Internship research project: Implementing advanced routing in the SPHY model using the convection-diffusion equation

Author: Femke Jansen

Supervisors

FutureWater:
Wilco Terink MSc.

Wageningen University:
Drs. Paul Torfs

HWM-70424
March 14, 2017

FutureWater
Onderzoek en advies, gericht op de duurzame toekomst van ons water

Wageningen UR
For quality of life
Abstract


Hydrological models, such as the Spatial Processes in HYdrology (SPHY) model, are often used as a tool to simulate and predict discharge dynamics by modelling hydrological processes in catchments. This can be used to address water-related challenges in the area. To accurately simulate discharge in a channel it is important to implement streamflow routing in the model, which describes the transport of water through an open-channel network. An advanced routing procedure using the convection-diffusion (cD) equation has been implemented in the SPHY model. This cD-routing module has been set up outside the PCRaster environment which is used by the SPHY model. This provides the opportunity to run the routing module with a different temporal and spatial resolution than the simulation of the SPHY model itself, contrary to the currently implemented simple routing scheme. The cD-routing procedure has been tested in the Tamakoshi river basin in Nepal. After calibration of the wave celerity (c) and the diffusion coefficient (D) the results obtained are comparable to the currently implemented routing scheme in terms of model performance. NS values of 0.80 and 0.72 were obtained for the two observation stations Busti and Rasnalu respectively. Computational efforts are comparable for the current simple routing scheme and for the new implemented cD-routing scheme.

Keywords: SPHY, routing, convection-diffusion, numerical discretization, Tamakoshi, Nepal
Contents

Abstract ................................................................. III

1 Introduction ......................................................... 6
   1.1 Problem description ......................................... 6
   1.2 Objectives and research questions ......................... 7
   1.3 Structure of report ........................................... 7

2 Application example ............................................... 8

3 SPHY model .......................................................... 9
   3.1 Model structure .................................................. 9
   3.2 Current routing procedure ..................................... 11
      3.2.1 Simple routing scheme .................................... 11
      3.2.2 Advanced routing scheme ................................... 12

4 Potential routing methods ......................................... 13
   4.1 Dynamic wave equations ....................................... 13
   4.2 Convection-diffusion approximation ......................... 14
   4.3 Kinematic wave approximation ................................. 15
   4.4 Selecting a routing method .................................... 15

5 Numerical approximation ......................................... 16
   5.1 Finite-difference schemes ..................................... 16
      5.1.1 Explicit finite-difference ................................ 16
      5.1.2 Implicit finite-difference ................................ 17
      5.1.3 Crank-Nicolson ............................................. 17
   5.2 Boundary conditions ........................................... 18
   5.3 Numerical discretization of cD-equation ...................... 18

6 Advanced routing module implementation ........................ 21
   6.1 Preprocessing .................................................... 21
      6.1.1 Preparing data and maps .................................. 21
      6.1.2 Define the stream network ................................ 22
      6.1.3 Determine stream order .................................... 24
      6.1.4 Identify reaches ............................................ 25
   6.2 Conservation of mass ........................................... 31
   6.3 Temporal and spatial grid ..................................... 31
   6.4 Initial and boundary conditions ............................... 31
   6.5 Lateral inflow ................................................... 31
   6.6 Parameter values to be specified in config-file ............... 34
   6.7 The cD-equation ................................................ 36
   6.8 Determine downstream hydrographs using cD-equation ......... 40
      6.8.1 Determine locations of inlets and outlets ............... 40
      6.8.2 Determine connecting points ............................... 42
      6.8.3 Locate connecting reaches ................................ 43
      6.8.4 Accumulated runoff at inlets and outlets ............... 45
CONTENTS

6.8.5 Downstream boundary conditions ................................................. 47
6.9 Apply cD-equation for routing ....................................................... 53
6.10 Visualisation of the results .......................................................... 59
6.11 Time series at observation stations ............................................... 60
6.12 Calibration ..................................................................................... 63
6.13 Adjustments made in SPHY script ................................................. 64

7 Results and Discussion ...................................................................... 66
  7.1 Model output and analysis .............................................................. 66
  7.2 Sensitivity analysis ......................................................................... 70
    7.2.1 Wave celerity ........................................................................ 70
    7.2.2 Diffusion coefficient ................................................................ 71
    7.2.3 Temporal resolution .............................................................. 71
    7.2.4 Spatial resolution .................................................................... 71
  7.3 Computational time .......................................................................... 72

8 Conclusion .......................................................................................... 77

9 Recommendations ............................................................................... 79

A Overview python tables .................................................................... 83
B Reach length based on ldd map ......................................................... 85
C Calibration log-file ............................................................................ 86
List of Figures

2.1 Tamakoshi river basin ........................................................................... 8

3.1 SPHY model structure. (copied from Terink et al., 2015). ................. 10

5.1 (a) Explicit finite-difference discretization, (b) implicit finite-difference discretization, (c) Crank-Nicolson finite-difference discretization. ................................. 16

6.1 River network demonstrating Strahlers principle to determine the stream order ................................................................. 23

6.2 Define the stream network using the DEM. ......................................... 24

6.3 River network where all sub-reaches of stream order 2 are highlighted and numbered. ........................................................ 26

6.4 Determine the streamorder for all reaches and identify all sub-reaches per streamorder. .......................................................... 30

6.5 Resulting hydrographs at several distances along the stream channel. .................................................................................................. 32

6.6 Resulting hydrographs at several distances along the stream channel after adding lateral inflow at two places along the channel. ......... 32

6.7 Resulting hydrographs at several distances along the stream channel after adding lateral inflow at two places along the channel. ......... 33

6.8 Components that together form the lateral inflow to be injected to the next reach at the connection point. ................................................................. 34

6.9 River network with examples of the inlet-, outlet- and connected grid cell visualised. ................................................................. 41

6.10 Codes linked to the local drain direction. ............................................. 42

6.11 Retrieve 'uphydro' and 'downhydro' for the reach of maximum streamorder from the time series of accumulated runoff (stored in TimeArray_inlets and TimeArray_pits respectively). These serve as input for the cD-equation of which the results are stored in 'Q_record_back' (Fig.(a)). Determine at which cell number the reaches are connected, in order to retrieve the time series for 'downhydro' from 'Q_record_back' (Fig.(b)). ........................................ 47

6.12 Retrieve 'uphydro' from the time series of accumulated runoff (stored in TimeArray_inlets). 'Downhydro' is retrieved from time series stored in 'Q_record_back' of the reach this stream flows into. These serve as input for the cD-equation of which the results are stored in 'Q_record_back' (Fig.(a)). Determine at which cell number the reaches are connected, in order to retrieve the time series for 'downhydro' from 'Q_record_back' (Fig.(b)). ........................................ 52

6.13 'Uphydro' and 'downhydro' were stored during the code run in section 6.8.5. These serve as input for the cD-equation of which the results are stored in 'Q_record'. The results of the outlet cell, as stored in 'Q_record', serves as lateral inflow of the connecting reach downstream (Fig.(b)).............................................................................. 58

7.1 Daily observed discharge, SPHY simulated discharge using the simple routing procedure and SPHY simulated discharge using the cD-routing procedure for the streamflow stations Busti (ID 647) and Rasnalu (ID 650) for 10 years (a & b) and for the year 2004 (c & d). Values of c and D were calibrated based on the 10 year data series. ................................................................. 67

7.2 Daily observed and SPHY simulated discharge using the cD-routing procedure for the streamflow stations Busti (ID 647) and Rasnalu (ID 650). Values of c and D were calculated using equations 6.2 and 6.3. ......................................................................................... 68

7.3 Example of the visualisation of the simulation results using the cD-routing procedure. Red colours indicate high streamflow (in m$^3$ s$^{-1}$) and purple colours low streamflow. ........................................................................ 70
7.4 Sensitivity analysis of wave celerity for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). Values of $c$ are given in km hr$^{-1}$. The diffusion coefficient is kept constant at 1.0 km$^2$ hr$^{-1}$.

7.5 Sensitivity analysis of the diffusion coefficient for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). Values of $D$ are given in km$^2$ hr$^{-1}$. The wave celerity is kept constant at 0.08 km hr$^{-1}$.

7.6 Sensitivity analysis of the temporal resolution for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). $D_t$ is given in hours and spatial resolution is kept constant at 0.25 km. Values of $c$ and $D$ are kept constant as well at 0.08 km hr$^{-1}$ and 0.1 km$^2$ hr$^{-1}$ respectively.

7.7 Sensitivity analysis of the spatial resolution for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). $D_x$ is given in metres and temporal resolution is kept constant at 24 hr. Values of $c$ and $D$ are kept constant as well at 0.08 km hr$^{-1}$ and 0.1 km$^2$ hr$^{-1}$ respectively.
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Example of the resulting ‘reachID’ table which stores the locations of the cells belonging to the specified stream order (r.126–128).</td>
<td>26</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of the resulting ‘ReachArrayID’ (r.165) and ‘SubReachArrayID’ (r.201) tables, which are similar to ‘reachID’ but now has been ordered based on the value of SlopeLength, needed to order the cells in downstream direction.</td>
<td>29</td>
</tr>
<tr>
<td>6.3</td>
<td>The different ‘IDreach’ tables contain information about the locations of e.g. the inlets or outlets of each reach (r.267–269). The different ‘ID’ tables contain similar information as the ‘IDreach’ tables, however now in the order that PCRaster scans the raster (r.287, 290, 294 and 297). ‘connectID’ shows the location where a stream flows into the next stream (r.328).</td>
<td>41</td>
</tr>
<tr>
<td>6.4</td>
<td>Example of the resulting table showing which reaches are coupled to each other (r.402).</td>
<td>44</td>
</tr>
<tr>
<td>6.5</td>
<td>The ‘TimeArrays’ store for each location of interest the accumulated runoff in m³ s⁻¹ at that point for every timestep (r.500–503).</td>
<td>46</td>
</tr>
<tr>
<td>6.6</td>
<td>‘Q_record_back’ and ‘Q_record’ store the results of the cD-equation for each cell of a reach for every timestep (r.668 &amp; 812). ‘back’ refers to running the cD-equation in upstream direction in order to provide the best possible estimation of the downstream boundary hydrographs for the individual reaches (see Sect. 6.8.5). The number of columns equals the number of cells of the considered reach and the number of rows equals the number of time steps.</td>
<td>51</td>
</tr>
<tr>
<td>6.7</td>
<td>‘Uphydro’ and ‘Downhydro’ store the hydrographs at the inlet and outlet of a reach respectively. The number of rows equals the number of time steps.</td>
<td>53</td>
</tr>
<tr>
<td>7.1</td>
<td>Overview of the results of the model performance.</td>
<td>69</td>
</tr>
<tr>
<td>7.2</td>
<td>Computational time in minutes for a simulation period of 10 years on a daily base with the original spatial resolution (= 250m) for the two different routing procedures.</td>
<td>72</td>
</tr>
<tr>
<td>A.1</td>
<td></td>
<td>84</td>
</tr>
</tbody>
</table>
1. Introduction

Water shortage, as well as water excess, increase food and drinking water insecurity and additionally may induce ecological threats. Additionally, hydropower facilities depend on a constant water supply to produce enough energy. With the intention to develop hydropower in a region, an understanding of climate change impacts on streamflow and its uncertainty is crucial, as well as to assess the changing probabilities and magnitudes of extreme events. These emerging challenging situations are expected to increase as a consequence of climate variability and change (Rockström et al., 2012; Vorosmarty et al., 2000; Terink et al., 2015). Adequate water supply is essential for the well-being of humans and the environment.

Hydrological models, such as the Spatial Processes in HYdrology (SPHY) model (Terink et al., 2015), are often used as a tool to simulate and predict discharge dynamics by modelling hydrological processes in catchments. This can be used to address water-related challenges in the area (Terink et al., 2015; Wagener and Wheater, 2006; Pechlivanidis et al., 2011). To accurately simulate discharge in a channel it is important to implement streamflow routing in the model, which describes the transport of water through an open-channel network by simulating the propagation of the magnitude, volume and temporal pattern of the flow (Terink et al., 2015; Fread, 1985). Routing procedures create the possibility to describe the hydrograph along the channel network, which provides information about the changes in water flow, for instance the attenuation (diffusion effect) of peak discharges (Ramírez, 2000; O’Sullivan et al., 2012). In the lower areas of a mountainous catchment that are prone to flooding, this attenuating effect can reduce the effects of a flood. Therefore assessment of this effect can support decision makers in defining strategies (Montaldo et al., 2004).

1.1 Problem description

Various types of methods have been developed to apply routing, which can roughly be subdivided into: i) hydrological routing and ii) hydraulic routing. Currently there are two routing schemes included in the SPHY model (Terink et al., 2015) that can be assigned to hydrological routing. The first option is a simple flow accumulation routing scheme, in which for each cell the accumulated amount of water flowing out of the cell into its neighbouring cell downstream is calculated, which thus comprises the amount of water in the cell itself plus the amount of water in the cells upstream. The second option is the fractional accumulation flux routing scheme, which is used in the presence of lakes or reservoirs. In this case part of the water volume stored will become available for routing depending on the actual lake/reservoir storage. Subsequently, this flux will follow the simple flow accumulation routing scheme. A flow recession coefficient (kx) has been implemented in SPHY to account for flow delay, which is needed to prevent all the specific runoff generated within the catchment on one day to end up at the most downstream part of the catchment on that same day.

The current implemented routing schemes in SPHY have proven their applicability under various conditions (Terink et al., 2015). However, it has also shown to favour either quick or slow flow routes, reflected in simulating either simulate peak flows or base flow accurately (p.c. W. Terink), depending on catchment characteristics. Introducing an advanced routing scheme could improve streamflow simulations, which is necessary to address water-related challenges in an area.
1.2 Objectives and research questions

The objective of this research project is to explore options and to implement an advanced routing procedure to improve the current routing scheme and therefore streamflow simulations in the SPHY model.

From this objective the following research question has been formulated:

*How can the routing procedure and therefore streamflow simulations of the SPHY model be improved?*

In order to answer this main research question the following sub-questions are identified:

1. How is the routing procedure currently implemented in the SPHY model?
2. What channel and catchment characteristics are typically available in projects where the SPHY model is applied?
3. What are potential routing methods, considering available data, and how can these be implemented in the SPHY model?
4. What is the effect of using these routing methods on model performance in terms of streamflow simulations and on model computation time?

1.3 Structure of report

A description of the research area will be provided first (Chapter 2), followed by a brief description of the SPHY model (Chapter 3). Subsequently an overview is provided of potential routing methods (Chapter 4) and in chapter 5 several numerical approximation methods are explained. The implementation of the advanced routing scheme and the accompanying programming code is presented in chapter 6. The main results are discussed in chapter 7 and the report ends with the conclusion (Chapter 8) and recommendations (Chapter 9).
2. Application example

This research focuses on the Tamakoshi River Basin in Nepal where FutureWater is performing hydrological research. The SPHY model with its current routing module has already been set up and calibrated for this area. This facilitates comparison of the results with the newly implemented routing module. In this area catchment response and routing is dependent on natural processes without (too much) human intervention. The latter is highly likely to be of influence on the routing when considering a catchment in for instance The Netherlands, which could require additional components to be implemented in the routing procedure. This lies beyond the scope of this internship research project.

The Tamakoshi river basin is located in the north-east of Nepal and lies partly in China and covers an area of 2926 km$^2$. The glaciated basin lies between 27$^\circ$37'42"N to 28$^\circ$19'23"N latitude and 86$^\circ$0'9"E to 86$^\circ$34'12"E longitude in the southern slope of eastern Hindu Kush Himalayan region. Elevation roughly ranges from 850 masl to 7300 masl. On average 20% of the area is estimated to be covered with snow and around 80 glaciers are present in the river basin covering a total area of 110.00 km$^2$ (Khadka et al., 2014).

![Figure 2.1: Tamakoshi river basin](image1)

(a) Location map of Tamakoshi basin in Nepal  
(b) Digital Elevation Map (DEM)
3. SPHY model

The Spatial Processes in HYdrology (SPHY) model developed by FutureWater is a spatially distributed leaky bucket type of model that includes the main terrestrial hydrological and glacier processes and has a flexible spatial resolution (Terink et al., 2015). The SPHY model is written in the Python programming language and makes use of the PCRaster (Karssenberg et al., 2001) dynamic modelling framework. The model code has been made freely available (version 2.1) (FutureWater, 2015). During this internship research project a yet to be released version of the model with improved glacier module has been used. Below a description of the model based on Terink et al. (2015) will be provided.

3.1 Model structure

SPHY is a spatially distributed leaky bucket type of model and is raster based. This means that changes in storage and fluxes are simulated on a cell-by-cell basis, which provides the opportunity to evaluate these changes over time and space. A schematic overview of the model structure is provided in figure 3.1. The soil structure consists of two upper soil reservoirs and a third groundwater reservoir, which complies with the VIC model structure. Drainage from these reservoirs occurs through surface runoff, lateral flow and baseflow respectively. The sum of these components, together with snowmelt and glacier melt if present, is the cell-specific runoff, which forms the water volume available for routing.

Precipitation that falls on a grid cell may be either classified as rain or snow, depending on temperature. Part of the precipitation will be intercepted and evaporated from the land surface, while another part of the liquid precipitation will become surface runoff or will infiltrate into the soil reservoir. There the remainder of the water that does not evaporate, either contributes to river discharge through lateral flow from the first soil layer, or as baseflow after it has percolated to the groundwater. Precipitation in the form of snow contributes to the snow storage, of which the balance is updated accordingly to the simulation of snowmelt and snow accumulation amounts.

Some of the hydrological processes incorporated in the model are contained in modules that can be switched on/off according to the relevance of these processes in the area of interest. This enables to reduce model run time and decrease the number of required input data. The decision on which modules to include in the modeling framework requires knowledge about the catchment characteristics and its relevant processes. The modules present are: glaciers, snow, groundwater, dynamic vegetation, simple routing, and lake/reservoir routing. A detailed description of all hydrological processes and modules incorporated in the SPHY model and how they are implemented can be found in Terink et al. (2015). A more detailed description of the routing procedures as currently implemented in the model will be given in the next section.
Figure 3.1: SPHY model structure. (copied from Terink et al. 2015).
3.2 Current routing procedure

Total cell-specific runoff available for routing through the stream network consists of different runoff components, namely: rainfall runoff, snow runoff, glacier runoff and baseflow (see Eq. 3.1). Rainfall runoff consists of surface runoff and lateral flow from the first soil layer. Baseflow is resulting from the groundwater module, however when this module is switched off, lateral flow from the second soil layer is considered baseflow.

\[ Q_{Tot} = RRo + SRo + GRo + BF \]  

(3.1)

Where:

- \( Q_{Tot} \): Specific runoff on day \( t \) (mm)
- \( RRo \): Rainfall runoff (mm)
- \( SRo \): Snow runoff (mm)
- \( GRo \): Glacier runoff (mm)
- \( BF \): Base Flow (mm)

The SPHY user can select a simple routing scheme or a more complex routing scheme in which discharge from reservoirs/lakes can be included as well through a \( Q-h \) relation. From a Digital Elevation Model (DEM) the flow direction network map is derived, which is used in both routing methods.

3.2.1 Simple routing scheme

Most advanced routing methods, e.g. solving the full dynamic Saint Venant equations and Manning equation, require data on channel geometry and morphology which are often not available for the spatial scale that SPHY generally is applied to. Therefore, SPHY uses a simplified routing method in which the accumulated cell-specific runoff flows into its neighbouring cell. This accumulated cell-specific runoff comprises the amount of water in the cell itself plus the amount of water in the cells upstream and is calculated using the PCRaster function \( \text{accuflux} \). A flow recession coefficient (\( kx \)) has been implemented in SPHY to account for flow delay, which is needed to prevent all the specific runoff generated within the catchment on one day to end up at the most downstream part of the catchment on that same day. If preferred the four streamflow contributors can also be routed separately [Terink et al., 2015].

\[ Q_{Tot_t} = \frac{Q_{Tot_t} \cdot 0.001 \cdot A}{24 \cdot 3600} \]  

(3.2)

\[ Q_{Tot accu,t} = \text{accuflux}(F_{dir}, Q_{Tot_t}) \]  

(3.3)

\[ Q_{Tot rout,t} = (1 - kx) \cdot Q_{Tot accu,t} + kx \cdot Q_{Tot rout,t-1} \]  

(3.4)

Where:

- \( Q_{Tot_t} \): Total cell-specific runoff on day \( t \) (\( m^3 \text{s}^{-1} \))
- \( Q_{Tot} \): Total cell-specific runoff on day \( t \) (mm)
- \( A \): Grid-cell area (\( m^2 \))
- \( Q_{Tot accu,t} \): Accumulated streamflow on day \( t \) without flow delay (\( m^3 \text{s}^{-1} \))
- \( Q_{Tot rout,t} \): Routed streamflow on day \( t \) (\( m^3 \text{s}^{-1} \))
- \( Q_{Tot rout,t-1} \): Routed streamflow on day \( t-1 \) (\( m^3 \text{s}^{-1} \))
- \( F_{dir} \): Flow direction network
- \( kx \): Flow recession coefficient, ranging from 0 (fast) to 1 (slow) (-)
3.2.2 Advanced routing scheme

The second more advanced routing option is the fractional accumulation flux routing scheme, which is used in the presence of lakes or reservoirs. Lakes/reservoirs act as a buffer from which the water is released delayed. The scheme uses the `accufractionflux` and `accufractionstate` PCRaster functions, which calculate for each cell the fraction of the accumulated water that flows out of the cell and the fraction that is stored in the cell respectively. The fraction that is transported out of the cell equals 1 for non-lake cells. For lake-cells however, the volume that becomes available for routing depends on the actual lake storage and the Q-h relation that is provided. Subsequently, this flux will follow the simple flow accumulation routing scheme. Specifics on data requirements and how to use this routing method are provided in section 2.8.2 of [Terink et al.] (2015).

\[
Q_{\text{Tot}_{\text{accu},t}} = \text{accufractionflux}(F_{\text{dir}}, S_{\text{act},t}, Q_{\text{frac},t}) \tag{3.5}
\]

\[
S_{\text{act},t+1} = \text{accufractionstate}(F_{\text{dir}}, S_{\text{act},t}, Q_{\text{frac},t}) \tag{3.6}
\]

Where:

- \( S_{\text{act},t} \) : Actual storage (m\(^3\))
- \( S_{\text{act},t+1} \) : Updated storage to be used in next time step (m\(^3\))
4. Potential routing methods

The dynamics of water in a channel can be described and approximated at different levels of complexity. The full dynamic wave equations, consisting of the continuity equation and the momentum equation, are considered to most accurately describe one-dimensional unsteady flow in open channels (Miller, 1984). However, depending on the application the importance of the various terms in these equations can reduce, which allows us to omit them to simplify the problem. These approximations comprise the continuity equation combined with various simplifications of the momentum equation (Brunner, 1992). This chapter will provide a short description of the full dynamic wave model and of two approximations, i.e. the convection-diffusion and kinematic wave approximation.

4.1 Dynamic wave equations

Dynamic wave routing involve solving the complete one-dimensional Saint Venant flow equations, which consists of the continuity equation (Eq. 4.1) and momentum equation (Eq. 4.2). Assumptions made in deriving these equations are: 1) hydrostatic pressure distribution prevails, hence vertical accelerations are negligible, 2) uniformly distributed velocity across any channel section, 3) small channel bed slope, 4) homogeneous and incompressible flow, and 5) momentum resulting from lateral flow is negligible (Brunner, 1992; Miller, 1984; Litrico and Fromion, 2009; Barati et al., 2012; Shultz, 2007).

\[
\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \tag{4.1}
\]

\[
\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} \left( \frac{Q^2}{A} \right) + gA \frac{\partial h}{\partial x} - gA(S_0 - S_f) = 0 \tag{4.2}
\]

Where:

- \(A\) : Cross-sectional area (m\(^2\))
- \(Q\) : Discharge (m\(^3\) s\(^{-1}\))
- \(x\) : Distance along channel (m)
- \(t\) : Time (s)
- \(g\) : Gravitational acceleration (m s\(^{-2}\))
- \(h\) : Water depth (m)
- \(S_0\) : Bed slope (m m\(^{-1}\))
- \(S_f\) : Friction slope (m m\(^{-1}\))

The terms of the equation of momentum can be described as: 1) local acceleration, 2) convective acceleration, 3) pressure gradient and 4) gravitational and frictional forces (Singh, 1996; Brunner, 1992; Miller, 1984; Shultz, 2007).

The dynamic wave equations are used in various applications, such as river flow forecasting, wave routing in shallow water bodies, dam break flood wave routing, sewer modelling and more in general when a system is subjected to backwater effects due to restrictions downstream such as weirs (Litrico and Fromion).
Since the equations have no general analytical solution it requires numerical techniques, such as the finite element method, finite volume method and finite difference method to solve the non-linear unsteady flow equations. Models such as HEC-RAS and MIKE 11 solve the full dynamic Saint Venant equations using an implicit finite difference method. To solve the Saint Venant equations a lot of data is required related to channel geometry and morphology. This data is not always available, which requires simplification of the equations in which some terms of the momentum equation are neglected.

4.2 Convection-diffusion approximation

The convection-diffusion wave (cD-wave) or diffusion wave is based on the continuity equation and a simplification of the momentum equation, in which the acceleration terms have been omitted. This assumption is made since the two acceleration terms counterbalance each other during rising and recession limbs of a flood wave. Furthermore, in most cases the acceleration terms are significantly smaller than the pressure gradient term. In most real world cases flood waves are diffusion waves. Equation 4.2 therefore reduces to equation 4.3 in which the pressure term is responsible for describing physical diffusion of the flood wave.

\[
\frac{\partial h}{\partial x} - (S_0 - S_f) = 0 \tag{4.3}
\]

The continuity equation can be expressed as:

\[
\frac{\partial h}{\partial t} + u \frac{\partial h}{\partial x} + h \frac{\partial u}{\partial x} = 0 \tag{4.4}
\]

Where:

- \( u \): velocity (m s\(^{-1}\))

Combining equations 4.3 and 4.4 the linearized form of the cD-equation can be written as:

\[
\frac{\partial Q}{\partial t} + c \frac{\partial Q}{\partial x} - D \frac{\partial^2 Q}{\partial x^2} = 0 \tag{4.5}
\]

Where:

- \( c \): wave celerity (m s\(^{-1}\))
- \( D \): diffusion coefficient (m\(^2\) s\(^{-1}\))

The cD-equation is able to describe the translation (c) and the attenuation (D) of a wave as it propagates through the channel. However, the cD-equation is not suitable to apply when the cross-sectional geometry of a river changes considerably or when the slope increases largely, as the acceleration terms would not be negligible in those cases. An advantage of this approximation is that it needs less hydro-morphological data and most natural slow to moderately rising flood waves can be described using this approximation. Implicit numerical schemes to solve the cD-equation require boundary conditions both upstream and downstream, implying that the downstream boundary condition should be known or can be estimated.

In literature little can be found on typical values for c and D. Vakgroep Hydraulica en Afvoerhydrologie found values for c ranging from 1.1 – 2.2 m s\(^{-1}\) and D values ranging from 10.000 – 40.000 m\(^2\) s\(^{-1}\) for different sections of the Rhine near Lobith. Although the conditions of the Rhine river and the Tamakoshi river are far from similar, these values give a sense of the order of magnitude of these parameters.
4.3 Kinematic wave approximation

An even more simplified approximation of the full dynamic wave model is the kinematic wave model, which additionally omits the pressure gradient term. A kinematic wave develops when gravitational and frictional forces approach equilibrium. This is achieved when changes in depth and velocity with respect to distance and change in velocity with respect to time are small compared to channel bed slope. Therefore equation 4.2 reduces further to equation 4.6 (Torfs, 2002; Brunner, 1992; Singh, 1996; Miller, 1984).

\[ S_0 = S_f \] (4.6)

The kinematic wave model does not allow for wave attenuation following the assumption that the pressure gradient term is negligible. Therefore the kinematic wave model is most accurate in urban environments where channels are fairly steep and that there is a relationship between discharge, flow depth and position along the channel (Lighthill and Whitham, 1955; Singh, 1996; Shultz, 2007). The linearized form of the kinematic wave equation can be written as:

\[
\frac{\partial Q}{\partial t} + c \frac{\partial Q}{\partial x} = 0
\] (4.7)

This equation describes the translation of a wave through a channel without hydrograph diffusion. However, depending on the numerical scheme used to solve this equation numerical diffusion can be introduced. Furthermore, backwater effects cannot be simulated using the kinematic wave equations because the model allows disturbances only to travel in downstream direction (Singh, 1996; Brunner, 1992; Miller, 1984).

4.4 Selecting a routing method

The appropriate routing method should be selected depending on the characteristics of a hydrological system, the characteristics of the flood wave, data availability, required computational efficiency and the degree of influence of backwater effects.

Comparing the three methods it can be concluded that the cD-wave equations provide a solid approximation of the full dynamic wave equations. It balances between the accuracy and high data demanding dynamic wave model and the simplicity of the kinematic wave model (Shultz, 2007). Because there is no data on channel geometry and discharge data is scarce for the Tamakoshi river the cD-equation is selected as most suitable routing method for the aims of this research project. Further analysis in this report will therefore focus on applying the cD-equation as a routing method.

Additionally, the routing procedure will be executed in a vector layer that is decoupled from the PCRaster environment. This will enable us to execute the dynamic routing procedure on a different temporal and spatial scale than the model simulation that is performed in the raster grid of PCRaster. The aim is therefore also not necessarily to exactly understand and calculate the dynamics of the open water using the full Saint Venant equations, rather than to analyse the pragmatic feasibility of implementing a routing procedure using a vector layer.
5. Numerical approximation

Non-linear partial differential equations have no general analytical solution, which is why numerical discretization techniques are employed to solve them (Miller, 1984). Examples of numerical discretization techniques to solve unsteady flow equations are the finite element method (FEM), finite volume method (FVM) and finite difference method (FDM). The FDM is appropriate when no complex geometries such as uniform grids need to be handled. Mass is more rigorously conserved using a FVM, and its accuracy and computational times is comparable to FDMs. The FEM is more complex to implement and would require more computational effort. When a problem can be solved with a regularly structured grid, FDM can be more efficient to use (Botte et al., 2000; Chung, 2010; Shukla et al., 2011). We consider the assumption of applying a regular structured grid to be appropriate for routing purposes, especially considering the lack of information and data about channel geometry. Therefore we will focus on FDMs.

5.1 Finite-difference schemes

The FDM is based on applying a local Taylor expansion to approximate the derivatives. It uses a regular network of nodes separated by distance dx to discretize a continuous model domain. In a similar way the time domain is discretized using a time step dt (Shultz, 2007; Koohafkan, 2016). There are many choices that can be made selecting an appropriate numerical scheme. Most FDMs will introduce artificial diffusion when applied to the convection part of the C-D-equation, which might be larger than the actual physical diffusion (Verma et al., 2012; Chung, 2010). Space derivatives are approximated using central difference, but temporal derivatives can be approximated using three different basic schemes: explicit, implicit and using Crank-Nicolson schemes which combines the two (see Fig. 5.1).

Figure 5.1: (a) Explicit finite-difference discretization, (b) implicit finite-difference discretization, (c) Crank-Nicolson finite-difference discretization.

5.1.1 Explicit finite-difference

A finite-difference scheme is said to be explicit when the value of a variable is computed forward in time explicitly depending on the value of the previous time step (see Fig. 5.1a). An explicit method is advantageous over an implicit method considering the ease of programming and the requirements for computer resources as it does not use iterative computations, however a disadvantage is that this method is subject to numerical stability constraints, and should therefore meet the requirements defined by the Courant condition (Equation 5.1) (Koohafkan, 2016). For numerically stable solutions C should
be smaller than 1. Therefore the size of the temporal resolution \( \Delta t \) is restricted, assuming a predefined spatial resolution \( \Delta x \) and a known wave celerity \( c \).

\[
C = \frac{\Delta t}{\Delta x c}
\]  

(5.1)

Where:

- \( C \) : Courant number (-)
- \( \Delta t \) : Time step (s)
- \( \Delta x \) : Spatial step (m)
- \( c \) : Wave celerity (m s\(^{-1}\))

The MacCormack scheme is an example of a 2-step explicit FDM which uses a backwards-looking predictor step, followed by a forward-looking corrector step. This order can also be reversed for each time step. This scheme is easier to apply and it provides more accurate results at coarser spatial and temporal resolutions compared to the Lax diffusive scheme. It has been widely used to solve non-linear equations [Hoffman and Frankel 2001; Pletcher et al. 2012]. The MacCormack scheme is one of the options provided in the rivr-package for the programming language R to solve the full Saint Venant equations [Koohafkan 2016].

### 5.1.2 Implicit finite-difference

An implicit finite-difference scheme numerically solves the equations iteratively as the value for the next time step depends on itself (see Fig. 5.1b). It is unconditionally numerically stable with no restrictions on the size of \( \Delta t \) and \( \Delta x \) [Barati et al. 2012]. However, to achieve high accuracy for a solution, \( \Delta t \) should approach the time step determined from the Courant condition.

### 5.1.3 Crank-Nicolson

A third basic finite-difference scheme is the Crank-Nicolson scheme (see Fig. 5.1c). Suppose a one-dimensional partial differential equation of the form:

\[
\frac{\partial u}{\partial t} = f(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2})
\]  

(5.2)

The Crank-Nicolson method combines the fully explicit (forward Euler) and fully implicit (backward Euler) schemes by computing the average of those:

\[
\frac{u^{t+1}_x - u^t_x}{\Delta t} = f_x^t \left( u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2} \right) \quad \text{(Explicit)}
\]  

(5.3)

\[
\frac{u^{t+1}_x - u^t_x}{\Delta t} = f_x^{t+1} \left( u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2} \right) \quad \text{(Implicit)}
\]  

(5.4)

\[
\frac{u^{t+1}_x - u^t_x}{\Delta t} = \frac{1}{2} \left[ f_x^{t+1} \left( u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2} \right) + f_x^t \left( u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2} \right) \right] \quad \text{(Crank-Nicolson)}
\]  

(5.5)

Since it is an implicit method because the solution does not explicitly depend on the value of the previous time step, it is numerically stable. However, it is prone to oscillations if the time step relative to the spatial step becomes large. It is a second order method because it is centered in time. For one-dimensional (convection-) diffusion problems this method is often resorted to and it generally yields accurate results [Crank and Nicolson 1947; Smith 1985].
5.2 Boundary conditions

To find a numerical solution for a differential equation, initial conditions and boundary conditions need to be specified. There are three commonly used types of boundary conditions. A Dirichlet boundary condition specifies that the state of the dependent variable at the boundary is a known function of time, i.e. in hydraulics this can be defined as fixed head. This boundary condition is relatively easy to implement (Singh, 1996; Shahedi, 2008). Neumann boundary condition constitute the case in which the derivative of the dependent variable is specified, i.e. in hydraulics a fixed flux. When a head-dependent flux is specified at the boundary this is called a mixed boundary condition or Robins boundary condition. It is a linear combination of the Dirichlet and Neumann boundary conditions and specifies a known total flux over the boundary which comprises a diffusion and convection component.

Upstream typically a stage or flow hydrograph is provided, while downstream also a rating curve (Q-h relation) or normal depth (the depth of flow when the slope of the water surface equals the slope of the channel bottom (Singh, 1996)) can be specified to define the boundary condition (Brunner, 2010). The downstream boundary should be chosen such that the backwater effect is minimal. Assuming that backwater effect can be neglected in case of a situation like Tamakoshi river is valid because of the topographical height differences. For a case in The Netherlands this assumption might not hold because of artificial constructions that regulate water levels. Minimizing the influence of backwater can be done in different ways for example: 1) let the river widen dramatically at the downstream point, i.e. artificially creating a situation that simulates conditions in which all water drains into a sea, 2) artificially extend the length of the river such that the possible backwater effect caused by errors in boundary specification will have minimal effect on the actual end of the river, or 3) Provide a Q-h relation when data is available.

The SPHY model simulates the specific runoff for each cell in the raster. This specific runoff consists of surface runoff, lateral flow, baseflow, snowmelt and glacier melt. From this the accumulated runoff for the upstream location of every reach of the steam network has been calculated. This hydrograph is used as the upstream boundary condition. Since there is no data available from which a Q-h relation could be determined the same method has been used for the outlet of the entire stream network. The downstream hydrographs for the smaller individual reaches are derived from results obtained by applying the cD-equation from down- to upstream as a preprocessing step (for further explanation see section 6.8).

5.3 Numerical discretization of cD-equation

The convection-diffusion equation is discretized using the Crank-Nicolson method. Consider the cD-equation:

\[ \frac{\partial Q}{\partial t} = D \frac{\partial^2 Q}{\partial x^2} - c \frac{\partial Q}{\partial x} + f(Q) \]  \hspace{1cm} (5.6)

Where:

\[ f : \text{Reaction term, e.g. lateral flow} \]

Applying the Crank-Nicolson method to the cD-equation, and adding the option to shift between predominantly implicit or explicit (i.e. \( \omega = 1 \rightarrow \text{fully implicit} \), \( \omega = 0 \rightarrow \text{fully explicit} \)) yields:

\[ \frac{Q_{x}^{t+1} - Q_{x}^{t}}{\Delta t} = \frac{D}{2\Delta x^2} \left( (1 - \omega) \left( Q_{x+1}^{t} - 2Q_{x}^{t} + Q_{x-1}^{t} \right) + \omega \left( Q_{x+1}^{t+1} - 2Q_{x}^{t+1} + Q_{x-1}^{t+1} \right) \right) \]

\[ - \frac{c}{4\Delta x} \left( (1 - \omega) \left( Q_{x+1}^{t} - Q_{x-1}^{t} \right) + \omega \left( Q_{x+1}^{t+1} - Q_{x-1}^{t+1} \right) \right) + f(Q_{x}^{t}) \]  \hspace{1cm} (5.7)
Multiplying with $\Delta t$ and defining $\sigma = \frac{D \Delta t}{\Delta x}$ and $\rho = -\frac{C \Delta t}{\Delta x}$ yields:

$$\omega \left( Q_{i+1}^{t+1} + 2\sigma Q_{i}^{t+1} - \sigma Q_{i-1}^{t+1} + \rho Q_{i-1}^{t+1} - \rho Q_{i+1}^{t+1} \right) = (1 - \omega) \left( Q_{i}^{x} - 2\sigma Q_{i}^{x} + \sigma Q_{i-1}^{x} - \rho Q_{i-1}^{x} + \sigma Q_{i+1}^{x} + \rho Q_{i+1}^{x} \right) + \Delta t f \left( Q_{i}^{x} \right)$$

(5.8)

After reordering the equation becomes:

$$(-\sigma + \rho)\omega Q_{i-1}^{t+1} + (1 + 2\sigma \omega)Q_{i}^{t+1} - (\sigma + \rho)\omega Q_{i+1}^{t+1} = (\sigma - \rho)(1 - \omega)Q_{i-1}^{t} + (1 - 2\sigma(1 - \omega))Q_{i}^{t} + (\sigma + \rho)(1 - \omega)Q_{i+1}^{t} + \Delta t f \left( Q_{i}^{t} \right)$$

(5.9)

This equation is valid for spatial indices $x=1$, $\ldots$, $X-2$, in which $X$ is the number of discrete spatial points in the grid. However, for the points at the boundaries, i.e. $x=0$ and $x=X-1$ the equation makes no sense because $Q_{i-1}^{t}$ and $Q_{i}^{X}$ are located outside the grid:

$$x = 0 : (-\sigma + \rho)\omega Q_{0}^{t+1} + (1 + 2\sigma \omega)Q_{0}^{t+1} - (\sigma + \rho)\omega Q_{1}^{t+1} = (\sigma - \rho)(1 - \omega)Q_{0}^{t} + (1 - 2\sigma(1 - \omega))Q_{0}^{t} + (\sigma + \rho)(1 - \omega)Q_{1}^{t} + \Delta t f \left( Q_{0}^{t} \right)$$

(5.10)

$$x = X-1 : (-\sigma + \rho)\omega Q_{X-1}^{t+1} + (1 + 2\sigma \omega)Q_{X-1}^{t+1} - (\sigma + \rho)\omega Q_{X}^{t+1} = (\sigma - \rho)(1 - \omega)Q_{X-1}^{t} + (1 - 2\sigma(1 - \omega))Q_{X-1}^{t} + (\sigma + \rho)(1 - \omega)Q_{X}^{t} + \Delta t f \left( Q_{X-1}^{t} \right)$$

(5.11)

However, by providing at both the upstream and downstream boundaries a specified flow hydrograph, which can be interpreted as a Dirichlet boundary as a function of time, the numerical approximation can be written compactly in a linear system using vector notation that looks like:

$$A \ast Q[t+1] = B \ast Q[t] + f[t]$$

in which:

$$A = \begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
(-\sigma + \rho)\omega & 1 + 2\sigma \omega & (-\sigma + \rho)\omega & 0 & \cdots & 0 \\
0 & (-\sigma + \rho)\omega & 1 + 2\sigma \omega & (-\sigma + \rho)\omega & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & (-\sigma + \rho)\omega & 1 + 2\sigma \omega & (-\sigma + \rho)\omega & 0 \\
0 & \cdots & 0 & (-\sigma + \rho)\omega & 1 + 2\sigma \omega & (-\sigma + \rho)\omega \\
0 & \cdots & 0 & 0 & 0 & 1 \\
\end{bmatrix}, \quad Q^{t+1} = \begin{bmatrix}
Q_{1}^{t+1} \\
Q_{2}^{t+1} \\
Q_{X-1}^{t+1} \\
Q_{X}^{t+1} \\
\end{bmatrix}$$

$$B = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 \\
(\sigma - \rho)(1 - \omega) & 1 - 2\sigma(1 - \omega) & (\sigma + \rho)(1 - \omega) & 0 & \cdots & 0 \\
0 & (\sigma - \rho)(1 - \omega) & 1 - 2\sigma(1 - \omega) & (\sigma + \rho)(1 - \omega) & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & (\sigma - \rho)(1 - \omega) & 1 - 2\sigma(1 - \omega) & (\sigma + \rho)(1 - \omega) & 0 \\
0 & \cdots & 0 & (\sigma - \rho)(1 - \omega) & 1 - 2\sigma(1 - \omega) & (\sigma + \rho)(1 - \omega) \\
0 & \cdots & 0 & 0 & 0 & 0 \\
\end{bmatrix}$$

$$Q^{t} = \begin{bmatrix}
Q_{1}^{t} \\
Q_{2}^{t} \\
Q_{X-1}^{t} \\
Q_{X}^{t} \\
\end{bmatrix}, \quad f = \begin{bmatrix}
\Delta t f(Q_{1}^{t}) \\
\Delta t f(Q_{2}^{t}) \\
\Delta t f(Q_{X-1}^{t}) \\
\Delta t f(Q_{X}^{t}) \\
\end{bmatrix}$$
When enough data is available a Q-h relation could be implemented at the downstream boundary. A Q-h relation can be described by a linear combination of the Dirichlet and Neumann boundary condition:

\[
\alpha \ast \text{Neumann} + \beta \ast \text{Dirichlet} = \gamma \\
\alpha \cdot \frac{\partial Q}{\partial x} (X-1,t) + \beta \cdot Q(X-1,t) = \gamma
\] (5.12) (5.13)

Discretization of equation 5.13 using centered difference for \( x = X-1 \) (i.e. boundary node), gives:

\[
\alpha \cdot \frac{Q_X^t - Q_{X-2}^t}{2\Delta x} + \beta \cdot Q_{X-1}^t = \gamma
\] (5.14)

\[
-Q_{X-2}^t + \frac{2\beta\Delta x}{\alpha} Q_{X-1}^t + Q_X^t = \frac{2\Delta x \gamma}{\alpha}
\] (5.15)

Following equation 5.15 the last three terms of the last row in matrix A and matrix B should be changed to \(-1\), \(\frac{2\beta\Delta x}{\alpha}\), and 1 from left to right respectively and the last term of vector f should become \(\frac{2\Delta x \gamma}{\alpha}\) for a Robin boundary condition. Due to the low data availability a Q-h relation could not be determined for the Tamakoshi river basin. Therefore we are restricted to use the flow hydrograph that could be constructed from the accumulated discharge produced in each cell as simulated by SPHY. However, to make the solution numerically more stable, the linear combination of the Dirichlet and Neumann boundary condition as described in equation 5.15 can be used. For this \(\beta\) will be chosen 1.0 and \(\gamma\) will be the 'observed' flow hydrograph, which in this case is the accumulated discharge simulated by SPHY. By also giving some weight to \(\alpha\) (e.g. \(\alpha = 0.2\)), and thus including the gradient of discharge (i.e. Neumann boundary condition), the solution will numerically become more stable because the gradient of the discharge will not take unrealistic values at the boundary.
6. Advanced routing module implementation

The advanced routing procedure comprises the use of the convection-diffusion equation (cD-equation) for channel flow routing through a drainage network. This method will allow to simulate the shift and attenuation of a wave as it propagates through the stream network. The cD-equation needs to be applied to every reach of the stream network using appropriate initial and boundary conditions. The simulated hydrograph at the outlet of each reach will act as lateral inflow into the connected next reach of a higher stream order. The implementation of this procedure will be described in this chapter. All programming code in this chapter is included in the python script 'CDrouting.py', unless stated otherwise.

6.1 Preprocessing

6.1.1 Preparing data and maps

Several preparatory steps need to be performed before the routing procedure can be started. The programming code for this is provided below (r.27–73). The command lines within the code explain which step is performed.

```python
# Read the input and output directories from the configuration file
inpath = config.get('DIRS', 'inputdir')
outpath = config.get('DIRS', 'outputdir')

# Missing value definition
MV = -9999

# Set the timing criteria
import datetime
datetime = datetime
sy = config.getint('TIMING', 'startyear')
sm = config.getint('TIMING', 'startmonth')
sd = config.getint('TIMING', 'startday')
ey = config.getint('TIMING', 'endyear')
em = config.getint('TIMING', 'endmonth')
ed = config.getint('TIMING', 'endday')
startdate = datetime.datetime(sy,sm,sd)
enddate = datetime.datetime(ey,em,ed)

# Visualisation of results in maps
visFLAG = config.getint('CDROUTING', 'VisualisationFLAG')

# Set clonemap
clonemap = inpath + config.get('GENERAL', 'mask')
pcr.setclone(clonemap)
```
53 clone = pcr.ifthen(pcr.readmap(clonemap), pcr.boolean(1))
54 pcr.report(clone, outpath+'clone.map')
56 DEMadj = pcr.lddcreatedem(pcr.readmap(inpath +
config.get('GENERAL', 'dem')), 1e31, 1e31, 1e31, 1e31)
58 pcr.report(DEMadj, outpath+'demadj.map')
59 DEM = pcr.readmap(outpath + 'demadj.map')
61 stations = pcr.readmap(inpath + config.get('GENERAL','locations'))
62 stations2 = pcr.ifthenelse(stations!=MV, pcr.boolean(1), pcr.boolean(0))
63 stations3 = pcr.ifthen(stations2,DEM)
65 stations_loc = numpy.argmax(pcr.pcr2numpy(stations3, MV)!=MV)
66 stations_loc = numpy.hstack((stations_loc, numpy.zeros((stations_loc.shape[0], 3))))
67 names_table = inpath + config.get('GENERAL','names')
68 stations_name = pcr.lookupscalar(names_table, stations)
69 names = pcr.pcr2numpy(stations_name, MV)
72 Runoff = outpath + config.get('CDROUTING','totrun')

6.1.2 Define the stream network

A map containing the local drain direction is calculated based on the Digital Elevation Model (DEM) using PCRaster in order to define the stream network of the basin (r.75–76). The PCRaster map of the local drain direction is converted into a numpy array (r.81). In this way the values can be adjusted such that pits are created at the inlets of the reaches of the smallest streamorder, and additionally at the outlets of all reaches in the stream network. This will be performed at a later stage (see Sect. 6.1.4). Every cell in the raster will be assigned a flow direction, and therefore every cell is considered to be part of the stream network. To reduce the refinement of the network and to create a network of well-defined streams in which not all cells are assigned to the stream network, a threshold needs to be specified in the SPHY config-file (r.83–85). This threshold specifies the minimal area upstream of a cell (in km$^2$) required for this cell to be assigned to the stream network. Using the spatial resolution of the raster (r.138–141) this area is converted into number of cells using equation 6.1 (r.91). To determine the number of cells upstream of a cell the ‘accuflux’ function as incorporated in PCRaster has been used (r.89–90). This function calculates the accumulated amount of material flowing out of a cell, which consists of the accumulated material produced upstream of the cell added to the material in the cell itself (Karssenberg et al., 2001). Using the number ‘1’ as material, will provide the number of cells upstream of a cell (including the cell itself). If this number is larger than the threshold value, the cell will be assigned to the stream network (r.92–93).

\[
\text{Threshold Number of cells} = \frac{\text{Threshold Area}}{\text{(Spatial resolution)}^2}
\]

(6.1)

Where:

- Threshold Number of cells : Threshold defining number of cells upstream of cell for this cell to be assigned to stream network (-)
- Threshold Area : Minimal area required upstream of a cell for this cell to be assigned to stream network (km$^2$)
- Spatial resolution : Spatial resolution of the raster (km)
6.1. PREPROCESSING

Figure 6.1: River network demonstrating Strahlers principle to determine the stream order.

```python
75 # Calculate local drain direction using Digital Elevation Model (DEM)
76 FlowDir_ini = pcr.lddcreate(DEM, 1e31, 1e31, 1e31, 1e31)
77 # Convert the PCRmap into numpy array 'ldd_inout'.
78 # This array will be used to create pits at the inlets of the reaches of
79 # the smallest streamorder and additionally pits at all outlets.
80 # This is needed to subtract the water from the system at these points.
81 ldd_inout = pcr.pcr2numpy(FlowDir_ini, MV)

83 # Threshold specifies the minimum area (in km²) required upstream of a cell,
84 # for this cell to be assigned to the stream network.
85 threshold_area = config.getfloat('CDROUTING', 'threshold')

#### FROM SCRIPT 'sphy.py' ####
138 # Spatial resolution of PCRaster grid
139 self.SpaceRes = pcr.cellarea()
141 self.SpaceRes = self.SpaceRes[0][0]**0.5/1000
####

87 # Determine stream network (if more than #...cells upstream, then it is considered)
88 # a stream). Assign '1' to streams and '0' to non-streams.
89 accuflux = pcr.accuflux(FlowDir_ini, 1)
90 pcr.report(accuflux, outpath + 'accuflux.map')
91 threshold_cells = int(threshold_area/((SpaceRes/1000)**2))
92 network = pcr.ifthenelse(accuflux > threshold_cells, pcr.nominal(1), pcr.nominal(0))
93 pcr.report(network, outpath + 'network.map')
```
6.1.3 Determine stream order

Once the stream network has been defined the stream order of all reaches belonging to the stream network are specified using the function 'streamorder' as implemented in PCRaster (r.99–101). This function again requires the local drain direction map as input, and therefore will use the map in which the ldd has been determined for all cells belonging to the stream network (r.95–97). To obtain the stream order solely for the defined stream network, the two maps are multiplied (r.103–105). The minimum and maximum stream order are retrieved from the resulting map (r.107–110).

95  # Determine flow direction only for stream network
96  FlowDir_network = pcr.ldd(pcr.scalar(network)*pcr.scalar(FlowDir_ini))
97  pcr.report(FlowDir_network, outpath + 'FlowDir_network.map')

99  # Determine the streamorder map from the FlowDir map
100 streamorder = pcr.streamorder(FlowDir_ini)
101 pcr.report(streamorder, outpath+'streamorder.map')

Figure 6.2: Define the stream network using the DEM.
6.1. PREPROCESSING

103 # Determine for the defined network what the streamorder is
104 orderNetwork = pcr.nominal(pcr.scalar(streamorder)*pcr.scalar(network))
105 pcr.report(orderNetwork, outpath + 'orderNetwork.map')

107 # Determine the minimum and maximum streamorder. For this convert to numpy array,
# make all missing values 0 and flatten it (hstack)
108 orders = numpy.hstack(pcr.pcr2numpy(orderNetwork, 0))
109 strordmin = int(min(i for i in orders if i > 0))
110 strordmax = int(max(i for i in orders if i > 0))

6.1.4 Identify reaches

From the 'orderNetwork' map all reaches per stream order are retrieved and saved in a separate map,
containing all reaches for the regarded stream order (r.119 & 122). Additionally, the location (x,y) of all
cells belonging to that stream order are stored in table reachID (r.126–128).

In general when performing a function using PCRaster, it will scan the raster row by row. This therefore
determines the order in which cells are stored when they meet a certain criteria. Since this order does not
necessarily coincide with downstream neighbouring cells another method was founded to determine neigh-
bouring cells. For this all cells belonging to a reach the value of the DEM (r.120 & 123) and SlopeLength
are calculated (r.121 & 124). At first it was expected that DEM values would systematically decrease in
downstream direction, however due to some small inconsistencies in the map this is not always the case.
SlopeLength is a function in PCRaster that calculates the accumulative-friction-distance of the longest
path upstream (r.112–115), which from a pragmatic point of view, means that this value systematically
increases in downstream direction.

112 # Calculate slopelength - calculates the accumulative friction-distance of the
113 # longest path upstream. Needed to sort the cells of a reach in downstream direction.
114 slopelength = pcr.slopelength(FlowDir_network,1)
115 pcr.report(slopelength, outpath+'slopelength.map')

117 # Determine the reaches and its indices on the map,
# as well as for these reaches subtract DEM and SlopeLength values.
118 for order in range(strordmin,strordmax+1):
119     reach = pcr.ifthenelse(orderNetwork==order, pcr.boolean(1), pcr.boolean(0))
120     demreach = pcr.ifthen(reach,DEM)# if part of the reach, then give DEM value
121     SLreach = pcr.ifthen(reach,slopelength)
122     pcr.report(reach, outpath + 'reach' + str(order) + '.map')
123     pcr.report(demreach, outpath + 'demreach' + str(order) + '.map')
124     pcr.report(SLreach, outpath + 'SLreach' + str(order) + '.map')

126 # Convert PCRmap to numpy array and determine for the reach the location
# (row and col), and store this.
127 reachID = numpy.argwhere(pcr.pcr2numpy(SLreach, MV)!=MV)
128 numpy.savetxt(outpath+'reachid'+str(order)+'.txt', reachID, delimiter=',')

The next step is to separate the individual reaches per stream order. First a matrix consisting of 5
columns is created 'ArrayID' (r.163–164). The first column is reserved to assign an ID number to each
cell. The second and third column contain the row and the column of the cells of all reaches respectively,
indicating their location in the raster (r.167–168). The fourth and fifth column are filled with the DEM
and SlopeLength values respectively (r.153–161 + 169–170). Subsequently the matrix is sorted from low to
high based on SlopeLength values (r.172–173), which means that the cells are now ordered in downstream
direction. The table is saved as 'ReachArrayID_order' (r.177). Now everything is prepared to retrieve
the individual reaches per stream order (r.179–260).
For this purpose the outlet of one reach is found and reported (r.186–187) by determining the maximum of the Slopelength map (i.e. the maximum slopelength will be found at the most downstream point of a reach), which is then assigned TRUE (r.182–183). In case there are more than one cells with the same maximum Slopelength value, then also the DEM value will be considered in order to find a just one unique outlet (r.184–185). For this outlet its catchment is determined using the function ‘catchment’ in PCRaster (r.189–191). This is a sub-catchment of the total considered area. By combining the map of the sub-catchment and the map containing all the reaches of the stream order (r.192–195), the cells of the reach that are located within the sub-catchment are retrieved (r.196) and its locations are saved in a matrix, which again is ordered in downstream direction using Slopelength values and saved in a table named ‘SubReachArrayID’ (r.198–213). Subsequently these cells are removed from the matrix (r.228–232) that contains all reaches of the regarded stream order so a new maximum can be found and therefore the outlet of the next sub-reach, repeating this procedure until all sub-reaches are identified.

In between it is determined in which specific reach the observation stations are located (r.215–226). At a later stage this allows us to retrieve time series of the routing results at these locations in order to compare these simulations to observations.

Furthermore, the array of the local drain direction map \textit{ldd\_inout} is changed into the number ‘5’ for every outlet and for all inlets of reaches with the smallest stream order, as this creates a pit (r.237 & 247). A pit in the local drain direction map will prevent water to flow past this point (see next Section (Sect. 6.2)). The location of the outlets, together with the stream order and sub-reach number are stored in the array \textit{outIDreach} (r.238–241). The next section will discuss on the need to create these pits. Furthermore a map is created in which only the outlets are given \textit{ldd\_out} (r.133 & 236), and similarly a map containing only the inlets of reaches of the smallest stream order \textit{ldd\_in} (r.132 & 246). These maps are needed to inject the water, either retrieved from the system or through the open channel routing, to the right locations. In addition a map is created containing only the inlets of reaches with a stream order $>1$ \textit{ldd\_ups} (r.134 & 250). The latter is used to determine the upstream boundary of each reach when running the cD-equation backwards which is needed to determine the downstream boundary conditions for each reach (see Sect. 6.8).

**Figure 6.3:** River network where all sub-reaches of stream order 2 are highlighted and numbered.
6.1. PREPROCESSING

# Preparation for inlet map (only inlets of smallest streamorder), outlet map and
# a map with inlets of all reaches with streamorder>1
ldd_in = pcr.pcr2numpy(pcr.ifthen(clone==1, pcr.boolean(0)), MV)
ldd_out = pcr.pcr2numpy(pcr.ifthen(clone==1, pcr.boolean(0)), MV)
ldd_ups = pcr.pcr2numpy(pcr.ifthen(clone==1, pcr.boolean(0)), MV)

# Create empty arrays to store the ID’s, ‘order’ and ‘sub’ of the outlets,
# inlets and inlets of all reaches with streamorder>1 respectively.
outIDreach = []
inIDreach = []
upsIDreach = []

##########
# For each stream order order the cells in downstream direction.
# An array is produced which contains: 1)Unique ID, 2)row, 3)col, 4)Elevation,
# 5)SlopeLength (this is used to sort the cells in downstream direction)
##########
for order in range(strordmin,strordmax+1):
    reach = pcr.readmap(outpath + 'reach' + str(order) + '.map')
    reachID = numpy.loadtxt(outpath+'reachid'+str(order)+'.txt', delimiter=',')
    dem = pcr.readmap(outpath + 'demreach' + str(order) + '.map')
    runoffDEM = pcr.ifthen(reach, dem)
    DEMArray = pcr.pcr2numpy(runoffDEM,MV)
    SL = pcr.readmap(outpath + 'SLreach' + str(order) + '.map')
    runoffSL = pcr.ifthen(reach, SL)
    SLArray = pcr.pcr2numpy(runoffSL,MV)

    # Create empty array with 5 columns and length equal to number of cells
    # in the reach.
    ArrayID = numpy.zeros((len(reachID),5))

    # Fill the array:
    for cell in range(0,len(reachID)):
        ArrayID[cell,1]=int(reachID[cell,0])
        ArrayID[cell,2]=int(reachID[cell,1])
        ArrayID[cell,3]=DEMArray[int(reachID[cell,0]),int(reachID[cell,1])]
        ArrayID[cell,4]=SLArray[int(reachID[cell,0]),int(reachID[cell,1])]

    # Sort the array according to SlopeLength
    ArrIDSorted=ArrayID[numpy.argsort(ArrayID[:,4])]

    # After sorting add ID value to the cells. Now the cells are ordered in
    # a downstream way.
    for ID in range(0,len(reachID)):
        ArrIDSorted[ID,0]=ID+1
        numpy.savetxt(outpath + 'ReachArrayID_' + str(order) + '.txt',
                       ArrIDSorted, delimiter=',')

    # Identify and subtract individual sub-reaches within each stream order
    sub=0
    while max(ArrIDSorted[:,4])!=MV:
        # Find the outlet of one of the sub-reaches and make it ‘TRUE’
SLoutlet = pcr.boolean(SL == max(ArrIDSorted[:,4]))
# Needed in case there are more points with the same maximum value
DEMoutlet = pcr.ifthen(SLoutlet,dem)
outlet = pcr.boolean(DEMoutlet == (ArrIDSorted[:,4]==
max(ArrIDSorted[:,4]))[0,3])
pcr.report(outlet, outpath + 'outlet' + str(order) + ' + str(sub+1) + '.map')

# Determine for point '1' its catchment
subcatch = pcr.catchment(FlowDir ini,outlet)
pcr.report(subcatch, outpath + 'subcatch' + str(order) + ' + str(sub+1) + '.map')
# Select from the reaches within this stream order the cells that lie within
# the defined catchment of the accompanying outlet point
subreach=pcr.ifthen(subcatch,reach)
subreach2 = pcr.ifthen(subreach,SL)
pcr.report(subreach2, outpath + 'subreach' + str(order) + ' + str(sub+1) + '.map')
subreachID = numpy.argwhere(pcr.pcr2numpy(subreach2, MV)!=MV)

# Create empty array with 6 columns and length equal to number of cells
# in the reach.
subarrayID = numpy.zeros((len(subreachID),6))
# Fill the array:
for cell in range(0,len(subreachID)):
    subarrayID[cell,1]=int(subreachID[cell,0])
    subarrayID[cell,2]=int(subreachID[cell,1])
    subarrayID[cell,3]=SLArray[int(subreachID[cell,0]),int(subreachID[cell,1])]
    subarrayID[cell,4]=int(order)
    subarrayID[cell,5]=int(sub+1)

# Sort the array according to SlopeLength
subarrIDSorted=subarrayID[numpy.argsort(subarrayID[:,3])]
# After sorting add ID value to the cells.
# Now the cells are ordered in a downstream way.
for ID in range(0,len(subreachID)):
    subarrIDSorted[ID,0]=ID+1
    numpy.savetxt(outpath + 'SubReachArrayID' + str(order) + ' + str(sub+1) + '.txt', subarrIDSorted, delimiter=',')

# Find in which reach (i.e. order and sub) the stations are located
# and store this in 'stations_loc'.
for loc in range(len(stations_loc)):
    # Exclude streams comprising 1 cell
    # (assuming that this will not be the location of a station)
    if numpy.size(subarrIDSorted)>6:
        if stations_loc[loc,0] in subarrIDSorted[:,1]:
            index = numpy.where(subarrIDSorted[:,1]==stations_loc[loc,0])
            if subarrIDSorted[index[0],2]==stations_loc[loc,1]:
                stations_loc[loc,2]=order
                stations_loc[loc,3]=sub+1
                stations_loc[loc,4]=index[0][0]

# Remove (=set to MV) the cells of the subreach regarded from
# the array 'ArrIDSorted' in order to find the next max SL
for subID in range(0,len(subreachID)):
6.1. PREPROCESSING

```python
6.1. PREPROCESSING

230 x=ArrIDSorted[ArrIDSorted[:,1]==subreachID[subID,0]]
231 y=x[:,2]==subreachID[subID,1]
232 ArrIDSorted[int(y[0,0])-1,:]=MV

234 # Create a pit (cell value is 5) in the ldd in at the outlet points of
235 # each subreach. Outlets are the last value of the 'sub array sorted'.
236 ldd_out[int(subarrIDSorted[-1][1]),int(subarrIDSorted[-1][2])]=1
237 ldd_inout[int(subarrIDSorted[-1][1]),int(subarrIDSorted[-1][2])]=5
238 outIDreach.append(int(order))
239 outIDreach.append(int(sub+1))
240 outIDreach.append(int(subarrIDSorted[-1][1]))
241 outIDreach.append(int(subarrIDSorted[-1][2]))

242 # Find index of the most upstream cell of each subreach
243 # and change this in the map to value '1'
244 # Make inlets also pits in ldd in array.
245 if order==strordmin:
246    ldd_in[int(subarrIDSorted[0][1]),int(subarrIDSorted[0][2])]=1
247    ldd_inout[int(subarrIDSorted[0][1]),int(subarrIDSorted[0][2])]=5
248
249 # Change the location of the inlets of reaches with streamorder>1 to '1'
250 if order>strordmin:
251    upsIDreach.append(int(order))
252    upsIDreach.append(int(sub+1))
253    upsIDreach.append(int(subarrIDSorted[0][1]))
254    upsIDreach.append(int(subarrIDSorted[0][2]))
255    inIDreach.append(int(order))
256    inIDreach.append(int(sub+1))
257    inIDreach.append(int(subarrIDSorted[0][1]))
258    inIDreach.append(int(subarrIDSorted[0][2]))

260 sub+=1
```

Table 6.2: Example of the resulting 'ReachArrayID' (r.165) and 'SubReachArrayID' (r.201) tables which are similar to 'reachID' but now has been ordered based on the value of SlopeLength, needed to order the cells in downstream direction.

(a) ReachArrayID

<table>
<thead>
<tr>
<th>ID</th>
<th>x</th>
<th>y</th>
<th>DEM</th>
<th>SlopeLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>350</td>
<td>156</td>
<td>4959.8</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>397</td>
<td>32</td>
<td>707.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>365</td>
<td>67</td>
<td>826.0</td>
<td>0.0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>665</td>
<td>377</td>
<td>87</td>
<td>616.1</td>
<td>38541.6</td>
</tr>
<tr>
<td>666</td>
<td>377</td>
<td>86</td>
<td>600.4</td>
<td>38791.6</td>
</tr>
</tbody>
</table>

(b) SubReachArrayID

<table>
<thead>
<tr>
<th>ID</th>
<th>x</th>
<th>y</th>
<th>SlopeLength</th>
<th>#order</th>
<th>#sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>280</td>
<td>180</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>281</td>
<td>179</td>
<td>353.6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>282</td>
<td>178</td>
<td>707.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>377</td>
<td>87</td>
<td>38541.6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>377</td>
<td>86</td>
<td>38791.6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
CHAPTER 6. ADVANCED ROUTING MODULE IMPLEMENTATION

Figure 6.4: Determine the streamorder for all reaches and identify all sub-reaches per streamorder.

(a) Streamorder determined based on Strahler.

(b) All sub-reaches of order 1

(c) Slope length of these sub-reaches.

(d) DEM of these sub-reaches
6.2 Conservation of mass

So-called ‘pits’ are created at every outlet and inlet of a reach to withdraw water from the system in order to prevent accounting for the volume of water twice through routing as open water and as groundwater through the soil system. With this pit the water is forced out of the system and cannot flow beyond this point. A pit is created by adjusting the local drain direction map at this point, i.e. it is changed into number ‘5’ as this creates a pit by definition. This way the total amount of water in the system is preserved. For a more elaborate explanation consult section 6.5 and see figure 6.8.

6.3 Temporal and spatial grid

The routing module is set up outside the PCRaster environment which provides the opportunity to run the routing module with a different temporal and spatial resolution than the simulation of the SPHY model itself. An advantage of this is that when preferred a finer resolution can be chosen, which improves the quality of the numerical solution. By specifying a smaller time step the numerical solution will approach the analytical solution, however this will increase computational time. Choosing a time step that is significantly larger than the system time scale may lead to instabilities or oscillations. It is therefore a balance between the quality of the numerical approximation and computational time. The temporal scale should also be in balance with the spatial scale to avoid instabilities and to limit excessive numerical diffusion. Time and space problems are coupled through the Courant number (see Eq. 5.1). This number should be smaller than 1 to limit the numerical diffusion. The user receives a warning when the Courant number exceeds 1.

The routing can be calculated on a different temporal resolution than on which the visualisation of the result are done. They are independent of each other, i.e. the routing calculations can be done on an hourly base, while the visualisation can be given on a daily base.

6.4 Initial and boundary conditions

A spatially constant discharge of 25.0 m$^3$s$^{-1}$ has been specified as initial condition for all reaches. This has been calculated from rough estimations of width, depth and flow velocity in the Tamakoshi river. At the upstream boundary a flow hydrograph needs to be specified. This comprises the accumulated discharge of the area upstream of this point. At the downstream boundary preferably a known Q-h relation is provided, however in data-scarce areas such as the Tamakoshi river basin this relation is unknown. Therefore, another option is to provide a flow hydrograph as well, similarly to the upstream boundary. However, to strengthen the numerical stability of the system, it was chosen to use a linear combination of the Dirichlet and Neumann boundary conditions as was explained in section 5.3. The flow hydrograph of the outlet of the entire basin comprises the accumulated discharge of the whole catchment as simulated by SPHY. The downstream flow hydrographs of all the other individual reaches of the stream network are determined by applying the cD-equation from downstream to upstream direction. This method will be explained more elaborately in section 6.8.

6.5 Lateral inflow

An artificial experiment has been done to demonstrate the effect of adding lateral inflow at a certain position along a stream. Figure 6.5 shows how an initial impulse to the stream system shifts and attenuates as the wave travels through the channel. Time is depicted along the x-axis time and the three coloured lines correspond to hydrographs simulated at distance $1/4^{th}$, $2/4^{th}$ and $3/4^{th}$ along the stream respectively.
Figure 6.5: Resulting hydrographs at several distances along the stream channel.

Figure 6.6 is the result of adding lateral flow at both \(1/4\)th and \(2/4\)th of the channel. This affects the hydrographs at these locations instantly.

Figure 6.6: Resulting hydrographs at several distances along the stream channel after adding lateral inflow at two places along the channel.

The same has been done in figure 6.7, but now the timing of the impulse of lateral flow at a distance of \(2/4\)th of the channel is shifted.
There are two sources of water that together form the lateral inflow for the next reach (see Fig. 6.8). The accumulated specific runoff produced by the cells upstream of the inlet of the reach is routed through the channel (dashed blue area). The amount at the outlet of this reach after routing is one source of lateral flow for the next downstream reach ($\text{Water}_{\text{routed}}$). Besides the injection of lateral flow originating from the open channel, an additional flow is added as lateral flow to this point. This water originates from the system and has not been routed yet. This is the total amount of accumulated specific runoff of the cells upstream of the point of injection minus the blue area (i.e. the red area).

Adding the outflow from one reach into the next as lateral inflow can be done using different methods. It can be injected at one single point or it can be divided along a transect. Although knowing that by adding a large volume of water at a single point it can lead to numerical unstable conditions, it has been chosen from a pragmatic point of view to use the point injection here. Choosing a smaller threshold value, and thereby refining the stream network, will decrease the effect of the point injection.

If the threshold to determine the stream network is relatively high, i.e. less cells are assigned to the stream network, the water from the system that is added as lateral flow at one point to the next stream can become considerable large (see Fig. 6.8). As a result a large amount of water that has not been routed yet will suddenly be added to the channel at a single point.
6.6 Parameter values to be specified in config-file

There are several parameters that need to be specified by the user in the SPHY config-file. First the user should specify a threshold value that specifies the minimal area upstream of a cell required for this cell to be assigned to the stream network. This should be provided in km\(^2\). Values for \(\alpha\) and \(\beta\) should be provided to specify the weight to be given to the Neumann and Dirichlet part of the boundary condition respectively. A value for \(\omega\) should be given to specify the Crank-Nicolson scheme to be more implicit or explicit (i.e. \(\omega = 1 \rightarrow\) fully implicit, \(\omega = 0 \rightarrow\) fully explicit).

Furthermore, as was explained in section 6.3, the calculations in the routing module can be done on a different temporal and spatial resolution. Therefore the user must specify the time step at which the routing calculations are performed, i.e. \(dt\), as well as the time step at which the results are provided and visualised, i.e. \(dt_{\text{visualize}}\). The user can opt for \(1/24\), \(3/24\), \(6/24\), \(12/24\) or \(24/24\) daily time steps. The spatial step \(dx\) should be chosen such that the result of the division: \(dx_{\text{clonemap}}/dx\) is an integer number, i.e. \(dx\) can be larger or smaller than the spatial resolution of the clone map that is used. If the user does not specify a value \(dx\) will equal the spatial resolution of the clone map.

In addition, values for the wave celerity (c) and diffusion coefficient (D) should be provided. The user can opt to directly provide a value for both parameters. The other option is to calculate c and D starting from the linearised convection convection-diffusion equation (see Eq. 6.2 and 6.3). For the latter option a representative width and depth of the river should be provided, as well as the roughness coefficient and bottom slope. For further elaboration on the derivation of the equations consult Torfs (2002). Estimation of these parameters are subject to a number of uncertainties and the results of the calculations of c and D are sensitive to slight changes of these parameters. Therefore most probably better results are obtained when c and D are calibrated.

\[
c = \frac{3}{2} S_0^{1/2} C h^{1/2} \tag{6.2}
\]

\[
D = \frac{C h^{3/2}}{2 S_0^{1/2}} \tag{6.3}
\]
Where:

c : wave celerity (m s\(^{-1}\))
D : diffusion coefficient (m\(^2\) s\(^{-1}\))
C : Chézy coefficient (m\(^{1/2}\) s\(^{-1}\))
S\(_0\) : Bed slope (m m\(^{-1}\))
h : Water depth (m)

These calculations are programmed in the script \(cD\):

```python
8  def calcCD(self, width, depth, roughness, slope):
    # Cross sectional area (m2)
10    A=float(width)*float(depth)
11    # Wetted perimeter (m)
12    P=2*depth+width
13    # Hydraulic radius (m)
14    R=A/P
17    # Chezy coefficient (m(1/2)*s-1)
18    Chezy = (1/float(roughness))*(R**(1/6))
20    # Wave celerity
21    c=(3/2.)*(float(slope)**0.5)*Chezy*(float(depth)**0.5)  # m s-1
22    c = c * 60*60  # m hr-1
23    c = c /1e3  # km hr-1
25    # Diffusion coefficient
26    D=(Chezy*(float(depth)**(3/2.)))/(2*(float(slope)**0.5))  # m2 s-1
27    D = D * 60*60  # m2 hr-1
28    D = D /1e6  # km2 hr-1
30    return c,D
```
6.7 The cD-equation

The following functions are used to solve the cD-equation (see Sect. 5.3). The former function \textit{runCDback} is called upon first in order to determine the downstream boundary conditions for all individual reaches (see Sect. 6.8) (r.28–103). The latter function \textit{runCD} is used for the main routing loop (see Sect. 6.9) (r.106–189). They are similar except for the fact that in the function \textit{runCD} lateral flow is included and that the reaches are artificially elongated in order to decrease the effect of the imposed downstream boundary conditions (see Sect. 5.2).

```python
32 def runCDback(up, down, init, length, dx, dt, c, D):
34     # Length of reach in km
35     L = length
37     # Number of spatial steps
38     Nx = int((L/dx)+1)
40     # Divide the volume of water over smaller time steps when the chosen temporal
41     # step is smaller than 24 hours.
42     if dt < 24:
43         up_smalldt = []
44         for i in range(len(up)-1):
45             t_up = (up[i+1]-up[i])/(24/dt)
46             for j in range(1,(24/dt)+1):
47                 up_smalldt.append(up[i]+t_up*j)
48         up = up_smalldt
50     # Number of time steps
51     Nt=len(up)
53     # Define sigma and rho in the context of numerical discretization
54     sigma = float(D*dt)/float(2.*dx*dx)
55     rho = float(-c*dt)/float(4.*dx)
56     print 'Courant [-] = ' + str(4*-rho)
57     print 'Peclet [-] = ' + str(dx*c/D)
58     print '100 numerical time steps [hr] = ' + str(100*dt)
59     print 'Travel distance in 100 numerical time steps [km] = ' + str(c*100*dt)
60     print '100 numerical spatial steps [km] = ' + str(100*dx)
61     print 'Travel time over 100 numerical spatial steps [hr] = ' + str(100*dx/c)
62     print 'Travel time over 50 km [hr] = ' + str(50./c)
64     # Specify the weight given to Neumann and Dirichlet part of the Robin b.c.
65     alpha = config.getfloat('CDROUTING','alpha')
66     beta = config.getfloat('CDROUTING','beta')
68     # Implicit/explicit (omega=1 --> fully implicit, omega=0 --> fully explicit)
69     omega = config.getfloat('CDROUTING','omega')
71     # Create tridiagonal matrices
72     A = np.diagflat([(-sigma+rho)*omega for i in range(Nx-1)], -1) +
73         np.diagflat([1.+2.*sigma*omega for i in range(Nx)]) +
74         np.diagflat([(-(sigma+rho))*omega for i in range(Nx-1)], 1)
75     A[0,:] = np.array([0 for i in range(0,Nx-1)])
77     # If reach is shorter than 4 cells:
78     if Nx<4:
79         A[-1,:] = np.array([0 for i in range(0,Nx-1)]) + [1]
```

36
6.7. THE CD-EQUATION

```python
# In all other cases:
else:
    A[-1, :] = np.array([0 for i in range(0, Nx-3)] + [-1] +
                        [(2*beta*dx)/alpha] + [1])

B = np.diagflat([(sigma-rho)*(1-omega) for i in range(Nx-1)], -1) +
np.diagflat([(1.-2.*sigma*(1-omega) for i in range(Nx)]) +
np.diagflat([(sigma+rho)*(1-omega) for i in range(Nx-1)], 1)
B[0, :] = np.array([0 for i in range(0, Nx)])
B[-1, :] = np.array([0 for i in range(0, Nx)])

# Create vector F
f_vec = np.array([0 for i in range(0, Nx)])

# Specify initial condition
Q = init

# Create empty list to store the results
Q_record = []

# First create a stationary solution which is subsequently provided to the
# main loop. To create this stationarity no impulse is given upstream.
for ti in range(1, 100):
    Q_new = np.linalg.solve(A, B.dot(Q) + f_vec)
    Q = Q_new
    Q_record.append(Q)

# Main loop to solve the system iteratively
for ti in range(1, Nt):
    # Provide upstream hydrograph
    f_vec[0] = up[ti]*dt
    # Provide downstream hydrograph
    f_vec[-1] = (2*dx*down[ti])/alpha*dt
    # Solve the system
    Q_new = np.linalg.solve(A, B.dot(Q) + f_vec)
    Q = Q_new
    Q_record.append(Q)

Q_record = np.array(Q_record)
return Q_record
```

def runCD(up, down, init, lat, check_lat, lat_distr, length, dx, dt, c, D):

    # Length of reach in km
    L = length
    # Artificially elongate the reach for the downstream b.c. to have less influence
    L_long = length * 1.2
    # Number of spatial steps
    Nx = int((L/dx)+1)
    Nx_long = int((L_long/dx)+1)
    # Divide the volume of water over smaller time steps when the chosen temporal
```
if dt < 24:
    up_smalldt = []
    for i in range(len(up)-1):
        t_up = (up[i+1]-up[i])/(24/dt)
        for j in range(1,(24/dt)+1):
            up_smalldt.append(up[i]+t_up*j)
    up = up_smalldt

# Number of time steps
Nt=len(up)

# Define sigma and rho in the context of numerical discretization
sigma = float(D*dt)/float(2.*dx*dx)
rho = float(-c*dt)/float(4.*dx)

print 'Courant [-] = ' + str(4*-rho)
print 'Peclet [-] = ' + str(dx*c/D)

print '100 numerical time steps [hr] = ' + str(100*dt)
print 'Travel distance in 100 numerical time steps [km] = ' + str(c*100*dt)
print '100 numerical spatial steps [km] = ' + str(100*dx)
print 'Travel time over 100 numerical spatial steps [hr] = ' + str(100*dx/c)

# Specify the weight given to Neumann and Dirichlet part of the Robin b.c.
alpha = config.getfloat('CDROUTING','alpha')
beta = config.getfloat('CDROUTING','beta')

# Implicit/explicit (omega=1 --> fully implicit, omega=0 --> fully explicit)
omega = config.getfloat('CDROUTING','omega')

# Create tridiagonal matrices
A = np.diagflat([-sigma+rho)*omega for i in range(Nx-1), -1] +
np.diagflat([1.+2.*sigma*omega for i in range(Nx)] +
np.diagflat([(-(sigma+rho))*omega for i in range(Nx-1)], 1)
A[0, :] = np.array([1] + [0 for i in range(0,Nx-1)])

# If reach is shorter than 4 cells:
if Nx<4:
    A[-1, :] = np.array([0 for i in range(0,Nx-1)] + [1])
else:
    A[-1, :] = np.array([0 for i in range(0,Nx-3)] + [-1] +
        [(2*beta*dx)/alpha] + [1])

B = np.diagflat([(sigma-rho)*(1-omega) for i in range(Nx-1)], -1) +
np.diagflat([1.-2.*sigma*(1-omega) for i in range(Nx)]) +
np.diagflat([(sigma+rho)*(1-omega) for i in range(Nx-1)], 1)
B[0, :] = np.array([0 for i in range(0,Nx)])
B[-1, :] = np.array([0 for i in range(0,Nx)])

# Create vector F
f_vec = np.array([0 for i in range(0,Nx)])

# Specify initial condition
Q = init

# Create empty list to store the results
Q_record = []

# First create a stationary solution which is subsequently provided to the
# main loop. To create this stationarity no impulse is given upstream.
for ti in range(1,100):
    Q_new = np.linalg.solve(A, B.dot(Q)+ f_vec)
    Q = Q_new
    Q_record.append(Q)

# Main loop to solve the system iteratively
for ti in range(1,Nt):
    # Provide upstream hydrograph
    f_vec[0] = up[ti]*dt
    # Add lateral inflows
    for i in range(len(check_lat)):
        f_vec[int(lat[i][0])-1] = (lat[i][ti] + lat[i][ti+Nt])*dt
    # Provide downstream hydrograph
    f_vec[-1] = (2*dx*down[ti])/alpha*dt
    # Solve the system
    Q_new = np.linalg.solve(A, B.dot(Q)+ f_vec)
    Q = Q_new
    Q_record.append(Q)
Q_record_long = np.array(Q_record)
# Cut of the elongated part of the reach
Q_record = Q_record_long[:,0:Nx]

return Q_record


6.8 Determine downstream hydrographs using cD-equation

The cD-equation is applied to every individual reach of the stream network, for which initial and boundary conditions should be provided. For the inlet of a reach the hydrograph that acts as upstream boundary condition is obtained from the table containing total accumulated runoff (see Sect. 6.8.4). This time series is based on simulations of the SPHY model. However, the downstream boundary conditions for each reach are not explicitly specified. In order to define the hydrograph downstream, the best approximation is to use the hydrograph of the cell of the next stream order to which the outlet is connected to. An iterative process is started in which the cD-equation firstly is applied to the highest stream order, coinciding with the main stream. From this the hydrograph of the connecting points to its sub-reaches is retrieved, which act as downstream boundary conditions for the sub-reaches. Subsequently, the cD-equation will be applied to these sub-reaches, from which the hydrograph of the subsub-reaches are obtained. This process continues until all reaches are included. For this process the locations of the inlets and outlets need to be know, as well as the connecting points and which reaches are connecting to each other.

6.8.1 Determine locations of inlets and outlets

In section 6.1 the inlets and outlets of all reaches were determined. Continuing on that, the locations (i.e. row and column in the raster) of all outlets and all inlets are stored in 'pitID' and 'inID' respectively. In addition 'uphydroID' contains the locations of the inlets of all reaches with stream order > 1, and in 'inoutID' the locations of all the outlets together with the inlets of the reaches of the smallest stream order are stored. For further analysis it is needed to convert the array of the adjusted local drain direction (i.e. containing pits) back to a PCRmap, as well as converting the array with the inlets to a PCRmap.

262 # Reshape the arrays to 4 columns.
263 # Save the arrays containing 'order', 'sub' and 'x and y'
264 outIDreach=numpy.reshape(outIDreach, (len(outIDreach)/4,4))
265 inIDreach=numpy.reshape(inIDreach, (len(inIDreach)/4,4))
266 upsIDreach=numpy.reshape(upsIDreach, (len(upsIDreach)/4,4))
267 numpy.savetxt(outpath + 'outIDreach.txt', outIDreach, delimiter=',')
268 numpy.savetxt(outpath + 'inIDreach.txt', inIDreach, delimiter=',')
269 numpy.savetxt(outpath + 'upsIDreach.txt', upsIDreach, delimiter=',')

271 # Convert ldd array 'ldd_inout' back to pcr-map. This map contains pits at the
272 # inlets of the reaches of the smallest streamorder and additionally pits at all outlets.
273 ldd_upd = pcr.numpy2pcr(Ldd, ldd_inout, MV)
274 pcr.report(ldd_upd, outpath + 'ldd_upd.map')
275 # Convert inlet array to pcr-map (inlets of streamorder 1)
276 inlets = pcr.numpy2pcr(Boolean, ldd_in, MV))
277 pcr.report(inlets, outpath + 'inlets.map')
278 # Convert outlet array to pcr-map (outlets of all streamorders)
279 outlets = pcr.numpy2pcr(Boolean, ldd_out, MV)
280 pcr.report(outlets, outpath + 'outlets.map')
281 # Convert inlets of streamorder>1 array to pcr-map
282 ups = pcr.numpy2pcr(Boolean, ldd_ups, MV)
283 pcr.report(ups, outpath + 'ups.map')

285 # Determine locations of only all the outlets
286 pitID = numpy.argwhere(ldd_out==1)
287 numpy.savetxt(outpath + 'pitID.txt', pitID, delimiter=',')
288 # Determine locations of only all the inlets of streamorder 1
289 inID = numpy.argwhere(ldd_in==1)
290 numpy.savetxt(outpath + 'inID.txt', inID, delimiter=',')
291 # Determine locations of inlets of all stream orders > 1
292 # Needed to have a first estimation of the upstream b.c. when runCDback.
293 uphydroID = numpy.argwhere(ldd_ups==1)
6.8. DETERMINE DOWNSTREAM HYDROGRAPHS USING CD-EQUATION

294  numpy.savetxt(outpath + 'uphydroID.txt', uphydroID, delimiter=',')
295  # Determine locations of only all the outlets plus inlets of streamorder 1
296  inoutID = numpy.argwhere(lddInOut==5)
297  numpy.savetxt(outpath + 'inoutID.txt', inoutID, delimiter=',')

Table 6.3: The different 'IDreach' tables contain information about the locations of e.g. the inlets or outlets of each reach (r.267–269). The different 'ID' tables contain similar information as the 'IDreach' tables, however now in the order that PCRaster scans the raster (r.287, 290, 294 and 297). 'connectID' shows the location where a stream flows into the next stream (r.328).

<table>
<thead>
<tr>
<th>(a) out/in/upsIDreach</th>
<th>(b) pit/in/uphydro/inoutID</th>
<th>(c) connectID</th>
</tr>
</thead>
<tbody>
<tr>
<td>#order</td>
<td>#sub</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>280</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>214</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>222</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>58</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>165</td>
</tr>
</tbody>
</table>

Figure 6.9: River network with examples of the inlet-, outlet- and connected grid cell visualised.
6.8.2 Determine connecting points

To find the points where two reaches are connected to each other the local drain direction code has been used. For every cell the local drain direction map assigns an arrow pointing to its downstream neighbouring cell, which is linked to a value from 1 to 9 according to figure 6.10. Starting from the location of the outlet the connecting point can be determined using this local drain direction by adding or subtracting a row and/or column.

![Figure 6.10: Codes linked to the local drain direction.](image)

```python
# Finding the connecting point to the stream of the next order using ldd-direction
ldd_array = pcr.pcr2numpy(FlowDir_ini, MV)
connectID=numpy.zeros((len(pitID),2)) # Create empty array with 2 columns
for i in range(0,len(pitID)):
    direction = ldd_array[pitID[i][0]][pitID[i][1]]
    if direction == 1:
        connectID[i,0] = pitID[i][0]+1
        connectID[i,1] = pitID[i][1]-1
    if direction == 2:
        connectID[i,0] = pitID[i][0]+1
        connectID[i,1] = pitID[i][1]
    if direction == 3:
        connectID[i,0] = pitID[i][0]+1
        connectID[i,1] = pitID[i][1]+1
    if direction == 4:
        connectID[i,0] = pitID[i][0]
        connectID[i,1] = pitID[i][1]-1
    if direction == 6:
        connectID[i,0] = pitID[i][0]
        connectID[i,1] = pitID[i][1]+1
    if direction == 7:
        connectID[i,0] = pitID[i][0]-1
        connectID[i,1] = pitID[i][1]-1
    if direction == 8:
        connectID[i,0] = pitID[i][0]-1
        connectID[i,1] = pitID[i][1]
    if direction == 9:
        connectID[i,0] = pitID[i][0]-1
        connectID[i,1] = pitID[i][1]+1
numpy.savetxt(outpath + 'connectID.txt', connectID, delimiter=',')
```

6.8.3 Locate connecting reaches

The following programming code is used to determine which reaches are connecting to each other:

```python
# Create empty list.
coupled_reaches=[]
for i in range(0,len(outIDreach)):
    # Find where in pitID the first entry of outIDreach is located
    index = numpy.where(numpy.logical_and(pitID[:,0]==outIDreach[i,2],
                                            pitID[:,1]==outIDreach[i,3]))[0][0]
    # At this same index in connectID the corresponding connecting point can be found
    row=connectID[index][0]
    col=connectID[index][1]
    # Find the order that is one higher than the one considered
    order=outIDreach[i,0]+1
    # Determine the number of sub-reaches that order has
    Nreaches=inIDreach[:,0].tolist().count(float(order))
    if Nreaches>1:
        # Continue this loop until the corresponding connecting reach is found
        sub=1
        found='false'
        while found!='true':
            if order>strordmax:
                found='true'
            else:
                # Read file containing the locations of the sub-reaches of the higher order
                reachIDs = numpy.loadtxt(outpath + 'SubReachArrayID_' + str(order) + '_.' + str(sub) + '.txt', delimiter=',')
                # If reach consists of only 1 cell
                if len(reachIDs)/6.==1:
                    # Check whether reachIDs contains the connecting cell that is considered
                    if numpy.logical_and(reachIDs[1]==float(row),
                                          reachIDs[2]==float(col)).tolist():
                        found = 'true'
                        coupled_reaches.append(int(outIDreach[i,0]))
                        coupled_reaches.append(int(outIDreach[i,1]))
                        coupled_reaches.append(int(order))
                        coupled_reaches.append(int(sub))
                    else:
                        found = 'false'
                        sub+=1
                else:
                    # Check whether reachIDs contains the connecting cell that is considered
                    order+=1
                    sub=1
                    Nreaches=inIDreach[:,0].tolist().count(float(order))
            else:
                # Check whether reachIDs contains the connecting cell that is considered
                # or when all reaches have been regarded
                sub+=1
                if sub==Nreaches+1:
                    order+=1
                    sub=1
                    Nreaches=inIDreach[:,0].tolist().count(float(order))
```

43
# If that is TRUE, then store the number of the order and the sub-reach # of the reaches that are connected to each other # If FALSE, check the next sub-reach
a = (numpy.logical_and(reachIDs[:,1]==float(row), reachIDs[:,2]==float(col))).tolist()
if a.count(True)==1:
    found = 'true'
coupled_reaches.append(int(outIDreach[i,0]))
coupled_reaches.append(int(outIDreach[i,1]))
coupled_reaches.append(int(order))
coupled_reaches.append(int(sub))
else:
    found='false'
sub+=1
if sub==Nreaches+1:
    order+=1
    sub=1
    Nreaches=inIDreach[:,0].tolist().count(float(order))

# Reshape the list.
# 1st column: order of sub-reach, 2nd column: sub of sub-reach,
# 3rd column: order to which sub-reach drains (=order of reach), 4th column: sub of reach
coupled_reaches=numpy.reshape(coupled_reaches, (len(coupled_reaches)/4,4))
numpy.savetxt(outpath + 'coupled_reaches.txt', coupled_reaches, delimiter=',')

An example of the format of the resulting table is shown in table 6.4:

Table 6.4: Example of the resulting table showing which reaches are coupled to each other (r.402).

<table>
<thead>
<tr>
<th>Order</th>
<th>Sub-reach</th>
<th>Order</th>
<th>Sub-reach</th>
</tr>
</thead>
<tbody>
<tr>
<td>upstream</td>
<td>downstream</td>
<td>upstream</td>
<td>downstream</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
6.8. DETERMINE DOWNSTREAM HYDROGRAPHS USING CD-EQUATION

6.8.4 Accumulated runoff at inlets and outlets

The following code is used to determine the flow hydrograph for each pit (\textit{TimeArrayPit}) and inlet of stream order 1 (\textit{TimeArrayIn}) as simulated by the SPHY model without routing. A separate table is created to store these hydrographs of only the inlets of all reaches with stream order > 1 (\textit{TimeArrayUphydro}), as well as for the hydrograph of the outlet of the entire catchment (\textit{TimeArrayOutlet}). Discharge is given in m$^3$ s$^{-1}$.

```python
##########
# Determine daily accumulated runoff for each pit and inlet for all timesteps
##########
timestep = (enddate-startdate).days+1
# Create empty arrays that have a number of rows equal to the number of timesteps, 
# and have a number of columns equal to the amount of outlets/inlets respectively
TimeArrayPit = numpy.zeros(((timestep+1),(len(pitID)+1)))
TimeArrayIn = numpy.zeros(((timestep+1),(len(inID)+1)))
TimeArrayUphydro = numpy.zeros(((timestep+1),(len(uphydroID)+1)))
TimeArrayOutlet = []

# For-loop over all timesteps
for time in range(1,timestep+1):
    # Read the runoff time-series
    runoff = pcr.readmap(pcrm.generateNameT(Runoff, time))
    runoff = pcr.ifthen(clone, runoff*0.001*pcr.cellarea()/(24*3600.))
    # Create map
    pcr.report(runoff, outpath + 'runoff_' + str(time) + '.map')
    # Calculate the accuflux of runoff for each pit and inlet and 
    # write it to numpy array
    accu_pit = pcr.ifthen(ldd_upd==5,pcr.accuflux(ldd_upd,runoff))
    pcr.report(accu_pit, outpath + 'accu_pit.map')
    accu_in = pcr.ifthen(inlets==1, pcr.accuflux(ldd_upd,runoff))
    pcr.report(accu_in, outpath + 'accu_in.map')
    accu_up = pcr.ifthen(ups==1, pcr.accuflux(FlowDir_ini,runoff))
    # Convert PCRmap to numpy array
    runoff_array_pit = pcr.pcr2numpy(accu_pit,MV)
    runoff_array_in = pcr.pcr2numpy(accu_in,MV)
    runoff_array_ups = pcr.pcr2numpy(accu_up,MV)

    # Fill the time array with time series of the accuflux of runoff for all pits
    for j in range(0,len(pitID)):
        TimeArrayPit[0,0] = MV
        TimeArrayPit[0,j+1] = j+1 # first row shows the pit number
        TimeArrayPit[time,0] = time # first column shows timestep
        # find amount of total runoff per pit
        TimeArrayPit[time,j+1] = runoff_array_pit[pitID[j][0]][pitID[j][1]]

    # Fill the time array with time series of the accuflux of runoff for all inlets 
    # of streamorder 1
    for j in range(0,len(inID)):
        TimeArrayIn[0,0] = MV
        TimeArrayIn[0,j+1] = j+1 # first row shows the inlet number
        TimeArrayIn[time,0] = time # first column shows timestep
        # find amount of total runoff per inlet
        TimeArrayIn[time,j+1] = runoff_array_in[inID[j][0]][inID[j][1]]
```

45
# Fill the time array with time series of the accuflux of runoff for all inlets
for j in range(0, len(uphydroID)):
    TimeArrayUphydro[0,0] = MV
    TimeArrayUphydro[0,j+1] = j+1 # first row shows the inlet number
    TimeArrayUphydro[time,0] = time # first column shows timestep
    # Find amount of total runoff per inlet
    TimeArrayUphydro[time,j+1] = runoff_array[uphydroID[j][0]][uphydroID[j][1]]

# Calculate the accuflux of the runoff for the outlet of the total area
accu_outlet = pcr.accuflux(FlowDir_ini, runoff)
runoff_outlet = pcr.mapmaximum(accu_outlet)
runoff_outlet = pcr.pcr2numpy(runoff_outlet,MV)
runoff_outlet = runoff_outlet[0][0]
TimeArrayOutlet.append(runoff_outlet)

# Save the time arrays
numpy.savetxt(outpath + 'TimeArray_pits.txt', TimeArrayPit, delimiter=',')
numpy.savetxt(outpath + 'TimeArray_inlets.txt', TimeArrayIn, delimiter=',')
numpy.savetxt(outpath + 'TimeArray_ups.txt', TimeArrayUphydro, delimiter=',')
numpy.savetxt(outpath + 'TimeArray_outlet.txt', TimeArrayOutlet, delimiter=',')

Table 6.5: The 'TimeArrays' store for each location of interest the accumulated runoff in m$^3$ s$^{-1}$ at that point for every timestep (r.500–503).

<table>
<thead>
<tr>
<th>Timestep</th>
<th>location 1</th>
<th>location 2</th>
<th>...</th>
<th>location N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0.010</td>
</tr>
<tr>
<td>2</td>
<td>0.034</td>
<td>0.028</td>
<td>...</td>
<td>0.304</td>
</tr>
<tr>
<td>3</td>
<td>0.052</td>
<td>0.044</td>
<td>...</td>
<td>0.328</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T-1</td>
<td>1.813</td>
<td>2.045</td>
<td>...</td>
<td>0.741</td>
</tr>
<tr>
<td>T</td>
<td>0.943</td>
<td>1.215</td>
<td>...</td>
<td>0.636</td>
</tr>
</tbody>
</table>
6.8. DETERMINE DOWNSTREAM HYDROGRAPHS USING CD-EQUATION

(a) Run cD-equation for most downstream reach (highest stream order).

(b) Retrieve 'downhydro' from 'Q_record' for connecting upstream reaches.

Figure 6.11: Retrieve 'uphydro' and 'downhydro' for the reach of maximum streamorder from the time series of accumulated runoff (stored in TimeArray_inlets and TimeArray_pits respectively). These serve as input for the cD-equation of which the results are stored in 'Q_record_back' (Fig.(a)). Determine at which cell number the reaches are connected, in order to retrieve the time series for 'downhydro' from 'Q_record_back' (Fig.(b)).

6.8.5 Downstream boundary conditions

Now the iterative process is started in which the cD-equation is applied to all reaches (r.667), starting at the main stream (Fig. 6.11) and working in upstream direction (Fig. 6.12). This is done to approximate the downstream boundary condition of each reach since this is the best approximation that can be done with the information available.

```
# Calculate cD-equation for every reach of the stream network to determine downstream boundary conditions, starting at the main reach (highest stream order), and continue the calculation towards the smaller branches.
print 'Starting cD-equation from down- to upstream to determine the conditions at the downstream point of each reach.'
for order in range(strordmax,strordmin-1,-1):
    # Number of sub-reaches of the order (by counting the number of rows that have the same order)
    Nsubs_this=inIDreach[:,0].tolist().count(float(order))
```

47
# Determine Q_record_back for the highest streamorder

if order == strordmax:
    print order

    # Determine location of inlet
    rowIDin = upsIDreach[numpy.where(numpy.logical_and(upsIDreach[:,0]==float(order), upsIDreach[:,1]==float(1)))[0][0]][2]
    colIDin = upsIDreach[numpy.where(numpy.logical_and(upsIDreach[:,0]==float(order), upsIDreach[:,1]==float(1)))[0][0]][3]

    # Find where in uphydroID (= which row) the regarded inlet is located
    indexIN = numpy.where(numpy.logical_and(uphydroID[:,0]==rowIDin, uphydroID[:,1]==colIDin))[0][0]

    # Select the timeseries from TimeArrayUphydro
    uphydro_maxorder = TimeArrayUphydro[1:,indexIN+1]
    numpy.savetxt(outpath + 'uphydro' + str(order) + '_' + str(1) + '.txt', uphydro_maxorder, delimiter=',')

    # Determine location of outlets/pits
    rowIDout = outIDreach[numpy.where(numpy.logical_and(outIDreach[:,0]==float(order), outIDreach[:,1]==float(1)))[0][0]][2]
    colIDout = outIDreach[numpy.where(numpy.logical_and(outIDreach[:,0]==float(order), outIDreach[:,1]==float(1)))[0][0]][3]

    # Find where in pitID (= which row) the regarded pit is located
    indexOUT = numpy.where(numpy.logical_and(pitID[:,0]==rowIDout, pitID[:,1]==colIDout))[0][0]

    # Select the timeseries from TimeArrayOutlet
    downhydro_maxorder = TimeArrayOutlet

    if dt < 24:
        # If dt<24: divide every timestep by 24/dt
        down_smalldt = []
        for i in range(len(downhydro_maxorder)-1):
            t_down = (downhydro_maxorder[i+1] - downhydro_maxorder[i]) / (24/dt)
            for j in range(1, (24/dt)+1):
                down_smalldt.append(downhydro_maxorder[i] + t_down*j)
        downhydro_maxorder = down_smalldt

    numpy.savetxt(outpath + 'downhydro' + str(order) + '_' + str(1) + '.txt', downhydro_maxorder, delimiter=',')

    # Determine number of timesteps
    Nt = len(downhydro_maxorder)

    # Read the locations of the cells belonging to the main reach
    reachIDs = numpy.loadtxt(outpath + 'SubReachArrayID' + str(order) + '_' + str(1) + '.txt', delimiter=',')

    # Length of stream in km (= number of cells * spatial resolution [m])
    if dx > SpaceRes:
        if len(reachIDs) % 2 == 0:
            length = (len(reachIDs) * grid_res)
        else:
            # Artificially elongate reach
            length = ((len(reachIDs)+1) * grid_res)
    else:
        length = (len(reachIDs) * grid_res)

    Nx = int((length/dx)+1)
    init = numpy.array([base for i in range(0, Nx)])
6.8. DETERMINE DOWNSTREAM HYDROGRAPHS USING CD-EQUATION

```python
# Run cD-equation for reach of highest stream order and store this as Q_record_back_maxorder
Q_record_back_maxorder = cD(uphydro_maxorder, downhydro_maxorder, init, length, dx, dt, c, D)
numpy.savetxt(outpath + 'Q_record_back_record_back_maxorder' + '.txt', Q_record_back_maxorder, delimiter=',')

# Consider all sub-reaches that flow into this orders reach
for sub in range(1,Nsubs_this+1):
    # Keeping track which reach is considered during the loop
    print 'order: ' + str(order) + ', sub: ' + str(sub)
    # Find in 'coupled_reaches' which sub-reaches flow into this orders reach
    subs_upstream = numpy.where(numpy.logical_and(coupled_reaches[:,2]==order, coupled_reaches[:,3]==sub))[0]
    # Stop in case the reach has no sub-reaches
    if len(subs_upstream)==0:
        print 'reach has no reaches draining into it.'
    else:
        # Special case when regarding the main stream (highest stream order)
        if order==strordmax:
            Q_record_back_order = Q_record_back_maxorder
        # In all other cases:
        else:
            Q_record_back_order = numpy.loadtxt(outpath + 'Q_record_back_record_back_order' + str(order) + str(sub) + '.txt', delimiter=',')

        # Read the locations of the cells belonging to the reach that is considered
        reachIDs = numpy.loadtxt(outpath + 'SubReachArrayID_record_sub' + str(order) + str(sub) + '.txt', delimiter=',')

        # For-loop over all sub-reaches of this order
        for i in subs_upstream:
            # Find the order- and sub number of the sub-reach upstream of the
            # considered reach
            subsub_order = coupled_reaches[i,0]
            subsub_sub = coupled_reaches[i,1]
            print 'sub-reach order: ' + str(subsub_order), ', sub-reach sub: ' + str(subsub_sub)
            # Calculate the following as long as order is not equal to the
            # smallest stream order
            if order!=strordmin:
                # Determine location of inlets
                rowIDin = inIDreach[numpy.where(numpy.logical_and(inIDreach[:,0]==float(subsub_order), inIDreach[:,1]==float(subsub_sub)))[0][0]][2]
                colIDin = inIDreach[numpy.where(numpy.logical_and(inIDreach[:,0]==float(subsub_order), inIDreach[:,1]==float(subsub_sub)))[0][0]][3]
                # Find where in inID (= which row) the
                # regarded inlet is located
                indexIN = numpy.where(numpy.logical_and(inID[:,0]==rowIDin, inID[:,1]==colIDin))[0][0]
                # Select the timeseries from TimeArrayIn
                uphydro = TimeArrayIn[1:,indexIN+1]
                numpy.savetxt(outpath + 'uphydro_record_uphydro' + str(subsub_order) + '.txt', uphydro, delimiter=',')
```

49
else:
    # Determine location of inlets
    rowIDin = upsIDreach[numpy.where(numpy.logical_and(upsIDreach[:,0]==float(subsuborder), upsIDreach[:,1]==float(subsubsub)))[0][0]][2]
    colIDin = upsIDreach[numpy.where(numpy.logical_and(upsIDreach[:,0]==float(subsuborder), upsIDreach[:,1]==float(subsubsub)))[0][0]][3]
    # Find where in uphydroID (= which row) the regarded inlet is located
    indexIN = numpy.where(numpy.logical_and(uphydroID[:,0]==rowIDin, uphydroID[:,1]==colIDin))[0][0]
    # Select the timeseries from TimeArrayUphydro
    uphydro = TimeArrayUphydro[1:,indexIN+1]
    numpy.savetxt(outpath + 'uphydro' + str(subsuborder) + str(subsubsub) + '.txt', uphydro, delimiter=',')

    # Determine location of outlets/pits of the 'smaller'-order reaches --> needed to determine connectID
    rowIDout = outIDreach[numpy.where(numpy.logical_and(outIDreach[:,0]==float(subsuborder), outIDreach[:,1]==float(subsubsub)))[0][0]][2]
    colIDout = outIDreach[numpy.where(numpy.logical_and(outIDreach[:,0]==float(subsuborder), outIDreach[:,1]==float(subsubsub)))[0][0]][3]
    # Find where in pitID (= which row) the regarded pit is located
    indexOUT = numpy.where(numpy.logical_and(pitID[:,0]==rowIDout, pitID[:,1]==colIDout))[0][0]
    # Find the how many’d cell (from upstream) of the reach forms
    indexstreamcell = numpy.where(numpy.logical_and(reachIDs[:,1]==connectID[indexOUT][0], reachIDs[:,2]==connectID[indexOUT][1]))
    cellnumber = indexstreamcell[0][0] + 1
    # Determine cell number when original spatial resolution of the model differs from spatial resolution used in cD-equation
    cellnumber = cellnumber * (grid_res/dx)
    numpy.save(outpath + 'cellnumber' + str(subsuborder) + str(subsubsub) + order + str(order) + str(sub), cellnumber)
    # Get flow hydrograph of the connecting point and use that as downstream b.c. of the reach of the stream order more upstream
    connecthydro = []
    for i in range(len(Q_record_backorder)):
        connecthydro.append(Q_record_backorder[i][cellnumber])
    numpy.savetxt(outpath + 'downhydro' + str(subsuborder) + str(subsubsub) + '.txt', connecthydro, delimiter=',')

    # Determine length of the regarded subreach of the 'smaller'
    # Length of stream in km (= number of cells * spatial resolution [m])
    lengthsubsub = numpy.loadtxt(outpath + 'SubReachArrayID' + str(subsuborder) + str(subsubsub) + '.txt', delimiter=',')
    if numpy.size(lengthsubsub)==6:
        length = grid_res
if dx>SpaceRes:
    # Artificially elongating a 1-cell reach
    length = grid_res*(dx/SpaceRes)
else:
    if dx>SpaceRes:
        if len(lengthsubsub)%2==0:
            length = (len(lengthsubsub)*grid_res)
        else:
            length = ((len(lengthsubsub)+1)*grid_res)
    else:
        length = (len(lengthsubsub)*grid_res)
Nx = int((length/dx)+1)
init = numpy.array([base for i in range(0,Nx)])

# Run cD-equation for the subreach and save the results in Q_record_back
Q_record_back_sub = cD(uphydro, connecthydro, init, length, dx, dt, c, D)
numpy.savetxt(outpath + 'Q_record_back' + str(subsub_order) + str(subsub_sub)+'.txt', Q_record_back_sub, delimiter=',')

else: print 'end'

Table 6.6: 'Q_record_back' and 'Q_record' store the results of the cD-equation for each cell of a reach for every timestep (r.668 & 812). 'back' refers to running the cD-equation in upstream direction in order to provide the best possible estimation of the downstream boundary hydrographs for the individual reaches (see Sect. 6.8.5). The number of columns equals the number of cells of the considered reach and the number of rows equals the number of time steps.

(a) Q_record_back

<table>
<thead>
<tr>
<th>Timestep</th>
<th>cell 1</th>
<th>cell 2</th>
<th>...</th>
<th>cell N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.025</td>
<td>0.004</td>
<td>...</td>
<td>0.652</td>
</tr>
<tr>
<td>2</td>
<td>0.034</td>
<td>0.052</td>
<td>...</td>
<td>0.158</td>
</tr>
<tr>
<td>3</td>
<td>0.138</td>
<td>0.253</td>
<td>...</td>
<td>0.089</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T-1</td>
<td>0.338</td>
<td>2.442</td>
<td>...</td>
<td>0.677</td>
</tr>
<tr>
<td>T</td>
<td>0.502</td>
<td>2.021</td>
<td>...</td>
<td>0.824</td>
</tr>
</tbody>
</table>

(b) Q_record

<table>
<thead>
<tr>
<th>Timestep</th>
<th>cell 1</th>
<th>cell 2</th>
<th>...</th>
<th>cell N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.520</td>
<td>0</td>
<td>...</td>
<td>0.520</td>
</tr>
<tr>
<td>2</td>
<td>0.682</td>
<td>0.038</td>
<td>...</td>
<td>0.601</td>
</tr>
<tr>
<td>3</td>
<td>0.692</td>
<td>0.021</td>
<td>...</td>
<td>0.639</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T-1</td>
<td>0.891</td>
<td>2.852</td>
<td>...</td>
<td>0.474</td>
</tr>
<tr>
<td>T</td>
<td>0.744</td>
<td>2.954</td>
<td>...</td>
<td>0.487</td>
</tr>
</tbody>
</table>
(a) Run cD-equation for the reaches of second highest stream order. Continue this process in upstream direction.

\[
\text{downhydro}_1.7 = Q_{\text{record} \_ \text{back} \_ 2.3} \quad \text{[cell number]}
\]

\[
\text{downhydro}_1.11 = Q_{\text{record} \_ \text{back} \_ 2.3} \quad \text{[cell number]}
\]

(b) Retrieve 'downhydro' from 'Q_record' for connecting upstream reaches.

**Figure 6.12**: Retrieve 'uphydro' from the time series of accumulated runoff (stored in TimeArray_inlets). 'Downhydro' is retrieved from time series stored in 'Q_record_back' of the reach this stream flows into. These serve as input for the cD-equation of which the results are stored in 'Q_record_back' (Fig.(a)). Determine at which cell number the reaches are connected, in order to retrieve the time series for 'downhydro' from 'Q_record_back' (Fig.(b)).
Table 6.7: ‘Uphydro’ and ‘Downhydro’ store the hydrographs at the inlet and outlet of a reach respectively. The number of rows equals the number of time steps.

<table>
<thead>
<tr>
<th>Timestep</th>
<th>Uphydro</th>
<th>Timestep</th>
<th>Downhydro</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.125</td>
<td>1</td>
<td>1.839</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>2</td>
<td>1.758</td>
</tr>
<tr>
<td>3</td>
<td>0.018</td>
<td>3</td>
<td>1.615</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>T-1</td>
<td>1.122</td>
<td>T-1</td>
<td>0.625</td>
</tr>
<tr>
<td>T</td>
<td>1.284</td>
<td>T</td>
<td>0.784</td>
</tr>
</tbody>
</table>

6.9 Apply cD-equation for routing

Now the final iterative process is started in which the cD-equation is applied to all reaches (r.811), starting at the smallest branches and working in downstream direction. Using this flow routing sequence we assume no backwater effects, which is a valid assumption in an area with substantial elevation differences. The outflow of an upstream reach is injected as lateral flow all at one point where the reach is connected to its downstream branch (r.728–798). Additionally, the accumulated runoff produced in the sub-catchment of the main outlet only (i.e. the runoff produced in the sub-catchments of all other reaches are subtracted) is added as distributed lateral inflow to the main stream (r.800–808). If visualisation of the results are preferred all data is stored in one table Qall which is a preparatory step for creating maps of the results (see Sect. 6.10) (r.821–874).

# Finally calculate cD-equation for every reach of the stream network, starting at the smallest reaches and working towards the main reach
print 'Starting cD-equation from up- to downstream for the entire stream network.'
print 'This is the main routing loop.'
# Create empty array to which Q_record is concatenated
Qall=numpy.empty([len(downhydro),maxorder],0)
# Start main loop
for order in range(strordmin,strordmax+1):
    print 'order: ' + str(order)
    # Number of sub-reaches of the order (by counting the number of rows that have the same order)
    Nsubs_this=inIDreach[:,0].tolist().count(float(order))
    # Consider all sub-reaches that flow into this orders reach)
    for sub in range(1,Nsubs_this+1):
        # Keeping track which reach is considered during the loop
        print 'sub: ' + str(sub)
        # Find in 'coupled_reaches' to which this reach drains
        sub_downstream = numpy.where((numpy.logical_and(coupled_reaches[:,0]==order,
coupled_reaches[:,1]==sub)))[0]
        if len(sub_downstream)>0:
            order_drain = coupled_reaches[sub_downstream[0],2]
            sub_drain = coupled_reaches[sub_downstream[0],3]
        # Read the locations of the cells belonging to the reach that is considered
        reachIDs = numpy.loadtxt(outpath + 'SubReachArrayID.' + sub + '_.txt', delimiter=',')
# Determine length of the regarded subreach of the 'smaller' order
# Length of stream in km (= number of cells * spatial resolution [m])
if numpy.size(reachIDs)==6:
    length = grid_res
    if dx>SpaceRes:
        # Artificially elongating a 1-cell reach
        length = grid_res*(dx/SpaceRes)
    else:
        if dx>SpaceRes:
            if len(reachIDs)%2==0:
                length = (len(reachIDs)*grid_res)
            else:
                length = ((len(reachIDs)+1)*grid_res)
        else:
            length = (len(reachIDs)*grid_res)
else:
    # For streams of the lowest streamorder: Read upstream hydrograph.
    uphydro = numpy.loadtxt(outpath + 'uphydro_' + str(order) + '_' + str(sub)+'.txt', delimiter=',')
    # Read downstream hydrograph
    downhydro = numpy.loadtxt(outpath + 'downhydro_' + str(order) + '_' + str(sub)+'.txt', delimiter=',')
    # Artificially elongate the streams (*1.2) and there place
    # the downstream b.c.
    L_long=length*1.2
    Nx_long = int((L_long/dx)+1)
    init = numpy.array([base for i in range(0,Nx_long)])
else:
    # This way the b.c. will have less effect on the actual end of the reach.
    L_long=length*1.2
    Nx_long = int((L_long/dx)+1)
    init = numpy.array([base for i in range(0,Nx_long)])

# Check if a reach drains into the considered reach. This is done by checking
# if the reach appears in column 2 and 3 of coupled_reaches
check_lateral = numpy.where(numpy.logical_and(coupled_reaches[:,2]==order, coupled_reaches[:,3]==sub))[0]
if len(check_lateral)==0:
    print 'reach ' + str(order) + '_' + str(sub) + ' has no sub-reaches'
laterals=[]
else:
    lateral=[]
    for i in check_lateral:
        # Append the cellnumber where the lateral flow should be added
        connect = numpy.load(outpath + 'cellnumber_' + str(coupled_reaches[i,0]) + str(coupled_reaches[i,1]) + str(order) + str(sub) + '.npy')
        lateral.append(int(connect))
        # Append the lateral flow originating from the routing through
        # the channel -> Water_routed
        lat = numpy.loadtxt(outpath + 'lateral_' + str(coupled_reaches[i,0]) + str(coupled_reaches[i,1]) + str(order) + str(sub)+'.txt', delimiter=',')
        for j in range(len(lat)):
            lateral.append(lat[j])
        # Append the lateral flow originating from the runoff produced in
        # the cells (TotR as stored in TimeArrayPit) -> Water_system.
        # Determine location of outlets/pits of the reach that drains
        # into the regarded reach.
6.9. APPLY CD-EQUATION FOR ROUTING

```python
rowIDout = outIDreach[numpy.where((numpy.logical_and(outIDreach[:,0]==float(order), outIDreach[:,1]==float(1)))==0)[0][0]][2]
collOut = outIDreach[numpy.where((numpy.logical_and(outIDreach[:,0]==float(order), outIDreach[:,1]==float(1)))==0)[0][0]][3]
# Find where in pitID (= which row) the regarded pit is located
indexOUT = numpy.where((numpy.logical_and(pitID[:,0]==rowIDout, pitID[:,1]==collOut))==0)[0][0]
latTotR = TimeArrayPit[1:,indexOUT+1]
# If dt<24: divide every timestep by 24/dt
if dt < 24:
    latTotR_smalldt = []
    for i in range(len(latTotR)-1):
        t_lat = (latTotR[i+1]-latTotR[i])/(24/dt)
        for j in range(1, (24/dt)+1):
            latTotR_smalldt.append(latTotR[i]+t_lat*j)
    latTotR = latTotR_smalldt
for k in range(len(latTotR)):
    lateral.append(latTotR[k])
# Every row in laterals covers the lateral hydrograph of one sub-reach
# that drains into the considered reach. The first column of laterals
# is the cellnumber where the lateral flow enters the considered reach.
laterals=numpy.reshape(lateral, (len(check_lateral),len(lat)+1))
# Find whether there are multiple reaches flowing into the same cell of
# the next reach. In that case these flows should be summed.
duplicates = [item for item, count in Counter(laterals[:,0]).iteritems() if count > 1]
# Find which rows in laterals have the same connect cell
for items in duplicates:
    index = numpy.argwhere(laterals[:,0]==items)
    # Add the time series of these lateral flows to 'temp'
    temp=[]
    for same in range(len(index)):
        temp.append(laterals[index[same][0],:])
    # Create empty array
    summed=numpy.zeros([len(temp[0]),0]).tolist()
    # Fill the list with zeros
    for j in range(1,len(temp[0])):
        summed[j]=0.
    # Add as first row the connectID cellnumber
    summed[0]=temp[0][0]
    # Add the summed time series to 'summed'
    for i in range(len(temp[0])):
        summed[j]+=temp[i][j]
    summed=numpy.reshape(summed, (1,len(summed)))
    # Remove the individual time series of those that were just summed
laterals = numpy.delete(laterals, index.tolist(), axis=0)
# Add the summed time series
laterals = numpy.concatenate([laterals,summed])
# Update 'check_lateral'
check_lateral = laterals[:,0]
```

# Prepare the distributed lateral inflow of the main stream.
# This comprises the accumulated runoff produced in the sub-catchment
# of the main outlet only, i.e. the sub-catchments of all other reaches

# are substracted from this. If not added laterally this volume of water
# would not be accounted for.
if order==strordmax:
    lat_distr_maxorder = downhydro/len(reachIDs)
else:
    lat_distr_maxorder = downhydro*0.

#### Apply cD-equation to the reach ####
Q_record = cD(uphydro, downhydro, init, laterals, check_lateral,
lat_distr_maxorder, length, dx, dt, c, D)
numpy.savetxt(outpath + 'Q_record' + str(order) + '_' + str(sub) + '.txt', Q_record, delimiter=',')

# The hydrograph of the last cell becomes the lateral flow for the reach
# it drains into
if len(sub_downstream)>0:
    lateral=[]
    for i in range(len(Q_record)):
        lateral.append(Q_record[i][-1])
    numpy.savetxt(outpath + 'lateral' + str(order) + str(sub) + str(order_drain) + str(sub_drain)+'.txt', lateral, delimiter=',')

# Visualisation of the results?
if visFLAG==1:
    # 3 options to add values to 'Qall' depending on relation dx:SpaceRes
    # In the end the length of 'Qall' should be the same for the 3 options
    # in order to visualize the results on a map with its
    # original spatial resolution
    if dx == SpaceRes:
        Qall=numpy.concatenate([Qall,Q_record[:,1:]],axis=1)
    else:
        Qtemp=[]
        if dx < SpaceRes:
            for times in range(len(Q_record)):
                Q_record_temp = Q_record[times,1:]
                # Take mean of every (SpaceRes/dx) cells
                means = numpy.mean(Q_record_temp.reshape(-1,int(SpaceRes/dx)), axis=1)
                for cell in range(len(means)):
                    Qtemp.append(means[cell])
            # In case the reach comprises only 1 cell
            if len(Q_record[0,:])==6 and len(reachIDs)==6:
                Qtemp=numpy.reshape(Qtemp, (len(Q_record),1))
            # In other cases
            else:
                Qtemp=numpy.reshape(Qtemp, (len(Q_record),len(reachIDs)))
        else:
            # In case the reach comprises only 1 cell
            if len(Q_record[0,:])==2 and len(reachIDs)==6:
                Qtemp=numpy.reshape(Qtemp,Q_record[:,1], (len(Q_record),1))
            else:
                # In other cases duplicate each cell (dx/SpaceRes) times
                Qall=numpy.concatenate([Qall,Qtemp],axis=1)
else:
    Qtimp=[]
    if dx > SpaceRes:
        # In case the reach comprises only 1 cell
        if len(Q_record[0,:])==6 and len(reachIDs)==6:
            Qtemp=numpy.reshape(Qtemp,Q_record[:,1], (len(Q_record),1))
        else:
            # In other cases duplicate each cell (dx/SpaceRes) times
            Qall=numpy.concatenate([Qall,Qtemp],axis=1)
    else:
for times in range(len(Q_record)):
    Q_record_temp = Q_record[times,1:]
    # Repeat the value of a cell dx/SpaceRes times
    rep = numpy.repeat(Q_record_temp, int(dx/SpaceRes))
    # Remove at the beginning of the reach the number
    # of cells that there are too many
    if len(rep)>len(reachIDs):
        rep = rep[(len(rep)-len(reachIDs)):]  
        # Repeat at the beginning of the reach the first value
        # len(SubreachID)-len(rep)-times (pragmatic solution)
        if len(rep)<len(reachIDs):
            for i in range(len(reachIDs)-len(rep)-times (pragmatic solution):
                rep=numpy.insert(rep, 0, rep[0])
    for cell in range(len(rep)):
        Qtemp.append(rep[cell])
        Qtemp=numpy.reshape(Qtemp, (len(Q_record),len(reachIDs)))
    # Add all values contained in rep to Qall
    for cell in range(len(rep)):
        Qtemp.append(rep[cell])
        Qtemp=numpy.reshape(Qtemp, (len(Q_record),len(reachIDs)))
    # Add all values contained in rep to Qall
    Qall=numpy.concatenate([Qall,Qtemp],axis=1)

if visFLAG==1:
    numpy.savetxt(outpath + 'Qall.txt', Qall, delimiter=',')
(a) Run cD-equation for the reaches most upstream (smallest stream order). Continue this process in downstream direction.

\[
\text{lateral}_2.3 = Q_{\text{record}}_1.7 \quad \text{[cell number]}
\]
\[
\text{lateral}_2.3 = Q_{\text{record}}_1.11 \quad \text{[cell number]}
\]

(b) Retrieve 'lateral' from 'Q_record' for connecting downstream reaches.

Figure 6.13: 'Uphydro' and 'downhydro' were stored during the code run in section 6.8.5. These serve as input for the cD-equation of which the results are stored in 'Q_record' (Fig.(a)). The results of the outlet cell, as stored in 'Q_record', serves as lateral inflow of the connecting reach downstream (Fig.(b)).
6.10 Visualisation of the results

The results of the cD-equation are visualised using again the original PCRaster environment. For every time-step a map of the stream network will be created providing the output of the cD-equation for each cell. For this purpose the matrix 'networkID' is created in which the locations of all the cells of the whole stream network are stored.

```python
407 # Put the locations of all cells of the network in table 'networkID'.
408 # Starting with the smallest stream order and per reach in downstream direction.
409 # This way the order will comply with the order as the values are stored in 'Q_record'
410 # after applying the cD-equation.
411 networkID = []
412 for order in range(strordmin,strordmax+1):
413     # Number of sub-reaches of the order (by counting the number of rows that have
414     # the same order)
415     Nsubs_this=inIDreach[:,0].tolist().count(float(order))
416     # Consider all sub-reaches that flow into this orders reach
417     for sub in range(1,Nsubs_this+1):
418         # Read the table in which row and column for each cell are stored
419         # (column '1' and '2')
420         SubreachID = numpy.loadtxt(outpath + 'SubReachArrayID' + str(order) +
421         '.txt', delimiter=',')
422         # Add row and column (= location of cell) to 'networkID'
423         # Special case when reach comprises only 1 cell
424         if numpy.size(SubreachID)==6:
425             networkID.append(SubreachID[1])
426             networkID.append(SubreachID[2])
427         # All other cases
428         else:
429             for ID in range(len(SubreachID)):
430                 networkID.append(SubreachID[ID,1])
431                 networkID.append(SubreachID[ID,2])
432     # Reshape the whole list into an matrix of 2 columns. Column '0' = row,
433     # column '1' = column
434     networkID=numpy.reshape(networkID, (len(networkID)/2,2))
435     numpy.savetxt(outpath + 'networkID.txt', networkID, delimiter=',')
```

The last step in the visualisation consists of combining 'networkID' and 'Qall' and create for each time step a map.

```python
879 if visFLAG==1:
880     # Use 'Qall' to visualise the results by creating maps with original
881     # spatial resolution. Create a map that only contains the stream network
882     # and convert it to a numpy array (float).
883     network_solely = pcr.ifthen(network,clone)
884     pcr.report(network_solely, outpath + 'network_solely.map')
885     output_array = pcr.pcr2numpy(network_solely, MV).astype(numpy.float)
886     # Set initials for dynamically naming the files
887     Q=0
888     count=0
889     pcrcounter=0.
890     endcount=dt.visualize/dt
891     # For each timestep create a map
892     for times in range(len(Qall)):
893         count+=1
```
6.11 Time series at observation stations

For calibration purposes time series of the simulated results at the observation stations are provided at a daily base (r.932–947) in the original spatial resolution (r.949–997). Discharge observations and simulated discharge using the cD-routing procedure are plotted, as well as discharge simulated with the current simple routing scheme are plotted for comparison (r.1012–1050). Nash-Sutcliffe values are displayed as an indication of model performance (r.1052–1064).

```python
# Time series of specified locations, e.g. streamflow locations
for loc in range(len(stations_loc)-1):  # -1 to discard the outlet of the total area
    # Determine in which reach the station is located
    order = int(stations_loc[loc,2])
    sub = int(stations_loc[loc,3])
    index = int(stations_loc[loc,4])
    index = int(index * (grid_res/dx))

    # Determine nameID of the station
    name = int(names[int(stations_loc[loc,0]),int(stations_loc[loc,1])])
    if name == 999:
        name = 'outlet'
    name = str(name)

    # Read the locations of the cells belonging to the reach that is considered
    reachIDs = numpy.loadtxt(outpath + 'SubReachArrayID_' + str(order) + str(sub) + '.txt', delimiter=',')

    ### Handling Q_station files when dt != 24 ###
    # Read simulated Q of the reach
    Q_station = numpy.loadtxt(outpath + 'Q_record_' + str(order) + str(sub) + '.txt', delimiter=',')
    if dt<24:
        Qstation_dt = []
        Q=0
        count=0
        endcount=24/dt
        # For each timestep create a map
        for times in range(len(Q_station)):
            count+=1
            Q+=Q_station[times]
```

CHAPTER 6. ADVANCED ROUTING MODULE IMPLEMENTATION
6.11. TIME SERIES AT OBSERVATION STATIONS

```python
if count == endcount:
    count=0
    Qstation_graph=Q/(24/dt)
    Q=0
    Qstation_dt.append(Qstation_graph)
    Qstation_dt=numpy.reshape(Qstation_dt, (len(Q.station)/(24/dt),len(Q.station[0])))
    Q_station = Qstation_dt

### Handling Q_station files when dx != SpaceRes ###
Qall=numpy.empty((len(downhydro_maxorder),0))
if dx != SpaceRes:
    Qtemp=[]
    if dx < SpaceRes:
        for times in range(len(Q.station)):
            Q_record_temp = Q.station[times,1:]
            means = numpy.mean(Q_record_temp.reshape(-1,int(SpaceRes/dx)), axis=1)
            for cell in range(len(means)):
                Qtemp.append(means[cell])
    else:
        Qtemp=numpy.reshape(Qtemp, (len(Q.station),len(reachIDs)))
        Qall=numpy.concatenate([Qall,Qtemp],axis=1)
        Q_station=Qall

if dx > SpaceRes:
    if len(Q.station[0,:])==2 and len(reachIDs)==6:
        Qtemp=numpy.reshape(Qstation[:,1], (len(Q.station),1))
        Qall=numpy.concatenate([Qall,Qtemp],axis=1)
        Q_station=Qall
    else:
        for times in range(len(Q.station)):
            Q_record_temp = Q.station[times,1:]
            rep = numpy.repeat(Q_record_temp, int(dx/SpaceRes))
            if len(rep)>len(reachIDs):
                rep = rep[(len(rep)-len(reachIDs)):
            if len(rep)<len(reachIDs):
                for i in range(len(reachIDs)-len(rep)):
                    rep=numpy.insert(rep, 0, rep[0])
            for cell in range(len(rep)):
                Qtemp.append(rep[cell])
            Qtemp=numpy.reshape(Qtemp, (len(Q.station),len(reachIDs)))
        Qall=numpy.concatenate([Qall,Qtemp],axis=1)
        Qall=numpy.concatenate([Qall,Qtemp],axis=1)
```

61
Q.station=Qall

# Read observed Q and Q simulated using the old routing procedure
Q.obs_sim = PlottingObsSim.ObsSim(name, inpath, sy, sm, sd)

# Create array containing dates.
date = numpy.arange(datetime.date(sy, sm, sd), datetime.date(ey, em, ed),
datetime.timedelta(days=1)).astype(datetime.date)

# Date formatter
years = mdates.YearLocator()
months = mdates.MonthLocator()
if len(date)>365:
 Fmt = mdates.DateFormatter('%Y')
if len(date)<366:
 Fmt = mdates.DateFormatter('%m-%Y')

# Create plots
fig, ax = pyplot.subplots()
ax.plot(date, Q.obs_sim[0][Q.obs_sim[1]:Q.obs_sim[1]+len(date)],
color='grey', label='Observed')
ax.plot(date, Q.obs_sim[2][Q.obs_sim[3]:Q.obs_sim[1]+len(date)],
color='dodgerblue', label='Simple routing')
if len(Q.station)>len(date):
  diff = abs(len(Q.station)-len(date))
  ax.plot(date, Q.station[diff:,index],color='darkorange', label='cD-routing')
else:
  ax.plot(date, Q.station[:,index],color='darkorange', label='cD-routing')

# Add label and legend
pyplot.ylabel('Discharge [m$^3$ s$^{-1}$]')
pyplot.legend(fontsize=9, frameon=False)

# format the ticks
if len(date)>365:
  ax.xaxis.set_major_locator(years)
  ax.xaxis.set_major_formatter(Fmt)
  ax.xaxis.set_minor_locator(months)
if len(date)>365:
  ax.xaxis.set_major_locator(months)
  ax.xaxis.set_major_formatter(Fmt)
datemin = datetime.date(date.min().year, 1, 1)
datemax = datetime.date(date.max().year + 1, 1, 1)
ax.set_xlim(datemin, datemax)

# rotates and right aligns the x labels, and moves the bottom of the
# axes up to make room for them
for ax in fig.get_axes():
  if ax.is_last_row():
    for label in ax.get_xticklabels():
      label.set_ha('right')
      label.set_rotation(30.)
  else:
    for label in ax.get_xticklabels():
      label.set_visible(False)
Apart from SPHY parameters that need to be calibrated (i.e., they are influencing simulated specific runoff as used by the routing module), there are two parameters in the routing module that need to be calibrated: wave celerity (c) and diffusion coefficient (D). Both can also be calculated from the linearised cD-equation (see Eq. 6.2 and 6.3). However, this may not always yield good results as was explained in section 6.6, e.g., erratic behaviour of discharge or too steep recession curves, resulting in low Nash-Sutcliffe values (Nash and Sutcliffe, 1970). Therefore for this application example both values have been calibrated manually. To evaluate the performance of the routing procedure in the SPHY model, the Nash-Sutcliffe efficiency (NS) of the discharge is computed as a measure of goodness of fit.

Values of these calibration parameters depend on the period of time that is simulated, as well as the selected spatial and temporal resolution. Changes made in one of these three variables require recalibration of the parameters.

Currently the parameters c and D are spatially constant for the entire basin, while spatially dependent values for these parameters may yield different and better results. However, this requires calibration of these parameters for each reach in the stream network or at locations in which the characteristics of the stream change substantial.

To test several combinations of parameters before running the whole routing module, the user can opt to run the script ‘cD_test.py’ (see Appendix C). For this the user needs to specify ranges for the parameters c, D, dt and dx and provide an estimation of the expected maximum and minimum discharge. The code will provide a log-file with the parameter values, Courant number, Peclet number, maximum and minimum simulated discharge, total discharge upstream and downstream and it will raise a warning if large oscillations occurred. The latter is an indication that the parameter combination should be changed.
6.13 Adjustments made in SPHY script

This whole routing procedure is applied at the end of the SPHY model when the total specific runoff for each cell has been determined. The following code has been added to the SPHY programming code:

```python
if self.curdate == self.enddate:
  if self.CDRoutFLAG == 1:
    self.dx = config.getfloat('CDROUTING', 'dx')
except:
    self.dx = self.SpaceRes

# Check the ratio between dx and SpaceRes
if self.dx > self.SpaceRes:
  test = self.dx/self.SpaceRes
  if test.is_integer():
    print ''
else:
  sys.exit('ERROR: dx/SpaceRes is not an integer: Change dx')

if self.dx < self.SpaceRes:
  test = self.SpaceRes/self.dx
  if test.is_integer():
    print ''
else:
  sys.exit('ERROR: SpaceRes/dx is not an integer: Change dx')

self.dt = config.getint('CDROUTING', 'dt')
dt_options = [1,3,6,12,24]
if self.dt not in dt_options:
  sys.exit('ERROR: chosen dt does not correspond to the options provided: change dt to 1, 3, 6, 12 or 24.')

self.dt_visualize = config.getint('CDROUTING', 'dt_visualize')
dt_options = [1,3,6,12,24]
if self.dt_visualize not in dt_options:
  sys.exit('ERROR: chosen dt_visualize does not correspond to the options provided: change dt to 1, 3, 6, 12 or 24.')

if self.dt_visualize < self.dt:
  sys.exit('ERROR: dt_visualize is smaller than dt: choose a value for dt_visualize equal to or larger than dt')

try:
  self.c = config.getfloat('CDROUTING', 'c')
  self.D = config.getfloat('CDROUTING', 'D')
except:
  self.width = config.getfloat('CDROUTING','width')
  self.depth = config.getfloat('CDROUTING','depth')
```
self.roughness = config.getfloat('CDROUTING','roughness')
self.slope = config.getfloat('CDROUTING','slope')
self.c = self.cD.calcCD(self,self.width,self.depth,
    self.roughness,self.slope)[0]
self.D = self.cD.calcCD(self,self.width,self.depth,
    self.roughness,self.slope)[1]
print 'c and D are calculated'
if self.c*(self.dt*60*60)/(self.dx*1000) > 1:
    print 'WARNING: Courant number is larger than 1, this could lead to
numerical instability: Decrease time step, or increase spatial step'
self.CDrouting.CDrouting(self,self.dx,self.dt,self.SpaceRes,
    self.dt_visualize,self.c,self.D)
7. Results and Discussion

7.1 Model output and analysis

Figures 7.1a and 7.1b show daily observed and SPHY simulated discharge using the current simple routing scheme and using the newly implemented cD-routing procedure. The flow hydrographs are shown for 10 consecutive years at the streamflow stations Busti (ID 647) and Rasnalu (ID 650), that both are located in the Tamakoshi river basin. Values of $c (= 0.08 \text{ km hr}^{-1})$ and $D (= 1.0 \text{ km}^2 \text{ hr}^{-1})$ were manually calibrated. However, these values make no sense physically as they are relatively low and as they are not comparable to values found in literature (Vakgroep Hydraulica en Afvoerhydraulica, Year unknown) and when calculated using equations 6.2 and 6.3. Despite this, the model performs well given a NS value of 0.80 and 0.72 for stations Busti and Rasnalu respectively for both routing procedures. In general, the average simulated discharge corresponds to the observed discharge. However, peak discharges are frequently underestimated, as well as the low flows during the months December – May.

During this application example the settings of the SPHY model, as have been provided from a previous study, have not been changed. Recalibration of the SPHY model could change the simulated specific runoff for each cell, which is used as input for the cD-routing module, and could further improve the results of the cD-routing module.

Comparing the new cD-routing procedure to the current implemented simple routing scheme in SPHY, it can be seen in figures 7.1a and 7.1b that the results of both simulations are comparable. Both yield a similar NS value when considering the 10 years data series. The strong seasonality has a strong influence on the resulting reasonably high NS value of 0.80. This, however, does not necessarily mean that the model performs well. In general, the simple routing scheme simulates higher peaks than using the cD-routing. Low flows during the months December – May are better simulated by the simple routing. Zooming into the year 2004 it is demonstrated that the observed streamflow is more erratic than both simulated runs (see Fig. 7.1a and 7.1b).

Figure 7.2 demonstrates the results of daily observed and SPHY simulated discharge for the same locations after calculating the values of $c (= 18.5 \text{ km hr}^{-1})$ and $D (= 0.18 \text{ km}^2 \text{ hr}^{-1})$ using equations 6.2 and 6.3. It can be seen that simulated discharge behaves erratic and the average discharge is too low, resulting in a bad model performance with NS values of -0.13 and -0.31 for stations Busti and Rasnalu respectively. The erratic behaviour is most probably due to the high wave celerity causing a wave to pass through the channels fast. This will result in a steep rising limb and recession limb of the hydrograph.

An example of the visualisation of the simulation results is shown in figure 7.3. It displays the streamflow for each cell of the stream network at a certain timestep. By displaying a number maps of consecutive time-steps after each other a movie can be created in which the propagation of a wave can be visualised as it travels through the stream network.
Figure 7.1: Daily observed discharge, SPHY simulated discharge using the simple routing procedure and SPHY simulated discharge using the cD-routing procedure for the streamflow stations Busti (ID 647) and Rasnalu (ID 650) for 10 years (a & b) and for the year 2004 (c & d). Values of c and D were calibrated based on the 10 year data series.
Figure 7.2: Daily observed and SPHY simulated discharge using the cD-routing procedure for the streamflow stations Busti (ID 647) and Rasnalu (ID 650). Values of c and D were calculated using equations 6.2 and 6.3.
Table 7.1: Overview of the results of the model performance.

(a) Overview of the results of the model performance with changing temporal resolution. The parameters c, D and dt are kept constant at 0.08 [km hr$^{-1}$], 0.1 [km$^2$ hr$^{-1}$] and 0.250 km respectively.

<table>
<thead>
<tr>
<th></th>
<th>Station Busti</th>
<th></th>
<th>Station Rasnalu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple</td>
<td>cD-routing</td>
<td>Simple</td>
<td>cD-routing</td>
</tr>
<tr>
<td>dt [hr]</td>
<td>24</td>
<td>12</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>NS [-]</td>
<td>0.80</td>
<td>0.74</td>
<td>0.80</td>
<td>0.74</td>
</tr>
<tr>
<td>Computational time [min]</td>
<td>-</td>
<td>7</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

(b) Overview of the results of the model performance with changing spatial resolution. The parameters c, D and dx are kept constant at 0.08 [km hr$^{-1}$], 0.1 [km$^2$ hr$^{-1}$] and 24 hr respectively.

<table>
<thead>
<tr>
<th></th>
<th>Station Busti</th>
<th></th>
<th>Station Rasnalu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple</td>
<td>cD-routing</td>
<td>Simple</td>
<td>cD-routing</td>
</tr>
<tr>
<td>dx [km]</td>
<td>0.25</td>
<td>0.25</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>NS [-]</td>
<td>0.80</td>
<td>0.80</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>Computational time [min]</td>
<td>-</td>
<td>4</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

CHAPTER 7. RESULTS AND DISCUSSION

Figure 7.3: Example of the visualisation of the simulation results using the cD-routing procedure. Red colours indicate high streamflow (in m$^3$ s$^{-1}$) and purple colours low streamflow.

7.2 Sensitivity analysis

A sensitivity analysis has been performed to assess the sensitivity of the routing module to changes in the parameters c and D, as well as changes in the temporal and spatial resolution, i.e. dt and dx respectively.

7.2.1 Wave celerity

Figures 7.4a and 7.4b display the results of the sensitivity analysis of parameter c for a period of 10 years for both stations respectively. In orange the default simulation with c = 0.08 km hr$^{-1}$ is depicted. The other lines demonstrate the results of decreasing and increasing the value of c with 25% and 50%. A higher value of c results in lower simulated discharges and a more erratic hydrograph. It was expected, supported by performed artificial examples, that discharge peaks would be higher with higher values of c, because the wave would arrive at the location sooner with less time for the development of diffusive effects. A small shift can be seen in the arrival of the peak with changing wave celerity. This effect might become more clearly visible when displaying over a shorter period of time of for instance a week. The peaks simulated with higher wave celerity are sharper as expected, because the wave takes less time to pass the station. Moreover, it becomes clear that mass is not preserved and this seems to be the largest issue here. No cause could be found yet to explain this strange and unexpected behaviour.

Zooming in to the year 2004 (Fig. 7.4c and 7.4d) it becomes visible for station Busti that during the recession limb the lines start to differ more than during the rising limb. This implies that at the arrival of peak discharges the routing module is less sensitive to changes in c than during the recession limb. This makes it more difficult to calibrate c such that it simulates the peaks well at this station. Station Rasnalu shows more sensitivity at the rising limb compared to the other station. This might be attributable to the size of the area upstream of the location, which is much smaller for Rasnalu than for Busti. Further analysis is required to assess this assumption.
7.2.2 Diffusion coefficient

Figures 7.5a and 7.5b display the results of the sensitivity analysis of parameter D for a period of 10 years for both stations respectively. In orange the default simulation with D = 0.1 km$^2$ hr$^{-1}$ is depicted. The other lines demonstrate the results of decreasing and increasing the value of D with 25% and 50%. Higher values of D result in smoother hydrographs and in higher simulated discharges. The latter is opposite to what is expected, namely higher values of D leading to more attenuated waves and therefore lower peaks. This might be a result of the numerical scale chosen too large with respect to the system time scale. Further analysis is needed to assess this. Again it is visible that mass is not preserved and this needs further exploration to be able to explain and solve this problem. From a pragmatic point of view the value of D can be seen as a parameter that can be used to calibrate the routing module.

Zooming in to the year 2004 (Fig. 7.5c and 7.5d) it can be seen that with higher D the total increase and decrease of discharge during peaks is higher. Peaks are therefore simulated higher, but the recession after is larger as well. Similar as to the parameter c, discharge at station Busti is less sensitive to changes in the parameter D than at station Rasnalu.

Considering the period of 10 years from 2000 – 2010 as a reference it can be seen that the cD-routing module is more sensitive to the wave celerity compared to the diffusion coefficient.

7.2.3 Temporal resolution

Sensitivity to changes in the temporal resolution is displayed in figure 7.6. Dt is given in hours and is indicating the temporal resolution on which the calculation of the routing is performed (i.e. visualisation is performed on a daily base). In orange the default simulation at dt = 24 hr is shown. An hourly temporal resolution is not included in the analysis over the period 2000 – 2010 since it required more storage memory (RAM memory) than available.

The routing module appears to be very sensitive to the temporal resolution at station Busti (Fig. 7.6c), where a finer temporal resolution results in significantly higher simulated discharge. It is not fully understood why calculations performed on an hourly base lead to rather different results than coarser temporal resolutions. Therefore it would require recalibration of the parameters c and D when changing the temporal resolution. Due to time restrictions it was not possible to assess the results of such a recalibration. In addition, the higher sensitivity to the temporal resolution at station Busti compared to station Rasnalu might partly be attributed to the size of the catchment upstream as this is the main difference between the two stations. This needs further exploration. And again there is the issue of the mass that is not conserved with changing temporal resolution.

7.2.4 Spatial resolution

In figure 7.7 the sensitivity of the routing module to changes in the spatial resolution are shown. Dx is given in metres and the temporal resolution is kept constant at 24 hours. Values of c and D are kept constant as well at 0.08 km hr$^{-1}$ and 0.1 km$^2$ hr$^{-1}$ respectively. In orange the default simulation at dx = 250 m is depicted. A spatial resolution of 50 m is not included in the analysis over the period 2000 – 2010 since it required more storage memory (RAM memory) than available.

A finer spatial resolution results in substantial lower simulated discharges at station Busti, while discharge simulated at station Rasnalu is insensitive such a change. And here as well mass is not preserved with changing spatial resolution. Simulating discharge at a larger spatial resolution for both stations does not significantly change the results. Again there is no explanation for this phenomenon yet and therefore needs further assessment. Again the size of the upstream catchment is a possible factor that could explain these results, or a mistake made in the programming code could be the explanation. Another difference between these stations is the stream order of the reach in which it is located. Station Busti is located in the main stream of the stream network, while station Rasnalu is located in a reach of a lower stream order. Only in the main stream the lateral inflow is added in a distributed way instead of point injection (see Sect. 6.5). Further analysis is needed to assess whether this interferes with the code for running the routing module on a smaller spatial resolution. In addition due to time restrictions it was not possible
TABLE 7.2: Computational time in minutes for a simulation period of 10 years on a daily base with the original spatial resolution (= 250m) for the two different routing procedures.

<table>
<thead>
<tr>
<th></th>
<th>Simple routing</th>
<th>cD-routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td>–</td>
<td>0.2</td>
</tr>
<tr>
<td>Calculating accumulated runoff</td>
<td>–</td>
<td>18</td>
</tr>
<tr>
<td>Routing without/with visualisation</td>
<td>–</td>
<td>4/7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>≈ 20</td>
<td>22/25</td>
</tr>
</tbody>
</table>

to assess the results of recalibration of the model with changing spatial resolutions.

7.3 Computational time

Preferably the computational time of the new cD-routing module does not exceed much the more simple current routing scheme. In table 7.1 the computational time for different scenarios is depicted. Note that these number indicate the time it takes to run only the routing part (i.e. applying the cD-equation) of the routing module (r.516 – r.1068) (without creating maps). This is the most relevant as this part will be used when calibrating the routing parameters. The computational time of all the preprocessing steps in which the stream network is defined and all the individual reaches are retrieved (r.27 – r.402) is 10 seconds. The time to compute and prepare the tables containing the accumulated runoff for all time-steps for several points (r.435 – r.503) depends on the period of time considered: for a period of 1 year this takes 1.5 minutes, while for a period of 10 years it takes 18 minutes. In table 7.2 an overview is provided of these computational times for a simulation period of 10 years on a daily base with a spatial resolution of 250m. Furthermore, the total computational time of the two routing procedures is provided.

In addition it should be noted that the routing module can be run independently of the SPHY model. Therefore the SPHY model should be run once to calculate and create the maps containing the total accumulated specific runoff for each cell. Subsequently the routing module can be run several times with different settings without having to run the whole SPHY model again.
Figure 7.4: Sensitivity analysis of wave celerity for 10 consecutive years (a) and (b) and for the year 2004 (c) and (d). Values of $c$ are given in km hr$^{-1}$. The diffusion coefficient is kept constant at 1.0 km$^2$ hr$^{-1}$.
Figure 7.5: Sensitivity analysis of the diffusion coefficient for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). Values of D are given in km$^2$ hr$^{-1}$. The wave celerity is kept constant at 0.08 km hr$^{-1}$.
7.3. COMPUTATIONAL TIME

Figure 7.6: Sensitivity analysis of the temporal resolution for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). Dt is given in hours and spatial resolution is kept constant at 0.25 km. Values of $c$ and $D$ are kept constant as well at 0.08 km hr$^{-1}$ and 0.1 km$^2$ hr$^{-1}$ respectively.
Figure 7.7: Sensitivity analysis of the spatial resolution for 10 consecutive years ((a) and (b)) and for the year 2004 ((c) and (d)). $D_x$ is given in meters and temporal resolution is kept constant at 24 hr. Values of $c$ and $D$ are kept constant as well at 0.08 km/hr and 0.1 km/hr, respectively.

Chapter 7. Results and Discussion

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
0 100 200 300 400 500
Discharge [m$^3$ s$^{-1}$]
$NS_{Dx} = 0.25 = 0.80$
$NS_{Dx} = 0.50 = 0.79$
$dx = 250$
$dx = 500$

(a) Station Busti (ID 647)

(b) Station Rasnalu (ID 650)
8. Conclusion

In this internship research project alternative routing procedure options were explored with the aim to improve the current routing scheme and therefore streamflow simulations in the SPHY model. Given the fact that SPHY is regularly applied in data-scarce research areas, it is concluded that the convection-diffusion (cD) approach is the most suitable routing method. The cD-wave equations provide a solid approximation of the full dynamic wave equations as it balances between the accuracy and high data demanding dynamic wave model and the simplicity of the kinematic wave model. The cD-equation was numerically discretized using the Crank-Nicolson scheme.

The cD-routing module is set up outside the PCRaster environment. This provides the opportunity to run the routing module with a different temporal and spatial resolution than the simulation of the SPHY model itself. Furthermore, visualisation of the results can be done on a different time-step than the routing calculations. These settings need to be specified by the user in the accompanying SPHY configuration file.

For the implementation of the cD-routing procedure it was necessary to first define the stream network and to identify all individual reaches of the different stream orders. In addition, it was determined which reaches are connecting to each other, since the output of one stream is added as lateral inflow into the next. Furthermore, time series containing the accumulated runoff for specified points (e.g. inlets and outlets of reaches) were stored in tables. After determining the downstream hydrographs for all reaches by applying the cD-routing procedure to all individual reaches starting at the downstream point and proceeding in upstream direction, the main loop starts. In final part the actual routing is performed, for which the cD-equation was applied to all reaches starting upstream and proceeding in downstream direction.

After manual calibration of the wave celerity (c) and the diffusion coefficient (D) the results obtained are comparable to the currently implemented simple routing scheme in terms of NS values. NS values are 0.80 and 0.72 for the stations Busti and Rasnalu respectively for the period 2000 – 2010. In general the average simulated discharge corresponds to the observed discharge. However, peak discharges are frequently underestimated, as well as the low flows during the months December – May.

The routing procedure is sensitive to changes in c and D, as well as to changes in the temporal and spatial resolution. Focussing on the latter two, it is not fully understood why a finer temporal and spatial resolution lead to substantial different results in simulated discharge for station Busti. Changes in c result in a slight shift in the arrival of a peak discharge as was expected. It was expected that higher values of D would lead to more attenuated waves and therefore lower peaks. However, the results of changes in D demonstrated the opposite. Further analysis is needed to explore this, especially focussing on the fact that mass is not preserved with changing parameter values.

Comparing computational time between the two routing schemes, it can be concluded that they are comparable. Depending on whether the user wants a visualisation of the results the cD-routing procedure takes 2 or 5 minutes longer than the currently implemented simple routing scheme.

It can be concluded that the newly implemented advanced cD-routing procedure performs well in terms of streamflow simulations and the results are comparable to the currently implemented simple routing scheme. Furthermore the cD-routing procedure has the advantage that it has the possibility to run the
routing on a different temporal and spatial resolution than the simulation of the SPHY model itself. It should be noted that currently the cD-routing procedure has only been applied to an area which is minimally regulated by humans and has substantial elevation differences. Further analyses should be done to test the routing procedure in different settings.
9. Recommendations

At this stage wave celerity (c) and diffusion coefficient (D) values were assumed to be constant over the whole research area. However, in reality these values will vary spatially depending on local channel geometry and characteristics. For instance lakes, reservoirs, channel contractions and structures like bridges, weirs and dams will influence these values. Making the c and D value spatial dependent could further improve the channel routing.

The sensitivity of the routing module to changes in c and D should be analysed further. This can be done by evaluating the effect of one parameter with the other parameter switched off. Furthermore, it would be interesting to take a closer look at a shorter time period to be able to analyse the response of the model to individual precipitation events and whether c and D can be calibrated to optimize the simulation for such individual events.

Calibration of the c and D values are done manually, but preferably this should be done in a consistent manner using a robust calibration method. Levenberg-Marquardt or HydroPSO are potential calibration methods to be used.

Adding the outflow from one reach to the next reach is done by injecting this as lateral inflow at one single point. However, another option that could still be implemented is to distribute this water along a transect of the next reach, rather than inject the whole volume of water at one point, because this might cause some disruptive effects on the routing in the next reach. These effects will become smaller when choosing a smaller threshold value.

Currently the length of the reaches are determined based on the number of cells of that reach multiplied with the length of one cell, i.e. the spatial resolution of the grid. This calculation is valid in the case when the water flows straight through the cell. However, water could also flow diagonally through a cell depending on the local drain direction, which would increase the flow path of the water. Taking this into account new calculations were performed based on the local drain direction (ldd) map, improving the estimation of the true length of the flow path that the water takes. However, the visualisation of the results were already programmed in such a way that it requires the length calculated based on the number of cells multiplied with the length of one cell. Due to time limitations it was not able to implement the improvement in calculations of the flow path length to the visualisation of the results and therefore the former, less accurate calculations were adopted again. Appendix B provides the programming code to calculate the length the water has to travel in every reach based on the ldd map.

Some pragmatic choices were made during the programming. For instance a reach is artificially elongated when it happens to consist of only one cell. This is needed for the calculations of the cD-equation among others. However, the effects of adding one cell is small in comparison with for instance the way the length of a reach is determined without considering the local drain direction.

Currently the downstream boundary condition is chosen to be the flow hydrograph of the accumulated runoff of the entire catchment. However, with this we force an amount of discharge at timestep t at the boundary that produced at that same timestep, while in reality this runoff would arrive later at the outlet. Considering this, the flow hydrograph at the downstream boundary preferably should be shifted.

Currently there is no option available to include lakes and reservoirs in the routing procedure. Like
in HEC-RAS there is an option to include a storage area to simulate a reservoir or a dam breach (Brummer, 2010). Using the vector layer as is used now in this new routing procedure, a reservoir could become a point in the vector, for which this cell would be assigned different characteristics (e.g. a larger width and lower c) than a normal river cell. How this can be executed exactly needs to be explored further.

PCRaster itself also has the option to solve the dynamic wave equation, as well as the kinematic wave equation. However, the latter concerns an experimental implementation, and stability and accuracy cannot be guaranteed. Furthermore, to solve the dynamic wave equations substantial data and information is required as was concluded earlier as well (Sect. 4.1). In addition, working outside the PCRaster environment, as is done with the implementation of the cD-routing scheme, enables us to choose different temporal and spatial resolution used for the routing calculations.
Bibliography


FutureWater, 2015. SPHY.
URL https://github.com/FutureWater/SPHY

Hayami, S., 1951. On the propagation of flood waves., bulletin 1, Disaster Prevention Research Institute, Kyoto University, Kyoto, Japan.

HI-AWARE, 2016. Hi-aware.
URL http://www.hi-aware.org/


URL https://cran.r-project.org/web/packages/rivr/vignettes/technical-vignette.html


BIBLIOGRAPHY


Vakgroep Hydraulica en Afvoerhydrologie, Year unknown. Afvoervoorspellingen voor de rijn bij lobith, nota 67.


A. Overview python tables
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Format</th>
<th>File name</th>
</tr>
</thead>
<tbody>
<tr>
<td>reachID</td>
<td>Locations of all cells belonging to a streamorder x, y</td>
<td>reachid</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>ReachArrayID</td>
<td>Similar to reachID with slopelength (SL) and DEM value</td>
<td>ID, x, y, SL, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>SubReachArrayID</td>
<td>Similar to ReachArrayID but now for each sub-reach within ID, x, y, SL, #order, #sub</td>
<td>ID, x, y, SL, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>outIDreach</td>
<td>Location of outlets of all reaches from all streamorders</td>
<td>ID, x, y, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>inIDreach</td>
<td>Location of inlets of all reaches from all streamorders</td>
<td>ID, x, y, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>upsIDreach</td>
<td>Location of inlets of all reaches with streamorder &gt; 1</td>
<td>ID, x, y, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>pitID</td>
<td>Same as outIDreach, however in the order that PCRaster scans x, y the raster. Needed to convert arrays to maps again.</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>inID</td>
<td>Similar to inIDreach, however in the order that PCRaster scans x, y the raster and only reaches with streamorder 1. Needed to convert arrays to maps again.</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>uphydroID</td>
<td>Contains the locations of inlets of reaches of x, y streamorder &gt; 1.</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>inoutID</td>
<td>Contains the locations of all outlets plus inlets of x, y reaches with streamorder 1.</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>connectID</td>
<td>Location where a stream flows into the next stream. x, y</td>
<td></td>
</tr>
<tr>
<td>order</td>
<td>coupled reaches</td>
<td>Lists which reaches are connecting to each other. First two columns contain information about the upstream reach. The latter 2 columns store information about the reach it flows into.</td>
<td>#order, #sub, #order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>TimeArrayPit</td>
<td>Stores for each outlet the accumulated runoff at that point for every timestep. TimeArrayPits</td>
<td>Timestep, outlet 1, outlet 2, ...</td>
</tr>
<tr>
<td>order</td>
<td>TimeArrayIn</td>
<td>Stores for each inlet with streamorder 1 the accumulated runoff at that point for every timestep. TimeArrayinlets</td>
<td>Timestep, inlet 1, inlet 2, ...</td>
</tr>
<tr>
<td>order</td>
<td>TimeArrayUphydro</td>
<td>Stores for each inlet with streamorder &gt; 1 the accumulated runoff at that point for every timestep. TimeArrayups</td>
<td>Timestep, inlet 1, inlet 2, ...</td>
</tr>
<tr>
<td>order</td>
<td>TimeArrayOutlet</td>
<td>Stores for the outlet of the entire catchment the accumulated runoff for every timestep. TimeArrayoutlet</td>
<td>Timestep, inlet 1, inlet 2, ...</td>
</tr>
<tr>
<td>order</td>
<td>Uphydro</td>
<td>Hydrograph at the inlet of a reach. #rows = #time steps uphydro order sub</td>
<td>#order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>Downhydro</td>
<td>Hydrograph at the outlet of a reach. #rows = #time steps downhydro order sub</td>
<td>#order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>Lateral</td>
<td>Hydrograph at the cell that laterally drains into the next reach. #rows = #time steps lateral order sub</td>
<td>#order, #sub</td>
</tr>
<tr>
<td>order</td>
<td>Q</td>
<td>Stores for each cell of a reach the results of the cD-equation for each time step. 'back' refers to running the cD-equation in upstream direction. Qback</td>
<td>#columns = #cells, #rows = #time steps</td>
</tr>
<tr>
<td>order</td>
<td>Q</td>
<td>Stores for each cell of a reach the results of the cD-equation for each time step. Now the cD-equation is applied in downstream direction. Q</td>
<td>#columns = #cells, #rows = #time steps</td>
</tr>
<tr>
<td>order</td>
<td>Qall</td>
<td>Stores for each cell of the entire stream network the results of the cD-equation for each time step. Used for visualizing the results. Qall.txt</td>
<td>#columns = #cells, #rows = #time steps</td>
</tr>
</tbody>
</table>
B. Reach length based on ldd map

# Determine the length of the reach in km. Based on the flow direction the flow path
# is either the spatial resolution of a cell (straight), or it flows diagonal
# over the cell. In the latter case Pythagorean theorem is applied (diag).
# Find for each cell its drain direction
lddreach = pcr.ifthen(subreach,FlowDir_ini)
ldd_reacharray = pcr.pcr2numpy(lddreach, MV)

# Count the number of cells in which the water flows diagonally or straight respectively.
diag = numpy.count_nonzero(ldd_reacharray==1) + numpy.count_nonzero(ldd_reacharray==3) +
       numpy.count_nonzero(ldd_reacharray==7) + numpy.count_nonzero(ldd_reacharray==9)
straight = numpy.count_nonzero(ldd_reacharray==2) + numpy.count_nonzero(ldd_reacharray==4) +
          numpy.count_nonzero(ldd_reacharray==6) + numpy.count_nonzero(ldd_reacharray==8)
pit = numpy.count_nonzero(ldd_reacharray==5)

# Calculate the length of the reach.
length_reach = diag*(math.sqrt((SpaceRes**2)*2)) + (straight+pit)*SpaceRes
numpy.save(outpath + 'length_' + str(order) + '_' + str(sub+1), length_reach)
C. Calibration log-file

This appendix is written after the first results were presented already and it continues on the results that were found during the sensitivity analyses (i.e. it was found that mass is not preserved). Artificial examples were performed to test the numerics of the routing module. Apart from minor revisions the code appears to be solid. However, it was found that the routing module is highly sensitive to parameter values and parameter combinations. Therefore a script called 'cD_test.py' was created in which many parameter combinations are tested and the preliminary results are written to a log-file. The user should specify the range for the parameters c, D, dt and dx in the config-file, as well as the range within which the simulated discharge should remain, otherwise a warning is raised.

```
# Length of reach in km
L = length

# Specify the weight given to Neumann and Dirichlet part of the Robin b.c.
alpha = config.getfloat('CDROUTING','alpha')
beta = config.getfloat('CDROUTING','beta')

# Implicit/explicit (omega=1 --> fully implicit, omega=0 --> fully explicit)
omega = config.getfloat('CDROUTING','omega')

# Generate TABLE to test/calibrate the results of several combinations of parameters c, D, dt, dx.
C_min = config.getfloat('CDROUTING','C_min')
C_max = config.getfloat('CDROUTING','C_max')
C_step = (C_max-C_min)/5 # 5 interval steps
C = np.arange(C_min,C_max+C_step,C_step)
if C_min == 0.0: # Cannot handle 0
    C[0] = 0.0001

D_min = config.getfloat('CDROUTING','D_min')
D_max = config.getfloat('CDROUTING','D_max')
D_step = (D_max-D_min)/5 # 5 interval steps
D = np.arange(D_min,D_max+D_step,D_step)
if D_min == 0.0: # Cannot handle 0
    D[0] = 0.0001

Dt_min = config.getfloat('CDROUTING','Dt_min')
Dt_max = config.getfloat('CDROUTING','Dt_max')
Dt_step = (Dt_max-Dt_min)/5 # 5 interval steps
DT = np.arange(Dt_min,Dt_max+Dt_step,Dt_step)

Dx_min = config.getfloat('CDROUTING','Dx_min')
Dx_max = config.getfloat('CDROUTING','Dx_max')
Dx_step = (Dx_max-Dx_min)/5 # 5 interval steps
DX = np.arange(Dx_min,Dx_max+Dx_step,Dx_step)
```
48 # Range within the simulated Q should stay, otherwise a warning is raised.
49 Q_max = config.getfloat('CDROUTING', 'Q_max')
50 Q_min = config.getfloat('CDROUTING', 'Q_min')
52 # Open/create log-file to write the results to.
53 f = open('logfile', 'w')
54 f.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s
56 # Try all parameter combinations:
57 for c in C:
58 for d in D:
59 for dt in DT:
60 for dx in DX:
61 warningFLAG = 0
62 # Number of spatial steps
63 Nx = int((L/dx)+1)
64 x_grid = np.array([j*dx for j in range(Nx)])
66 # Number of time steps
67 Nt = 50
68 t_grid = np.array([n*dt for n in range(Nt)])
70 # Define sigma and rho in the context of numerical discretization
71 sigma = float(d*dt)/float(dx*dx)
72 rho = float(-c*dt)/float(2.*dx)
73 Courant = 4*-rho
74 Peclet = dx*c/d
76 # Create tridiagonal matrices A and B
77 A = np.diagflat([(-sigma+rho)*omega for i in range(Nx-1)], -1) +
78 np.diagflat([1.+2.*sigma*omega for i in range(Nx)]) +
79 np.diagflat([(-(sigma+rho))*omega for i in range(Nx-1)], 1)
80 A[-1, :] = np.array([1] + [0 for i in range(0, Nx-1)])
81 A[-1, :] = np.array([0 for i in range(0, Nx-3)] + [-1] +
82 [(2*beta*dx)/alpha] + [1])
83 B = np.diagflat([(sigma-rho)*(1-omega) for i in range(Nx-1)], -1) +
84 np.diagflat([1.-2.*sigma*(1-omega) for i in range(Nx)]) +
85 np.diagflat([(sigma+rho)*(1-omega) for i in range(Nx-1)], 1)
86 B[0, :] = np.array([0 for i in range(0, Nx)])
87 B[-1, :] = np.array([0 for i in range(0, Nx-3)] + [-1] +
88 [(2*beta*dx)/alpha] + [1])
89 # Create vector F
90 f_vec = np.array([0 for i in range(0, Nx)])*dt
92 # Specify initial condition
93 base = config.getfloat('CDROUTING', 'Q')
94 init = np.array([base for i in range(0, Nx)])
95 Q = init
97 # Impulse upstream
98 impulsFLAG = config.getint('CDROUTING', 'impulsFLAG')
99 if impulsFLAG == 0:
100     value_impuls = 50
width_impuls = 2
up = np.array(np.ndarray.tolist(np.linspace(base, value_impuls, width_impuls))
+ np.ndarray.tolist(np.linspace(value_impuls, base, width_impuls))
+ [base for i in range(0, Nt-(2*width_impuls))])
else:
amplitude = 50
width_impuls = 5
freq = 1.1
x = np.arange(width_impuls)
y = base*amplitude*np.sin(np.pi * freq * x / width_impuls)
up = np.array(np.ndarray.tolist(y)+[base for i in range(0, Nt-width_impuls)])

# Create empty list to store the results
Q_record = []

# First create a stationary solution which is subsequently provided to
# the main loop. To create this stationarity no impulse is given
# upstream.
for ti in range(1, 100):
    Q_new = np.linalg.solve(A, B.dot(Q)+ f_vec)
    Q = Q_new
    Q_record.append(Q)

# Main loop to solve the system iteratively
for ti in range(1, Nt):
    # Provide upstream hydrograph
    f_vec[0] = up[ti]*dt
    # Solve the system
    Q_new = np.linalg.solve(A, B.dot(Q)+ f_vec)
    Q = Q_new
    if max(Q)>Q_max or min(Q)<Q_min:
      warningFLAG = 1
      print 'WARNING: large oscillations: choose other parameter values'
      break
    Q_record.append(Q)

# If warning is raised: write the parameter values to the log-file
# without continuing the calculations
if warningFLAG==1:
    f.write("%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%s,%s,%s,%s,%s\n"
    (c,d,dt,dx,Courant,Peclet, 'NaN','NaN','NaN','NaN','WARNING'))
continue

# If no warning is raised: create array Q_record
Q_record = np.array(Q_record)

# Calculate the maximum and minimum simulated Q
Qmax=[]
Qmin=[]
for time in range(0, Nt):
    Qmax.append(max(Q_record[time]))
    Qmin.append(min(Q_record[time]))
Qmax = max(Qmax)
Qmin = min(Qmin)
# Calculate the total Q at the upstream and downstream end of the reach.
# If the wave is passed completely, these totals should be equal.
Qtot_up = 0
for j in range(len(Q.record)):
  Qtot_up += Q.record[j][0]

Qtot_down = 0
for j in range(len(Q.record)):
  Qtot_down += Q.record[j][-1]

# Write the results of this parameter combination to the log-file
f.write("%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%s\n" %
(c,d,dt,Courant,Peclet,Qmax,Qmin,Qtot_up,Qtot_down,'-'))