ChanNet

Generated by Doxygen 1.8.5

Wed Nov 19 2014 16:58:02

Contents

Chapter 1

ChanNet

This code simulates flow through a network of open channels.

COMPILING THE CODE

To compile the code, type the following in the directory where the Makefile is.

1. make

For *wetting and drying purposes*, the code has to be recompiled with wetting and drying turned on. So in the directory where the Makefile and the source files are, do the following:

1. make clean
2. make WD=-DWDON

RUNNING THE CODE:

./ChanNet pathToChannelNodes.in pathToJunctionMesh.in

DESCRIPTION OF THE INPUT FILES

- **ChannelNodes.in**: This is a file that contains the grid information associated with the channels, including the coordinates, the width at the nodes and the Manning's n values.
- **JunctionMesh.in**: This is a file that contains the gird and connectivity information associated with the junctions and their discretizations. It should also have information about the connectivity of the channels.

DESCRIPTION OF FILES THAT NEED TO BE CHANGED FOR EACH RUN

- **main.c**: The parameters H0 and VEL_ZERO are defined here. These might need to be tweaked for different wetting/drying cases.
- **initialize_channels.c**: File where the initial conditions on the channels are specified.
- **initialize_junctions.c**: File where the initial conditions on the junctions are specified.
- **boundary_conditions.c**: File where the boundary conditions on the channels are specified.

Author

Prapti Neupane

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 channel Struct Reference

Data Fields

- int NumEl
- int NumNodes
- int * GlobalNodeNum
- double * x
- double * y
- double * b
- double * z
- double * nFriction
- double mindh
- double * beta
- int NumInflowJunctionEdges
- int * InflowJunctionEdges
- int NumOutflowJunctionEdges
- int * OutflowJunctionEdges
- double * A
- double * Q
- double max_lambda
- int * WD

4.1.1 Field Documentation

4.1.1.1 double* channel::A

Stores the wet cross-sectional area solution. Contains the solution at each node. Size = 2*NumNodes

4.1.1.2 double* channel::b

width of the channel at each node

4.1.1.3 double* channel::beta

The momentum correction coefficient for the channel prescribed at each node

4.1.1.4 int* channel::GlobalNodeNum

Global node numbers of the nodes of the channel

4.1.1.5 int* channel::InflowJunctionEdges

NumInflowJunctionEdges X 2 array stored as a 1-D array in row-major order. For row i, element (i,0) stores the (local) junction edge number connected to the inflow end (represented by the first node) of the channel. Element (i,1) stores the junction number connected to the channel.

4.1.1.6 double channel::max_lambda

Maximum eigenvalue of the jacobian across all elements. This information is recalculated at each time step and is used to set the next time step according to the CFL condition

4.1.1.7 double channel::mindh

size of the smallest element of the channel.

4.1.1.8 double* channel::nFriction

manning's coefficient for the channel prescribed at each node

4.1.1.9 int channel::NumEl

Number of elements used for the discretization of the channel

4.1.1.10 int channel::NumInflowJunctionEdges

Number of junction edges connected to the inflow end (represented by the first node) of the channel

4.1.1.11 int channel::NumNodes

Total number of nodes (counts one node only once)

4.1.1.12 int channel::NumOutflowJunctionEdges

Number of junction edges connected to the outflow end (represented by the last node) of the channel

4.1.1.13 int* channel::OutflowJunctionEdges

NumOutflowJunctionEdges X 2 array stored as a 1-D array in row-major order. For row i, element (i,0) stores the (local) junction edge number connected to the outflow end (represented by the last node) of the channel. Element (i,1) stores the junction number connected to the channel.

4.1.1.14 double* channel::Q

Stores the volumetric discharge solution. Contains the solution at each node. Size = 2*NumNodes

4.1.1.15 int* channel::WD

Stores the wet/dry status of each element. Size = NumNodes-1

4.1.1.16 double* channel::x

x-coordinates of the nodes of the channel

4.1.1.17 double* channel::y

y-coordinates of the nodes of the channel

4.1.1.18 double* channel::z

bathymetry at each node

The documentation for this struct was generated from the following file:

- ChannelsAndJunctions.h

4.2 junction Struct Reference

`#include <ChannelsAndJunctions.h>`

Data Fields

- int TotalNumEdges
- int NumEl
- int NumNodes
- double * x
- double * y
- double * z
- double minEdgLength
- int * EdgtoEls
- int * EltoVert
- int * EdgtoVert
- int * BdryPrescribed
- int * ChannelNumber
- double * zeta
- double * Qx
- double * Qy
- double * bzeta
- double * bQn
- double * nFriction
- int * WD
- double max_lambda

4.2.1 Detailed Description

A structure used to store all the variables associated with the discretization and the physical properties of a 2-D junction element. Other than the solutions (zeta, Qx and Qy), the maximum eigenvalue (max_lambda) and the wet/dry status (WD), every other field is either obtained or can be determined from the grid file.

4.2.2 Field Documentation

4.2.2.1 int* junction::BdryPrescribed

For each edge, stores the information about whether or not the edge is connected to a channel. size = TotalNumEdges x 1. BdryPrescribed(i) is 1 if edge i is connected to an inflow channel, 2 if edge i is connected to an outflow channel and 0 if its neither.

4.2.2.2 double* junction::bQn

Stores the value of normal flow prescribed at an edge. size = TotalNumEdges x 1. 9999 if not connected to a channel.

4.2.2.3 double* junction::bzeta

Stores the value of zeta prescribed at an edge. size = TotalNumEdges x 1. 0 if not connected to a channel.

4.2.2.4 int* junction::ChannelNumber

Stores the channel number associated with each edge. size = TotalNumEdges x 1.

4.2.2.5 int* junction::EdgtoEls

In its row, EdgtoEls stores the elements that are connected by an edge. Size = TotalNumEdges X 2; (i,0) and (i,1) elements are the two elements connected by edge i. If edge i is a boundary edge then it stores the same element in both columns.

4.2.2.6 int* junction::EdgtoVert

Stores the two vertices connected by the edges. size = totalNumEdges X 2; (i,0) and (i,1) are the global vertex numbers of the vertices connected by edge i.

4.2.2.7 int* junction::EltoVert

Stores the global vertex number of the vertices of the elements. size = NumEl X 3. (i,j)th element is the global vertex number of the jth vertex of element i.

4.2.2.8 double junction::max_lambda

Maximum eigenvalue of the jacobian across all elements. This information is recalculated at each time step and is used to set the next time step according to the CFL condition

4.2.2.9 double junction::minEdgLength

length of the smallest edge of an element.

4.2.2.10 double* junction::nFriction

Stores the Manning's coefficient at each node. size = NumNodes

4.2.2.11 int junction::NumEl

Number of elements in the junction

4.2.2.12 int junction::NumNodes

Number of nodes in the junction. Each node is counted only once.

4.2.2.13 double* junction::Qx

Stores the momentum in the x-direction. size = NumEl X 3;

4.2.2.14 double* junction::Qy

Stores the momentum in the y-direction. size = NumEl X 3;

4.2.2.15 int junction::TotalNumEdges

Total number of the edges of the junction element. Each edge is counted only once.

4.2.2.16 int* junction::WD

Stores the wet/dry status of each element. size = NumEl

4.2.2.17 double* junction::x

x-coordinates of the nodes of the junction

4.2.2.18 double* junction::y

y-coordinates of the nodes of the junction

4.2.2.19 double* junction::z

bathymetry at the nodes

4.2.2.20 double* junction::zeta

Stores the water surface height solution. size = NumEl x 3;

The documentation for this struct was generated from the following file:

- ChannelsAndJunctions.h

Chapter 5

File Documentation

5.1 1DInnerProducts.c File Reference

Functions

- void Compute1DInnerProducts (struct channel *Chan, int el, double *IP)

5.1.1 Function Documentation

5.1.1.1 void Compute1DInnerProducts (struct channel * *Chan,* int *el,* double * *IP*)

Function for evaluating the inner products that occur on the right hand side of the DG equations for the 1-D channels

Parameters

in	*Chan*	the channel structure corresponding to the channel that is currently being worked on
in	*el*	integer element number of the channel element that is currently being worked on
out	*IP*	Pointer to a double array of size 10 in which the ten inner products are returned

5.2 2DInnerProducts.c File Reference

Functions

- void Compute2DInnerProducts (struct junction *junc, double *Fhat1dotn, double *Fhat2dotn, double *Fhat3dotn, int el, double *SI, double *VI)

5.2.1 Function Documentation

5.2.1.1 void Compute2DInnerProducts (struct junction * *junc,* double * *Fhat1dotn,* double * *Fhat2dotn,* double * *Fhat3dotn,* int *el,* double * *SI,* double * *VI*)

Function for evaluating the inner products that occur on the right hand side of the DG equations for the 2-D junctions

Parameters

in	*junc*	a pointer to the junction structure corresponding to the junction that is currently being worked on.
in	*Fhat1dotn*	a pointer to the array that contains the first component of the numerical normal flux vector at the edges. Size = NumEl x 3
in	*Fhat2dotn*	a pointer to the array that contains the second component of the numerical normal flux vector at the edges. Size = NumEl x 3
in	*Fhat3dotn*	a pointer to the array that contains the third component of the numerical normal flux vector at the edges. Size = NumEl x 3
in	*el*	integer element number of the element that is currently being worked on
out	*SI*	pointer to a double array of size 9 in which all the surface integrals are returned
out	*VI*	pointer to a double array of size 21 in which all the area integrals are returned

5.3 boundary_conditions.c File Reference

5.3.1 Detailed Description

Specify the boundary conditions on the inflow and outflow channel ends in this file. Currently these conditions need to be compiled. But eventually we might have to change the way these conditions are provided so that they can be read from a time series data file and won't hvave to be compiled.

5.4 ChannelsAndJunctions.h File Reference

Data Structures

- struct channel
- struct junction

5.4.1 Detailed Description

This file contains channel and junction structure definitions.

5.5 compute2DL.c File Reference

Functions

- void compute2DL (struct junction ∗junc, double time, double ∗RHSZeta, double ∗RHSQx, double ∗RHSQy)

5.5.1 Detailed Description

This file contains code to evaluate the right hand side of the following discrete ODE obtained from the DG discretization of the 2-D shallow water equations:

$\frac{\partial \hat{\mathbf{w}}}{\partial t} = M^{-1} L(\hat{\mathbf{w}}, t)$

5.5.2 Function Documentation

5.5.2.1 void compute2DL (struct junction * *junc,* double *time,* double * *RHSZeta,* double * *RHSQx,* double * *RHSQy*)

Function for evaluating the right hand side of the discrete ODE obtained from the DG discretization of the 2-D shallow water equations

Parameters

in	*junc*	a pointer to the junction structure corresponding to the junction that is currently being worked on
in	*time*	a double representing the current time of the simulation
out	*RHSZeta*	a pointer to an array of size NumEl x 3 in which the right hand side for the water surface elevation will be stored
out	*RHSQx*	a pointer to an array of size NumEl x 3 in which the right hand side for the momentum in the x-direction is stored
out	*RHSQy*	a pointer to an array of size NumEl x 3 in which the right hand side for the momentum in the y-direction is stored

5.6 computeL.c File Reference

Functions

- void computeL (struct channel *Chan, double time, int channelNumber, double *RHSA, double *RHSQ)

5.6.1 Detailed Description

This file contains code to evaluate the right hand side of the following discrete ODE obtained from the DG discretization of the 1-D St. Venant equations:

$\frac{\partial \hat{\mathbf{w}}}{\partial t} = M^{-1} L(\hat{\mathbf{w}}, t)$

5.6.2 Function Documentation

5.6.2.1 void computeL (struct channel * *Chan*, double *time*, int *channelNumber*, double * *RHSA*, double * *RHSQ*)

Function for evaluating the right hand side of the discrete ODE obtained from the DG discretization of the 1-D St. Venant equations

Parameters

in	*Chan*	a pointer to the channel structure corresponding to the channel that is currently being worked on
in	*time*	a double representing the current time of the simulation
in	*channelNumber*	an integer number designated for the channel that is currently being worked on. This is only there for debugging purposes and we might not need it anymore.
out	*RHSA*	a pointer to an array of size NumEl x 2 in which the right hand side for the wet cross-sectional area will be stored
out	*RHSQ*	a pointer to an array of size NumEl x 2 in which the right hand side for the volumetric discharge will be stored

5.7 constitutive_equations.c File Reference

Functions

- double getI1 (double A, double b)
- double getI2 (double A, double b, double db)
- double getS_f (double A, double Q, double b, double n)

5.7.1 Detailed Description

This file contains definitions for calculating the pressure terms (I1 and I2) along with the friction slope (S_f using Manning's formula) that appear in the 1-D St. Venant equations. Currently, calculation of I1 and I2 assumes that the channels have rectangular cross-sections. The code can be easily extended for trapezoidal cross-sections.

5.7.2 Function Documentation

5.7.2.1 double getI1 (double *A*, double *b*)

Function that calculates and returns a double value (I1) that represents the hydrostatic pressure term at any point in a channel with rectangular cross-sections

Parameters

in	*A*	cross-sectional area at the quadrature point
in	*b*	width of the channel at the quadrature point

5.7.2.2 double getI2 (double *A*, double *b*, double *db*)

Function that calculates and returns a double value (I2) that represents the wall pressure term at any point in a channel with rectangular cross-sections

Parameters

in	*A*	cross-sectional area at the quadrature point
in	*b*	width at the quadrature point
in	*db*	value of the derivative of width in the element we are currently working on

5.7.2.3 double getS_f (double *A*, double *Q*, double *b*, double *n*)

Function that calculates and returns a double value (S_f) that represents the friction slope term at any point in a channel. The friction slope is calculated using Manning's formula.

Parameters

in	*A*	cross-sectional area at the quadrature point
in	*Q*	volumetric discharge at the quadrature point
in	*b*	width at the quadrature point
in	*n*	value of Manning's coefficient at the quadrature point

5.8 create_channel_network.c File Reference

Functions

- void create_channel_network (char ∗ChannelNodes, char ∗JunctionNodes)

5.8.1 Detailed Description

This file contains code to create a channel network from the grid files provided for the channels and the junctions.

5.8.2 Function Documentation

5.8.2.1 void create_channel_network (char * *ChannelNodes*, char * *JunctionNodes*)

Function that reads in the grid files and creates a channel structure for each channel in the network and a junction structure for each junction in the network. The channel and the junction structures contain all the information about channels and the junctions respectively and their connectivity. The channel structures are then stored in an array called ChannelList and the junction structures are stored in an array called JunctionList.

Parameters

in	*ChannelNodes*	string name of the gid file for channels
in	*JunctionNodes*	string name of the grid file for junctions

5.9 initialize_channels.c File Reference

Functions

- void initialize_channels ()

5.9.1 Detailed Description

This file contains code to initialize the channel structures created in create_channel_network.c and their member fields.

5.9.2 Function Documentation

5.9.2.1 void initialize_channels ( )

This function allocates necessary space for all the member fields of the channel structure. Then it assigns initial values for the quantities A and Q. It also assigns an initial wet/dry state for each element of the channel.

5.10 initialize_junctions.c File Reference

Functions

- void initialize_junctions ()

5.10.1 Detailed Description

This file contains code to initialize the junction structures created in create_channel_network.c and their member fields.

5.10.2 Function Documentation

5.10.2.1 void initialize_junctions ( )

This function allocates necessary space for all the member fields of the junction structure. Then it assigns initial values for the quantities zeta, Qx and Qy. It also assigns an intiial wet/dry state for each element of the junction.

5.11 main.c File Reference

Variables

- const double g = 9.810000000000
- const double H0 = 1e-3
- const double VELZERO = 1e-2

5.11.1 Detailed Description

This file is the driver for ChanNet. When wetting and drying is on, the minimum water threshold and the momentum threshold are also allocated in this file. This needs to be changed later so that it will be a parameter than can be read from a file.

5.11.2 Variable Documentation

5.11.2.1 const double g = 9.810000000000

gravitational acceleration constant

5.11.2.2 const double H0 = 1e-3

Minimum water height threshold that is maintained in all elements. Nodes whose water height is below this threshold are considered dry.

5.11.2.3 const double VELZERO = 1e-2

a momentum threshold. If the magnitude of the momentum at a dry node is below this threshold value, then this momentum will not be transferred to another wet node in the element.

5.12 mathfunctions.c File Reference

Functions

- int sign (double a)
- void addVectors (double vector1[], double vector2[], double sum[], int size, double a, double b)
- void MatrixVectorMultiply (double *mat, double vec[], double prodvec[], int row, int col)

5.12.1 Detailed Description

This file contains some simple math functions that I wasn't sure is included in the C math library.

5.12.2 Function Documentation

5.12.2.1 void addVectors (double *vector1[],* double *vector2[],* double *sum[],* int *size,* double *a,* double *b*)

a function to compute s = a*vector1 + b*vector2

Parameters

in	*vector1*	an array containing doubles
in	*vector2*	an array containing doubles
in	*size*	size of the two vectors being added
in	*a*	a double number by which the first vector is scaled
in	*b*	a double number by which the second vector is scaled
out	*sum*	a double array where the result will be stored

5.12.2.2 void MatrixVectorMultiply (double * *mat,* double *vec[],* double *prodvec[],* int *row,* int *col*)

a function to compute perform a matrix-vector multiplication: a = M*b

Parameters

in	*mat*	a pointer to the matrix M stored as an array in row-major order
in	*vec*	a pointer to the vector b
in	*row*	integer number of rows of matrix M
in	*col*	integer number of columns of matrix M
out	*prodvec*	an array where the result will be stored

5.12.2.3 int sign (double *a*)

a function that returns -1 if a <0, 0 if a = 0 and 1 if a >1

Parameters

in	*a*	a double value whose sign we want to determine

5.13 mathfunctions.h File Reference

5.13.1 Detailed Description

This file contains some simple math macros and function prototypes that I wasn't sure is included in the math library.

5.14 MeshAttributes.h File Reference

Variables

- int NumChannels
- int NumJunctions
- struct channel ** ChannelList
- struct junction ** JunctionList

5.14.1 Detailed Description

This file contains global variables that are used throughout the software for storing the channel and junction structures.

5.14.2 Variable Documentation

5.14.2.1 struct channel** ChannelList

An array that holds the pointers to the channel structures created for each channel in the network

5.14.2.2 struct junction** JunctionList

An array that holds the pointers to the junction structures created for each channel in the network

5.14.2.3 int NumChannels

Total number of channels in the network

5.14.2.4 int NumJunctions

Total number of junctions in the network

5.15 minmod.c File Reference

Functions

- void minmod (struct channel *Chan)

Variables

- const double H0

5.15.1 Detailed Description

This file contains code to apply the minmod slope limiter on the water surface height and the volumetric discharge obtained from the 1-D RKDG scheme.

5.15.2 Function Documentation

5.15.2.1 void minmod (struct channel * *Chan*)

Function to apply the minmod slope limiter on the 1-D conserved variables

Parameters

in	*Chan*	a pointer to the channel structure that we are currently working on

5.15.3 Variable Documentation

5.15.3.1 const double H0

Minimum water height threshold that is maintained in all elements. Nodes whose water height is below this threshold are considered dry.

5.16 numericalFlux1D.c File Reference

Functions

- void RoeFlux1D (double A_L, double A_R, double Q_L, double Q_R, double b, double localG, double beta, double *Fhat)
- void LF (double A_L, double A_R, double Q_L, double Q_R, double b, double localG, double *Fhat)

5.16.1 Detailed Description

This file contains code to compute different types of numerical fluxes for the 1-D St. Venant equations.

5.16.2 Function Documentation

5.16.2.1 void LF (double *A_L*, double *A_R*, double *Q_L*, double *Q_R*, double *b*, double *localG*, double * *Fhat*)

Function to calculate local Lax-Friedrich's flux for the 1-D St. Venant equations at an interface (node).

Parameters

in	*A_L*	value of the wet cross-sectional area at the node coming from the left element.
in	*A_R*	value of the wet cross-sectional area at the node coming from the right element
in	*Q_L*	value of the volumetric discharge at the node coming from the left element
in	*Q_R*	value of the volumetric discharge at the node coming from the right element
in	*b*	value of the width of the channel at the node. We assume that the width of the channel is prescribed at each node and thus the value will be continuous at nodes
in	*localG*	value of the gravitational acceleration constant. This will be the same as g, except when we set it to zero to handle wetting and drying.
out	*Fhat*	a pointer to an array where the numerical flux computed will be stored

5.16.2.2 void RoeFlux1D (double *A_L*, double *A_R*, double *Q_L*, double *Q_R*, double *b*, double *localG*, double *beta*, double * *Fhat*)

Function to calculate Roe's flux for the 1-D St. Venant equations at an interface (node).

Parameters

in	*A_L*	value of the wet cross-sectional area at the node coming from the left element.
in	*A_R*	value of the wet cross-sectional area at the node coming from the right element
in	*Q_L*	value of the volumetric discharge at the node coming from the left element
in	*Q_R*	value of the volumetric discharge at the node coming from the right element
in	*b*	value of the width of the channel at the node. We assume that the width of the channel is prescribed at each node and thus the value will be continuous at nodes
in	*localG*	value of the gravitational acceleration constant. This will be the same as g, except when we set it to zero to handle wetting and drying.
in	*beta*	value of the momentum correction coefficient at the node
out	*Fhat*	a pointer to an array of size 2 where the numerical flux computed will be stored

5.17 numericalFlux2D.c File Reference

Functions

- double RoeFlux2D (double zeta_in, double zeta_ex, double Qx_in, double Qx_ex, double Qy_in, double Qy_ex, double z_edge, double nx, double ny, double localG, double *Fhatdotn)

5.17.1 Detailed Description

This file contains code to compute different types of numerical fluxes for the 2-D shallow water equations.

5.17.2 Function Documentation

5.17.2.1 double RoeFlux2D (double *zeta_in*, double *zeta_ex*, double *Qx_in*, double *Qx_ex*, double *Qy_in*, double *Qy_ex*, double *z_edge*, double *nx*, double *ny*, double *localG*, double * *Fhatdotn*)

Function to calculate Roe's flux for the 2-D shallow water equations at an interface (element edges).

Parameters

in	*zeta_in*	value of the water surface height at the edge coming from the interior element
in	*zeta_ex*	value of the water surface height at the edge coming from the exterior element
in	*Qx_in*	value of the momentum in the x-direction coming from the interior element
in	*Qx_ex*	value of the momentum in the x-direction coming from the exterior element
in	*Qy_in*	value of the momentum in the y-direction coming from the interior element
in	*Qy_ex*	value of the momentum in the y-direction coming from the exterior element
in	*z_edge*	value of the bathymetry at the edge. We assume that the bathymetry information is provided at the edge and thus will be continuous at an edge. In the future we might need to revise this assumption and figure out how to handle it if the bathymetry information is provided at the nodes instead.
in	*nx*	x-component of the outward unit normal vector to the interior element at the edge
in	*ny*	y-component of the outward unit normal vector to the interior element at the edge
in	*localG*	value of the gravitational acceleration constant. This will be the same as g, except when we set it to zero to handle wetting and drying.
out	*Fhatdotn*	a pointer to an array of size 3 where the numerical flux computed will be stored

Returns

current_max_lambda the maximum eigenvalue at the interface

5.18 oneTimeStep.h File Reference

5.18.1 Detailed Description

This file contains function prototypes for the functions that are exectued in order to step forward in time.

5.19 SimulationSteps.h File Reference

5.19.1 Detailed Description

This file contains the function prototypes for functions that represent different stages of the simulation, i.e. function to create the channels and junction structure, functions to initialize those structures and the function to evolve them in time.

5.20 SlopeLimiter2D.c File Reference

Functions

- void SlopeLimiter (struct junction *junc)

Variables

- const double g
- const double H0

5.20.1 Detailed Description

This file contains code to apply a slope limiter on the 2-D conserved variables obtained from the 2-D RKDG scheme.

5.20.2 Function Documentation

5.20.2.1 void SlopeLimiter (struct junction * *junc*)

Function to apply the BDS slope limiter on the 2-D conserved variables

Parameters

in	*junc*	a pointer to the junction structure that we are currently working on

5.20.3 Variable Documentation

5.20.3.1 const double g

gravitational acceleration constant

5.20.3.2 const double H0

Minimum water height threshold that is maintained in all elements. Nodes whose water height is below this threshold are considered dry.

5.21 time_evolution.c File Reference

Functions

- void oneTimeStep (struct channel *Chan, double time, double dt, int channelNumber)
- void oneTimeStep2D (struct junction *junc, double time, double dt)
- void time_evolution (double FinalTime)

5.21.1 Detailed Description

This file contains code to advance the solutions one step forward in time

5.21.2 Function Documentation

5.21.2.1 void oneTimeStep (struct channel * *Chan*, double *time*, double *dt*, int *channelNumber*)

Function to advance a channel structure one step forward in time using 2-stage RKDG scheme

Parameters

in	*Chan*	channel structure that we are currently working on
in	*time*	current time of the simulation
in	*dt*	size of the time step
in	*channelNumber*	identity (number) of the channel structure that we are currently working on

5.21.2.2 void oneTimeStep2D (struct junction * *junc*, double *time*, double *dt*)

Function to advance a junction structure one step forward in time using 2-stage RKDG scheme

Parameters

in	*junc*	the junction tructure that we are currently working on
in	*time*	current time of the simulation
in	*dt*	size of the time step

5.21.2.3 void time_evolution (double *FinalTime*)

Function to evolve the channels as well as the junctions through time until the final simulation time. It also outputs the data at a certain time

5.22 wetDry.c File Reference

Functions

- int BigTheta (double a)
- void wetDryStatus1D (struct channel *Chan)
- void PDop1D (struct channel *Chan)
- void wetDryStatus2D (struct junction *junc)
- void PDop2D (struct junction *junc)

5.22.1 Detailed Description

This file contains all the functions associated with the wetting and drying treatment.

5.22.2 Function Documentation

5.22.2.1 int BigTheta (double *a*)

A function that checks to see if a value is positive.

Parameters

a	a double value to be tested

Returns

1 if positive, 0 if negative

5.22.2.2 void PDop1D (struct channel * *Chan*)

This function does some post processing to ensure that the water depth remains positive in the 1-D channels

Parameters

in	*Chan*	pointer to the channel structure that we are currently working on

5.22.2.3 void PDop2D (struct **junction** ∗ *junc*)

This function does some post processing to ensure that the water depth remains positive in the 2-D junctions

Parameters

in	*junc*	pointer to the junction structure that we are currently working on

5.22.2.4 void wetDryStatus1D (struct **channel** ∗ *Chan*)

A function that assigns a wet or dry status to each element of a channel and stores it in channel::WD. 0 represents a dry element and 1 represents a wet element.

Parameters

in	*Chan*	pointer to the channel structure that we are currently working on

5.22.2.5 void wetDryStatus2D (struct **junction** ∗ *junc*)

A function that assigns a wet or dry status to each element of a junction and stores it in junction::WD. 0 represents a dry element and 1 represents a wet element.

Parameters

in	*junc*	pointer to the the junction structure that we are currently working on

Index