ELV

ELV

User Manual & Technical Reference

`Version: 1.18`

Víctor Chavarrías[1] and Liselot Arkesteijn[1]

[1]Delft University of Technology

April 19, 2017

Contents

Preface

ELV is a 1D morphodynamic model, written in Matlab and used as a research model. The model is constructed in a modular way, where the hydrodynamic and morphodynamic behaviour of a 1D single branch river system are solved in a decoupled fashion. This document explains how the model ELV is organized and how to run it. At the end of this manual a tutorial for a first run will be included in the future. Many thanks to the amazing V for creating the structure of ELV and his determination to keep it organized.

1 Model organization

1.1 Introduction

1.1.1 Typographical Conventions

Directory names are expressed in between angle brackets and with the backslashes (e.g. <\directory\>).

File names in between angle brackets (e.g. <filename.ext>).

Variables in the model are written in Courier New (e.g. `variable`).

Units are written between vertical brackets (e.g. $[m]$).

The kind of variable is written in between vertical brackets and in Courier New font. The size may be specified as a product (e.g. [100x1 `double`])

1.1.2 Directories Organization

The model is composed of several functions stored in one folder. The folder name is irrelevant, we use <\ELV\>.

A specific run needs a folder. All the input and output files will be stored in this folder. The folder name is irrelevant, e.g. <\r01\>.

The relative location of the folders is irrelevant.

The user may use his/her own working folder with auxiliary scripts, e.g. <\code\>.

1.2 Model layout

We need to add a nice flowchart;

1.3 SVN version control

To be updated by the amazing V;

2 Governing equations

2.1 Flow model

2.1.1 Saint-Venant

For the unsteady flow solver; the 1D Saint-Venant equations are solved. ADD MORE ONCE WIDTH VARIATIONS ARE INCLUDED

2.1.2 Diffusive wave model

2.1.3 Backwater model

At the moment two implementations of the backwater model are present, one using the conservation of energy, the other starting from momentum conservation. The latter implementation accounts for compound channels.

Solver 1

The first solver uses the energy conservation principle.

$$H = \eta + h + \frac{u^2}{2g} \tag{1}$$

$$S_f = \frac{c_f u^2}{gh} \tag{2}$$

$$\frac{\partial H}{\partial s} = -S_f \tag{3}$$

The solution procedure is as follows:

1. Compute the specific discharge at each location
2. Compute the downstream energy, based on the downstream boundary condition and local specific discharge;
3. Compute the friction slope
4. March upstream using an Euler forward method

All variables are computed at the cell's centers.

Solver 2

See Appendix B for details;

2.2 Sediment mass conservation model

2.2.1 Exner

2.2.2 Hirano

Add some details on the substrate?

2.3 Equilibrium models

2.3.1 Analytical model

2.3.2 Space-marching model

3 Model setup

3.1 Workflow

1. The user creates the simulation directory (e.g. <\r01\>).
2. The user creates a Matlab variable called <input.mat> containing the input for the simulation in that folder.
3. The user calls the function <run_ELV.m> in which the argument is <input.mat>.
4. The simulation runs and the results are stored in <input.mat> in the same folder.
5. The postprocessing runs and the figures are stored under a folder called <\r01\figures\>.

How to input is explained in Section 4. The run process is explained in Section 3.2. The model itself is explained in Section 3.3.

3.2 Run

Once the input file is created, to run a simulation the user needs to call the function <run_ELV> (Appendix **??**). This consist of:

1. Initialization <\output\> and <\figures\>.
2. Call to the main function <ELV> (Section 3.3).
3. Call to the postprocessing routines.

3.2.1 Initialization

In this part the folder where the figures are going to be written are created and the log file is created.

3.2.2 Call to ELV

In this part the main function is called.

3.2.3 Postprocessing

In this part the postprocessing routines are called.

3.3 ELV

The mathematically dependent variables of the model are u, h, η_b, and M_{ak}:

- `u` : mean flow velocity $[m/s]$; [1xn_x `double`]
- `h` : flow depth $[m]$; [1xn_x `double`]
- `etab`: bed elevation $[m]$; [1xn_x `double`]
- `Mak` : sediment mass in the active layer per unit area $[m]$; [n_fxn_x `double`]

Other variables in the model are:

- `qbk` : specific sediment transport per grain size excluding pores $[m^2/s]$; [n_fxn_x `double`]

- `Cf` : dimensionless friction coefficient $[-]$; [1xn_x `double`]
- `La` : active layer thickness $[m]$; [1xn_x `double`]
- `Msk` : sediment mass in the substrate per unit area $[m]$; [n_fxn_xxn_{sl} `double`]

All the variables are organized as follows:

- in dimension 1 (row) they store the data per size fraction (k).
- in dimension 2 (colums) they store the data streamwise direction (x).
- in dimension 3 they store the data per layer (only the substrate data) (sl).

The main function is called <ELV> (Appendix **??**). This function contains the following parts:

1. Initialization (Section **??**)
2. Preprocessing (Section **??**)
3. Time Loop (Section **??**)

3.4 Equilibrium models

ELV contains a special option to compute the equilibrium state of the river under the prescribed boundary conditions. There are three options for this, each of which are accesible within ELV and as a seperate model funtion.

3.4.1 Equilibrium with ELV

The equilibrium state using a time-marching model within ELV is found by setting the total simulation time of the run to NaN. The remainder of the workflow stays the same. An automatic criterion for equilibrium is implemented, based on the periodicity of the external forcings.

3.4.2 Normal flow analytical equilibrium solution

The equilibrium state in the normal flow segment, as described by the analytical equilibrium solution can be found using the same workflow as for ELV, but rather than that calling the function <run_nfanalyt.m> in which the argument is <input.mat>. The variable <input.mat> can be specified the same as in ELV, however, not all variables needed for a transient simulation are needed here aswell.

3.4.3 Space-marching solution

The equilibrium state computed using a space marching procedure can be obtained by running <run_spacemarching.m>, rather than <run_nfanalyt.m>. This is one opition, in addition when a transient simulation is started from a space-marching equilibrium solution, the entire workspace after space-marching is stored in <output_sp.mat>.

4 Input parameter specification in <input.mat>

There are 2 ways of creating the variable called <input.mat>:

1. Via script
2. Via ascii-files

For option 1 we provide the script <input_single_ELV.m>. The values are written in the script and when executed the variable is created in the target folder. Similar scripts can be used to automatically generate several simulations.

For option 2 we provide the script <input_dir_ELV.m>. The input is the folder containing the ascii-files. The files can be either the input files of the old model or Delft3D input files. <input.mat> contains one single variable called `input`. The variable input is a structure array that contains

- `input.run`: run identificator [`char`]
- `input.mdv`: master definition variables [`struct`]
- `input.grd`: grid [`struct`]
- `input.mor`: morphology [`struct`]
- `input.sed`: sediment characteristics [`struct`]
- `input.tra`: sediment transport [`struct`]
- `input.ini`: initial condition [`struct`]
- `input.bch`: hydrodynamic boundary conditions [`struct`]
- `input.bcm`: morphodynamic boundary conditions [`struct`]
- `input.nour`: variables related to including a nourishment [`struct`]
- `input.oth`: remaining variables or conditions, waiting to be categorized at a later stage [`struct`]

In this section we will use the following notation:

- n_f stands for the number of sediment size fractions.
- n_x stands for the number of nodes in streamwise direction.
- n_{sl} stands for the number of substrate layers.
- n_i stands for the number of input values.

4.1 Run Identificator (`input.run`)

`input.run` contains the tag that identifies a run.

- `input.run`: simulation name [`char`]; e.g. `'L_05'`

4.2 Master Definition (`input.mdv`)

`input.mdv` contains the general variables and flags of the run.

- `input.mdv.flowtype` : flow assumption: 1=steady; 2=quasi-steady; 3=unsteady explicit; 4 = unsteady implicit (Preismann); 5 = space-marching model [1x1 `double`]; e.g. [1]
- `input.mdv.fluxtype` : fluxtupe in case of unsteady explicit method: 1=..; 2=..; 3=...; [1x1 `double`]; e.g. [1]
- `input.mdv.frictiontype`: friction type: 1=constant; 2=related to grain size; 3=related to flow depth; [1x1 double]; e.g. [1]
- `input.mdv.Tstop` : simulation time [s]; [1x1 `double`]; e.g. [3600]. When NaN is specified as input, the simulation is ran until equilibrium is reached.
- `input.mdv.dt` : time step [s]; [1x1 `double`]; e.g. [2]
- `input.mdv.Flmap_dt` : printing map-file interval time [s]; [1x1 `double`]; e.g. [60]
- `input.mdv.Cf` : friction coefficient [$-$]; [1x1 `double`]; e.g. [0.008]
- `input.mdv.rhow` : water density [kg/m^3]; [1x1 `double`]; e.g. [1000]
- `input.mdv.g` : gravity constant [m/s^2]; [1x1 `double`]; e.g. [9.81]

When the friction is [1x1 `double`], or a vector [1xn_x `double`], then the cross-section is rectangular. For the latter, rather than the same value along the entire domain the vector specifies the value per node. For the space-marching method the friction is interpolated. In the solver-option of flowtype 6, it is possible to specify a non-rectangular cross-section. The input should then be specified as [4x1 `double`] (constant along the streamwise) or [4xn_x `double`], varying within the cross-section and per node. The rows are respectively 1) representative friction (NaN); 2) main channel friction; 3) left flood plain friction; and 4) right flood plain friction.

4.3 Grid (`input.grd`)

`input.grd` contains the general variables and flags of the run.

- `input.grd.L` : domain length [m]; [1x1 `double`]; e.g. [100]
- `input.grd.B` : domain width [m]; [1x1 `double`]; e.g. [1]
- `input.grd.dx`: streamwise discretizations [m]; [1x1 `double`]; e.g. [0.1]

When the width is [1x1 `double`], or a vector [1xn_x `double`], then the cross-section is rectangular. For the latter, rather than the same value along the entire domain the vector specifies the value per node. For the space-marching method the width is interpolated. In the solver-option of flowtype 6, it is possible to specify a non-rectangular cross-section. The input should then be specified as [4x1 `double`] (constant cross-section) or [4xn_x `double`], cross-section per node. The rows are respectively 1) total width; 2) main channel width; 3) left flood plain width; and 4) right flood plain width.

4.4 Morphology (`input.mor`)

input.mor contains the morphology input.

- `input.mor.bedupdate` : flag to update the bed: 0=No; 1=Yes [−]; [1x1 `double`]; e.g. [1]
- `input.mor.gsdupdate` : flag to update the grain size distribution: 0=No; 1=Yes [−]; [1x1 `double`]; e.g. [1]
- `input.mor.ellcheck` : flag to check for ellipticity: 0=No; 1=Yes [−]; [1x1 `double`]; e.g. [1]
- `input.mor.Latype` : active layer assumption: 1=constant thickness; 2=linked to grain size; 3=linked to flow depth; [1x1 `double`]; e.g. [1]
- `input.mor.La` : active layer thickness $[m]$; [1x1 `double`]; e.g. [0.1]
- `input.mor.ThUnLyr` : thickness of each underlayer $[m]$; [1x1 `double`]; e.g. [0.15]
- `input.mor.interfacetype` : fractions at the interface 1=Hirano; 2=Hoey and Ferguson [-]; [1x1 `double`]; e.g. [1]
- `input.mor.total_ThUnLyr`: thickness of the entire bed $[m]$; [1x1 `double`]; e.g. [2]
- `input.mor.MorStt` : spin-up time $[s]$; [1x1 `double`]; e.g. [60]
- `input.mor.MorFac` : morphological accelerator factor [−]; [1x1 `double`]; e.g. [10]
- `input.mor.porosity` : bed porosity [-]; [1x1 double]; e.g. [0.4]

4.5 Sediment Characteristics (`input.sed`)

`input.sed` contains the sediment characteristics.

- `input.sed.dk` : characteristic grain sizes $[m]$; [n_fx1 `double`]; e.g. [0.0005;0.003;0.005]
- `input.sed.rhos`: sediment density $[kg/m^3]$; [1x1 `double`]; e.g. [2650]

4.6 Sediment Transport (`input.tra`)

`input.tra` contains the sediment transport input.

- `input.tra.cr` : sediment transport closure relation: 1=Meyer-Peter-Müller; 2=Engelund-Hansen; 3=Ashida-Michiue; 4=Wilcock-Crowe; 5= Generalized Load relation [1x1 `double`]; e.g. [1]
- `input.tra.param`: sediment transport parameters (depending on the closure relation) Default values (to be implemented are:)
  1. MPM: e.g. [8, 1.5, 0.047]
  2. EH: e.g. [0.05, 5]
  3. AM: e.g. [17, 0.05]
  4. WC, no parameters required
  5. GL: e.g. [,]
- `input.tra.hid` : hiding function: 0=NO function; 1=Egiazaroff; 2=Power-Law; 3=Ashida-Mishihue;

4.7 Initial Condition (`input.ini`)

`input.ini` contains the initial condition.

- `input.ini.initype`: kind of initial condition: 1=normal flow equilibrium (given an upstream sediment load); 2=free (manually define); 3 =from file; 4= space-marching equilibrium (given an upstream sediment load) [1x1 `double`]; e.g. [1]

If the initial condition is set to be free, input is required for:

- `input.ini.u` : flow velocity $[m/s]$; [1x1 `double`]; e.g. [10]
- `input.ini.h` : flow depth $[m]$; [1x1 `double`]; e.g. [10]
- `input.ini.slopeb` : bed slope $[-]$; [1x1 `double`]; e.g. [10]
- `input.ini.etab0` : downstream bed elevation [m]; [1x1 `double`]; e.g. [10]
- `input.ini.etab` : bed elevation $[m]$; [1x1 `double`]; e.g. [10]
- `input.ini.Fak` : effective fractions at the active layer $[-]$; [$(n_f - 1)$x1 `double`]; e.g. [0.2,0.3]

for all except `input.ini.etab0` and `input.ini.Fak`, the input can also be a vector [1xn_x `double`]. For `input.ini.Fak`, the input can also be a matrix with dimensions [$(n_f - 1)$xn_xxn_{sl} `double`]. Then, rather than the same value along the entire domain the vector specifies the value per node. You can either specify `input.ini.slopeb` and `input.ini.etab0` or `input.ini.etab`. All of them is redundant and throws an error.

For the restart from file, a path to another result folder and a time step at which the results should be loaded are required:

- `input.ini.rst_file`: path to the folder (be aware of Linux/Windows)!
- `input.ini.rst_resultstimestep`: result time step with the initial condition $[-]$; [1x1 `double`], e.g. [70]; if NaN it will be the last one.

For all other cases, necessary input is:

- `input.ini.fsk` : effective fractions at the substrate $[-]$; [$(n_f - 1)$x1 `double`]; e.g. [0.2,0.3]

the input can also be a matrix [$(n_f - 1)$xn_xxn_{sl} `double`]. Then, rather than the same value along the entire domain the matrix specifies the values per node and per layer in the substrate. The input can also be the path to a file containing the full matrix. Also, the input can be a string 'Fak'; which indicates the values of 'Fak' (either via input or automatically computed) are copied.

For the space-marching equilibrium, an extra struct input.ini.sp is required, with following fields:

- `input.ini.sp.intype` : specification of upstream boundary condition; 1 = hydrograph series (Q,t) , 2=discrete probability distribution (Q,p(Q)) $[-]$; e.g. [1]
- `input.ini.sp.outtype`: specification of output; 1 = mean variables; 2 = mean bed elevation and dominant discharge flow variables; 3 = first time step variables (suited for periodic bcs); 4 = full output (variables during all discharges)

- `input.ini.sp.nmodes`: Number of modes [−] to discretize the pdf in (only applicable when no time reconstruction is performed), e.g. [100]
- `input.ini.sp.dx`: Space marching step [m]; [1x1 `double`], e.g. [1]. Overwrites the default automatic guess.
- `input.ini.sp.al`: Average annual load, use this as an overwrite to start from a different equilibrium condition.

4.8 Hydrodynamic Boundary Conditions (`input.bch`)

`input.bch` contains the boundary condition for hydraulics.

- `input.bch.uptype` : type of hydrodynamic boundary condition at the upstream end: 1,11=water discharge from input; 12 = water discharge from matlab file; 13 = daily discharge series from ascii-file; 14 = SOBEK-discharge file [-]; e.g. [1]
- `input.bch.dotype` : type of hydrodynamic boundary condition at the downstream end: 1,11=water level from input; 12 = water level from file; 2,21=water depth from input; 22=water depth from file [-]; [1x1 `double`]; e.g. [1]

For the input specification from file; if a too short time series is prescribed, the series automatically repeated until the end of the total simulation time. For the boundary conditions from input this is not the case!

When the boundary conditions are specified from file, they can by default be saved as a time series at the result folder. Otherwise, define:

- `input.bch.path_file_Q` : pathname where data is located [-]
- `input.bch.path_file_etaw0`: pathname where data is located [-]

When the variables are declared from input; the following are required:

- `input.bch.timeQ0` : time at which the specific water discharge is specified [s]; [n_ix1 `double`]; e.g. [1800;3600]
- `input.bch.Q0` : water discharge at the specified times [m^3/s]; [n_tx1 `double`]; e.g. [1;2]
- `input.bch.timeetaw0`: time at which the downstream water level is specified [s]; [n_ix1 `double`]; e.g. [1800;3600]
- `input.bch.etaw0` : downstream water level at the specified times [m]; [n_tx1 `double`]; e.g. [1;1.5]

4.9 Morphodynamic Boundary Condition (`input.bcm`)

`input.bcm` contains the boundary condition for morphology.

- `input.bcm.type` : type of morphodynamic boundary condition: 1,11=sediment discharge from input; 12 = sediment discharge from file; 13 = normal flow load distribution
- `input.bcm.NFLtype` : type of variables specified for normal flow load computation: 1 = slope and active layer bed fractions; 2 = average annual load and bed fractions; 3 = average annual load per fraction

- `input.bcm.NFLparam` : input parameters

lope, frac1, frac2, ... fracN . The fractions should add up to one. For unisize simulations only the slope is required.

AL, frac1,frac2,...,fracN . The fractions should add up to one. For unisize simulations only the average load is required.

Al1,AL2,...ALN .

- `input.bcm.NFLiparam` : (optional) provide a sophisticated initial guess to have better changes to find the solution

  1. not implemented yet
  2. not implemented yet
  3. $$AL1, AL2, ...ALN$$

  .

- `input.bcm.timeQbk0`: time at which the input is specified [s]; [n_ix1 `double`]; e.g. [1800;3600]
- `input.bcm.Qbk0` : volume of sediment transported excluding pores per unit time, and per size fraction at the specified times [m^3/s]; [n_txn_f `double`]; e.g. [2e-4,4e-4;3e-4,5e-4]
- `input.bcm.path_file_Qbk0`: pathname where data is located [-], default is the result folder.

4.10 Nourishment variables (`input.nour`)

Add nourishment things

5 Functions

5.1 ELV.m

ELV is the main function of the model

ELV(path_file_input,fid_log)

INPUT:
-path_file_input = path to the file input.mat; [char];
-fid_log = log file identifier

OUTPUT:
-

5.2 ELV_version.m

ELV_version is a function that writes the version of each function in the log file

ELV_version(fid_log)

INPUT:
-fid_log = file identifier

OUTPUT:
-

5.3 active_layer_mass_update.m

active_layer_mass_update updates the mass (volume) of sediment at the active layer.

`Mak_new=active_layer_mass_update(Mak,detaLa,fIk,qbk,bc,input,fid_log,kt)`

INPUT:
-`Mak` = effective volume of sediment per unit of bed area in the active layer [m]; [(nf-1)x(nx) double]
-`detaLa` = variation in elevation of the interface between the active layer and the substarte [m]; [(1)x(nx) double]
-`fIk` = effective volume fraction content of sediment at the interface between the active layer and the substrate [-]; [(nf-1)x(nx) double]
-`qbk` = volume of sediment transported excluding pores per unit time and width and per size fraction [m^2/s]; [(nf)x(nx) double]
-`bc` = boundary conditions structure
-`input` = input structure
-`fid_log` = identificator of the log file
-`kt` = time step counter [-]; [(1)x(1) double]

OUTPUT:
-`Mak_new` = new effective volume of sediment per unit of bed area in the active layer [m]; [(nf-1)x(nx) double]

5.4 active_layer_mass_update_pmm.m

function_name does this and that

Mak_new=active_layer_mass_update_pmm(Mak,detaLa,fIk,qbk,bc,pmm,input,fid_log,kt)

INPUT:
-

OUTPUT:
-

5.5 active_layer_thickness_update.m

active_layer_thickness_update is a function that updates the active layer thickness

La=active_layer_thickness_update(h,Mak,La_old,input,fid_log,kt)

INPUT:
-

OUTPUT:
-

5.6 add_nourishment.m

add_nourishment modifies the bed adding nourished sediment

[Mak_new,msk_new,Ls_new,La_new,etab_new,h_new]=add_nourishment(Mak_old,msk_old,Ls_old,La_old,etab_

INPUT:
-

OUTPUT:
-

5.7 add_sedflags.m

add_sedflags parse the sediment input and adds the sediment transport relation flags to the input structure

input=add_sedflags(input)

INPUT:
-input = input structure

OUTPUT:
-input = input structure

5.8 analytical_cubic_root.m

analytical_cubic_root returns the three roots of a cubic polynomial, provided they are real.

root=analytical_cubic_root(a_coeff, b_coeff, c_coeff, d_coeff)

INPUT:
-a_coeff = a coefficient of the polynomial
-b_coeff = b coefficient of the polynomial
-c_coeff = c coefficient of the polynomial
-d_coeff = d coefficient of the polynomial

OUTPUT:
-root = three roots

5.9 aux_ini_normalflow.m

aux_ini_normalflow is an auxiliary function that returns the difference between the boundary condition values of q and qbk and the computed ones

F=aux_ini_normalflow(X,objective,flg,cnt,cf,La,dk,sed_trans_param,hiding_param,mor_fac)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-a .bcm compatible with D3D is created in file_name

5.10 aux_swc.m

aux_swc is an auxiliary function of side_wall_corection

F=aux_swc(X,u,Sf,input)

INPUT:
-X =

-u =
-Sf =
-input =

OUTPUT:
-F =

5.11 backwater.m

backwater does this and that

[U,H]=backwater(ib,Cf,Hdown,Q,input)

INPUT:
-ib = slope vector
-Cf = dimensionless representative friction
-Q = upstream discharge (constant), or a discharge vector;

OUTPUT:
-U =
-H =

5.12 backwater_step.m

backwater_step computes just one step of the solver u,h,etab,Cf should be single values;

[U,H]=backwater_step(x,h,ib,Cf,Q,input)

INPUT:
-

OUTPUT:
-

5.13 bed_level_update.m

bed_level_update updates the bed elevation

etab_new=bed_level_update(etab,qbk,bc,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.14 bed_level_update_pmm.m

bed_level_update_pmm updates the bed elevationconsidering PMM

etab_new=bed_level_update_pmm(etab,qbk,bc,pmm,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.15 beta_pmm.m

beta_pmm computes the value of beta such that the eigenvalues of the modified matrix A have the same value of than in the original system matrix.

beta_o=beta_pmm(A,alpha_i,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.16 boundary_conditions_construction.m

boundary_condition_construction is a function that interpolates the boundary conditions

bc=boundary_conditions_construction(u,h,Mak,La,Cf,input,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-bc = boundary conditions [struct]

5.17 check_input.m

check_input is a function that checks that the input is enough and makes sense

input_out=check_input(input,path_file_input,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-input = variable containing the input [struct] e.g. input

5.18 check_simulation.m

check_simulation does this and that

check_simulation(u,h,Mak,Mak_old,msk,msk_old,La,La_old,Ls,Ls_old,qbk,bc,ell_idx,celerities,pmm,input,fid_l

INPUT:
-

OUTPUT:
-

5.19 compare_results.m

compare_results compares 2 output file. 0 means they are the same

[equal_bol_t,equal_bol,dif_idx,dif_res,dif_max]=compare_results(output_ref,output_chk,var)

INPUT:
-

OUTPUT:
-

5.20 condition_construction.m

condition_construction does this and that

[u,h,etab,Mak,La,msk,Ls,Cf,bc] = condition_construction (input,fid_log)

INPUT:
-input = input structure
-fid_log = identificator of the log file

OUTPUT:
-

5.21 derivatives.m

function_name does this and that

[dqb_dq,dqbk_dq,dqb_dMal,dqbk_dMal]=derivatives(u,h,Cf,La,qbk,Mak,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.22 display_tloop.m

display_tloop is a function that displays in command at the end of every time step

display_tloop(time_loop,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.23 dv_bouss_coef.m

dv_bouss_coef does this and that

[a1, a2] = dv_bouss_coef(input,x,h)

INPUT:
-

OUTPUT:
-

5.24 dv_cross_aux.m

dv_cross_aux does this and that

diff = dv_cross_aux(input, x, term)

INPUT:
-

OUTPUT:

-

5.25 dv_cross_section.m

dv_cross_section compute the coefficient resulting from the derivative of the cross section:
dAf/dx = a1*(dh/dx) + a2(h)
The computation is performed in different parts for the main channel and flood planes seperately.

[a1, a2, I_a1, I_a2, II_a1, II_a2, III_a1, III_a2] = dv_cross_section(input,x,hh,variable)

INPUT:
-x = should be a node identifier
-h = vector with h(1,x) the location at x!

OUTPUT:
-

5.26 eli_1.m

eli_1 does this and that

La_new=eli_1(La,ell_idx,in_f,input,fid_log,kt)

INPUT:
-

OUTPUT:
-

5.27 elliptic_nodes.m

elliptic_nodes does this and that

[ell_idx,out]=elliptic_nodes(u,h,Cf,La,qbk,Mak,fIk,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.28 errorprint.m

errorprint is a function that prints in the log file the error catched

errorprint(error_obj,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.29 flow_update.m

flow_update updates the flow depth and the mean flow velocity.

[u_new,h_new]=flow_update(u,h,etab,Cf,bc,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.30 folders_creation.m

folders_creation is a function that created the folder where output may be stored.

folders_creation(path_file_input,fid_log)

INPUT:
-

OUTPUT:
-

5.31 friction.m

friction is a function that computed the dimensionless friction coefficient

Cf=friction(h,Mak,Cf_old,input,fid_log,kt)

INPUT:
-

OUTPUT:

-

5.32 friction_correction.m

friction_correction corrects the friction coefficient for wall friction and ripples

[Cf_b]=friction_correction(u,h,Cf,input,fid_log,kt)

INPUT:
-

OUTPUT:
-

5.33 function_layout.m

function_name does this and that

`[output]=function_name(input)`

INPUT:
`-input` = variable containing the input [struct] e.g. input

OUTPUT:
`-output` = variable containing the output [struct] e.g. input

5.34 get_analytical_solution.m

get_analytical_solution does this and that

get_analytical_solution(folder_run)

INPUT:
-

OUTPUT:
-

5.35 get_bouss_coef.m

get_bouss_coef does this and that

alpha_b = get_bouss_coef(input,x,hh)

INPUT:
-

OUTPUT:
-

5.36 get_cross_section.m

get_cross_section does this and that

[T,I,II,III] = get_cross_section(input,x,h,variable)

INPUT:
-

OUTPUT:
-

5.37 get_equislope.m

get_equislope does this and that

S = get_equislope(Qw,input,AL,Fk,X0)

INPUT:
-input = input structure

OUTPUT:
-

5.38 get_equivals.m

get_equivals computes the normal load distribution given a hydrograph and the annual load

[S,Fk] = get_equivals(Qw,input,AL,sedp,X0)

INPUT:
-input = input structure

OUTPUT:
-

5.39 get_equivals2_pdf.m

get_equivals2_pdf does this and that

[B,Fk] = get_equivals2_pdf(pq,dq,input,AL,sedp,S,X0)

INPUT:
-input = input structure

OUTPUT:
-

5.40 get_equivals_V.m

get_equivals_V is a slight modification of get_equivals to not display output

[S0,Fak0,sum_Fak_obj,max_rel_error] = get_equivals_V(Qw,input_1,AL,sedp_1)

INPUT:
-input = input structure

OUTPUT:
-

5.41 get_equivals_pdf.m

get_equivals_pdf does this and that

[S,Fk] = get_equivals_pdf(pq,dq,input,AL,sedp,X0)

INPUT:
-input = input structure

OUTPUT:
-

5.42 get_flow_velocities.m

get_flow_velocities does this and that

U = get_flow_velocities(input,x,Q,h,ib)

INPUT:
-

OUTPUT:

-

5.43 get_rep_fric.m

get_rep_fric does this and that

Cf_rep = get_rep_fric(input,x,h)

INPUT:
-

OUTPUT:
-

5.44 get_sedigraph.m

get_sedigraph does this and that

Qb = get_sedigraph(Qw,input,S,Fk)

INPUT:
-input = input structure

OUTPUT:
-

5.45 grain_size_distribution_update.m

grain_size_distribution_update updates the mass at the active layer and the substrate and the substrate thickness

[Mak_new,msk_new,Ls_new,La_ne,etab_new,ell_idx,out_en,pmm]=grain_size_distribution_update(Mak,msk,Ls,

INPUT:
-

OUTPUT:
-

5.46 ini_Cf.m

ini_Cf is a function that creates the initial friction coefficient vector

Cf=ini_Cf(input,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.47 ini_La.m

ini_La this function creates the initial active layer thickness vector

La=ini_La(input,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.48 ini_msk.m

ini_msk is a function that creates the substrate variable

[msk,Ls]=ini_msk(La,input,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.49 ini_normalflow.m

ini_normalflow is a function that finds the normal flow conditions at the first time step.

[u,h,slopeb,Fak]=ini_normalflow(input,Cf,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.50 ini_normalflow_L.m

ini_normalflow_L is a function that finds the normal flow conditions at the first time step.

[u,h,slopeb,Fak]=ini_normalflow_L(input,Cf,fid_log)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.51 ini_spacem.m

ini_spacem approximates the alternating steady equilibrium profile using the space marching algorithm

[u,h,etab,qb,Fak] = ini_spacem(input,fid_log,bc)

INPUT:
-input: general ELV input file;
input.ini.initype: 12 alternating steady, return mean condition
13 alternating steady + time reconstrcution,
return first condition
14 do not use in combination with ELV
input.ini.initype: 5- space marching model + time reconstruction,
return the first condition as initial condition
51=12 - space marching, return mean condition
52 = 5 =13
53 = 15 =parstudy settings, loading, (dq,pq) from
file

-fid_log: variable for log-file, may be empty
-bc: struct consisting of the boundary conditions
-bc.q0
-bc.qbk
-bc.etaw0

OUTPUT:
-

5.52 initial_condition_construction.m

initial_condition_construction does this and that

[u,h,etab,Mak,La,msk,Ls,Cf]=initial_condition_construction(input,fid_log,bc)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.53 join_results.m

join_results does this and that

join_results(input,fid_log)

INPUT:
-

OUTPUT:
-

5.54 log_file_creation.m

log_file_creation is a function that creates the log file

fid_log=log_file_creation(path_file_input)

INPUT:
-

OUTPUT:
-

5.55 output_creation.m

output_creation is a function that creates the file where the results will be saved and saves the initial condition.

output_creation(input,fid_log)

INPUT:
-

OUTPUT:
-

5.56 phi_func.m

phi_func does this and that

phi = phi_func(theta, input)

INPUT:
-input = input structure

OUTPUT:
-

5.57 preconditioning_mass_matrix.m

preconditioning_mass_matrix does this and that

pmm=preconditioning_mass_matrix(ell_idx,out_en,u,input,fid_log,kt)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.58 preissmann.m

preissmann is the Preismmann scheme for unsteady flow update Implict scheme

[U,H] = preissmann(u,h,etab,Cf,Hdown,qwup,input,fid_log,kt)

INPUT:
-input = input structure

OUTPUT:
-

5.59 print_tloop.m

print_tloop is a function that print in the log file the time to finish

print_tloop(time_loop,input,fid_log,kt,kts)

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:

-

5.60 run_ELV.m

run_ELV is a function that creates the folders for the output, calls the main function and then the postprocessing functions

run_ELV(path_file_input)

INPUT:
-path_file_input = path to the file input.mat; [char];

OUTPUT:
-

5.61 run_spacemarching.m

run_ELV is a function that creates the folders for the output, calls the main function and then the postprocessing functions

run_spacemarching(path_file_input)

INPUT:
-path_file_input = path to the file input.mat; [char];

OUTPUT:
-

5.62 sediment_transport.m

sediment transport calculation

[qbk,Qbk]=sediment_transport(flg,cnt,h,q,cf,La,Mak,dk,sed_trans_param,hiding_param,mor_fac,fid_log,kt)

Symbols used in the size definition:

5.63 side_wall_correction.m

side_wall_correction

[Cf_t,Cf_b,Cf_w,u_st_b]=side_wall_correction(input_i,u,h,Sf)

INPUT:
-Mak = effective volume of sediment per unit of bed area in the active layer [m]; [(nf-1)x(nx) double]
-detaLa = variation in elevation of the interface between the active layer and the substarte [m]; [(1)x(nx) double]
-fIk = effective volume fraction content of sediment at the interface between the active layer and the substrate [-]; [(nf-1)x(nx) double]
-qbk = volume of sediment transported excluding pores per unit time and width and per size fraction [m^2/s]; [(nf)x(nx) double]
-bc = boundary conditions structure
-input = input structure
-fid_log = identificator of the log file
-kt = time step counter [-]; [(1)x(1) double]

OUTPUT:
-Mak_new = new effective volume of sediment per unit of bed area in the active layer [m]; [(nf-1)x(nx) double]

5.64 slope2elevation.m

slope2elevation computes the bed elevation given the slope

INPUT:
-input = variable containing the input [struct] e.g. input

OUTPUT:
-

5.65 solve_Hdown.m

unction obj = solve_Hdown(X,input,dQ,pQ,AL)
solve_Hdown computes H_down
VERSION 3

INPUT:
-input

OUTPUT:
-

5.66 solve_equibed.m

unction obj = solve_equibed(X,H,input,dQ,pQ,AL)
solve_Hdown computes H_down
VERSION 3

INPUT:
-input (ELV)
-X: mean bed elevation
-H: water surface elevation (element or vector)
-dQ: vector with discharge values
-pQ: vector with probability of occurence of discharge values
-AL: average load per fraction
OUTPUT:
-

5.67 solve_mixed.m

unction obj = solve_mixed(X,input,Fak_old, Qbk, dQbkdu, h_old, Fr_old, u_old, pq, K, AL, dxi, dq)
solve_mixed computes an update of the space marching algorithm
VERSION 3

INPUT:
-input

OUTPUT:
-

5.68 solve_nfbc.m

unction obj = solve_nfbc(X,input,Qw,AL)
solve_sedigraph computes the slope and surface fraction
VERSION 1

INPUT:
-X(1) for slope, remainder for fractions
-input for parmaters
-Qw: equidistant spaced hydrograph
-AL: mean annual load per fraction

OUTPUT:
-

5.69 solve_nfbc2_pdf.m

unction obj = solve_nfbc2_pdf(X,input,dq,pq,AL,ib)
solve_sedigraph computes the slope and surface fraction
VERSION 1

INPUT:
-X(1) for slope, remainder for fractions
-input for parmaters
-Qw: equidistant spaced hydrograph
-AL: mean annual load per fraction

OUTPUT:
-

5.70 solve_nfbc_pdf.m

unction obj = solve_nfbc_pdf(X,input,dq,pq,AL)
solve_sedigraph computes the slope and surface fraction
VERSION 1

INPUT:
-X(1) for slope, remainder for fractions
-input for parmaters
-Qw: equidistant spaced hydrograph
-AL: mean annual load per fraction

OUTPUT:
-

5.71 solve_qdom.m

unction obj = solve_qdom(X,input,ib,F,AL,k)
solve_sedigraph computes the slope and surface fraction
VERSION 1

INPUT:
-X: dominant discharge
-input for parmaters
-ib: slope
-F: fraction of gravel
-Qw: equidistant spaced hydrograph
-AL: mean annual load per fraction

OUTPUT:
-

5.72 solve_qdom_pdf.m

unction obj = solve_qdom_pdf(X, input,ib,F,AL,k)
solve_sedigraph computes the slope and surface fraction

VERSION 1

INPUT:
-X: dominant discharge
-input for parmaters
-ib: slope
-F: fraction of gravel
-Qw: equidistant spaced hydrograph
-AL: mean annual load per fraction

OUTPUT:
-

5.73 substrate_update.m

function_name does this and that

INPUT:
-

OUTPUT:
-

5.74 warningprint.m

warningprint does this and that

warningprint(fid_log, text)

INPUT:
-input = input structure

OUTPUT:
-

5.75 write_results.m

write_results does this and that

write_results(input,fid_log,kts)

INPUT:
-

OUTPUT:

-

6 Example

A Steady flow solver

For the backwater solution, under opt. 6 we have implemented a function that just computes one upstream step. This function is also called from the space-marching solution routines. We start from the momentum equation

B Implementation details (remove from manual later)

We allow for a compound channel, with specification of width etc. as illustrated in Figure xx. Mathematically, the expressions are presented below. For now, friction is assumed to spatially constant and hence it cannot be dependent on the local flow depth.

B.1 Main channel

B.1.1 Block Ia

$$\begin{aligned} A &= Bh \\ \frac{\partial A}{\partial x} &= \frac{\partial h}{\partial x}(B) + \frac{\partial B}{\partial x}(h) \\ P &= B \\ \frac{\partial P}{\partial x} &= \frac{\partial B}{\partial x} \end{aligned}$$

B.1.2 Block Ib1

$$\begin{aligned} A_f &= \frac{\min(h,h_l)^2}{2}\frac{B_l}{h_l} \\ &= \frac{(h+h_l-|h-h_l|)^2}{8}\frac{B_l}{h_l} \\ \frac{\partial A_f}{\partial x} &= \frac{1}{8}\left((h+h_l-|h-h_l|)^2\left(\frac{1}{h_l}\frac{\partial B_l}{\partial x}-\frac{B_l}{h_l^2}\frac{\partial h_l}{\partial x}\right)+2\,(h+h_l-|h-h_l|)\,\frac{\partial(h+h_l-|h-h_l|)}{\partial x}\frac{B_l}{h_l}\right) \\ &= \frac{\partial h}{\partial x}\left(\frac{1}{4}\,(h+h_l-|h-h_l|)\,\frac{B_l}{h_l}\left(1-\frac{h-h_l}{|h-h_l|}\right)\right)+ \\ &\quad \frac{\partial h_l}{\partial x}\left(\frac{1}{4}\,(h+h_l-|h-h_l|)\,\frac{B_l}{h_l}\left(\frac{h-h_l}{|h-h_l|}-1\right)-\frac{1}{8}(h+h_l-|h-h_l|)^2\frac{B_l}{h_l^2}\right)+ \\ &\quad \frac{\partial B_l}{\partial x}\left(\frac{1}{8}\,(h+h_l-|h-h_l|)^2\,\frac{1}{h_l}\right) \\ P &= \frac{\min(h,h_l)}{h_l}\sqrt{B_l^2+h_l^2} \\ &= (h+h_l-|h-h_l|)\frac{\sqrt{B_l^2+h_l^2}}{2h_l} \\ \frac{\partial P}{\partial x} &= \frac{\partial h}{\partial x}\left(\frac{\sqrt{B_l^2+h_l^2}}{2h_l}\left(1-\frac{h-h_l}{|h-h_l|}\right)\right)+ \\ &\quad \frac{\partial h_l}{\partial x}\left(\frac{\sqrt{B_l^2+h_l^2}}{2h_l}\left(\frac{h-h_l}{|h-h_l|}-1\right)-\left(\frac{(h+h_l-|h-h_l|)}{2h_l}\sqrt{B_l^2+h_l^2}\right)+\left(\frac{(h+h_l-|h-h_l|)}{2\sqrt{B_l^2+h_l^2}}\right)\right) \\ &\quad +\frac{\partial B_l}{\partial x}\left(\frac{(h+h_l-|h-h_l|)}{2h_l\sqrt{B_l^2+h_l^2}}B_l\right) \end{aligned}$$

where

$$\frac{\partial|h-h_l|}{\partial x}=\frac{h-h_l}{|h-h_l|}\frac{\partial h-h_l}{\partial x}=\frac{h-h_l}{|h-h_l|}\frac{\partial h}{\partial x}-\frac{h-h_l}{|h-h_l|}\frac{\partial h_l}{\partial x}$$

B.1.3 Block Ib2

$$
\begin{aligned}
A_f &= \max(h-h_l,0)B_l \\
&= \frac{(h-h_l+|h-h_l|)}{2}B_l \\
\frac{\partial A_f}{\partial x} &= \frac{\partial h}{\partial x}\left(\frac{B_l}{2}\left(1-\frac{h-h_l}{|h-h_l|}\right)\right)+\frac{\partial h_l}{\partial x}\left(\frac{B_l}{2}\left(\frac{h-h_l}{|h-h_l|}-1\right)\right)+\frac{\partial B_l}{\partial x}\left(\frac{(h-h_l+|h-h_l|)}{2}\right) \\
P &= 0 \\
\frac{\partial P}{\partial x} &= 0
\end{aligned}
$$

B.1.4 Blocks Ic1 and Ic2

These are similar to blocks Ib1, Ib2, and II, where $h_l = h_r$, $B_l = B_r$, and $B_{fl} = B_{fr}$.

B.2 Flood plains

B.2.1 Blocks II and III

$$
\begin{aligned}
A_f &= \max(h-h_l,0)B_{fl} \\
\frac{\partial A_f}{\partial x} &= \frac{\partial h}{\partial x}\left(\frac{B_{fl}}{2}\left(1-\frac{h-h_l}{|h-h_l|}\right)\right)+\frac{\partial h_l}{\partial x}\left(\frac{B_{fl}}{2}\left(\frac{h-h_l}{|h-h_l|}-1\right)\right)+\frac{\partial B_{fl}}{\partial x}\left(\frac{(h-h_l+|h-h_l|)}{2}\right) \\
P &= B_{fl}+h-h_l \qquad \text{if wet} \\
\frac{\partial P}{\partial x} &= \frac{\partial B_{fl}}{\partial x}+\frac{\partial h}{\partial x}-\frac{\partial h_l}{\partial x}
\end{aligned}
$$

Blocks II and III are identical.

B.3 Boussinesq coefficient

For now the friction is assumed to be spatially constant. Then the coefficient is given by:

$$
\begin{aligned}
\alpha_\beta &= \frac{\sum_j (C_j^2 A_{fj} R_j) A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^2} \\
\frac{\partial \alpha_\beta}{\partial x} &= \frac{1}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^2}\frac{\partial \sum_j (C_j^2 A_{fj} R_j) A_f}{\partial x} - \frac{\sum_j (C_j^2 A_{fj} R_j) A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^4}\frac{\partial \left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^2}{\partial x} \\
&= \frac{A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^2}\sum_j\left(C_j^2\frac{\partial A_{fj}^2 P_j^{-1}}{\partial x}\right) - 2\frac{\sum_j (C_j^2 A_{fj} R_j) A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^3}\sum_j\left(C_j\frac{\partial A_{fj}^{3/2} P_j^{-1/2}}{\partial x}\right) \\
&= \frac{A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^2}\sum_j\left(C_j^2\left(2\frac{A_{fj}}{P_j}\frac{\partial A_{fj}}{\partial x}-\frac{A_{fj}^2}{P_j^2}\frac{\partial P_j}{\partial x}\right)\right) - \\
&\quad 2\frac{\sum_j (C_j^2 A_{fj} R_j) A_f}{\left(\sum_j C_j A_{fj}\sqrt{R_j}\right)^3}\sum_j\left(C_j\left(\frac{3}{2}\frac{\sqrt{A_{fj}}}{\sqrt{P_j}}\frac{\partial A_{fj}}{\partial x}-\frac{1}{2}\frac{A_{fj}\sqrt{A_{fj}}}{P_j\sqrt{P_j}}\frac{\partial P_j}{\partial x}\right)\right)
\end{aligned}
$$

B.4 Momentum equation

For the backwater solver we solve the following equation:

$$\frac{\partial}{\partial x}\left(\alpha_\beta \frac{Q^2}{A_f}\right) + gA_f\frac{\partial h}{\partial x} + gA_f\frac{\partial \eta}{\partial x} + \frac{c_f|Q|Q}{RA_f} = 0 \tag{4}$$

where for the symbols we refer to the SOBEK-RE manual.

For the backwater solver we assume that $\frac{\partial Q}{\partial x} = 0$ and we isolate $\frac{\partial h}{\partial x}$ from the equation:

$$\begin{aligned}
-\alpha_\beta \frac{Q^2}{A_f^2}\frac{\partial A_f}{\partial x} + \frac{Q^2}{A_f}\frac{\partial \alpha_\beta}{\partial x} + gA_f\frac{\partial h}{\partial x} + gA_f\frac{\partial \eta}{\partial x} + \frac{c_f|Q|Q}{RA_f} &= 0 \\
-\alpha_\beta \frac{Q^2}{A_f^2}\left(a_1\frac{\partial h}{\partial x} + a_2\right) + \frac{Q^2}{A_f}\left(b_1\frac{\partial h}{\partial x} + b_2\right) + gA_f\frac{\partial h}{\partial x} + gA_f\frac{\partial \eta}{\partial x} + \frac{c_f|Q|Q}{RA_f} &= 0
\end{aligned}$$

$$\begin{aligned}
\left(-a_1\alpha_\beta\frac{Q^2}{A_f^2} + b_1\frac{Q^2}{A_f} + gA_f\right)\frac{\partial h}{\partial x} &= a_2\alpha_\beta\frac{Q^2}{A_f^2} - b_2\frac{Q^2}{A_f} - gA_f\frac{\partial \eta}{\partial x} - \frac{c_f|Q|Q}{RA_f} \\
\frac{\partial h}{\partial x} &= \frac{a_2\alpha_\beta\frac{Q^2}{A_f^2} - b_2\frac{Q^2}{A_f} - gA_f\frac{\partial \eta}{\partial x} - \frac{c_f|Q|Q}{RA_f}}{-a_1\alpha_\beta\frac{Q^2}{A_f^2} + b_1\frac{Q^2}{A_f} + gA_f}
\end{aligned} \tag{5}$$

where the coefficients a_1, a_2, b_1, b_2 are internally computed based on the change in wetted perimeter and cross-section as a function of the water depth and streamwise coordinate.