(c3, col="gray85", add=TRUE)
plot(slot(watershed,"station"),pch=24, bg="blue",add= TRUE)
plot.PolyLineAttribute(c1, "order", 450, 0.8)
plot.PolyLineAttribute(c2, "order", 450, 0.8)
plot.PolyLineAttribute(c3, "order", 450, 0.8)
plot(c1_river, col="blue", add=TRUE)
plot(node_trib,pch=21,bg="red",cex=.8,add=TRUE)
plot.PointAttribute(node_trib,"ELEV",600,0.7)
title(main="Current zhyd watershed (1)",
sub="First order tributary nodes (1.1, 1.2)")
```

`WatershedsData` *A dataset of the ECRINS database for the river Weser basin, Germany.*

Description

The European Environment Agency (EEA) has been developed the Catchments and Rivers Network System (ECRINS) version 1.1. The ECRINS is the hydrographical system currently in use at the European level as well as widely serving as the reference system for the Water Information System (WISE) (EEA,2012). The current version of ECRINS is based on previous work carried out by the Joint Research Centre (JRC) Catchment Characterisation and Modelling (CCM) and the EEA (European Lakes, Dams and Reservoirs Database (Eldred2), European Rivers and Catchments (ERICA)), (EEA,2012).

Usage

```
data(WatershedsData)
```

format

`basin`**:** an object `SpatialPolygonsDataFrame` as is defined in package `sp` that represents the river Weser basin. The `data` slot contains 6 variables as attributes of 1 obaservation.

`ctry:` an object `SpatialPolygonsDataFrame` as is defined in package `sp` that represents the administrative boundary of Germany. The `data` slot contains 6 variables as attributes of 1 obaservation.

`node:` an object `SpatialPointsDataFrame` as is defined in package `sp` that represents the nodes of the ECRINS river network of the river Weser basin. The `data` slot contains 13 variables as attributes of 3882 obaservations.

`rAller` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Aller, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 88 observations.

`rDiemel` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Diemel, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 39 observations.

`rFulda` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Fulda, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 82 observations.

`rHunte` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Hunte, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 34 observations.

`river` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the ECRINS river network of the river Weser basin. The `data` slot contains 52 variables as attributes of 3874 observations.

`rWerra` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Werra, a major tributary of the river Weser. The `data` slot contains 74 variables as attributes of 120 observations.

`rWeser` an object `SpatialLinesDataFrame` as is defined in package `sp` that represents the basin of the river Weser. The `data` slot contains 74 variables as attributes of 104 observations.

rWiumme an object SpatialLinesDataFrame as is defined in package sp that represents the basin of the river Wiumme, a major tributary of the river Weser. The data slot contains 74 variables as attributes of 18 observations.

station an object SpatialPoints as is defined in package sp that represents a point of interest for which the watershed will be aggregated an ordered. Could be a point with the coordinates of a measurement station.

subbasin an object SpatialPolygonsDataFrame as is defined in package sp that represents the subbasins of the tributaries of the river Weser. The data slot contains 4 variables as attributes of 4 observations.

zhyd an object SpatialPolygonsDataFrame as is defined in package sp that contains the primary hydrological units of the river Weser basin accordingly with ECRINS. The data slot contains 50 variables as attributes and 915 observations.

References

European Environment Agency - EAA. (2012). EEA catchments and rivers network system, ECRINS v1.1. rationales, building and improving for widening uses to Water Accounts and WISE applications (EEA Technical report No. 7/2012). (Luxembourg: Publications Office of the European Union).

Examples

```
data(WatershedsData)

# plotting river Weser basin
plot(WatershedsData$ctry)
plot(WatershedsData$basin, col="green4", add=TRUE)
title("River Weser basin, Germany")

# plotting river Weser basin
plot(WatershedsData$ctry)
plot(WatershedsData$basin, col="green4", add=TRUE)
title("River Weser basin, Germany")

# plotting subbasins river Weser basin
plot(WatershedsData$basin)
plot(WatershedsData$subbasin, col="green3",add=TRUE)
plot(WatershedsData$rWeser,col="blue",lwd=2,add=TRUE)
plot(WatershedsData$rAller,col="blue",lwd=1,add=TRUE)
plot(WatershedsData$rDiemel,col="blue",lwd=1,add=TRUE)
plot(WatershedsData$rFulda,col="blue",lwd=1,add=TRUE)
plot(WatershedsData$rHunte,col="blue",lwd=1,add=TRUE)
plot(WatershedsData$rWerra,col="blue",lwd=1,add=TRUE)
plot(WatershedsData$rWiumme,col="blue",lwd=1,add=TRUE)
title("Subbasins River Weser")

# plotting primary zhyd watersheds and drainage network inside river Werra subbasin
# subsetting the river Werra subbasin
id = list(gIntersects(WatershedsData$rWerra, WatershedsData$subbasin,byid=TRUE))
subbasin_rWerra = SpDF_Subset(id,WatershedsData$subbasin)
plot(subbasin_rWerra)

# subsetting the river Werra zhyd watersheds
id = list(gIntersects(WatershedsData$rWerra, WatershedsData$zhyd,byid=TRUE))
zhyd_rWerra = SpDF_Subset(id,WatershedsData$zhyd)
```

```
plot(WatershedsData$rWerra,col="blue",lwd=1,add=TRUE)
plot(zhyd_rWerra,col="green3",add=TRUE)
title("Subbasin River Weser and primary zhyd watersheds")

# subsetting the river Werra river drainage watersheds
id = list(gIntersects(subbasin_rWerra, WatershedsData$river,byid=TRUE))
river_rWerra = SpDF_Subset(id,WatershedsData$river)
plot(subbasin_rWerra)
plot(WatershedsData$rWerra,col="blue",lwd=3,add=TRUE)
plot(river_rWerra,col="blue1",add=TRUE)
title("Subbasin River Weser and drainage network")
```

Index

Masters
Program
in **Geospatial**
Technologies