Documentation of SWASHES

v1.04.00 (2022-09-02)

Generated by Doxygen 1.8.13
on Fri Sep 2 2022 10:23:27

Chapter 1

Todo List

Member Choice_solution::Choice_solution (Parameters &)

Exceptions should be treated.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

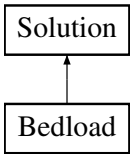
Class Documentation

5.1 Bedload Class Reference

Computes solutions with bedload.

`#include <bedload.hpp>`

Inheritance diagram for Bedload:



Public Member Functions

- Bedload (Parameters &)

  Constructor.
- virtual ~Bedload ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- void paramwarning () const

  Writes a warning about the the solution.

Additional Inherited Members

5.1.1 Detailed Description

Computes solutions with bedload.

Class that computes the solutions where the bed is moving with bedload, see Berthon et al. [2012].

Definition at line 70 of file bedload.hpp.

5.1.2 Constructor & Destructor Documentation

Bedload()

```
Bedload::Bedload (
            Parameters & par )  [explicit]
```

Constructor.
Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Warning

Problem: allocation of z0 failed

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex.

Note

If the vector z0 cannot be allocated, the code will exit with failure termination code.

Definition at line 59 of file bedload.cpp.

~Bedload()

```
Bedload::~Bedload ( )  [virtual]
```

Destructor.
Definition at line 151 of file bedload.cpp.

5.1.3 Member Function Documentation

compute()

```
void Bedload::compute ( )  [override], [virtual]
```

Computes the solution.
Computes the chosen bedload solution, see Berthon et al. [2012].

Modifies

Solution::hex, Solution::uex, Solution::zex.

Implements Solution.
Definition at line 156 of file bedload.cpp.

param()

```
void Bedload::param (
            SCALAR L,
            SCALAR dx_ex,
            SCALAR T,
            SCALAR uexl,
            SCALAR hexl,
```

```
SCALAR z0l,
SCALAR zexl,
SCALAR uexr,
SCALAR hexr,
SCALAR z0r,
SCALAR zexr,
SCALAR alpha,
SCALAR beta,
SCALAR A,
SCALAR q,
SCALAR C,
SCALAR p ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step
in	*T*	final time
in	*uexl*	value of the velocity on the left boundary
in	*hexl*	value of the water height on the left boundary
in	*z0l*	value of the inital topography on the left boundary
in	*zexl*	value of the final topography on the left boundary
in	*uexr*	value of the velocity on the right boundary
in	*hexr*	value of the water height on the right boundary
in	*z0r*	value of the inital topography on the right boundary
in	*zexr*	value of the final topography on the right boundary
in	*alpha*	parameter for Exner equation
in	*beta*	parameter for Exner equation
in	*A*	parameter for Exner equation
in	*q*	parameter for Exner equation
in	*C*	parameter for Exner equation
in	*p*	parameter for Exner equation

Definition at line 176 of file bedload.cpp.

paramwarning()

```
void Bedload::paramwarning ( ) const
```

Writes a warning about the the solution.

Warning

WARNING: to compare your numerical result to this solution, you must be able to remove friction from the Shallow-Water part (see doc).

Definition at line 214 of file bedload.cpp.

The documentation for this class was generated from the following files:

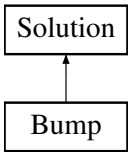
- Headers/bedload.hpp
- Sources/bedload.cpp

5.2 Bump Class Reference

Computes bump solutions.

`#include <bump.hpp>`

Inheritance diagram for Bump:



Public Member Functions

- Bump (Parameters &)

  Constructor.
- virtual ∼Bump ()

  Destructor.
- void compute () override

  Computes the solution.
- SCALAR p (SCALAR, SCALAR, SCALAR) const

  Coefficient p for Cardano method.
- SCALAR q (SCALAR, SCALAR, SCALAR, SCALAR) const

  Coefficient q for Cardano method.
- SCALAR determinant (SCALAR, SCALAR) const

  Determinant for Cardano method.
- SCALAR height (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Computation of the 3rd order polynomia roots.
- void abcd (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR &, SCALAR &, SCALAR &, SCALAR &)

  Defines a, b, c, d in order to solve $ah^3 + bh^2 + ch + d$.
- SCALAR RHJump (SCALAR, SCALAR, SCALAR) const

  Steady state RH relation.
- void param (SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.2.1 Detailed Description

Computes bump solutions.

Class that computes the solutions with a bump for the topography, see Delestre et al. [2013] and Goutal and Maurel [1997].

Definition at line 71 of file bump.hpp.

5.2.2 Constructor & Destructor Documentation

Bump()

```
Bump::Bump (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.
The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::zex to have the bump configuration.

Definition at line 60 of file bump.cpp.

~Bump()

```
Bump::~Bump ( )  [virtual]
```

Destructor.

Definition at line 171 of file bump.cpp.

5.2.3 Member Function Documentation

abcd()

```
void Bump::abcd (
            SCALAR q_in,
            SCALAR h_out,
            SCALAR zbx,
            SCALAR zbfin,
            SCALAR & a,
            SCALAR & b,
            SCALAR & c,
            SCALAR & d )
```

Defines a, b, c, d in order to solve $ah^3 + bh^2 + ch + d$.

Enters the coefficients of the 3rd order polynomia we want to solve: $ah^3 + bh^2 + ch + d$.

Parameters

in	*q_in*	inflow discharge
in	*h_out*	water height at the outflow
in	*zbx*	bottom topography of the current cell
in	*zbfin*	bottom topography at the outflow
out	*a*	coefficient of the 3rd order polynomia
out	*b*	coefficient of the 3rd order polynomia
out	*c*	coefficient of the 3rd order polynomia
out	*d*	coefficient of the 3rd order polynomia

Definition at line 368 of file bump.cpp.

compute()

```
void Bump::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the chosen bump solution, see Delestre et al. [2013] and Goutal and Maurel [1997].

Modifies

Solution::hex.

Implements Solution.

Definition at line 174 of file bump.cpp.

determinant()

```
SCALAR Bump::determinant (
            SCALAR p,
            SCALAR q ) const
```

Determinant for Cardano method.

Determinant in the Cardano method/related to number of roots.

Parameters

in	*p*	computed by Bump::p
in	*q*	computed by Bump::q

Returns

Value of $q^2 + \frac{4}{27}p^3$.

Definition at line 294 of file bump.cpp.

height()

```
SCALAR Bump::height (
            SCALAR p,
            SCALAR q,
            SCALAR a,
            SCALAR b,
            SCALAR hnear ) const
```

Computation of the 3rd order polynomia roots.

Parameters

in	*p*	computed by Bump::p
in	*q*	computed by Bump::q
in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*hnear*	height of the previous or following cell (depending on the height computation direction)

Warning

Error: no positive height.
Error: Probably irregular solution.

Returns

h, the water height.

Definition at line 308 of file bump.cpp.

p()

```
SCALAR Bump::p (
            SCALAR a,
            SCALAR b,
            SCALAR c ) const
```

Coefficient p for Cardano method.

Parameters

in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*c*	coefficient of the 3rd order polynomia

Returns

Value of $-\frac{b^2}{3a^2}+\frac{c}{a}$.

Definition at line 265 of file bump.cpp.

param()

```
void Bump::param (
            SCALAR L,
            SCALAR dx_ex ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step

Definition at line 404 of file bump.cpp.

q()

```
SCALAR Bump::q (
            SCALAR a,
            SCALAR b,
            SCALAR c,
            SCALAR d ) const
```

Coefficient q for Cardano method.

Parameters

in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*c*	coefficient of the 3rd order polynomia
in	*d*	coefficient of the 3rd order polynomia

Returns

Value of $\frac{b}{27a}\left(\frac{2b^2}{a^2}-9\frac{c}{a}\right)$.

Definition at line 278 of file bump.cpp.

RHJump()

```
SCALAR Bump::RHJump (
            SCALAR hplus,
            SCALAR hminus,
            SCALAR q ) const
```

Steady state RH relation.

Parameters

in	*hplus*	water height on the right side
in	*hminus*	water height on the left side
in	*q*	discharge

Returns

Value of $\left|q^2\left(\frac{1}{hplus}-\frac{1}{hminus}\right)+\frac{g}{2}\left(hplus^2-hminus^2\right)\right|$.

Definition at line 390 of file bump.cpp.
The documentation for this class was generated from the following files:

- Headers/bump.hpp
- Sources/bump.cpp

5.3 Choice_solution Class Reference

Choice of the solution.

```
#include <choice_solution.hpp>
```

Public Member Functions

- Choice_solution (Parameters &)

  Constructor.
- void compute ()

  Computes the solution.
- virtual ~Choice_solution ()

  Destructor.

5.3.1 Detailed Description

Choice of the solution.

Class that calls the chosen solution.

Definition at line 147 of file choice_solution.hpp.

5.3.2 Constructor & Destructor Documentation

Choice_solution()

```
Choice_solution::Choice_solution (
            Parameters & par )  [explicit]
```

Constructor.

Parameters

in	*par*	contains all the values from the parameter

Warning

Error: the dimension is ***
This *** solution for L=*** does not exist!
*** solutions for the domain *** do not exist!

Note

If the solution does not exists, the code will exit with failure termination code.

Todo Exceptions should be treated.

Definition at line 61 of file choice_solution.cpp.

~Choice_solution()

```
Choice_solution::~Choice_solution ( )  [virtual]
```

Destructor.

Definition at line 779 of file choice_solution.cpp.

5.3.3 Member Function Documentation

compute()

```
void Choice_solution::compute ( )
```

Computes the solution.

Definition at line 775 of file choice_solution.cpp.

The documentation for this class was generated from the following files:

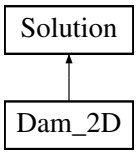
- Headers/choice_solution.hpp
- Sources/choice_solution.cpp

5.4 Dam_2D Class Reference

Computes Static dam solutions in 2D.

`#include <dam_2d.hpp>`

Inheritance diagram for Dam_2D:



Public Member Functions

- Dam_2D (Parameters &)

  Constructor.
- virtual ~Dam_2D ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- SCALAR norm (SCALAR, SCALAR)

  Computes the norm of a vector.
- SCALAR ring (SCALAR, SCALAR, SCALAR, SCALAR)

  Computes the topography of the center ring for the second domain.
- SCALAR cross (SCALAR, SCALAR, SCALAR, SCALAR)

  Computes the topography of the cross for the second domain.

Additional Inherited Members

5.4.1 Detailed Description

Computes Static dam solutions in 2D.

Class that computes the solutions with a dam in 2d, see Delestre et al. [2013].

Definition at line 68 of file dam_2d.hpp.

5.4.2 Constructor & Destructor Documentation

Dam_2D()

```
Dam_2D::Dam_2D (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::l, Solution::xex, Solution::yex, Dam_2D::zex2D, Dam_2D::uex2D, Dam_2D::vex2D to have Dam_2D configuration.

Definition at line 57 of file dam_2d.cpp.

~Dam_2D()

```
Dam_2D::~Dam_2D ( )  [virtual]
```

Destructor.

Definition at line 160 of file dam_2d.cpp.

5.4.3 Member Function Documentation

compute()

```
void Dam_2D::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the chosen Dam_2D solution.

Modifies

Dam_2D::hex2D.

Implements Solution.

Definition at line 174 of file dam_2d.cpp.

cross()

```
SCALAR Dam_2D::cross (
            SCALAR x,
            SCALAR y,
            SCALAR alpha,
            SCALAR beta )
```

Computes the topography of the cross for the second domain.

computes the height of the cross shaped dam at the point (x,y) with a topography of the form z(x,y) = min(dam_h, max(0, exp((g(x,y) ^2)/alpha - beta))

Parameters

in	*x*	first coordinate of the point
in	*y*	second coordinate of the point
in	*alpha*	parameter of the dam shape
in	*beta*	parameter of the dam shape

Definition at line 266 of file dam_2d.cpp.

norm()

```
SCALAR Dam_2D::norm (
            SCALAR x,
```

```
            SCALAR y )
```

Computes the norm of a vector.

computes the norm of the point (x,y)

Parameters

in	*x*	first coordinate of the point
in	*y*	second coordinate of the point

Definition at line 242 of file dam_2d.cpp.

param()

```
void Dam_2D::param (
            SCALAR L,
            SCALAR l,
            SCALAR dam_d,
            SCALAR dam_h,
            SCALAR dam_w,
            SCALAR dx_ex,
            SCALAR dy_ex ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain in x
in	*l*	length of the domain in y
in	*dam_h*	height of the dam
in	*dam_d*	distance of the center of the dam to the upstream border
in	*dam_w*	width of the flat section at the top of the dam
in	*dx_ex*	space step in x
in	*dy_ex*	space step in y

Definition at line 214 of file dam_2d.cpp.

ring()

```
SCALAR Dam_2D::ring (
            SCALAR x,
            SCALAR y,
            SCALAR alpha,
            SCALAR beta )
```

Computes the topography of the center ring for the second domain.

computes the height of the center ring at the point (x,y) with a topography of the form z(x,y) = min(dam_h, max(0, exp((g(x,y) ^2)/alpha - beta))

Parameters

in	*x*	first coordinate of the point

Parameters

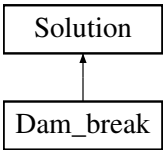
in	*y*	second coordinate of the point
in	*alpha*	parameter of the dam shape
in	*beta*	parameter of the dam shape

Definition at line 253 of file dam_2d.cpp.
The documentation for this class was generated from the following files:

- Headers/dam_2d.hpp
- Sources/dam_2d.cpp

5.5 Dam_break Class Reference

Computes dam break solutions.
`#include <dam_break.hpp>`
Inheritance diagram for Dam_break:



Public Member Functions

- Dam_break (Parameters &)

  Constructor.
- virtual ~Dam_break ()

  Destructor.
- void compute () override

  Computes the solution.
- SCALAR function (SCALAR, SCALAR, SCALAR) const

  Function $x^6 - 9v_{right}^2 x^4 + 16v_{left}\, v_{right}^2 x^3 - v_{right}^2 (v_{right}^2 + 8v_{left}^2)x^2 + v_{right}^6$ *to get the roots by dichotomy.*
- void param (SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.5.1 Detailed Description

Computes dam break solutions.
Class that computes the solutions for a dam break without friction, see Ritter [1892] Stoker [1957].
Definition at line 69 of file dam_break.hpp.

5.5.2 Constructor & Destructor Documentation

Dam_break()

```
Dam_break::Dam_break (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex to have the dam break configuration.

Definition at line 58 of file dam_break.cpp.

~Dam_break()

```
Dam_break::~Dam_break ( )  [virtual]
```

Destructor.

Definition at line 115 of file dam_break.cpp.

5.5.3 Member Function Documentation

compute()

```
void Dam_break::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the chosen dam break solution, see Ritter [1892] Stoker [1957].

Modifies

Solution::hex.

Implements Solution.

Definition at line 120 of file dam_break.cpp.

function()

```
SCALAR Dam_break::function (
            SCALAR x,
            SCALAR v_left,
            SCALAR v_right ) const
```

Function $x^6 - 9v_{right}^2 x^4 + 16v_{left}\, v_{right}^2 x^3 - v_{right}^2 (v_{right}^2 + 8v_{left}^2)x^2 + v_{right}^6$ to get the roots by dichotomy.

Function to solve by dichotomy the equation $cm^6 - 9v_{right}^2 cm^4 + 16v_{left}\, v_{right}^2 cm^3 - v_{right}^2 (v_{right}^2 + 8v_{left}^2)cm^2 + v_{right}^6 = 0$.

Returns

Value of $x^6 - 9v_{right}^2 x^4 + 16v_{left}\, v_{right}^2 x^3 - v_{right}^2 (v_{right}^2 + 8v_{left}^2)x^2 + v_{right}^6$.

Definition at line 195 of file dam_break.cpp.

param()

```
void Dam_break::param (
            SCALAR L,
            SCALAR xdam,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*xdam*	position of the dam
in	*dx_ex*	space step
in	*T*	final time

Definition at line 207 of file dam_break.cpp.

The documentation for this class was generated from the following files:

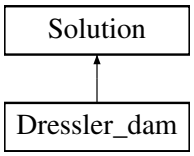
- Headers/dam_break.hpp
- Sources/dam_break.cpp

5.6 Dressler_dam Class Reference

Computes Dressler dam break solution.

`#include <dressler_dam.hpp>`

Inheritance diagram for Dressler_dam:



Public Member Functions

- Dressler_dam (Parameters &)

  Constructor.
- virtual ~Dressler_dam ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.6.1 Detailed Description

Computes Dressler dam break solution.

Class that computes the solutions for a dam break with friction, see Dressler [1952].

Definition at line 70 of file dressler_dam.hpp.

5.6.2 Constructor & Destructor Documentation

Dressler_dam()

```
Dressler_dam::Dressler_dam (
            Parameters & par )  [explicit]
```

Constructor.
Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Warning

Problem: allocation of hexd failed.

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::zex to have Dressler dam break configuration.

Note

If the vector hexd cannot be allocated, the code will exit with failure termination code.

Definition at line 60 of file dressler_dam.cpp.

~Dressler_dam()

```
Dressler_dam::~Dressler_dam ( )  [virtual]
```

Destructor.
Definition at line 115 of file dressler_dam.cpp.

5.6.3 Member Function Documentation

compute()

```
void Dressler_dam::compute ( )  [override], [virtual]
```

Computes the solution.
Computes Dressler solution, see Dressler [1952].

Modifies

Solution::hex, Solution::uex.

Implements Solution.
Definition at line 119 of file dressler_dam.cpp.

param()

```
void Dressler_dam::param (
            SCALAR L,
            SCALAR xdam,
            SCALAR C,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*xdam*	position of the dam
in	*C*	Chezy friction coefficient
in	*dx_ex*	space step
in	*T*	final time

Definition at line 220 of file dressler_dam.cpp.
The documentation for this class was generated from the following files:

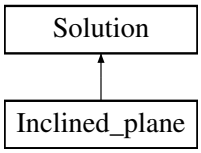
- Headers/dressler_dam.hpp
- Sources/dressler_dam.cpp

5.7 Inclined_plane Class Reference

Computes the solutions over an inclined plane.

`#include <inclined_plane.hpp>`

Inheritance diagram for Inclined_plane:



Public Member Functions

- Inclined_plane (Parameters &)
  Constructor.
- virtual ~Inclined_plane ()
  Destructor.
- void compute () override
  Computes the solution.
- SCALAR p (SCALAR, SCALAR, SCALAR) const
  Coefficient p for Cardano method.
- SCALAR q (SCALAR, SCALAR, SCALAR, SCALAR) const
  Coefficient q for Cardano method.
- SCALAR determinant (SCALAR, SCALAR) const
  Determinant for Cardano method.
- SCALAR height (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const
  Computation of the 3rd order polynomia roots.

- void abcd (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR &, SCALAR &, SCALAR &, SCALAR &)

  Defines a, b, c, d in order to solve $ah^3 + bh^2 + ch + d$.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.7.1 Detailed Description

Computes the solutions over an inclined plane.

Class that computes the solutions over an inclined plane, see Delestre et al. [2012].

Definition at line 71 of file inclined_plane.hpp.

5.7.2 Constructor & Destructor Documentation

Inclined_plane()

```
Inclined_plane::Inclined_plane (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.

The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::zex to have the inclined plane configuration.

Definition at line 60 of file inclined_plane.cpp.

~Inclined_plane()

```
Inclined_plane::~Inclined_plane ( )  [virtual]
```

Destructor.

Definition at line 100 of file inclined_plane.cpp.

5.7.3 Member Function Documentation

abcd()

```
void Inclined_plane::abcd (
            SCALAR q_in,
            SCALAR h_in,
            SCALAR alpha,
            SCALAR x,
            SCALAR & a,
            SCALAR & b,
```

```
            SCALAR & c,
            SCALAR & d )
```

Defines a, b, c, d in order to solve $ah^3 + bh^2 + ch + d$.
Enters the coefficients of the 3rd order polynomia we want to solve: $ah^3 + bh^2 + ch + d$.

Parameters

in	*q_in*	inflow discharge
in	*h_in*	water height at the inflow
in	*alpha*	the slope
in	*x*	the position
out	*a*	coefficient of the 3rd order polynomia
out	*b*	coefficient of the 3rd order polynomia
out	*c*	coefficient of the 3rd order polynomia
out	*d*	coefficient of the 3rd order polynomia

Definition at line 234 of file inclined_plane.cpp.

compute()

```
void Inclined_plane::compute ( )  [override], [virtual]
```

Computes the solution.
Computes the solution on an inclined plane, see Delestre et al. [2012].

Modifies

Solution::hex.

Implements Solution.
Definition at line 103 of file inclined_plane.cpp.

determinant()

```
SCALAR Inclined_plane::determinant (
            SCALAR p,
            SCALAR q ) const
```

Determinant for Cardano method.
Determinant in the Cardano method/related to number of roots.

Parameters

in	*p*	computed by Inclined_plane::p
in	*q*	computed by Inclined_plane::q

Returns

Value of $q^2 + \frac{4}{27}p^3$.

Definition at line 160 of file inclined_plane.cpp.

height()

```
SCALAR Inclined_plane::height (
```

```
            SCALAR p,
            SCALAR q,
            SCALAR a,
            SCALAR b,
            SCALAR hnear ) const
```

Computation of the 3rd order polynomia roots.

Parameters

in	*p*	computed by Inclined_plane::p
in	*q*	computed by Inclined_plane::q
in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*hnear*	height of the previous or following cell (depending on the height computation direction)

Warning

Error: no positive height.
Error: Probably irregular solution.

Returns

h, the water height.

Definition at line 174 of file inclined_plane.cpp.

p()

```
SCALAR Inclined_plane::p (
            SCALAR a,
            SCALAR b,
            SCALAR c ) const
```

Coefficient p for Cardano method.

Parameters

in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*c*	coefficient of the 3rd order polynomia

Returns

Value of $-\frac{b^2}{3a^2}+\frac{c}{a}$.

Definition at line 131 of file inclined_plane.cpp.

param()

```
void Inclined_plane::param (
            SCALAR L,
            SCALAR dx_ex,
            SCALAR alpha,
```

```
            SCALAR beta,
            SCALAR h0,
            SCALAR q0 ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step
in	*alpha*	the slope of the plane
in	*beta*	the value of the topography for x=0
in	*h0*	the left (imposed) water height
in	*q0*	the left (imposed) water discharge

Definition at line 260 of file inclined_plane.cpp.

q()

```
SCALAR Inclined_plane::q (
            SCALAR a,
            SCALAR b,
            SCALAR c,
            SCALAR d ) const
```

Coefficient q for Cardano method.

Parameters

in	*a*	coefficient of the 3rd order polynomia
in	*b*	coefficient of the 3rd order polynomia
in	*c*	coefficient of the 3rd order polynomia
in	*d*	coefficient of the 3rd order polynomia

Returns

Value of $\frac{b}{27a}\left(\frac{2b^2}{a^2} - 9\frac{c}{a}\right)$.

Definition at line 144 of file inclined_plane.cpp.

The documentation for this class was generated from the following files:

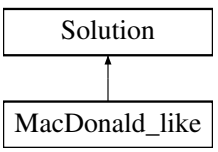
- Headers/inclined_plane.hpp
- Sources/inclined_plane.cpp

5.8 MacDonald_like Class Reference

Computes Mac Donald solutions.

`#include <macdonald_like.hpp>`

Inheritance diagram for MacDonald_like:



Public Member Functions

- MacDonald_like (Parameters &)

  Constructor.
- virtual ~MacDonald_like ()

  Destructor.
- void compute () override

  Computes the solution.
- SCALAR Delta_topo_Manning (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Evaluation of the slope variation for Manning friction law.
- SCALAR Delta_topo_Darcy_Weisbach (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Evaluation of the slope variation for Darcy-Weisbach friction law.
- void param (SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.8.1 Detailed Description

Computes Mac Donald solutions.

Class that computes Mac Donald solutions in 1d, see MacDonald [1996], MacDonald et al. [1997], Delestre et al. [2013] and Vo T. N. [2008].

Definition at line 73 of file macdonald_like.hpp.

5.8.2 Constructor & Destructor Documentation

MacDonald_like()

```
MacDonald_like::MacDonald_like (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.
The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Warning

Problem: allocation of dhex failed.

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::hex, Solution::qex to have Mac Donald configuration.

Note

If the vector dhex cannot be allocated, the code will exit with failure termination code.

Definition at line 59 of file macdonald_like.cpp.

~MacDonald_like()

```
MacDonald_like::~MacDonald_like ( )  [virtual]
```

Destructor.

Definition at line 436 of file macdonald_like.cpp.

5.8.3 Member Function Documentation

compute()

```
void MacDonald_like::compute ( )  [override], [virtual]
```

Computes the solution.

Computes Mac Donald solutions, see MacDonald [1996], MacDonald et al. [1997], Delestre et al. [2013] and Vo T. N. [2008].

Modifies

Solution::zex.

Implements Solution.

Definition at line 441 of file macdonald_like.cpp.

Delta_topo_Darcy_Weisbach()

```
SCALAR MacDonald_like::Delta_topo_Darcy_Weisbach (
            SCALAR q,
            SCALAR h,
            SCALAR dh,
            SCALAR Rain,
            SCALAR c ) const
```

Evaluation of the slope variation for Darcy-Weisbach friction law.

Parameters

in	*q*	discharge
in	*h*	water height
in	*dh*	variation of the water height
in	*Rain*	rain quantity
in	*c*	friction coefficient

Returns

Value of $\left(1-\frac{q^2}{gh^3}\right)dh+2Rain\frac{q}{gh^2}+c\frac{q^2}{8gh^3}$.

Definition at line 498 of file macdonald_like.cpp.

Delta_topo_Manning()

```
SCALAR MacDonald_like::Delta_topo_Manning (
            SCALAR q,
            SCALAR h,
            SCALAR dh,
```

```
            SCALAR Rain,
            SCALAR c ) const
```

Evaluation of the slope variation for Manning friction law.

Parameters

in	*q*	discharge
in	*h*	water height
in	*dh*	variation of the water height
in	*Rain*	rain quantity
in	*c*	friction coefficient

Returns

Value of $\left(1-\frac{q^2}{gh^3}\right)dh+2Rain\frac{q}{gh^2}+\frac{c^2q^2}{h^{10/3}}$.

Definition at line 483 of file macdonald_like.cpp.

param()

```
void MacDonald_like::param (
            SCALAR L,
            SCALAR dx_ex ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step

Definition at line 514 of file macdonald_like.cpp.
The documentation for this class was generated from the following files:

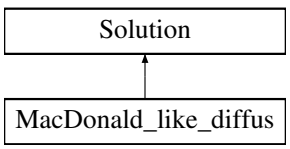
- Headers/macdonald_like.hpp
- Sources/macdonald_like.cpp

5.9 MacDonald_like_diffus Class Reference

Computes Mac Donald solutions with diffusion.

```
#include <macdonald_like_diffus.hpp>
```

Inheritance diagram for MacDonald_like_diffus:



Public Member Functions

- MacDonald_like_diffus (Parameters &)

  Constructor.

- virtual ~MacDonald_like_diffus ()

  Destructor.
- void compute () override

  Computes the solution.
- SCALAR Delta_topo_diffus (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Evaluation of the slope variation.
- void param (SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.9.1 Detailed Description

Computes Mac Donald solutions with diffusion.

Class that computes Mac Donald solutions in 1d with diffusion, see Delestre and Marche [2010].

Definition at line 70 of file macdonald_like_diffus.hpp.

5.9.2 Constructor & Destructor Documentation

MacDonald_like_diffus()

```
MacDonald_like_diffus::MacDonald_like_diffus (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.
The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Warning

Problem: allocation of dhex failed.
Problem: allocation of ddhex failed.

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::hex, Solution::qex to have Mac Donald configuration.

Note

If the vector dhex (or ddhex) cannot be allocated, the code will exit with failure termination code.

Definition at line 58 of file macdonald_like_diffus.cpp.

~MacDonald_like_diffus()

```
MacDonald_like_diffus::~MacDonald_like_diffus ( )  [virtual]
```

Destructor.

Definition at line 156 of file macdonald_like_diffus.cpp.

5.9.3 Member Function Documentation

compute()

```
void MacDonald_like_diffus::compute ( )  [override], [virtual]
```

Computes the solution.

Computes Mac Donald solutions with diffusion, see Delestre and Marche [2010].

Modifies

Solution::zex.

Implements Solution.

Definition at line 162 of file macdonald_like_diffus.cpp.

Delta_topo_diffus()

```
SCALAR MacDonald_like_diffus::Delta_topo_diffus (
            SCALAR q,
            SCALAR h,
            SCALAR dh,
            SCALAR ddh,
            SCALAR kt,
            SCALAR kl,
            SCALAR muv,
            SCALAR muh ) const
```

Evaluation of the slope variation.

Parameters

in	*q*	discharge
in	*h*	water height
in	*dh*	variation of the water height
in	*ddh*	second order derivative of h
in	*kt*	turbulent coefficient
in	*kl*	laminar coefficient
in	*muv*	vertical viscosity
in	*muh*	horizontal viscosity

Returns

Value of $\left(1-\frac{q^2}{gh^3}\right)dh+\frac{kl\,q}{gh^2(1+\frac{kl\,h}{3muv})}+\frac{kt\,q^2}{gh^2(1+\frac{kl\,h}{3muv})^2}+4muh\frac{q\,ddh-\frac{q\,dh^2}{h}}{gh^2}$.

Definition at line 197 of file macdonald_like_diffus.cpp.

param()

```
void MacDonald_like_diffus::param (
            SCALAR L,
            SCALAR dx_ex ) const
```

Writes the parameters of the solution.

Parameters

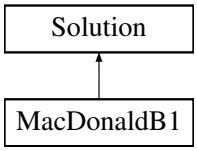
in	*L*	length of the domain
in	*dx_ex*	space step

Definition at line 215 of file macdonald_like_diffus.cpp.
The documentation for this class was generated from the following files:

- Headers/macdonald_like_diffus.hpp
- Sources/macdonald_like_diffus.cpp

5.10 MacDonaldB1 Class Reference

Computes Mac Donald pseudo 2d solutions.
`#include <macdonaldb1.hpp>`
Inheritance diagram for MacDonaldB1:



Public Member Functions

- MacDonaldB1 (Parameters &)

  Constructor.
- virtual ~MacDonaldB1 ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- SCALAR Delta_topo (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Evaluation of the slope variation.

Additional Inherited Members

5.10.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.
Class that computes Mac Donald pseudo 2d solutions with bottom B1, see MacDonald [1996].
Definition at line 69 of file macdonaldb1.hpp.

5.10.2 Constructor & Destructor Documentation

MacDonaldB1()

```
MacDonaldB1::MacDonaldB1 (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.
The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::hex to have Mac Donald configuration.

Definition at line 58 of file macdonaldb1.cpp.

~MacDonaldB1()

```
MacDonaldB1::~MacDonaldB1 ( ) [virtual]
```

Destructor.

Definition at line 229 of file macdonaldb1.cpp.

5.10.3 Member Function Documentation

compute()

```
void MacDonaldB1::compute ( ) [override], [virtual]
```

Computes the solution.

Computes Mac Donald solutions with bottom B1, see MacDonald [1996].

Modifies

Solution::zex.

Implements Solution.

Definition at line 164 of file macdonaldb1.cpp.

Delta_topo()

```
SCALAR MacDonaldB1::Delta_topo (
            SCALAR h,
            SCALAR hp,
            SCALAR b,
            SCALAR bp,
            SCALAR Q,
            SCALAR n,
            SCALAR Z,
            SCALAR exp1,
            SCALAR exp2 ) const
```

Evaluation of the slope variation.

Parameters

in	*h*	water height
in	*hp*	derivative of the water height
in	*b*	boundary function
in	*bp*	derivative of the boundary function
in	*Q*	discharge
in	*n*	friction coefficient
in	*Z*	slope
in	*exp1*	exponent, equal to 4/3
in	*exp2*	exponent, equal to 10/3

Returns

Value of $hp\left(\frac{Q^2(b+2Zh)}{g(h(b+Zh))^3}-1\right)-Q^2n^2\frac{(b+2h\sqrt{1+Z^2})^{exp1}}{(h(b+Zh))^{exp2}}+\frac{Q^2bp}{gh^2(b+Zh)^3}$.

Definition at line 188 of file macdonaldb1.cpp.

param()

```
void MacDonaldB1::param (
            SCALAR L,
            SCALAR dx_ex,
            SCALAR n ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step
in	*n*	friction coefficient

Definition at line 207 of file macdonaldb1.cpp.

The documentation for this class was generated from the following files:

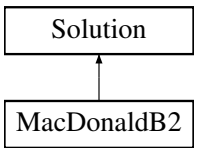
- Headers/macdonaldb1.hpp
- Sources/macdonaldb1.cpp

5.11 MacDonaldB2 Class Reference

Computes Mac Donald pseudo 2d solutions.

`#include <macdonaldb2.hpp>`

Inheritance diagram for MacDonaldB2:



Public Member Functions

- MacDonaldB2 (Parameters &)

  Constructor.
- virtual ~MacDonaldB2 ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- SCALAR Delta_topo (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Evaluation of the slope variation.

Additional Inherited Members

5.11.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.

Class that computes Mac Donald pseudo 2d solutions with bottom B2, see MacDonald [1996].

Definition at line 70 of file macdonaldb2.hpp.

5.11.2 Constructor & Destructor Documentation

MacDonaldB2()

```
MacDonaldB2::MacDonaldB2 (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.
The solution is saved at the steady state.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::hex to have Mac Donald configuration.

Definition at line 58 of file macdonaldb2.cpp.

~MacDonaldB2()

```
MacDonaldB2::~MacDonaldB2 ( )  [virtual]
```

Destructor.

Definition at line 199 of file macdonaldb2.cpp.

5.11.3 Member Function Documentation

compute()

```
void MacDonaldB2::compute ( )  [override], [virtual]
```

Computes the solution.

Computes Mac Donald solutions with bottom B2, see MacDonald [1996].

Modifies

Solution::zex.

Implements Solution.

Definition at line 134 of file macdonaldb2.cpp.

Delta_topo()

```
SCALAR MacDonaldB2::Delta_topo (
            SCALAR h,
            SCALAR hp,
            SCALAR b,
            SCALAR bp,
            SCALAR Q,
            SCALAR n,
            SCALAR Z,
            SCALAR exp1,
            SCALAR exp2 ) const
```

Evaluation of the slope variation.

Parameters

in	*h*	water height
in	*hp*	derivative of the water height
in	*b*	boundary function
in	*bp*	derivative of the boundary function
in	*Q*	discharge
in	*n*	friction coefficient
in	*Z*	slope
in	*exp1*	exponent, equal to 4/3
in	*exp2*	exponent, equal to 10/3

Returns

Value of $hp\left(\frac{Q^2(b+2Zh)}{g(h(b+Zh))^3}-1\right)-Q^2n^2\frac{(b+2h\sqrt{1+Z^2})^{exp1}}{(h(b+Zh))^{exp2}}+\frac{Q^2bp}{gh^2(b+Zh)^3}$.

Definition at line 180 of file macdonaldb2.cpp.

param()

```
void MacDonaldB2::param (
            SCALAR L,
            SCALAR dx_ex,
            SCALAR n ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step
in	*n*	friction coefficient

Definition at line 158 of file macdonaldb2.cpp.
The documentation for this class was generated from the following files:

- Headers/macdonaldb2.hpp
- Sources/macdonaldb2.cpp

5.12 Parameters Class Reference

Gets parameters.

`#include <parameters.hpp>`

Public Member Functions

- Parameters (int, char **)

  Constructor.
- virtual ~Parameters ()

  Destructor.
- void help () const

  Prints help.
- int get_nxex () const

  Gives the number of cells in x.
- int get_nyex () const

  Gives the number of cells in y.
- SCALAR get_choicedim () const

  Gives the dimension.
- int get_choicetype () const

  Gives the type.
- int get_choice () const

  Gives the chosen solution.
- int get_choicedomain () const

  Gives the domain.

Protected Attributes

- int nx_ex
- int ny_ex
- SCALAR choicedim
- int choicetype
- int choice
- int choicedomain

5.12.1 Detailed Description

Gets parameters.

Class that reads the parameters, checks their values and contains all the common declarations to get the values of the parameters.

Definition at line 69 of file parameters.hpp.

5.12.2 Constructor & Destructor Documentation

Parameters()

```
Parameters::Parameters (
            int argc,
            char ** argv )
```

Constructor.

Checks the arguments

Parameters

in	*argc*	number of arguments
in	*argv*	value of the arguments

Warning

The number of cells in x must be positive!
The number of cells in y must be positive!

Modifies

Parameters::choicedim, Parameters::choicetype, Parameters::choicedomain, Parameters::choice with the values given in argument.

Note

If the arguments are incompatible, the code will exit with failure termination code.

Definition at line 59 of file parameters.cpp.

~Parameters()

```
Parameters::~Parameters ( )  [virtual]
```

Destructor.

Definition at line 110 of file parameters.cpp.

5.12.3 Member Function Documentation

get_choice()

```
int Parameters::get_choice ( ) const
```

Gives the chosen solution.

Returns

The chosen solution.

Definition at line 240 of file parameters.cpp.

get_choicedim()

```
SCALAR Parameters::get_choicedim ( ) const
```

Gives the dimension.

Returns

The dimension of the solution.

Definition at line 250 of file parameters.cpp.

get_choicedomain()

```
int Parameters::get_choicedomain ( ) const
```

Gives the domain.

Returns

The domain of the solution.

Definition at line 270 of file parameters.cpp.

get_choicetype()

```
int Parameters::get_choicetype ( ) const
```

Gives the type.

Returns

The type of the solution.

Definition at line 260 of file parameters.cpp.

get_nxex()

```
int Parameters::get_nxex ( ) const
```

Gives the number of cells in x.

Returns

The number of cells in x.

Definition at line 220 of file parameters.cpp.

get_nyex()

```
int Parameters::get_nyex ( ) const
```

Gives the number of cells in y.

Returns

The number of cells in y.

Definition at line 230 of file parameters.cpp.

help()

```
void Parameters::help ( ) const
```

Prints help.

Prints how to use the code.

Definition at line 113 of file parameters.cpp.

5.12.4 Member Data Documentation

choice

```
int Parameters::choice  [protected]
```

Value corresponding to the chosen solution.

Definition at line 81 of file parameters.hpp.

choicedim

```
SCALAR Parameters::choicedim  [protected]
```

Value corresponding to the dimension of the solution.

Definition at line 77 of file parameters.hpp.

choicedomain

```
int Parameters::choicedomain  [protected]
```

Value corresponding to the domain of the solution.

Definition at line 83 of file parameters.hpp.

choicetype

```
int Parameters::choicetype  [protected]
```

Value corresponding to the type of the solution.

Definition at line 79 of file parameters.hpp.

nx_ex

```
int Parameters::nx_ex  [protected]
```

Number of cells in x.

Definition at line 73 of file parameters.hpp.

ny_ex

```
int Parameters::ny_ex  [protected]
```

Number of cells in y.

Definition at line 75 of file parameters.hpp.

The documentation for this class was generated from the following files:

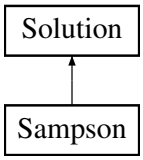
- Headers/parameters.hpp
- Sources/parameters.cpp

5.13 Sampson Class Reference

Computes Sampson solution.

`#include <sampson.hpp>`

Inheritance diagram for Sampson:



Public Member Functions

- Sampson (Parameters &)

  Constructor.
- virtual ~Sampson ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.13.1 Detailed Description

Computes Sampson solution.

Class that computes the solution for Sampson parabola with friction, see Sampson et al. [2006] Sampson et al. [2008].

Definition at line 70 of file sampson.hpp.

5.13.2 Constructor & Destructor Documentation

Sampson()

```
Sampson::Sampson (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex to have Sampson configuration.

Definition at line 58 of file sampson.cpp.

~Sampson()

```
Sampson::~Sampson ( )  [virtual]
```

Destructor.

Definition at line 93 of file sampson.cpp.

5.13.3 Member Function Documentation

compute()

```
void Sampson::compute ( )  [override], [virtual]
```

Computes the solution.

Computes Sampson solution, see Sampson et al. [2006] Sampson et al. [2008].

Modifies

Solution::hex, Solution::uex.

Implements Solution.

Definition at line 97 of file sampson.cpp.

param()

```
void Sampson::param (
            SCALAR L,
            SCALAR h0,
            SCALAR a,
            SCALAR B,
            SCALAR tau,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*h0*	value of the topography in the center of the domain
in	*a*	parameter of the topography
in	*B*	constant for the initial condition
in	*tau*	friction coefficient
in	*dx_ex*	space step
in	*T*	final time

Definition at line 124 of file sampson.cpp.

The documentation for this class was generated from the following files:

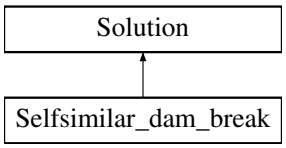
- Headers/sampson.hpp
- Sources/sampson.cpp

5.14 Selfsimilar_dam_break Class Reference

Computes self-similar dam break solutions.

`#include <selfsimilar_dam_break.hpp>`

Inheritance diagram for Selfsimilar_dam_break:



Public Member Functions

- Selfsimilar_dam_break (Parameters &)

  Constructor.
- virtual ~Selfsimilar_dam_break ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.14.1 Detailed Description

Computes self-similar dam break solutions.

Class that computes the self-similar solutions for dam break with friction, see Self-similar_solutions.pdf in the doc folder or in the bibliography of sourcesup.

Definition at line 71 of file selfsimilar_dam_break.hpp.

5.14.2 Constructor & Destructor Documentation

Selfsimilar_dam_break()

```
Selfsimilar_dam_break::Selfsimilar_dam_break (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex to have the self-similar dam break configuration.

Definition at line 59 of file selfsimilar_dam_break.cpp.

~Selfsimilar_dam_break()

```
Selfsimilar_dam_break::~Selfsimilar_dam_break ( )  [virtual]
```

Destructor.
Definition at line 138 of file selfsimilar_dam_break.cpp.

5.14.3 Member Function Documentation

compute()

```
void Selfsimilar_dam_break::compute ( )  [override], [virtual]
```

Computes the solution.
Computes the chosen self-similar dam break solution, see Self-similar_solutions.pdf in the doc folder or in the bibliography of sourcesup.

Modifies

Solution::hex.

Implements Solution.
Definition at line 143 of file selfsimilar_dam_break.cpp.

param()

```
void Selfsimilar_dam_break::param (
            SCALAR L,
            SCALAR xL,
            SCALAR xR,
            SCALAR hinit,
            SCALAR k1,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

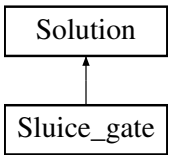
in	*L*	length of the domain
in	*xL*	left bound for the water
in	*xR*	right bound for the water
in	*hinit*	initial height of the fluid
in	*k1*	inverse of the friction coefficient in SW equations
in	*dx_ex*	space step
in	*T*	final time

Definition at line 181 of file selfsimilar_dam_break.cpp.
The documentation for this class was generated from the following files:

- Headers/selfsimilar_dam_break.hpp
- Sources/selfsimilar_dam_break.cpp

5.15 Sluice_gate Class Reference

Computes dam break with a sluice gate solutions.
`#include <sluice_gate.hpp>`
Inheritance diagram for Sluice_gate:



Public Member Functions

- Sluice_gate (Parameters &)

  Constructor.
- virtual ~Sluice_gate ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- SCALAR ff (SCALAR)

  Computes the value of the free flow function.
- SCALAR r1 (SCALAR, SCALAR, SCALAR)

  Computes the value of the rarefaction function from the left state.
- SCALAR spshock1 (SCALAR, SCALAR, SCALAR)

  Computes the speed of the shock wave in the first characteristic field.
- SCALAR spshock2 (SCALAR, SCALAR, SCALAR)

  Computes the speed of the shock wave in the second characteristic field.
- SCALAR s2 (SCALAR, SCALAR, SCALAR)

  Computes the value of the shock function.
- SCALAR dichotomie (int)

  Finds the intersection between two locus of admissible states to, the locus are chosen with the variable choice.

Additional Inherited Members

5.15.1 Detailed Description

Computes dam break with a sluice gate solutions.

Class that computes the solutions for a dam break with a sluice gate without friction, see Cozzolino et al. [2015].

Definition at line 68 of file sluice_gate.hpp.

5.15.2 Constructor & Destructor Documentation

Sluice_gate()

```
Sluice_gate::Sluice_gate (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex, Sluice_gate::h_left, Sluice_gate::h_↩ right, Sluice_gate::gate_size, Sluice_gate::solu, Sluice_gate::Cc, Sluice_gate::xdam to have the sluice gate opening configuration.

Definition at line 57 of file sluice_gate.cpp.

~Sluice_gate()

```
Sluice_gate::~Sluice_gate ( )  [virtual]
```

Destructor.
Definition at line 118 of file sluice_gate.cpp.

5.15.3 Member Function Documentation

compute()

```
void Sluice_gate::compute ( )  [override], [virtual]
```

Computes the solution.
Computes the chosen sluice gate solution.

Modifies

Solution::hex.

Implements Solution.
Definition at line 123 of file sluice_gate.cpp.

dichotomie()

```
SCALAR Sluice_gate::dichotomie (
            int choice )
```

Finds the intersection between two locus of admissible states to, the locus are chosen with the variable choice.
Finds the intersection between two locus of admissible states to, the locus are chosen with the variable choice

Parameters

in	*choice*	choice of which dichotomy to apply

Definition at line 383 of file sluice_gate.cpp.

ff()

```
SCALAR Sluice_gate::ff (
            SCALAR h )
```

Computes the value of the free flow function.
Computes the value of the free flow function

Parameters

in	*h*	water height, corresponds here to the where we compute the function

Definition at line 329 of file sluice_gate.cpp.

param()

```
void Sluice_gate::param (
            SCALAR L,
            SCALAR xdam,
            SCALAR gate_size,
            SCALAR h_left,
            SCALAR h_right,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*xdam*	position of the dam
in	*gate_size*	size of the opening of the sluice gate
in	*h_left*	height of the left side water
in	*h_right*	height of the right side water
in	*dx_ex*	space step
in	*T*	final time

Definition at line 303 of file sluice_gate.cpp.

r1()

```
SCALAR Sluice_gate::r1 (
            SCALAR h_r,
            SCALAR h_l,
            SCALAR u_l )
```

Computes the value of the rarefaction function from the left state.

Computes the value of the speed of the rarefaction function in the first characteristic field

Parameters

in	*h_r*	water height of the right side state
in	*h_l*	water height of the left side state, origin of the rarefacion wave
in	*u_l*	water speed of the left side state, origin of the rarefacion wave

Definition at line 339 of file sluice_gate.cpp.

s2()

```
SCALAR Sluice_gate::s2 (
            SCALAR h_l,
            SCALAR u_l,
            SCALAR h_r )
```

Computes the value of the shock function.

Computes the value of second characteristic field shock function

Parameters

in	h_l	water height of the left side state, origin of the rarefacion wave
in	u_l	water speed of the left side state, origin of the rarefacion wave
in	h_r	water height of the right side state

Definition at line 372 of file sluice_gate.cpp.

spshock1()

```
SCALAR Sluice_gate::spshock1 (
            SCALAR h_l,
            SCALAR u_l,
            SCALAR h_r )
```

Computes the speed of the shock wave in the first characteristic field.

Computes the speed of the shock wave in the first characteristic field

Parameters

in	h_l	water height of the left side state, origin of the rarefacion wave
in	u_l	water speed of the left side state, origin of the rarefacion wave
in	h_r	water height of the right side state

Definition at line 350 of file sluice_gate.cpp.

spshock2()

```
SCALAR Sluice_gate::spshock2 (
            SCALAR h_l,
```

```
SCALAR u_l,
SCALAR h_r )
```

Computes the speed of the shock wave in the second characteristic field.
Computes the speed of the shock wave in the second characteristic field

Parameters

in	h_l	water height of the left side state, origin of the rarefacion wave
in	u_l	water speed of the left side state, origin of the rarefacion wave
in	h_r	water height of the right side state

Definition at line 361 of file sluice_gate.cpp.

The documentation for this class was generated from the following files:

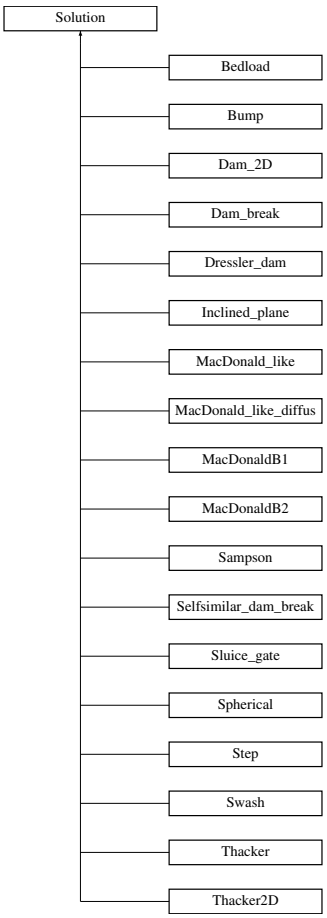
- Headers/sluice_gate.hpp
- Sources/sluice_gate.cpp

5.16 Solution Class Reference

Analytic solution.

```
#include <solution.hpp>
```

Inheritance diagram for Solution:



Public Member Functions

- Solution (Parameters &)

  Constructor.
- void allocation ()

  Allocations of the tables.
- void deallocation ()

  deallocation of the tables
- virtual void compute ()=0

  Function to be specified in case.
- void savefinalcritical (const SCALAR ∗, const SCALAR ∗, SCALAR ∗, const SCALAR ∗) const

  Saves the analytic solution at the final time with the critical height.
- void savefinalcriticalinit (const SCALAR ∗, const SCALAR ∗, SCALAR ∗, const SCALAR ∗, const SCALAR ∗) const

  Saves the analytic solution at the final time with the critical height and the initial topography.
- void savefinalmu (const SCALAR ∗, const SCALAR ∗, const SCALAR ∗) const

  Saves the analytic solution at the final time without u.
- void savefinal2D (const SCALAR ∗, const SCALAR ∗, TAB, TAB, TAB, TAB) const

  Saves the analytic solution at the final time in 2D.
- void savefinalSpherical (const SCALAR ∗, const SCALAR ∗, TAB, TAB, TAB, TAB) const

  Saves the analytic solution at the final time in a spherical geometry.
- void head (const Parameters &, const string &, const string &) const

  Writes the version of the software and the choice of the solution.
- virtual ∼Solution ()

  Destructor.

Protected Attributes

- const int NX_EX
- const int NY_EX
- SCALAR T
- SCALAR L
- SCALAR I
- SCALAR dx_ex
- SCALAR dy_ex
- SCALAR * xex
- SCALAR * yex
- SCALAR * hex
- SCALAR * uex
- SCALAR * qex
- SCALAR * zex

5.16.1 Detailed Description

Analytic solution.

Class that contains all the common declarations for the solutions.

Definition at line 69 of file solution.hpp.

5.16.2 Constructor & Destructor Documentation

Solution()

```
Solution::Solution (
            Parameters & par )
```

Constructor.

Parameters

in	*par*	contains all the values from the parameters file

Definition at line 59 of file solution.cpp.

~Solution()

```
Solution::~Solution ( )  [virtual]
```

Destructor.

Definition at line 223 of file solution.cpp.

5.16.3 Member Function Documentation

allocation()

```
void Solution::allocation ( )
```

Allocations of the tables.

Allocation of Solution::xex, Solution::yex, Solution::hex, Solution::uex, Solution::qex, Solution::zex.

Warning

Problem: allocation of xex failed.
Problem: allocation of yex failed.
Problem: allocation of hex failed.
Problem: allocation of uex failed.
Problem: allocation of qex failed.
Problem: allocation of zex failed.

Note

If a vector cannot be allocated, the code will exit with failure termination code.

Definition at line 228 of file solution.cpp.

compute()

```
virtual void Solution::compute ( )  [pure virtual]
```

Function to be specified in case.

Implemented in MacDonald_like, Bump, Inclined_plane, Selfsimilar_dam_break, Bedload, Dressler_dam, MacDonald_like_diffus, MacDonaldB2, Sampson, Dam_break, MacDonaldB1, Swash, Thacker, Thacker2D, Dam_2D, Sluice_gate, Spherical, and Step.

deallocation()

```
void Solution::deallocation ( )
```

deallocation of the tables

deallocation of Solution::xex, Solution::yex, Solution::hex, Solution::uex, Solution::qex, Solution::zex.

Definition at line 280 of file solution.cpp.

head()

```
void Solution::head (
            const Parameters & par,
            const string & solutiontype,
            const string & solutionchoice ) const
```

Writes the version of the software and the choice of the solution.

Parameters

in	*par*	parameter, contains all the values from the parameters file
in	*solutiontype*	name of the type of the solution
in	*solutionchoice*	name of the solution

Definition at line 203 of file solution.cpp.

savefinal2D()

```
void Solution::savefinal2D (
            const SCALAR * xex,
            const SCALAR * yex,
```

```
            TAB hex2D,
            TAB uex2D,
            TAB vex2D,
            TAB zex2D ) const
```

Saves the analytic solution at the final time in 2D.

Saves x and y (the position), h (the water height), u and v (the flow velocities in x and y), z+h (the free surface), z (the topography), U (the norm of the velocity), Fr (the Froude number), qx, qy and q (the flow discharge in x, y and its norm).

Parameters

in	*xex*	abscissae
in	*yex*	ordinates
in	*hex2D*	water height
in	*uex2D*	flow velocity in x
in	*vex2D*	flow velocity in y
in	*zex2D*	topography

Definition at line 143 of file solution.cpp.

savefinalcritical()

```
void Solution::savefinalcritical (
            const SCALAR * xex,
            const SCALAR * hex,
            SCALAR * uex,
            const SCALAR * zex ) const
```

Saves the analytic solution at the final time with the critical height.

Saves x (the position), h (the water height), u (the flow velocity), z (the topography), q (the flow discharge), z+h (the free surface), Fr (the Froude number) and z+hc (the critical surface).

Parameters

in	*xex*	abscissae
in	*hex*	water height
in	*uex*	flow velocity
in	*zex*	topography

Definition at line 77 of file solution.cpp.

savefinalcriticalinit()

```
void Solution::savefinalcriticalinit (
            const SCALAR * xex,
            const SCALAR * hex,
            SCALAR * uex,
            const SCALAR * zex,
            const SCALAR * z0 ) const
```

Saves the analytic solution at the final time with the critical height and the initial topography.

Saves x (the position), h (the water height), u (the flow velocity), z (the topography), q (the flow discharge), z+h (the free surface), Fr (the Froude number), z+hc (the critical surface), z0 (the initial topography) and z0+h (the initial surface).

Parameters

in	*xex*	abscissae
in	*hex*	water height
in	*uex*	flow velocity
in	*zex*	topography
in	*z0*	initial topography

Definition at line 101 of file solution.cpp.

savefinalmu()

```
void Solution::savefinalmu (
            const SCALAR * xex,
            const SCALAR * hex,
            const SCALAR * zex ) const
```

Saves the analytic solution at the final time without u.

Saves x (the position), h (the water height), z (the topography) and z+h (the free surface).

Parameters

in	*xex*	abscissae
in	*hex*	water height
in	*zex*	topography

Definition at line 126 of file solution.cpp.

savefinalSpherical()

```
void Solution::savefinalSpherical (
            const SCALAR * lambdaex,
            const SCALAR * thetaex,
            TAB hex2D,
            TAB uex2D,
            TAB vex2D,
            TAB rhoex ) const
```

Saves the analytic solution at the final time in a spherical geometry.

Saves x and y (the position), h (the water height), u and v (the flow velocities in x and y), z+h (the free surface), z (the topography), U (the norm of the velocity), Fr (the Froude number), qx, qy and q (the flow discharge in x, y and its norm).

Parameters

in	*lambdaex*	longitudinal angle
in	*thetaex*	latitudinal angle
in	*hex2D*	water height
in	*uex2D*	longitudinale flow velocity component
in	*vex2D*	latitudinale flow velocity component
in	*rhoex*	topography

Definition at line 173 of file solution.cpp.

5.16.4 Member Data Documentation

dx_ex

```
SCALAR Solution::dx_ex [protected]
```

Space step in x.

Definition at line 84 of file solution.hpp.

dy_ex

```
SCALAR Solution::dy_ex [protected]
```

Space step in y.

Definition at line 86 of file solution.hpp.

hex

```
SCALAR* Solution::hex [protected]
```

Array for the water height.

Definition at line 93 of file solution.hpp.

L

```
SCALAR Solution::L [protected]
```

Dimensions of the domain in x.

Definition at line 80 of file solution.hpp.

l

```
SCALAR Solution::l [protected]
```

Dimensions of the domain in y.

Definition at line 82 of file solution.hpp.

NX_EX

```
const int Solution::NX_EX [protected]
```

Number of cells in x.

Definition at line 73 of file solution.hpp.

NY_EX

```
const int Solution::NY_EX [protected]
```

Number of cells in y.

Definition at line 75 of file solution.hpp.

qex

```
SCALAR* Solution::qex [protected]
```

Array for the flow discharge.
Definition at line 97 of file solution.hpp.

T

```
SCALAR Solution::T [protected]
```

Final time.
Definition at line 78 of file solution.hpp.

uex

```
SCALAR* Solution::uex [protected]
```

Array for the flow velocity.
Definition at line 95 of file solution.hpp.

xex

```
SCALAR* Solution::xex [protected]
```

Array for the first coordinate.
Definition at line 89 of file solution.hpp.

yex

```
SCALAR* Solution::yex [protected]
```

Array for the second coordinate.
Definition at line 91 of file solution.hpp.

zex

```
SCALAR* Solution::zex [protected]
```

Array for the topography.
Definition at line 99 of file solution.hpp.
The documentation for this class was generated from the following files:

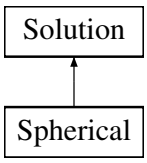
- Headers/solution.hpp
- Sources/solution.cpp

5.17 Spherical Class Reference

Computes Static solutions in spherical geometry.

```
#include <spherical.hpp>
```

Inheritance diagram for Spherical:



Public Member Functions

- Spherical (Parameters &)

  Constructor.
- virtual ~Spherical ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.17.1 Detailed Description

Computes Static solutions in spherical geometry.

Class that computes different static solutions in spherical geometry, see Williamson et al. [1992].

Definition at line 68 of file spherical.hpp.

5.17.2 Constructor & Destructor Documentation

Spherical()

```
Spherical::Spherical (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters.

Parameters

in	*par*	contains all the values from the parameters

Warning

Problem: allocation of lambdaex failed

Problem: allocation of thetaex failed

Modifies

Solution::dx_ex, Solution::dy_ex, Spherical::rhoex, Spherical::uex2D, Spherical::vex2D, Spherical::lambdaex, Spherical::thetaex, Spherical::alpha, Spherical::omega, Spherical::radius, to have a Spherical configuration and the domain paramters.

Definition at line 58 of file spherical.cpp.

~Spherical()

```
Spherical::~Spherical ( )  [virtual]
```

Destructor.

Definition at line 154 of file spherical.cpp.

5.17.3 Member Function Documentation

compute()

```
void Spherical::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the chosen Spherical solution.

Modifies

Spherical::hex2D.

Implements Solution.

Definition at line 170 of file spherical.cpp.

param()

```
void Spherical::param (
            SCALAR radius,
            SCALAR alpha,
            SCALAR omega,
            SCALAR dx_ex,
            SCALAR dy_ex ) const
```

Writes the parameters of the solution.

Parameters

in	*radius*	radius of the sphere
in	*omega*	pulsation of the rotation of the sphere
in	*alpha*	angle between the sphere's rotation axis and the polar axis
in	*dx_ex*	angle step in lambda
in	*dy_ex*	angle step in theta

Definition at line 199 of file spherical.cpp.

The documentation for this class was generated from the following files:

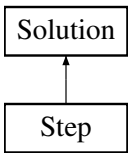
- Headers/spherical.hpp
- Sources/spherical.cpp

5.18 Step Class Reference

Computes dam break with a step solutions.

`#include <step.hpp>`

Inheritance diagram for Step:



Public Member Functions

- Step (Parameters &)

  Constructor.
- virtual ∼Step ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- SCALAR r1 (SCALAR, SCALAR, SCALAR)

  Computes the value of the rarefaction function from the left state.
- SCALAR spshock (SCALAR, SCALAR, SCALAR)

  Computes the speed of the shock wave in the second characteristic field.

Additional Inherited Members

5.18.1 Detailed Description

Computes dam break with a step solutions.

Class that computes the solutions for a dam break with a step without friction, see HAN and WARNECKE [2014].

Definition at line 68 of file step.hpp.

5.18.2 Constructor & Destructor Documentation

Step()

```
Step::Step (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex, Step::h_left, Step::h_right, Step::step_size, Step::solu, Step::Cc, Step::xdam to have the step opening configuration.

Definition at line 57 of file step.cpp.

~Step()

```
Step::~Step ( )  [virtual]
```

Destructor.

Definition at line 113 of file step.cpp.

5.18.3 Member Function Documentation

compute()

```
void Step::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the chosen step solution.

Modifies

Solution::hex.

Implements Solution.

Definition at line 118 of file step.cpp.

param()

```
void Step::param (
            SCALAR L,
            SCALAR xdam,
            SCALAR step_size,
            SCALAR h_left,
            SCALAR h_right,
            SCALAR u_left,
            SCALAR u_right,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*xdam*	position of the dam
in	*step_size*	size of the opening of the step
in	*h_left*	height of the left side water
in	*u_left*	water speed on the left of the step
in	*h_right*	height of the right side water
in	*u_right*	water speed on the right of the step
in	*dx_ex*	space step
in	*T*	final time

Definition at line 189 of file step.cpp.

r1()

```
SCALAR Step::r1 (
            SCALAR h_r,
            SCALAR h_l,
            SCALAR u_l )
```

Computes the value of the rarefaction function from the left state.
Computes the value of the speed of the rarefaction function in the first characteristic field

Parameters

in	h_r	water height of the right side state
in	h_l	water height of the left side state, origin of the rarefacion wave
in	u_l	water speed of the left side state, origin of the rarefacion wave

Definition at line 219 of file step.cpp.

spshock()

```
SCALAR Step::spshock (
            SCALAR h_l,
            SCALAR u_l,
            SCALAR h_r )
```

Computes the speed of the shock wave in the second characteristic field.
Computes the speed of the shock wave in the second characteristic field

Parameters

in	h_l	water height of the left side state, origin of the rarefacion wave
in	u_l	water speed of the left side state, origin of the rarefacion wave
in	h_r	water height of the right side state

Definition at line 230 of file step.cpp.
The documentation for this class was generated from the following files:

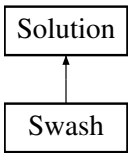
- Headers/step.hpp
- Sources/step.cpp

5.19 Swash Class Reference

Computes the solutions of the swash over an inclined plane.

```
#include <swash.hpp>
```

Inheritance diagram for Swash:



Public Member Functions

- Swash (Parameters &)

  Constructor.
- virtual ∼Swash ()

  Destructor.
- void compute () override

  Computes the solution.
- void ua_eta (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR ∗, int)

  Computes the non-dimensional speed ua and the non-dimensional free surface eta.
- void param (SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.
- void leftcondition (int, SCALAR, SCALAR) const

  Writes the left boundary condition (at each dt = 0.01s)
- SCALAR J (int, SCALAR)

  Computes the kth Bessel function for k=0,1 or 2.

Additional Inherited Members

5.19.1 Detailed Description

Computes the solutions of the swash over an inclined plane.

Class that computes the solutions of the swash over an inclined plane, see Marche [2005], Carrier and Greenspan [1958].

Definition at line 69 of file swash.hpp.

5.19.2 Constructor & Destructor Documentation

Swash()

```
Swash::Swash (
            Parameters & par )  [explicit]
```

Constructor.

Defines the physical parameters and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::xex, Solution::zex to have the inclined plane configuration.

Definition at line 58 of file swash.cpp.

~Swash()

```
Swash::~Swash ( )  [virtual]
```

Destructor.

Definition at line 140 of file swash.cpp.

5.19.3 Member Function Documentation

compute()

```
void Swash::compute ( )  [override], [virtual]
```

Computes the solution.

Computes the swash solutions of a wave on an inclined plane, see Marche [2005], Carrier and Greenspan [1958].

Implements Solution.

Definition at line 144 of file swash.cpp.

J()

```
SCALAR Swash::J (
            int k,
            SCALAR x )
```

Computes the kth Bessel function for k=0,1 or 2.

Computes the value of the kth Bessel function (k=0,1 or 2) at x.

Parameters

in	*k*	the number of the Bessel function (k=0,1 or 2)
in	*x*	the point to evaluate the Bessel function

Definition at line 329 of file swash.cpp.

leftcondition()

```
void Swash::leftcondition (
            int n,
            SCALAR hexl,
            SCALAR uexl ) const
```

Writes the left boundary condition (at each dt = 0.01s)

Saves the left boundary condition (h and q=hu) in the file transient_leftbc.txt or periodic_leftbc.txt.

Parameters

in	*n*	current time iteration
in	*hexl*	left value of the water height
in	*uexl*	left value of the velocity

Definition at line 302 of file swash.cpp.

param()

```
void Swash::param (
            SCALAR L,
            SCALAR dx_ex,
            SCALAR alpha,
            SCALAR e ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*dx_ex*	space step
in	*alpha*	the slope of the plane
in	*e*	the initial curvature of the wave

Definition at line 278 of file swash.cpp.

ua_eta()

```
void Swash::ua_eta (
            SCALAR x0,
            SCALAR e,
            SCALAR a,
            SCALAR ta,
            SCALAR xa,
            SCALAR * result,
            int sol )
```

Computes the non-dimensional speed ua and the non-dimensional free surface eta.
Computes the velocity and free surface (using Newton algorithm) at one point and one time.

Parameters

in	*x0*	abscissa changement
in	*e*	the initial curvature of the wave
in	*a*	the parameter a in the references
in	*ta*	non dimentional time
in	*xa*	non dimentional abscissa
in	*result*	non-dimensional velocity and free surface at (xa,ta)
in	*sol*	to choose the transient (sol=1) or periodic (sol=2) solution to compute

Definition at line 181 of file swash.cpp.
The documentation for this class was generated from the following files:

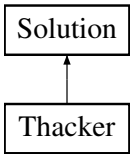
- Headers/swash.hpp
- Sources/swash.cpp

5.20 Thacker Class Reference

Computes Thacker solution.

`#include <thacker.hpp>`

Inheritance diagram for Thacker:



Public Member Functions

- Thacker (Parameters &)

  Constructor.
- virtual ∼Thacker ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.20.1 Detailed Description

Computes Thacker solution.

Class that computes the solution for Thacker parabola, see Thacker [1981].

Definition at line 69 of file thacker.hpp.

5.20.2 Constructor & Destructor Documentation

Thacker()

```
Thacker::Thacker (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::T, Solution::xex, Solution::zex to have Thacker configuration.

Definition at line 58 of file thacker.cpp.

∼Thacker()

```
Thacker::~Thacker ( ) [virtual]
```

Destructor.

Definition at line 91 of file thacker.cpp.

5.20.3 Member Function Documentation

compute()

```
void Thacker::compute ( )  [override], [virtual]
```

Computes the solution.

Computes Thacker solution, see Thacker [1981].

Modifies

Solution::hex, Solution::uex.

Implements Solution.

Definition at line 96 of file thacker.cpp.

param()

```
void Thacker::param (
            SCALAR L,
            SCALAR h0,
            SCALAR a,
            SCALAR dx_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain
in	*h0*	value of the topography in the center of the domain
in	*a*	parameter of the topography
in	*dx_ex*	space step
in	*T*	final time

Definition at line 125 of file thacker.cpp.

The documentation for this class was generated from the following files:

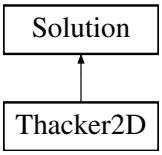
- Headers/thacker.hpp
- Sources/thacker.cpp

5.21 Thacker2D Class Reference

Computes Thacker solutions in 2D.

```
#include <thacker2d.hpp>
```

Inheritance diagram for Thacker2D:



Public Member Functions

- Thacker2D (Parameters &)

  Constructor.
- virtual ~Thacker2D ()

  Destructor.
- void compute () override

  Computes the solution.
- void param (SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR, SCALAR) const

  Writes the parameters of the solution.

Additional Inherited Members

5.21.1 Detailed Description

Computes Thacker solutions in 2D.

Class that computes the solutions for Thacker paraboloid, see Thacker [1981].

Definition at line 69 of file thacker2d.hpp.

5.21.2 Constructor & Destructor Documentation

Thacker2D()

```
Thacker2D::Thacker2D (
            Parameters & par ) [explicit]
```

Constructor.

Defines the physical parameters, the final time and prints the header with the configuration.

Parameters

in	*par*	contains all the values from the parameters

Modifies

Solution::dx_ex, Solution::L, Solution::l, Solution::T, Solution::xex, Solution::yex, Thacker2D::zex2D to have Thacker 2D configuration.

Definition at line 58 of file thacker2d.cpp.

~Thacker2D()

```
Thacker2D::~Thacker2D ( ) [virtual]
```

Destructor.

Definition at line 131 of file thacker2d.cpp.

5.21.3 Member Function Documentation

compute()

```
void Thacker2D::compute ( ) [override], [virtual]
```

Computes the solution.
Computes the chosen Thacker 2D solution, see Thacker [1981].

Modifies

Thacker2D::hex2D, Thacker2D::uex2D, Thacker2D::vex2D.

Implements Solution.
Definition at line 147 of file thacker2d.cpp.

param()

```
void Thacker2D::param (
            SCALAR L,
            SCALAR l,
            SCALAR h0,
            SCALAR a,
            SCALAR dx_ex,
            SCALAR dy_ex,
            SCALAR T ) const
```

Writes the parameters of the solution.

Parameters

in	*L*	length of the domain in x
in	*l*	length of the domain in y
in	*h0*	value of the topography in the center of the domain
in	*a*	parameter of the topography
in	*dx_ex*	space step in x
in	*dy_ex*	space step in y
in	*T*	final time

Definition at line 184 of file thacker2d.cpp.
The documentation for this class was generated from the following files:

- Headers/thacker2d.hpp
- Sources/thacker2d.cpp

Chapter 6

File Documentation

6.1 Headers/bedload.hpp File Reference

Computes solutions with bedload.

```
#include "solution.hpp"
```

Classes

- class Bedload

  Computes solutions with bedload.

Macros

- #define BEDLOAD_HPP

6.1.1 Detailed Description

Computes solutions with bedload.

Author

Minh Hoang Le lemhoang@math.cnrs.fr (2012)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: the bed is moving with bedload, see Berthon et al. [2012].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.1.2 Macro Definition Documentation

BEDLOAD_HPP

```
#define BEDLOAD_HPP
```

Definition at line 61 of file bedload.hpp.

6.2 Headers/bump.hpp File Reference

Computes bumps solutions.

```
#include "solution.hpp"
```

Classes

- class Bump

  Computes bump solutions.

6.2.1 Detailed Description

Computes bumps solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Anne-Celine Boulanger anne-celine.boulanger@inria.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: with a bump, see Delestre et al. [2013] and Goutal and Maurel [1997].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA - Universite Pierre et Marie Curie (France)

6.3 Headers/choice_solution.hpp File Reference

Choice of the solution.

```
#include "solution.hpp"
#include "dam_break.hpp"
#include "selfsimilar_dam_break.hpp"
#include "dressler_dam.hpp"
#include "sluice_gate.hpp"
#include "step.hpp"
#include "inclined_plane.hpp"
#include "bump.hpp"
#include "dam_2d.hpp"
#include "spherical.hpp"
```

```
#include "macdonald_like.hpp"
#include "macdonald_like_diffus.hpp"
#include "thacker.hpp"
#include "bedload.hpp"
#include "thacker2d.hpp"
#include "macdonaldb1.hpp"
#include "macdonaldb2.hpp"
#include "sampson.hpp"
#include "swash.hpp"
```

Classes

- class Choice_solution

  Choice of the solution.

Macros

- #define CHOICE_SOLUTION_HPP

6.3.1 Detailed Description

Choice of the solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)
Noemie Gaveau noemie.gaveau@gmail.com (2015)
Maxime Rougier rougiermaxime01@gmail.com (2022)

Version

1.04.00

Date

2022-06-29

From the value of the corresponding parameter, calls the chosen solution.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.3.2 Macro Definition Documentation

CHOICE_SOLUTION_HPP

```
#define CHOICE_SOLUTION_HPP
```

Definition at line 138 of file choice_solution.hpp.

6.4 Headers/dam_2d.hpp File Reference

Computes Static dam solutions in 2D.

```
#include "solution.hpp"
```

Classes

- class Dam_2D

  Computes Static dam solutions in 2D.

6.4.1 Detailed Description

Computes Static dam solutions in 2D.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Analytic solution: with a dam in 2d, see Delestre et al. [2013].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.5 Headers/dam_break.hpp File Reference

Computes dam break solutions.

```
#include "solution.hpp"
```

Classes

- class Dam_break

  Computes dam break solutions.

6.5.1 Detailed Description

Computes dam break solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: dam break without friction, see Ritter [1892] Stoker [1957].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.6 Headers/dressler_dam.hpp File Reference

Computes Dressler dam break solution.

```
#include "solution.hpp"
```

Classes

- class Dressler_dam

  Computes Dressler dam break solution.

6.6.1 Detailed Description

Computes Dressler dam break solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: dam break with friction, see Dressler [1952].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.7 Headers/inclined_plane.hpp File Reference

Computes the solution over an inclined plane.

```
#include "solution.hpp"
```

Classes

- class Inclined_plane

  Computes the solutions over an inclined plane.

6.7.1 Detailed Description

Computes the solution over an inclined plane.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Anne-Celine Boulanger anne-celine.boulanger@inria.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2014-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Delestre et al. [2012].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA - Universite Pierre et Marie Curie (France)

6.8 Headers/macdonald_like.hpp File Reference

Computes Mac Donald solutions.

```
#include "solution.hpp"
```

Classes

- class MacDonald_like

  Computes Mac Donald solutions.

6.8.1 Detailed Description

Computes Mac Donald solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: Mac Donald solutions in 1d, see MacDonald [1996] MacDonald et al. [1997], Delestre et al. [2013] and Vo T. N. [2008].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.9 Headers/macdonald_like_diffus.hpp File Reference

Computes Mac Donald solutions with diffusion.

```
#include "solution.hpp"
```

Classes

- class MacDonald_like_diffus

  Computes Mac Donald solutions with diffusion.

6.9.1 Detailed Description

Computes Mac Donald solutions with diffusion.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: Mac Donald solutions in 1d with diffusion, see Delestre and Marche [2010].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.10 Headers/macdonaldb1.hpp File Reference

Computes Mac Donald pseudo 2d solutions.

```
#include "solution.hpp"
```

Classes

- class MacDonaldB1

  Computes Mac Donald pseudo 2d solutions.

6.10.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald pseudo 2d solutions with bottom B1, see MacDonald [1996].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.11 Headers/macdonaldb2.hpp File Reference

Computes Mac Donald pseudo 2d solutions.

```
#include "solution.hpp"
```

Classes

- class MacDonaldB2

  Computes Mac Donald pseudo 2d solutions.

6.11.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald pseudo 2d solutions with bottom B2, see MacDonald [1996].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.12 Headers/misc.hpp File Reference

Definitions.

```
   #include <vector>
#include <iomanip>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <fstream>
#include <complex>
#include <cstdlib>
```

Macros

- #define MAX(a, b) (a>=b?a:b)
- #define MIN(a, b) (a<=b?a:b)
- #define GRAV 9.81
- #define GRAV_DEM 4.905
- #define PI 3.14159265
- #define EPSILON_H 1.e-12
- #define EPSILON 1.e-12
- #define VERSION "SWASHES version 1.04.00, 2022-09-02"

Typedefs

- typedef double SCALAR
- typedef vector< vector< SCALAR > > TAB

6.12.1 Detailed Description

Definitions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2015)
Rougier Maxime maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-09-02

Defines the constants, the types used in the code and contains the ‘include'.

Copyright

6.12.2 Macro Definition Documentation

EPSILON

```
#define EPSILON 1.e-12
```

Definition at line 73 of file misc.hpp.

EPSILON_H

```
#define EPSILON_H 1.e-12
```

Definition at line 72 of file misc.hpp.

GRAV

```
#define GRAV 9.81
```

Definition at line 69 of file misc.hpp.

GRAV_DEM

```
#define GRAV_DEM 4.905
```

Definition at line 70 of file misc.hpp.

MAX

```
#define MAX(
            a,
            b ) (a>=b?a:b)
```

Definition at line 66 of file misc.hpp.

MIN

```
#define MIN(
            a,
            b ) (a<=b?a:b)
```

Definition at line 67 of file misc.hpp.

PI

```
#define PI 3.14159265
```

Definition at line 71 of file misc.hpp.

VERSION

```
#define VERSION "SWASHES version 1.04.00, 2022-09-02"
```

Definition at line 75 of file misc.hpp.

6.12.3 Typedef Documentation

SCALAR

```
typedef double SCALAR
```

Definition at line 79 of file misc.hpp.

TAB

```
typedef vector< vector< SCALAR > > TAB
```

Definition at line 80 of file misc.hpp.

6.13 Headers/parameters.hpp File Reference

Gets parameters.

```
#include "misc.hpp"
```

Classes

- class Parameters

  Gets parameters.

6.13.1 Detailed Description

Gets parameters.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2015)

Version

1.03.00

Date

2015-10-28

Reads the parameters, checks their values, returns the use if needed.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.14 Headers/sampson.hpp File Reference

Computes Sampson solution.

```
#include "solution.hpp"
```

Classes

- class Sampson

  Computes Sampson solution.

6.14.1 Detailed Description

Computes Sampson solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Sampson parabola with friction, see Sampson et al. [2006] Sampson et al. [2008].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.15 Headers/selfsimilar_dam_break.hpp File Reference

Computes self-similar dam break solutions.

```
#include "solution.hpp"
```

Classes

- class Selfsimilar_dam_break

  Computes self-similar dam break solutions.

6.15.1 Detailed Description

Computes self-similar dam break solutions.

Author

Serge Bodjona (2013)
Carine Lucas carine.lucas@univ-orleans.fr (2014-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: self-similar solution for dam break with friction,
see Self-similar_solutions.pdf in the doc folder or in the bibliography of sourcesup.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.16 Headers/sluice_gate.hpp File Reference

Computes dam break with a sluice gate solutions.

```
#include "solution.hpp"
```

Classes

- class Sluice_gate

  Computes dam break with a sluice gate solutions.

6.16.1 Detailed Description

Computes dam break with a sluice gate solutions.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Analytic solution: dam break with a sluice gate without friction, see Cozzolino et al. [2015].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.17 Headers/solution.hpp File Reference

Common file.

```
#include "parameters.hpp"
```

Classes

- class Solution

  Analytic solution.

6.17.1 Detailed Description

Common file.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)
Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Common part for all the solutions.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.18 Headers/spherical.hpp File Reference

Computes Static solutions in spherical geometry.

```
#include "solution.hpp"
```

Classes

- class Spherical

  Computes Static solutions in spherical geometry.

6.18.1 Detailed Description

Computes Static solutions in spherical geometry.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Analytic solution: different static solutions in spherical geometry, see Williamson et al. [1992].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.19 Headers/step.hpp File Reference

Computes dam break with a step solutions.

```
#include "solution.hpp"
```

Classes

- class Step

  Computes dam break with a step solutions.

6.19.1 Detailed Description

Computes dam break with a step solutions.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-28

Analytic solution: dam break with a step without friction, see HAN and WARNECKE [2014].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.20 Headers/swash.hpp File Reference

Computes the solutions of the swash over an inclined plane.

```
#include "solution.hpp"
```

Classes

- class Swash

  Computes the solutions of the swash over an inclined plane.

6.20.1 Detailed Description

Computes the solutions of the swash over an inclined plane.

Author

Noemie Gaveau noemie.gaveau@gmail.com (2015)
Carine Lucas carine.lucas@univ-orleans.fr (2015-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: swash solutions, see Marche [2005], Carrier and Greenspan [1958]

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.21 Headers/thacker.hpp File Reference

Computes Thacker solution.

```
#include "solution.hpp"
```

Classes

- class Thacker

  Computes Thacker solution.

6.21.1 Detailed Description

Computes Thacker solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Thacker parabola, see Thacker [1981].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.22 Headers/thacker2d.hpp File Reference

Computes Thacker solutions in 2D.

```
#include "solution.hpp"
```

Classes

- class Thacker2D

  Computes Thacker solutions in 2D.

6.22.1 Detailed Description

Computes Thacker solutions in 2D.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Thacker paraboloid, see Thacker [1981].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.23 Sources/bedload.cpp File Reference

Computes solutions with bedload.

```
#include "bedload.hpp"
```

6.23.1 Detailed Description

Computes solutions with bedload.

Author

Minh Hoang Le lemhoang@math.cnrs.fr (2012)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: the bed is moving with bedload, see Berthon et al. [2012].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.24 Sources/bump.cpp File Reference

Computes bumps solutions.

```
#include "bump.hpp"
```

6.24.1 Detailed Description

Computes bumps solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Anne-Celine Boulanger anne-celine.boulanger@inria.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-30

Analytic solution: with a bump, see Delestre et al. [2013], Goutal and Maurel [1997].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA - Universite Pierre et Marie Curie (France)

6.25 Sources/choice_solution.cpp File Reference

Choice of the solution.

```
#include "choice_solution.hpp"
```

6.25.1 Detailed Description

Choice of the solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)
Noemie Gaveau noemie.gaveau@gmail.com (2015)
Maxime Rougier rougiermaxime01@gmail.com (2022)

Version

1.04.0

Date

2022-07-28

From the value of the corresponding parameter, calls the chosen solution.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.26 Sources/dam_2d.cpp File Reference

Computes Static dam solutions in 2D.

```
#include "dam_2d.hpp"
```

6.26.1 Detailed Description

Computes Static dam solutions in 2D.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Analytic solution: with a dam in 2d, see Delestre et al. [2013].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.27 Sources/dam_break.cpp File Reference

Computes dam break solutions.

```
#include "dam_break.hpp"
```

6.27.1 Detailed Description

Computes dam break solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: dam break without friction, see Ritter [1892] Stoker [1957].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.28 Sources/dressler_dam.cpp File Reference

Computes Dressler dam break solution.

```
#include "dressler_dam.hpp"
```

6.28.1 Detailed Description

Computes Dressler dam break solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: dam break with friction, see Dressler [1952].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.29 Sources/inclined_plane.cpp File Reference

Computes the solution over an inclined plane.

```
#include "inclined_plane.hpp"
```

6.29.1 Detailed Description

Computes the solution over an inclined plane.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Anne-Celine Boulanger anne-celine.boulanger@inria.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2014-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Delestre et al. [2012].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA - Universite Pierre et Marie Curie (France)

6.30 Sources/macdonald_like.cpp File Reference

Computes Mac Donald solutions.

```
#include "macdonald_like.hpp"
```

6.30.1 Detailed Description

Computes Mac Donald solutions.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald solutions in 1d, see MacDonald [1996], MacDonald et al. [1997], Delestre et al. [2013] and Vo T. N. [2008].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.31 Sources/macdonald_like_diffus.cpp File Reference

Computes Mac Donald solutions with diffusion.

```
#include "macdonald_like_diffus.hpp"
```

6.31.1 Detailed Description

Computes Mac Donald solutions with diffusion.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald solutions in 1d with diffusion, see Delestre and Marche [2010].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.32 Sources/macdonaldb1.cpp File Reference

Computes Mac Donald pseudo 2d solutions.

```
#include "macdonaldb1.hpp"
```

6.32.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald pseudo 2d solutions with bottom B1, see MacDonald [1996].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.33 Sources/macdonaldb2.cpp File Reference

Computes Mac Donald pseudo 2d solutions.

```
#include "macdonaldb2.hpp"
```

6.33.1 Detailed Description

Computes Mac Donald pseudo 2d solutions.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Mac Donald pseudo 2d solutions with bottom B2, see MacDonald [1996].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.34 Sources/parameters.cpp File Reference

Gets parameters.

```
#include "parameters.hpp"
```

6.34.1 Detailed Description

Gets parameters.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)
Rougier Maxime maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-08-24

Reads the parameters, checks their values, returns the use if needed.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.35 Sources/sampson.cpp File Reference

Computes Sampson solution.

```
#include "sampson.hpp"
```

6.35.1 Detailed Description

Computes Sampson solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Sampson parabola with friction, see Sampson et al. [2006] Sampson et al. [2008].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.36 Sources/selfsimilar_dam_break.cpp File Reference

Computes self-similar dam break solutions.

```
#include "selfsimilar_dam_break.hpp"
```

6.36.1 Detailed Description

Computes self-similar dam break solutions.

Author

Serge Bodjona (2013)
Carine Lucas carine.lucas@univ-orleans.fr (2014-2015)

Version

1.03.00

Date

2015-10-28

Analytic solution: self-similar solution for dam break with friction,
see Self-similar_solutions.pdf in the doc folder or in the bibliography of sourcesup.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.37 Sources/sluice_gate.cpp File Reference

Computes dam break with a sluice gate solutions.

```
#include "sluice_gate.hpp"
```

6.37.1 Detailed Description

Computes dam break with a sluice gate solutions.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-13

Analytic solution: dam break with a sluice gate without friction, see Cozzolino et al. [2015].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.38 Sources/solution.cpp File Reference

Common file.

```
#include "solution.hpp"
```

6.38.1 Detailed Description

Common file.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2010-2022)
Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-18

Common part for all the solutions.

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.39 Sources/spherical.cpp File Reference

Computes Static solutions in spherical geometry.

```
#include "spherical.hpp"
```

6.39.1 Detailed Description

Computes Static solutions in spherical geometry.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-18

Analytic solution: different static solutions in spherical geometry, see Williamson et al. [1992].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.40 Sources/step.cpp File Reference

Computes dam break with a step solutions.

```
#include "step.hpp"
```

6.40.1 Detailed Description

Computes dam break with a step solutions.

Author

Maxime Rougier maximerougier01@gmail.com (2022)

Version

1.04.00

Date

2022-07-28

Analytic solution: dam break with a step without friction, see HAN and WARNECKE [2014].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.41 Sources/swash.cpp File Reference

Computes the solutions of the swash over an inclined plane.

```
#include "swash.hpp"
```

6.41.1 Detailed Description

Computes the solutions of the swash over an inclined plane.

Author

Noemie Gaveau noemie.gaveau@gmail.com (2015)
Carine Lucas carine.lucas@univ-orleans.fr (2015-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: swash solutions, see Marche [2005], Carrier and Greenspan [1958].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.42 Sources/swashes.cpp File Reference

Main file.

```
#include "choice_solution.hpp"
```

Functions

- int main (int argc, char **argv)

6.42.1 Detailed Description

Main file.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2015)

Version

1.03.00

Date

2015-10-28

For more details, we refer to Delestre et al. [2013].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.42.2 Function Documentation

main()

```
int main (
            int argc,
            char ** argv )
```

Main function of SWASHES, see Delestre et al. [2013].

Parameters

in	*argc*	number of the arguments.
in	*argv*	value of the arguments.

Definition at line 58 of file swashes.cpp.

6.43 Sources/thacker.cpp File Reference

Computes Thacker solution.

```
#include "thacker.hpp"
```

6.43.1 Detailed Description

Computes Thacker solution.

Author

Olivier Delestre olivierdelestre41@yahoo.fr (2010)
Carine Lucas carine.lucas@univ-orleans.fr (2012-2022)

Version

1.03.01

Date

2022-03-29

Analytic solution: Thacker parabola, see Thacker [1981].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

6.44 Sources/thacker2d.cpp File Reference

Computes Thacker solution in 2D.

```
#include "thacker2d.hpp"
```

6.44.1 Detailed Description

Computes Thacker solution in 2D.

Author

Pierre-Antoine Ksinant pierreantoine.ksinantgarcia@gmail.com (2011)
Carine Lucas carine.lucas@univ-orleans.fr (2011-2015)

Version

1.03.00

Date

2015-10-28

Analytic solution: Thacker paraboloid, see Thacker [1981].

Copyright

License Cecill-V2
http://www.cecill.info/licences/Licence_CeCILL_V2-en.html

(c) CNRS - Universite d'Orleans - INRA (France)

Bibliography

C. Berthon, S. Cordier, O. Delestre, and M.-H. Le. An analytical solution of the Shallow Water system coupled to the Exner equation. *C. R. Acad. Sci. Paris, Ser. I*, 350(3–4):183–186, 2012. doi:10.1016/j.crma.2012.01.007. URL http://hal.archives-ouvertes.fr/hal-00648343. 9, 10, 73, 90

G. F. Carrier and H. P. Greenspan. Water waves of finite amplitude on a sloping beach. *Journal of Fluid Mechanics*, 4:97–109, 1958. doi:10.1017/S0022112058000331. URL http://journals.cambridge.org/article_S0022112058000331. 65, 66, 88, 99

L. Cozzolino, L. Cimorelli, C. Covelli, R. Della Morte, and D. Pianese. The analytic solution of the shallow-water equations with partially open sluice-gates: The dam-break problem. *Advances in Water Resources*, 80,90-102, 2015. doi:https://doi.org/10.1016/j.advwatres.2015.03.010. 48, 85, 97

O. Delestre and F. Marche. A numerical scheme for a viscous shallow water model with friction. *Journal of Scientific Computing*, pages 1–11, 2010. ISSN 0885-7474. doi:10.1007/s10915-010-9393-y. 33, 34, 79, 94

O. Delestre, S. Cordier, F. Darboux, and F. James. A limitation of the hydrostatic reconstruction technique for Shallow Water equations. *Comptes Rendus Mathématique*, 350(13-14):677–681, 2012. doi:10.1016/J.crma.2012.08.004. URL http://hal.archives-ouvertes.fr/hal-00710654. 26, 27, 78, 93

O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T. N. T. Vo, F. James, and S. Cordier. SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies. *International Journal of Numerical Methods in Fluids*, 72(3):269–300, May 2013. doi:10.1002/fld.3741. URL http://hal.archives-ouvertes.fr/hal-00628246. See Annex on the HAL page for complementary results (with illustrations of each case). 12, 14, 18, 30, 31, 74, 76, 79, 90, 91, 93, 100

R. F. Dressler. Hydraulic resistance effect upon the dam-break functions. *Journal of Research of the National Bureau of Standards*, 49(3):217–225, Sept. 1952. 23, 24, 77, 92

N. Goutal and F. Maurel. Proceedings of the 2^{nd} workshop on dam-break wave simulation. Technical Report HE-43/97/016/B, Electricité de France, Direction des études et recherches, 1997. 12, 14, 74, 90

E. HAN and G. WARNECKE. Exact riemann solutions to shallow water equations. *Quarterly of applied mathematics*, LXXII, NUMBER 3, 2014. 62, 87, 98

I. MacDonald. *Analysis and computation of steady open channel flow*. PhD thesis, University of Reading — Department of Mathematics, Sept. 1996. 30, 31, 35, 36, 38, 39, 79, 80, 81, 93, 94, 95

I. MacDonald, M. J. Baines, N. K. Nichols, and P. G. Samuels. Analytic benchmark solutions for open-channel flows. *Journal of Hydraulic Engineering*, 123(11):1041–1045, Nov. 1997. 30, 31, 79, 93

F. Marche. *Theoretical and numerical study of shallow water models ; applications to nearshore hydrodynamics*. Phd, Université Bordeaux 1, 2005. 65, 66, 88, 99

A. Ritter. Die Fortpflanzung der Wasserwellen. *Zeitschrift des Vereines Deuscher Ingenieure*, 36(33):947–954, 1892. 21, 22, 77, 92

J. Sampson, A. Easton, and M. Singh. Moving boundary shallow water flow above parabolic bottom topography. In A. Stacey, B. Blyth, J. Shepherd, and A. J. Roberts, editors, *Proceedings of the* 7^{th} *Biennial Engineering Mathematics and Applications Conference, EMAC-2005*, volume 47 of *ANZIAM Journal*, pages C373–C387. Australian Mathematical Society, oct 2006. URL http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/1050. 44, 45, 84, 96

J. Sampson, A. Easton, and M. Singh. Moving boundary shallow water flow in a region with quadratic bathymetry. In G. N. Mercer and A. J. Roberts, editors, *Proceedings of the* 8^{th} *Biennial Engineering Mathematics and Applications Conference, EMAC-2007*, volume 49 of *ANZIAM Journal*, pages C666–C680. Australian Mathematical Society, 2008. URL http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/306. 44, 45, 84, 96

J. J. Stoker. *Water Waves: The Mathematical Theory with Applications*. Pure and Applied Mathematics. Interscience Publishers, New York, USA, 1957. 21, 22, 77, 92

W. C. Thacker. Some exact solutions to the nonlinear shallow-water wave equations. *Journal of Fluid Mechanics*, 107:499–508, 1981. doi:10.1017/S0022112081001882. URL http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=389055&fulltextType=RA&fileId=S0022112081001882. 68, 69, 70, 71, 89, 100, 101

T. Vo T. N. One dimensional Saint-Venant system. Master's thesis, Université d'Orléans, France, June 2008. URL http://dumas.ccsd.cnrs.fr/dumas-00597434. 30, 31, 79, 93

D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *Journal of Computational Physics*, 102,211-224, 1992. doi:https://doi.org/10.1016/S0021-9991(05)80016-6. 60, 87, 98

Index