MOL: METHOD OF LINES APPLICATION

John R. Rice and Mikel Luján

October 13, 2000

Abstract

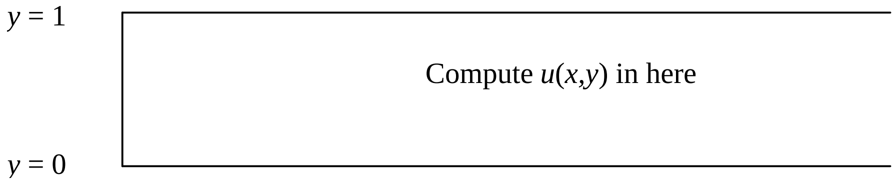
MOL is a C program that has the behavior of a realistic scientific application: the solution of time dependent partial differential equations by the "Method of Lines". MOL is an abstraction that eliminates or greatly simplifies those components of a realistic code which are required for robust and accurate results, but which have little or no effect on run time behavior. As a result, MOL has only 4,000 lines of code instead of 20,000 or 30,000 or more for a realistic code. Further, MOL is highly object oriented to facilitate analysis, instrumentation and translation. MOL should be viewed as producing the display of the PDE solution and the computation should produce a good "quality of service" (e.g., 30 solutions a second). MOL has four types (output speed, solution behavior, accuracy and parallelism) of externally controllable parameters to adjust execution behavior with a total of nine parameters. Currently MOL has one time and one space dimension. It is designed to be extendible in various ways, e.g., more space dimensions and adding communication parameters.

I. THE METHOD OF LINES

We start with the simplest case and later show relevant variations. The PDE problem is

$$u_x = f * u_{yy}$$

on the 3-sided domain: $x = 0, 0 \le y \le 1$; $y = 0, 0 \le x$; $x = 1, 0 \le x$. The usual presentation is $u_t = f * u_{xx}$ but I reserve the variable t to be *real time*. Values of $u(x,y)$ are given on 3-sides of the domain:



The computation starts at $x = 0$ and evolves (this PDE problem is often called evolutionary) through increasing x values.

The *Quality of Service* (QoS) requirement is **Produce a solution* $u(x_i,y)$ *at a fixed rate of progress*, (ROP). That is, given values x_i (say $x_i = i$*.01 for $i = 0$ to 1 million), produce the snapshot $u(x_i,y)$ at the rate ROP of, say, 30 per second.

The method goes as follows:

1. Discretize the y variable into K intervals, of size Δy by $y_k = k * \Delta y$, $k = 0,\dots, K$. This defines K-2 new lines parallel to $y = 0$ and $y = 1$. $u(x,y)$ is given on the boundaries for $k = 0, K$
2. On line k, discretize u_{yy} by the simple finite difference formula

$$u_{yy}(x,y) = \frac{u(x,y+\Delta y) - 2u(x,y) + u(x,y-\Delta y)}{(\Delta y)^2}$$

3. This creates K-2 ordinary differential equations (ODE) to be solved on the interior lines. Let $V_k(x) = u(y_k,x)$ to simplify the notation and these ODEs are

   $$dV_k(x)/dx = f * [V_{k+1}(x) - 2V_k(x) + V_{k-1}(x)]/ \Delta^2 y \qquad (1.1)$$

   for $k = 1,\ldots, K$-1
4. One now applies a standard (and for our application, a simple) ODE solving formula to each of these ODEs. An explicit formula generates a computation at $x = x_i$ of the form

   $$V_k(x_\mathrm{i} + \Delta x) = \text{some function of information for } x \le x_\mathrm{i} .$$

   Thus the ODE solver operates on line k in a mode of (1) make a step forward of size Δx, (2) exchange information with lines k+1 and k–1, (3) Repeat.

II. MOL Object Oriented Model — Model-A

The PDE problem is defined by four functions; three functions determining the values of *u(x,y)* on 3-sides of the domain, and a discretized function on the y-dimension. We represent the first three functions as objects (solutionAtT0, solutionAtY0, solutionAtYLast) of class Function2D, while the discretized function is represented as an object (functionDiscretisedFirstDerivative) of class FunctionDiscretised2D.

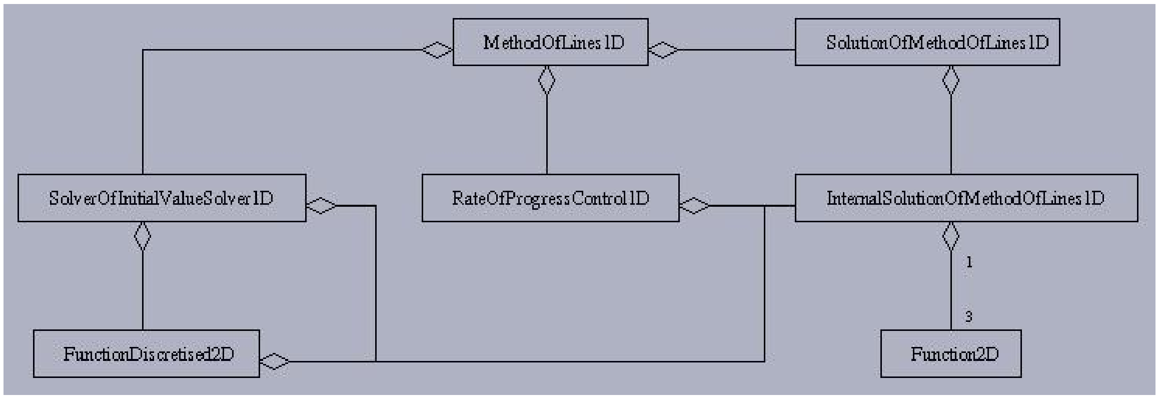
The main difference between these two classes is that FunctiondDiscretised2D has an attribute of class InternalSolutionOfMethodOfLines1D which enables a discretized function to access values of $V_k(x)$ (i.e. previously obtained solutions) in any line (see 1.1).

The class InternalSolutionOfMethodOfLines1D is a container whose objects store the set of internal values computed by the ODE solver (solver of Initial Value Problem). Internal is used to differentiate between values computed by the solver of Initial Value Problem (IVP), from the values required by users and specified by the snapshots $u(x_i,y)$ (with $x_i = i * userDeltaX$). The IVP solver can be computing the solution with a Δx different from *userDeltaX*. Thus, users simply have to access objects of class SolutionOfMethodOfLines1D which store the solution values for each snapshot.

An object of class RateOfControl1D represents the ROP. This class will receive information of how long took to compute the last *x*-step, how many steps are needed for the next snapshot and how much time has been consumed since was computed the previous snapshot. This information will be used to determine whether the ROP could be met or whether the accuracy should be relaxed.

An IVP solver is represented by an object of class SolverOfInitialValueProblem1D. This object has an attribute of class InternalSolutionOfMethodOfLines and another attribute of class FunctionDiscretised2D. The first attribute enables the solver to access solution values computed in a previous step. The second attribute enables the solver to evaluate the discretized first derivative function (1.1).

The UML class diagram for MOL model-A is:


MethodOfLines1D
SolutionOfMethodOfLines1D
SolverOfInitialValueSolver1D
RateOfProgressControl1D
InternalSolutionOfMethodOfLines1D
1
3
FunctionDiscretised2D
Function2D

III. THE SPECIFIC APPLICATION — SIMPLEST CASE OF MOL

We choose the PDE to have the true solution

$$u(x,y) = \cos(by) * [2 + \sin(ax)]$$

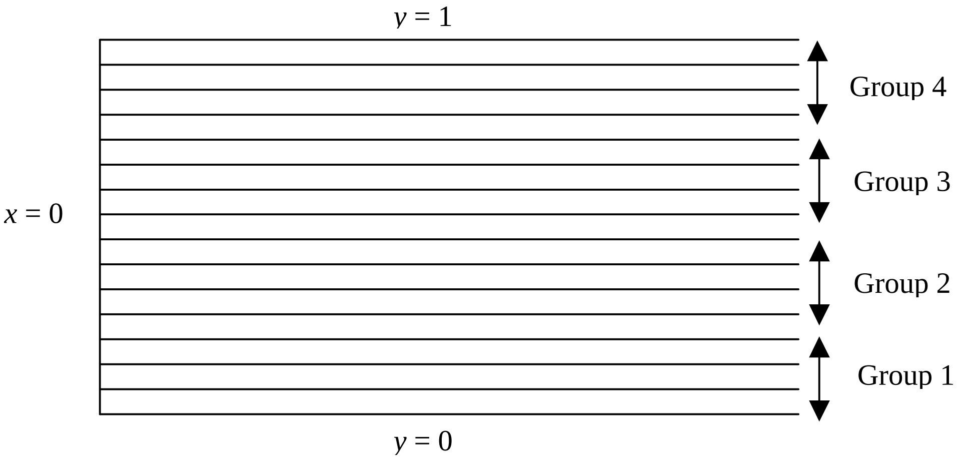
So the PDE is

$$\begin{aligned} u_x &= -[(a/b^2) * \cos(ax)/(2 + \sin(ax))]u_{yy} \\ &= f(a,b,x) * u_{yy} \end{aligned}$$

The parameters *a* and *b* are the *knobs* one varies to change the solution and thus the work required. We assume these are varied smoothly as a function of real time. We restrict the computation to

- Use constant Δy (at any value of x).
- Use constant Δx (at any value of y).
- Use the same ODE method on each line at any value of x.
- Adapt Δx and Δy to achieve an accuracy TOL.
- Be able to set a switch *Meth* to choose between two ODE methods.
- Group NG neighboring lines together to form a process for parallel execution.

The step size Δx is determined automatically by the ODE methods and Δy is determined by a simple function inside MOL. This leaves three knobs to control the computation: *Tol*, *Meth*, and *NG*. The computation geometry is illustrated as follows:



for the case of 16 lines partitioned into groups of 4.

As the control knows *a*, *b*, *Tol*, *Meth* and *NG* are changed, the rate of progress of the computation changes. The controls *a* and *b* are external, while *Meth* and *NG* are internal. The control *Tol* is ambiguous, one wants to set it externally, but if the computing capacity is inadequate, one can rationally choose between not meeting the QoS requirement or relaxing the accuracy requirement.

IV. MOL Object Oriented Model — Model-B

The restrictions of MOL make us modify Model-A. We need to accommodate Model-A so that:

- Δx and Δy can be adapted to achieve TOL, and
- switch the IVP solver among a group of them[*].

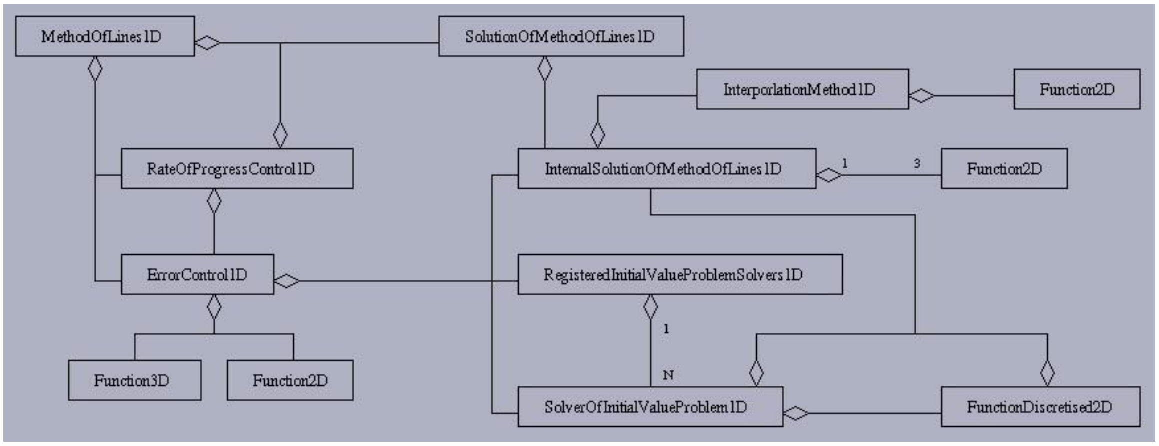
How to change Δx and Δy, and how to switch the IVP solver are encapsulated into an object of class ErrorControl1D. Such an obje ct will use an estimation of the computational error to, accordingly, increase or decrease Δx and Δy, and to switch among higher or lower order IVP solvers. For this purpose, users will have to provide a function that estimates the error due to the IVP sol ver (see 6.1) and a function that estimates the error due to the discrimination on *y*-dimension (see 6.2). The function that estimates the error due to the IVP solver takes as parameters the order of the solver, Δx and *x*; it is represented as an object of c lass Function3D. The other function takes as parameters Δy and *x*; it is represented as an object of class Function2D. Thus, the class ErrorControl1D has attributes of classes Function2D and Function3D.

The list of IVP solvers, among which MOL selects, is stored in an ordered list, class named RegisteredInitialValueProblemSolver1D.

Since Δx and Δy are allowed to change and IVP solvers use solution values $u(x_i,y)$ with x_i less than current *x*, sometimes $u(x_i,y)$ has not been computed. In these cases, we estima te $u(x_i,y)$ by interpolation. The class InterpolationMethod1D represents an interpolation algorithm.

[*] We do not introduce parallelism in Model-B. This will be introduced in Model-Par.

The UML class diagram for Model-B is:



V. GENERALIZATIONS OF MOL

We now present various ways the above application can be generalized to create more interesting computations.

III. Increase the Dimension [*]. One can add more space dimensions (x is time and y is space above) and solve

$$u_x = f * (u_{yy} + u_{zz})$$
$$u_x = f * (u_{yy} + u_{zz} + u_{ww})$$

The complexity of the code increases little with these additions, provided one maintains the groups of lines as equal rectangular sets. One can allow a border of smaller rectangular groups with somewhat more effort. Otherwise, one is restricted to 1, 4, 9, 16, 25, 36, … groups in 3D and 1, 8, 27, 64, 125, 216, 343, … groups in 4D.

The computing requirements increase rapidly with dimension, being of the order of K, K^2 and K^3 for space dimensions 1, 2, and 3, respectively.

III.2 Change the Number of Lines Dynamically. The number of lines in y (and z and w) is determined by *Tol*. We assume (reasonably) that we have a function D so that $\Delta y = D(Tol)$. As we vary b, we should change Δy, the relationship is, roughly, $\Delta y^2 = Tol/b^4$. In the previous simple case we are implicitly assuming that *Tol* is determined by the resolution of the visualization and not by the numerical accuracy needed.

III.3 Change the ODE Method Dynamically. This is actually fairly easy to do and the pause in the solution should be short, perhaps the order of the time to make 2 –5 steps in x. It is interesting because it can have a large effect on accuracy (and thus step size Δx) and adds to the challenge of management.

[*] Not to be implemented, since 1D is already complex enough (> 1000 lines).

[+] See Model-Par.

III.4 Change the Number of Groups Dynamically[+]. This is equivalent to adding processors in a parallel computation. It requires the redistribution of data among the groups. If one changes, say, from 6 to 14 groups using 42 lines, then essentially everything must be redistributed. If one simply divides each group of lines, then to makes the change is much simpler, e.g., going from 7 to 14 groups using 42 lines.

III.5 Variable Group Sizes[*]. A group represents a processor of a parallel computing environment. If this environment is heterogeneous, then the group sizes should vary so as to maintain a constant rate of progress. Suppose there are M processors, each of power m_j, then the $M = NG$ groups should have a number of lines g_j with m_j/g_j = constant.

If the group sizes are fixed, then the computation is essentially the same as before, but the code data structures must be more flexible. If the group sizes change dynamically, then further flexibility is needed in the codes for each group.

VI. MOL PARAMETERS

These are defined as follows:

* **ROP**: **Rate of Progress**. The rate of advancing x as a function of real (computing) time.
* **a**: **Time Variation Factor**. The parameter a in sin (ax).
* **b**: **Space Variation Factor**. The parameter b in cos(by).
* **Tol**: **Accuracy**. The tolerance needed in solving the PDE.
* **K**: **Number of Lines**. The number of lines in the method = $1/\Delta y$.
* **Meth**: **Method Selection**. Indicator of the ODE method being used along the lines.
* **MethOr**: **Method Order**. The order of the ODE method.
* **NG**: **Number of Groups**. Number of (nearly?) equal sized groups of lines assigned to a process. Equivalent to the number of processors in most parallel computing environments.

The parameters *ROP*, *a*, and *b* are totally external to the management process. The accuracy parameters (*Tol*, *K*, *Meth*, *Method*) are related approximately by

$$SolverError = C_x \times \Delta x^{\text{MethOr}} * a^{\text{MethOr}+1} \qquad (6.1)$$

$$DiscritizationError = C_y \times \Delta y^2 * b^4 \qquad (6.2)$$

$$Tol = SolverError + DiscritizationError$$

where C_x and C_y are unknown constants that can be estimated rather well (with some effort). The ODE methods automatically adapt the computation to control the term $C_x * \Delta x^M * a^M$ but the management analysis could suggest a change *Meth* to affect the performance. The control of the $C_y * \Delta y^2 * b^4$ term is made by MOL itself as a separate computation. The parameter *NG* depends on the hardware.

VII. MODEL COMPLEXITY AND ANALYSIS

It is obvious that MOL and its generalization have substantial complexity and a wide range of needs for computing power. The interactions between parameters and performance are non-linear and opaque. There are multiple choices for maintaining QoS, e.g., if the QoS is too low, one can

[*] See Model-Par.

increase NG, increas e the power of the processor (a parameter not visible here), change *Meth*, change *Tol*, or take some combination of these actions.

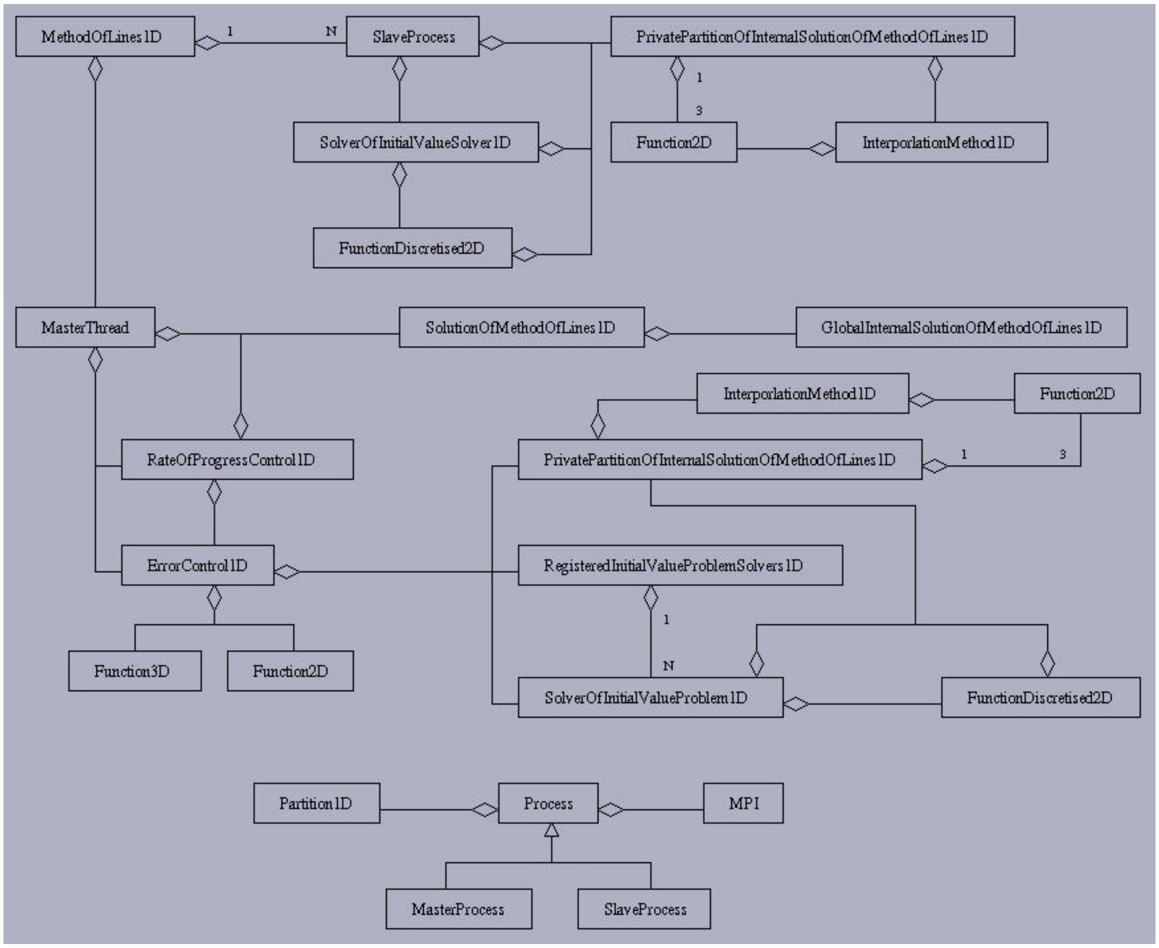
The communication needs exist in MOL but are, so far, derived from other features of MOL. Parameters of the communication syst em (bandwidth, connectivity, latency…) are not present in MOL, but this is as it should be.

VIII. Parallel MOL Object Oriented Model — Model-Par

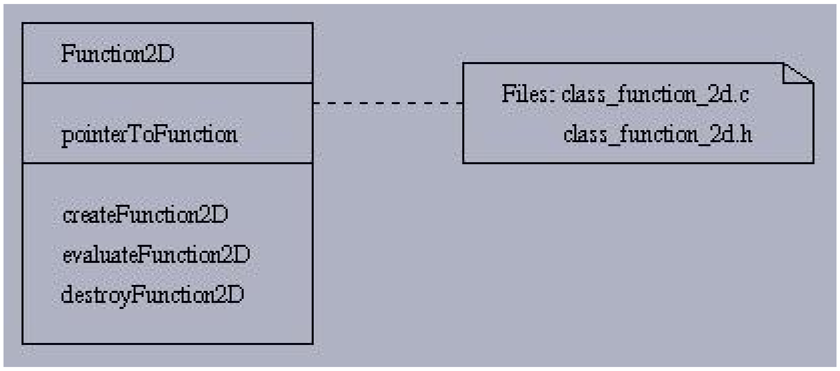
The parallel computation has been defined as a data partition of lines forming groups. The restriction is that a group should be composed of neighboring lines (see Section III). This is represented by an object of class Partition1D. Such an object is replicated for each process and provides methods to determine the lower and upper bounds of the loop that traverses t he lines at each step on *x*. Since the parallel implementation is based on Message Passing (MPI), we create a class GlobalInternalSolutionOfMethodOfLines1D and a class PrivatePartitionOfInternalSolutionOfMethodOfLines1D. These classes substitute the class InternalSolutionOfMethodOfLines1D in Model-A and Model-B.

The parallel computation of MOL will create a number of processes and each process will have its own object of class PrivatePartitionOfInternalSolutionOfMethodOfLines1D. In addition, the master th read will have an object of class GlobalInternalSolutionOfMethodOfLines1D. At each *x*-step, each process will compute the solutions for the lines that are in its partition (`do k = lowerBound, upperBound`) and then exchange the information with neighboring processes. Also at each x step the error control and the ROP are carried by the master, that then broadcasts to all the other processes the Δx, Δy and *Meth.* If necessary, the processes will communicate to repartition the lines.

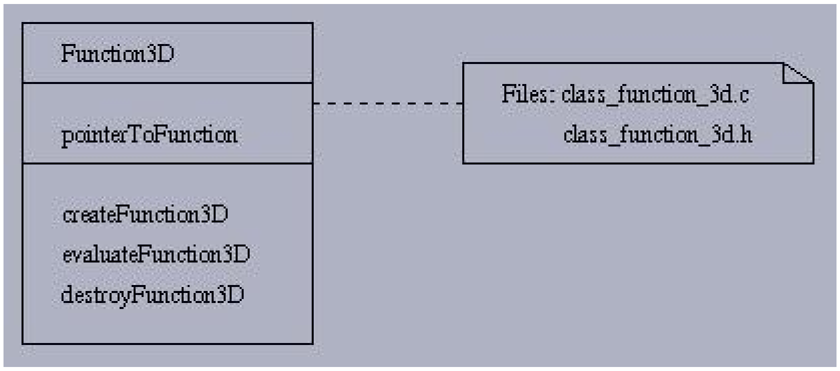
The UML class diagram for Model-Par is:


MethodOfLines1D
1
N
SlaveProcess
PrivatePartitionOfInternalSolutionOfMethodOfLines1D
1
3
SolverOfInitialValueSolver1D
Function2D
InterporlationMethod1D
FunctionDiscretised2D
MasterThread
SolutionOfMethodOfLines1D
GlobalInternalSolutionOfMethodOfLines1D
InterporlationMethod1D
Function2D
RateOfProgressControl1D
PrivatePartitionOfInternalSolutionOfMethodOfLines1D
1
3
ErrorControl1D
RegisteredInitialValueProblemSolvers1D
1
N
Function3D
Function2D
SolverOfInitialValueProblem1D
FunctionDiscretised2D
Partition1D
Process
MPI
MasterProcess
SlaveProcess

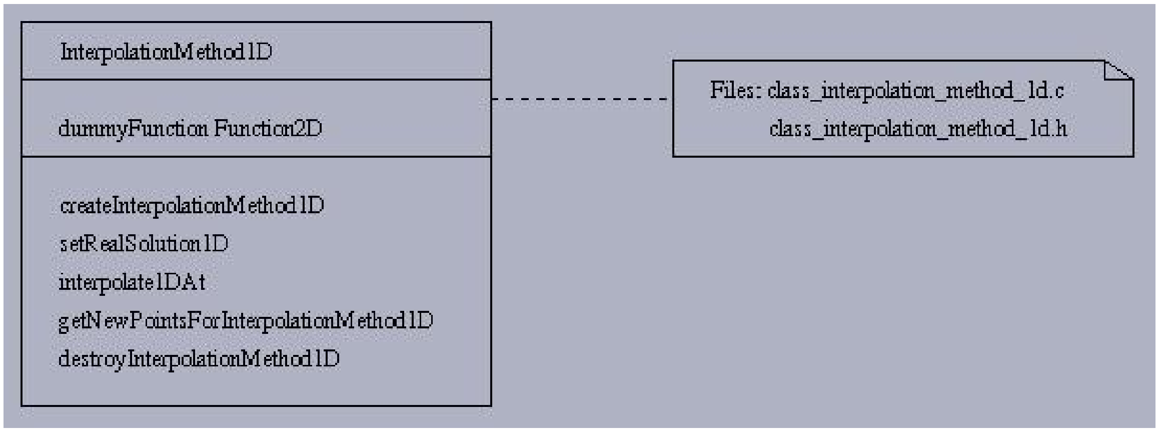
MOL Appendix - UML Class Diagrams with Attributes and Methods



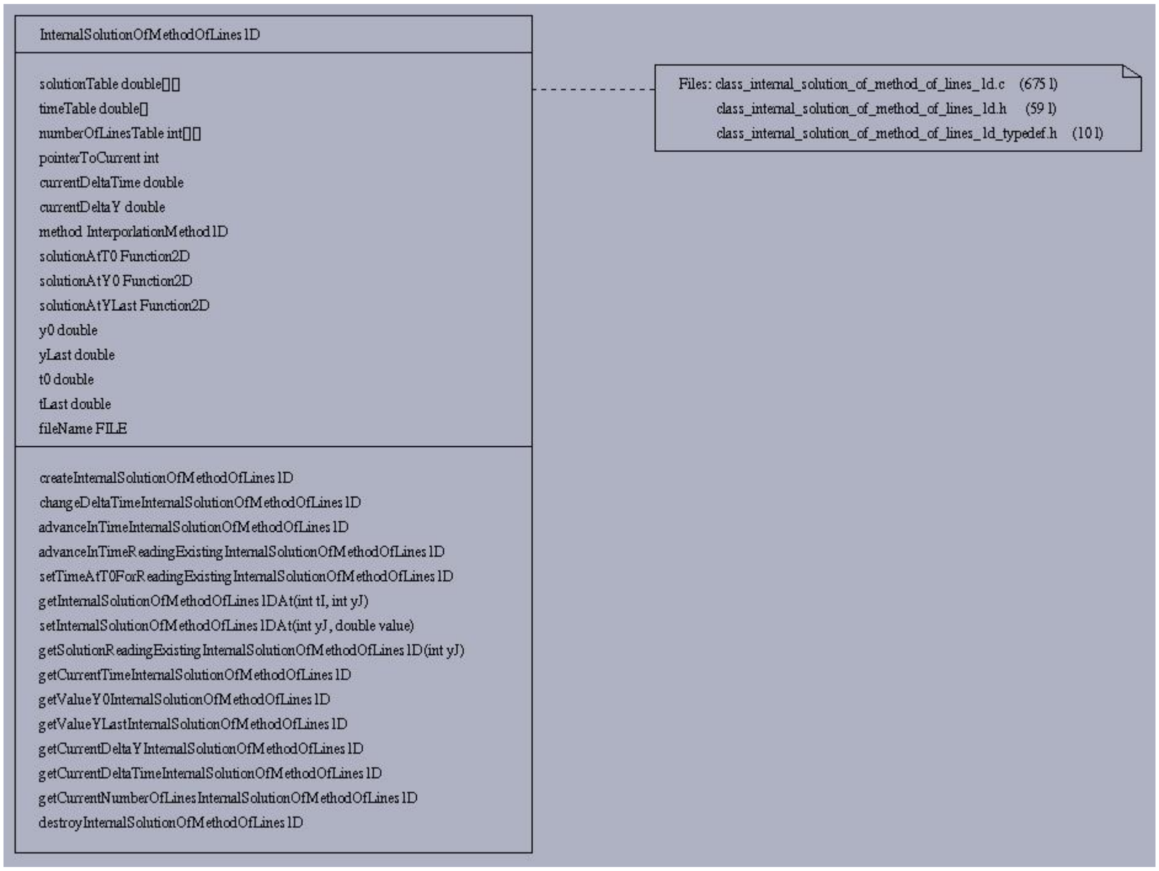
```
struct ClassFunction2D_rep
{
        double (*evaluate) (double, double);
};// end struct

typedef struct ClassFunction2D_rep *ClassFunction2D;

ClassFunction2D createFunction2D
        (double (*evaluate)(double x, double y));

double evaluateFunction2DAt(ClassFunction2D f, double x, double y);

void changeEvaluationFunction2D (ClassFunction2D f,
                       double (*newEvaluate)(double x, double y));

void destroyFunction2D (ClassFunction2D f);
```



```
struct ClassFunction3D_rep
{
   double (*evaluate) (double, double, double);
};// end struct

typedef struct ClassFunction3D_rep *ClassFunction3D;

ClassFunction3D createFunction3D
        (double (*evaluate)(double x, double y, double z));

double evaluateFunction3DAt(ClassFunction3D f, double x,
                            double y, double z);

void changeEvaluationFunction3D (ClassFunction3D f,
            double (*newEvaluate)(double x, double y, double z));

void destroyFunction3D (ClassFunction3D f);
```


InterpolationMethod1D
dummyFunction Function2D
createInterpolationMethod1D
setRealSolution1D
interpolate1DAt
getNewPointsForInterpolationMethod1D
destroyInterpolationMethod1D
Files: class_interpolation_method_1d.c
class_interpolation_method_1d.h

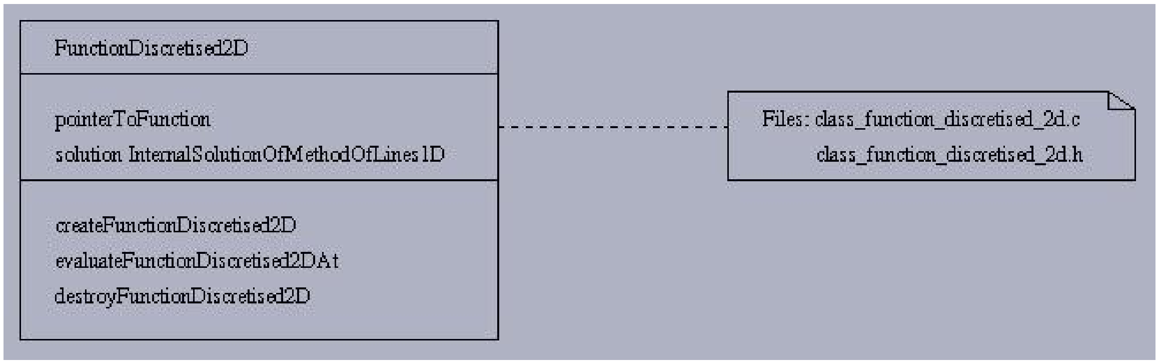
```
struct ClassInterpolationMethod1D_rep
{
   ClassFunction2D f;
}; // end struct

typedef struct ClassInterpolationMethod1D_rep *ClassInterpolationMethod1D;

ClassInterpolationMethod1D createInterpolationMethod1D ();

void setRealSolution1D (ClassInterpolationMethod1D method, ClassFunction2D f);
// this method is included only because this is a dummy interpolation.

double interpolate1DAt (ClassInterpolationMethod1D method,
                        double x, double y);

void getNewPointsForInterpolationMethod1D
                (ClassInterpolationMethod1D method,
                 ClassInternalSolutionOfMethodOfLines1D solution,
                 double x, double y);

void destroyInterpolationMethod1D (ClassInterpolationMethod1D method);
```


InternalSolutionOfMethodOfLines1D
solutionTable double[][]
timeTable double[]
numberOfLinesTable int[][]
pointerToCurrent int
currentDeltaTime double
currentDeltaY double
method InterporlationMethod1D
solutionAtT0 Function2D
solutionAtY0 Function2D
solutionAtYLast Function2D
y0 double
yLast double
t0 double
tLast double
fileName FILE
createInternalSolutionOfMethodOfLines1D
changeDeltaTimeInternalSolutionOfMethodOfLines1D
advanceInTimeInternalSolutionOfMethodOfLines1D
advanceInTimeReadingExistingInternalSolutionOfMethodOfLines1D
setTimeAtT0ForReadingExistingInternalSolutionOfMethodOfLines1D
getInternalSolutionOfMethodOfLines1DAt(int tI, int yJ)
setInternalSolutionOfMethodOfLines1DAt(int yJ, double value)
getSolutionReadingExistingInternalSolutionOfMethodOfLines1D(int yJ)
getCurrentTimeInternalSolutionOfMethodOfLines1D
getValueY0InternalSolutionOfMethodOfLines1D
getValueYLastInternalSolutionOfMethodOfLines1D
getCurrentDeltaYInternalSolutionOfMethodOfLines1D
getCurrentDeltaTimeInternalSolutionOfMethodOfLines1D
getCurrentNumberOfLinesInternalSolutionOfMethodOfLines1D
destroyInternalSolutionOfMethodOfLines1D
Files: class_internal_solution_of_method_of_lines_1d.c (675 l)
class_internal_solution_of_method_of_lines_1d.h (59 l)
class_internal_solution_of_method_of_lines_1d_typedef.h (10 l)

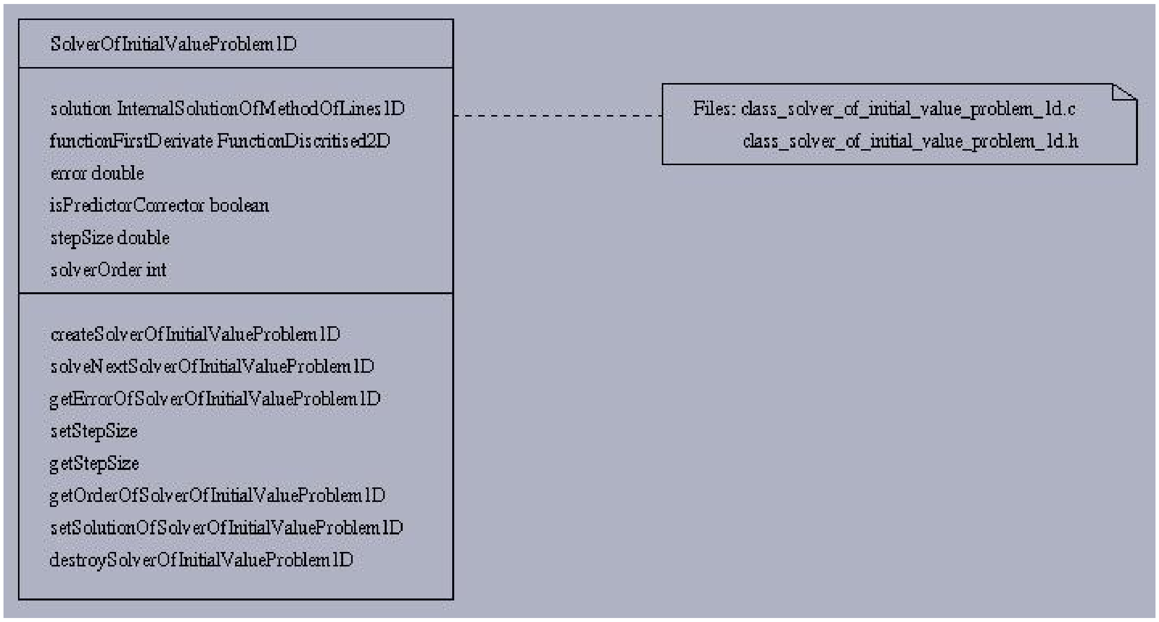
```
struct ClassInternalSolutionOfMethodOfLines1D_rep
{
   double solutionTable[maxIntermediateSolutionsInMemory][maxNumberOfLines-2];
        // maxIntermediateSolutionsInMemory =
maxIntermediateSolutionsPerUserSolution
        //                                      * maxNumberOfWindowsInMemory;
        // the intermediate solutions of a user solution is called
        // a window.
   double timeTable[maxIntermediateSolutionsInMemory];
   int numberOfLinesTable[maxIntermediateSolutionsInMemory];
   int pointerToCurrentTime;
   double currentDeltaTime;
   double currentDeltaY;

   ClassInterpolationMethod1D method;
   ClassFunction2D solutionAtT0;
   ClassFunction2D solutionAtY0;
   ClassFunction2D solutionAtYLast;
   double y0;
   double yLast;
   double t0;
   double tLast;
   double tLastCalculated;
   char *fileName;
};// end struct

typedef struct ClassInternalSolutionOfMethodOfLines1D_rep
                *ClassInternalSolutionOfMethodOfLines1D;

ClassInternalSolutionOfMethodOfLines1D createInternalSolutionOfMethodOfLines1D
```

```
      (ClassInterpolationMethod1D method,
       ClassFunction2D solutionAtT0,
       ClassFunction2D solutionAtY0,
       ClassFunction2D solutionAtYLast,
       double t0,
       double tLast,
       double y0,
       double yLast,
       int numberOfLines,
       double deltaY,
       double deltaTime,
       char *fileName);

void changeDeltaTimeInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution,
         double newDeltaTime);

void changeNumberOfLinesInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution,
                int newNumberOfLines, double newDeltaY);

void advanceInTimeInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution);

void advanceInTimeReadingExistingInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution);

double getSolutionReadingExistingInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution, int yJ);

void setTimeAtT0ForReadingExistingInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution);

int isLastReadingExistingInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution);

int isLastInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D  solution);

double getInternalSolutionOfMethodOfLines1DAt
(ClassInternalSolutionOfMethodOfLines1D solution,
              int tI, int yJ);

void setInternalSolutionOfMethodOfLines1DAt
(ClassInternalSolutionOfMethodOfLines1D  solution,
              int yJ, double value);

double getCurrentTimeInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

double getValueY0InternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

double getValueYLastInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

double getValueT0InternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

double getValueTLastInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);
```

```
double getCurrentDeltaYInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

double getCurrentDeltaTimeInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

int getCurrentNumberOfLinesInternalSolutionOfMethodOfLines1D
(ClassInternalSolutionOfMethodOfLines1D solution);

void destroyInternalSolutionOfMethodOfLines1D
               (ClassInternalSolutionOfMethodOfLines1D solution);
```

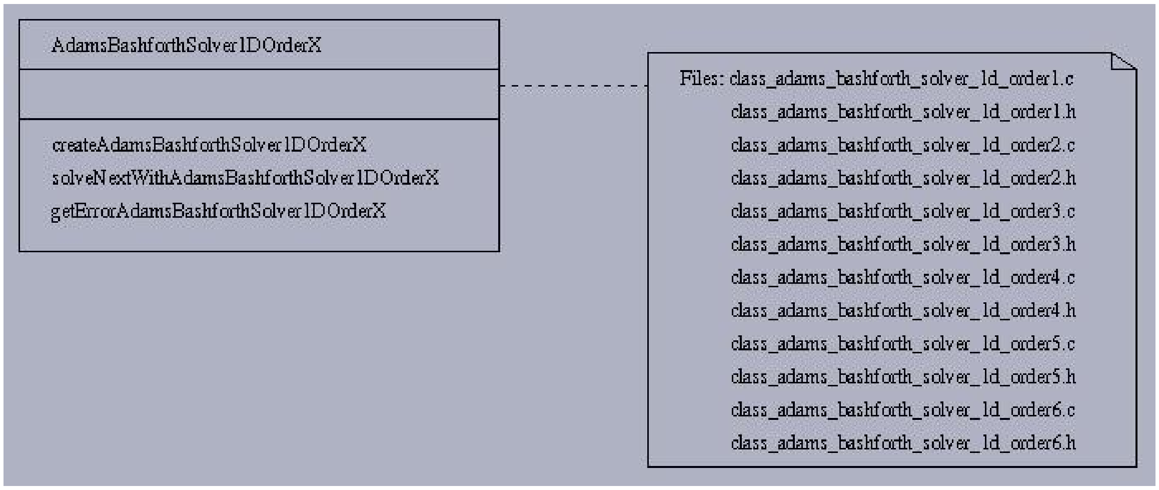

FunctionDiscretised2D
pointerToFunction
solution InternalSolutionOfMethodOfLines1D
createFunctionDiscretised2D
evaluateFunctionDiscretised2DAt
destroyFunctionDiscretised2D
Files: class_function_discretised_2d.c
class_function_discretised_2d.h

```
struct ClassFunctionDiscretised2D_rep
{
      double (*evaluate) (double, int, double, int, double,
ClassInternalSolutionOfMethodOfLines1D);
      ClassInternalSolutionOfMethodOfLines1D solution;
};// end struct

typedef struct ClassFunctionDiscretised2D_rep *ClassFunctionDiscretised2D;

ClassFunctionDiscretised2D createFunctionDiscretised2D
        (double (*evaluate)(double, int, double, int, double,
ClassInternalSolutionOfMethodOfLines1D));

double evaluateFunctionDiscretised2DAt(ClassFunctionDiscretised2D f, double x,
int xi, double y, int yi, double deltaY);

void setSolutionFunctionDiscretised2D (ClassFunctionDiscretised2D f,
ClassInternalSolutionOfMethodOfLines1D solution);

void changeEvaluationFunctionDiscretised2D (ClassFunctionDiscretised2D f,
                       double (*newEvaluate)(double, int, double, int, double,
ClassInternalSolutionOfMethodOfLines1D));

void destroyFunctionDiscretised2D (ClassFunctionDiscretised2D f);
```

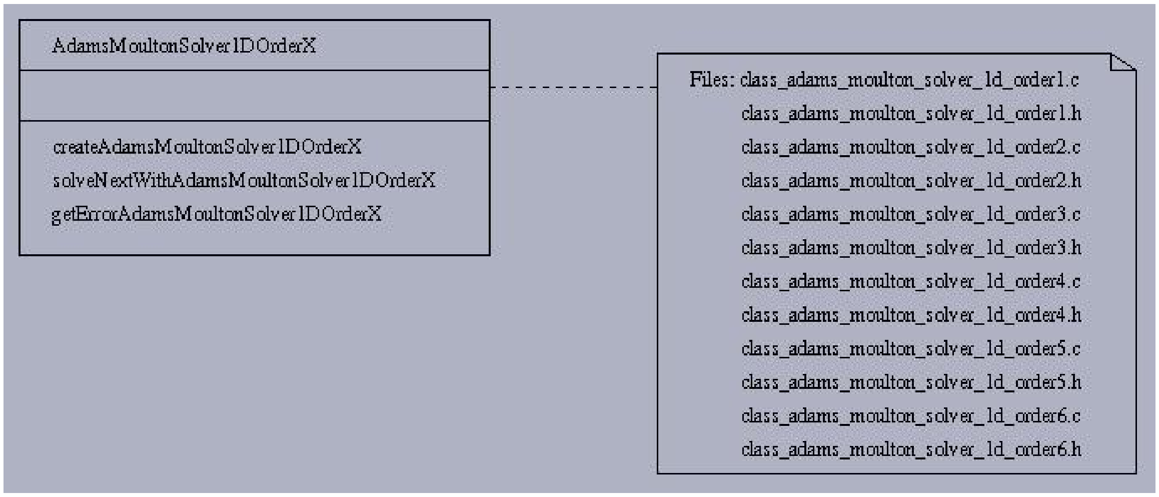

SolverOfInitialValueProblem1D
solution InternalSolutionOfMethodOfLines1D
functionFirstDerivate FunctionDiscritised2D
error double
isPredictorCorrector boolean
stepSize double
solverOrder int
createSolverOfInitialValueProblem1D
solveNextSolverOfInitialValueProblem1D
getErrorOfSolverOfInitialValueProblem1D
setStepSize
getStepSize
getOrderOfSolverOfInitialValueProblem1D
setSolutionOfSolverOfInitialValueProblem1D
destroySolverOfInitialValueProblem1D
Files: class_solver_of_initial_value_problem_1d.c
class_solver_of_initial_value_problem_1d.h

```
struct ClassSolverOfInitialValueProblem1D_rep
{
   ClassInternalSolutionOfMethodOfLines1D solution;
   ClassFunctionDiscretised2D functionFirstDerivate;
   void (*solveNextAt)(ClassInternalSolutionOfMethodOfLines1D solution,
                  ClassFunctionDiscretised2D functionFirstDerivate,
                  double stepSize, int yI);
   void (*solveNextPredictorCorrectorAt)(ClassInternalSolutionOfMethodOfLines1D
solution,
                                ClassFunctionDiscretised2D
functionFirstDerivate,
                                double stepSize, int yI, double *error);
   double (*getError) (double stepSize);
   double error;
   int isPredictorCorrector;
   int solverOrder;
};//end struct

typedef struct ClassSolverOfInitialValueProblem1D_rep
               *ClassSolverOfInitialValueProblem1D;

ClassSolverOfInitialValueProblem1D createSolverOfInitialValueProblem1D
          (void (*solveNextAt)(ClassInternalSolutionOfMethodOfLines1D solution,
           ClassFunctionDiscretised2D functionFirstDerivate,
           double stepSize, int yI),
           double (*getError)(double stepSize),
           int solverOrder);

ClassSolverOfInitialValueProblem1D
createPredictorCorrectorSolverOfInitialValueProblem1D
           (void (*solveNextAt)(ClassInternalSolutionOfMethodOfLines1D
solution,
                        ClassFunctionDiscretised2D functionFirstDerivate,
double stepSize,
                        int yI, double *error),
         double (*getError)(double stepSize),
         int solverOrder
         );

void solveNextSolverOfInitialValueProblem1D
```

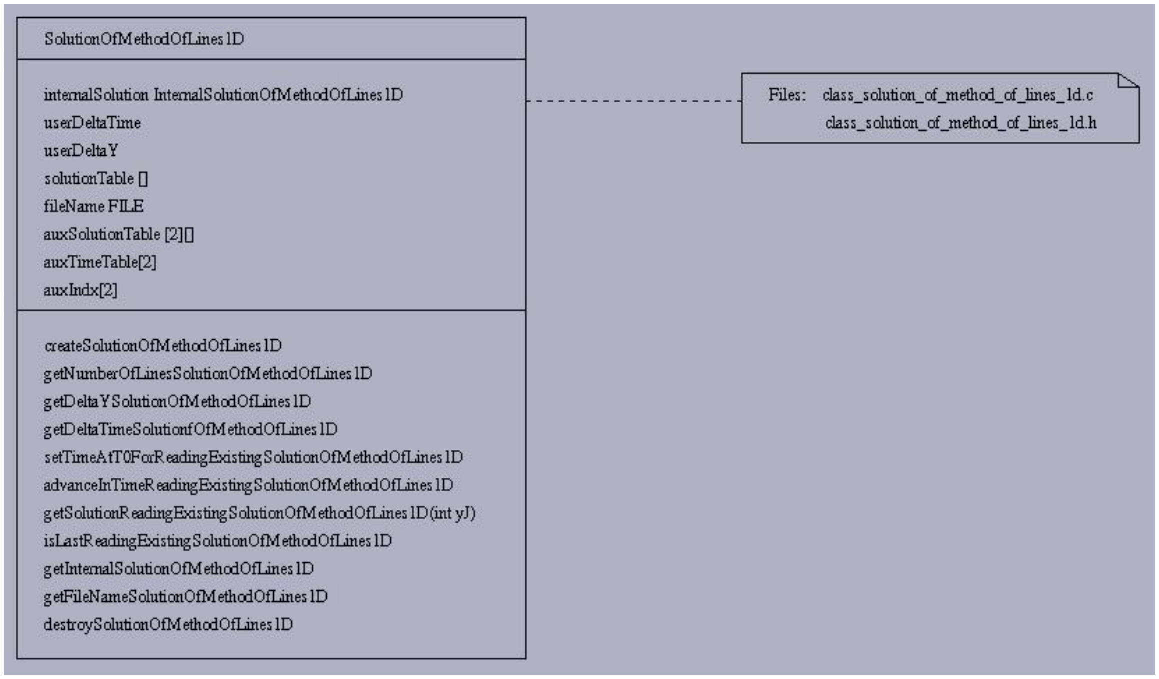
```
         (ClassSolverOfInitialValueProblem1D solver, int yJ);

double getErrorOfSolverOfInitialValueProblem1D
         (ClassSolverOfInitialValueProblem1D solver);

double getStepSize(ClassSolverOfInitialValueProblem1D solver);

int getOrderOfSolverOfInitialValueProblem1D
            (ClassSolverOfInitialValueProblem1D solver);

void setSolutionOfSolverOfInitialValueProblem1D
            (ClassSolverOfInitialValueProblem1D solver,
             ClassInternalSolutionOfMethodOfLines1D solution);

void setFunctionFirstDerivateSolverOfInitialValueProblem1D
            (ClassSolverOfInitialValueProblem1D solver,
             ClassFunctionDiscretised2D functionFirstDerivate);

void destroySolverOfInitialValueProblem1D(ClassSolverOfInitialValueProblem1D
solver);
```


AdamsBashforthSolver1DOrderX
createAdamsBashforthSolver1DOrderX
solveNextWithAdamsBashforthSolver1DOrderX
getErrorAdamsBashforthSolver1DOrderX
Files: class_adams_bashforth_solver_1d_order1.c
class_adams_bashforth_solver_1d_order1.h
class_adams_bashforth_solver_1d_order2.c
class_adams_bashforth_solver_1d_order2.h
class_adams_bashforth_solver_1d_order3.c
class_adams_bashforth_solver_1d_order3.h
class_adams_bashforth_solver_1d_order4.c
class_adams_bashforth_solver_1d_order4.h
class_adams_bashforth_solver_1d_order5.c
class_adams_bashforth_solver_1d_order5.h
class_adams_bashforth_solver_1d_order6.c
class_adams_bashforth_solver_1d_order6.h

```
ClassSolverOfInitialValueProblem1D createAdamsBashforthSolver1DOrder1();

void solveNextWithAdamsBashforthSolver1DOrder1At
(ClassInternalSolutionOfMethodOfLines1D solution,
             ClassFunctionDiscretised2D functionFirstDerivate, double stepSize,
             int yJ);

double getErrorAdamsBashforthSolver1DOrder1(double stepSize);
```

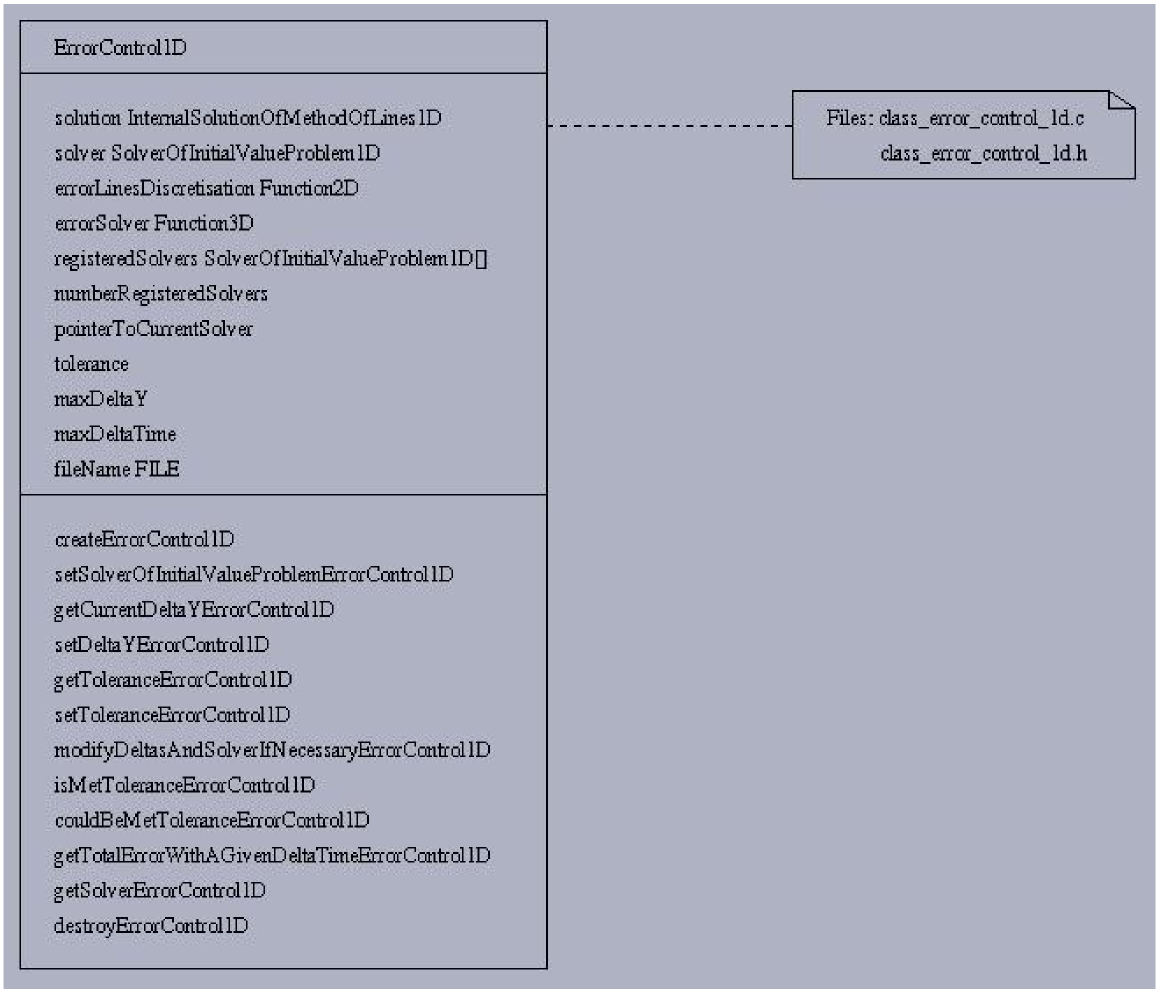

AdamsMoultonSolver1DOrderX
createAdamsMoultonSolver1DOrderX
solveNextWithAdamsMoultonSolver1DOrderX
getErrorAdamsMoultonSolver1DOrderX
Files: class_adams_moulton_solver_1d_order1.c
class_adams_moulton_solver_1d_order1.h
class_adams_moulton_solver_1d_order2.c
class_adams_moulton_solver_1d_order2.h
class_adams_moulton_solver_1d_order3.c
class_adams_moulton_solver_1d_order3.h
class_adams_moulton_solver_1d_order4.c
class_adams_moulton_solver_1d_order4.h
class_adams_moulton_solver_1d_order5.c
class_adams_moulton_solver_1d_order5.h
class_adams_moulton_solver_1d_order6.c
class_adams_moulton_solver_1d_order6.h

```
ClassSolverOfInitialValueProblem1D createAdamsMoultonSolver1DOrder1 ();

void
solveNextWithAdamsMoultonSolver1DOrder1At(ClassInternalSolutionOfMethodOfLines1
D solution,
             ClassFunctionDiscretised2D functionFirstDerivate, double stepSize,
             int yJ, double *error);

double getErrorAdamsMoultonSolver1DOrder1(double stepSize);
```


SolutionOfMethodOfLines1D
internalSolution InternalSolutionOfMethodOfLines1D
userDeltaTime
userDeltaY
solutionTable []
fileName FILE
auxSolutionTable [2][]
auxTimeTable[2]
auxIndx[2]
createSolutionOfMethodOfLines1D
getNumberOfLinesSolutionOfMethodOfLines1D
getDeltaYSolutionOfMethodOfLines1D
getDeltaTimeSolutionfOfMethodOfLines1D
setTimeAtT0ForReadingExistingSolutionOfMethodOfLines1D
advanceInTimeReadingExistingSolutionOfMethodOfLines1D
getSolutionReadingExistingSolutionOfMethodOfLines1D(int yJ)
isLastReadingExistingSolutionOfMethodOfLines1D
getInternalSolutionOfMethodOfLines1D
getFileNameSolutionOfMethodOfLines1D
destroySolutionOfMethodOfLines1D
Files: class_solution_of_method_of_lines_1d.c
class_solution_of_method_of_lines_1d.h

```
struct ClassSolutionOfMethodOfLines1D_rep
{
   ClassInternalSolutionOfMethodOfLines1D internalSolution;
   double userDeltaTime;
   double userDeltaY;
   int userNumberOfLines;
   double userCurrentTime;
   double *solutionTable;
   char *fileName;
   FILE *fp;
   double auxSolutionTable[2][maxNumberOfLines];
   double auxNumberOfLines[2];
   double auxTimeTable[2];
   int auxIndx[2];
};// end struct

typedef struct ClassSolutionOfMethodOfLines1D_rep
*ClassSolutionOfMethodOfLines1D;

ClassSolutionOfMethodOfLines1D createSolutionOfMethodOfLines1D
      (double userDeltaTime,
        double userDeltaY,
        int userNumberOfLines,
        ClassInterpolationMethod1D method,
        ClassFunction2D solutionAtT0,
        ClassFunction2D solutionAtY0,
        ClassFunction2D solutionAtYLast,
        double t0,
        double tLast,
        double y0,
        double yLast,
        int numberOfLines,
        double deltaY,
        double deltaTime,
        char *fileName);

double getTimeForNextSnapshot(ClassSolutionOfMethodOfLines1D solution);
```

```
int getNumberOfLinesSolutionOfMethodOfLines1D (ClassSolutionOfMethodOfLines1D
solution);

double getDeltaYSolutionOfMethodOfLines1D (ClassSolutionOfMethodOfLines1D
solution);

double getDeltaTimeSolutionOfMethodOfLines1D (ClassSolutionOfMethodOfLines1D
solution);

void setTimeAtT0ForReadingExistingSolutionOfMethodOfLines1D
(ClassSolutionOfMethodOfLines1D  solution);

void advanceInTimeReadingExistingSolutionOfMethodOfLines1D
(ClassSolutionOfMethodOfLines1D  solution);

double getSolutionReadingExistingSolutionOfMethodOfLines1D
(ClassSolutionOfMethodOfLines1D  solution, int yJ);

int isLastReadingExistingSolutionOfMethodOfLines1D
(ClassSolutionOfMethodOfLines1D  solution);

ClassInternalSolutionOfMethodOfLines1D
getInternalSolutionSolutionOfMethodOfLines1D (ClassSolutionOfMethodOfLines1D
solution);

char * getFileNameSolutionOfMethodOfLines1D (ClassSolutionOfMethodOfLines1D
solution);

void destroySolutionOfMethodOfLines1D(ClassSolutionOfMethodOfLines1D solution);
```

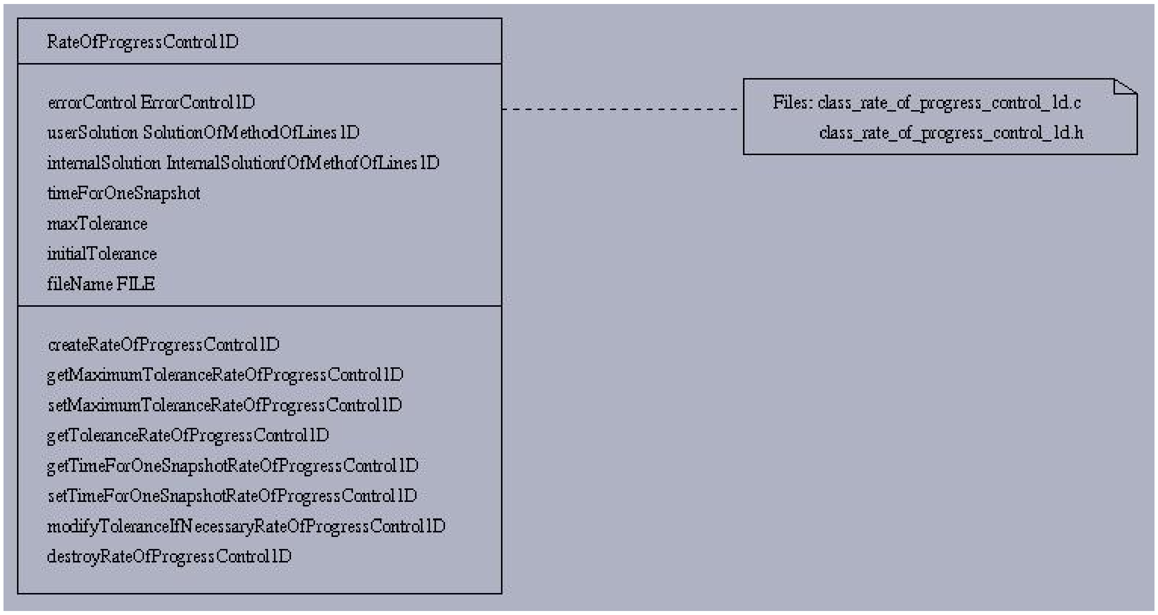

ErrorControl1D

solution InternalSolutionOfMethodOfLines1D
solver SolverOfInitialValueProblem1D
errorLinesDiscretisation Function2D
errorSolver Function3D
registeredSolvers SolverOfInitialValueProblem1D[]
numberRegisteredSolvers
pointerToCurrentSolver
tolerance
maxDeltaY
maxDeltaTime
fileName FILE

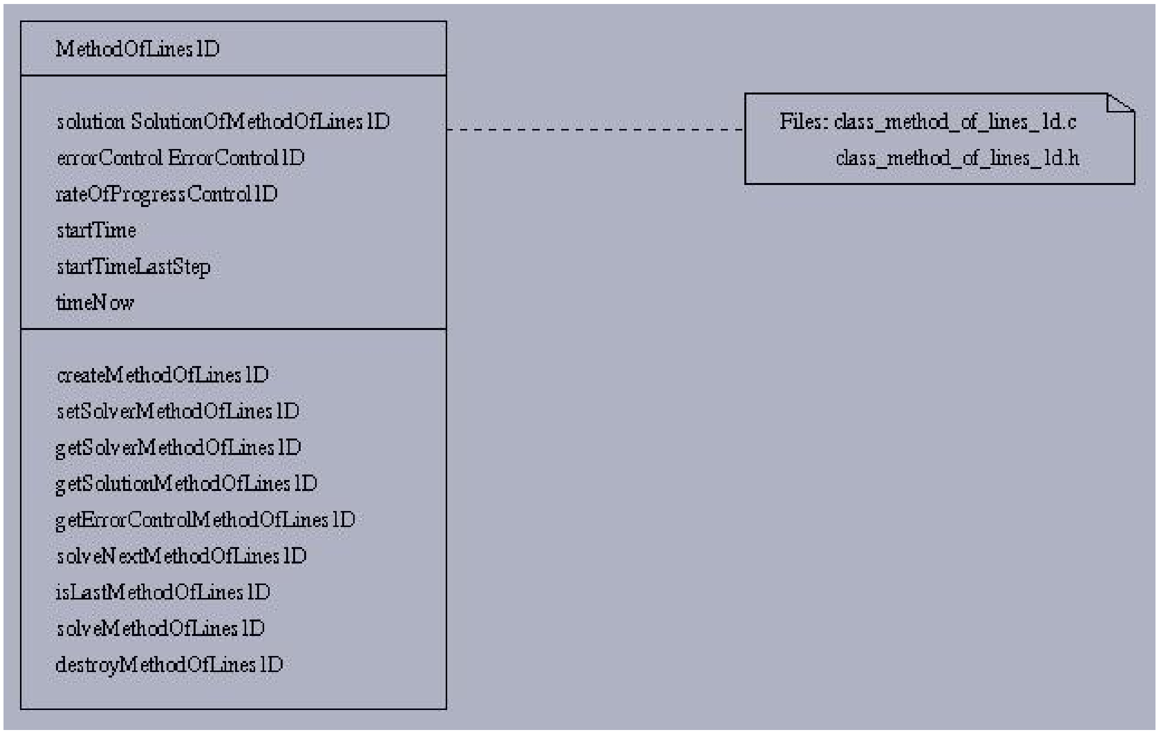
createErrorControl1D
setSolverOfInitialValueProblemErrorControl1D
getCurrentDeltaYErrorControl1D
setDeltaYErrorControl1D
getToleranceErrorControl1D
setToleranceErrorControl1D
modifyDeltasAndSolverIfNecessaryErrorControl1D
isMetToleranceErrorControl1D
couldBeMetToleranceErrorControl1D
getTotalErrorWithAGivenDeltaTimeErrorControl1D
getSolverErrorControl1D
destroyErrorControl1D

Files: class_error_control_1d.c
class_error_control_1d.h

```
struct ClassErrorControl1D_rep
{
   ClassInternalSolutionOfMethodOfLines1D solution;
   ClassSolverOfInitialValueProblem1D solver;
   int pointerToCurrentSolver;
   ClassFunction2D functionLinesDiscretisationError;
   ClassFunction3D functionSolverError;
   ClassSolverOfInitialValueProblem1D *registeredSolvers;
   int numRegisteredSolvers;
   double tolerance;
   double maxDeltaY;
   double maxDeltaTime;
   char *fileName;
};// end struct

typedef struct ClassErrorControl1D_rep *ClassErrorControl1D;

ClassErrorControl1D createErrorControl1D
(ClassInternalSolutionOfMethodOfLines1D solution,
         ClassSolverOfInitialValueProblem1D initialSolver,
         int keySolver,
         ClassFunction2D functionLinesDiscretisationError,
         ClassFunction3D functionSolverError,
         ClassSolverOfInitialValueProblem1D *registeredSolvers,
         int numRegisteredSolvers, double tolerance, double initialDeltaY,
         double maxDeltaY, double initialDeltaTime, double maxDeltaTime,
         char *userSolutionFileName);

void setSolverOfInitialValueProblemErrorControl1D (ClassErrorControl1D
errorControl,
                ClassSolverOfInitialValueProblem1D solver, int keySolver);

double getCurrentDeltaTimeErrorControl1D(ClassErrorControl1D errorControl);

void setDeltaTimeErrorControl1D(ClassErrorControl1D errorControl, double
newDeltaTime);

double getCurrentDeltaYErrorControl1D(ClassErrorControl1D errorControl);

void setDeltaYErrorControl1D (ClassErrorControl1D errorControl, double
newDeltaY);

double getToleranceErrorControl1D (ClassErrorControl1D errorControl);

void setToleranceErrorControl1D (ClassErrorControl1D errorControl, double
newTolerance);

void modifyDeltasAndSolverIfNecessaryErrorControl1D(ClassErrorControl1D
errorControl);

int isMetToleranceErrorControl1D(ClassErrorControl1D errorControl);

int couldBeMetToleranceErrorControl1D (ClassErrorControl1D errorControl);

double getTotalErrorWithAGivenDeltaTimeErrorControl1D (ClassErrorControl1D
errorControl, double deltaTime);

ClassSolverOfInitialValueProblem1D getSolverErrorControl(ClassErrorControl1D
errorControl);


void destroyErrorControl1D (ClassErrorControl1D errorControl);
```


RateOfProgressControl1D
errorControl ErrorControl1D
userSolution SolutionOfMethodOfLines1D
internalSolution InternalSolutionfOfMethofOfLines1D
timeForOneSnapshot
maxTolerance
initialTolerance
fileName FILE
createRateOfProgressControl1D
getMaximumToleranceRateOfProgressControl1D
setMaximumToleranceRateOfProgressControl1D
getToleranceRateOfProgressControl1D
getTimeForOneSnapshotRateOfProgressControl1D
setTimeForOneSnapshotRateOfProgressControl1D
modifyToleranceIfNecessaryRateOfProgressControl1D
destroyRateOfProgressControl1D
Files: class_rate_of_progress_control_1d.c
class_rate_of_progress_control_1d.h

```
struct ClassRateOfProgressControl1D_rep
{
   ClassErrorControl1D errorControl;
   ClassSolutionOfMethodOfLines1D userSolution;
   ClassInternalSolutionOfMethodOfLines1D internalSolution;
   double timeForOneSnapshot;
   double maxTolerance;
   double initialTolerance;
   char *fileName;
}; // end struct

typedef struct ClassRateOfProgressControl1D_rep *ClassRateOfProgressControl1D;

ClassRateOfProgressControl1D createRateOfProgressControl1D
                  (ClassErrorControl1D errorControl,
                   ClassSolutionOfMethodOfLines1D userSolution,
                   double timeForOneSnapshot,
                   double maxTolerance,
                   double initialTolerance);

double getMinimumToleranceRateOfProgressControl1D (ClassRateOfProgressControl1D
ropControl);

void setMinimumToleranceRateOfProgressControl1D (ClassRateOfProgressControl1D
ropControl,
                    double minTolerance);

double getTimeForOneSnapshotRateOfProgressControl1D
(ClassRateOfProgressControl1D ropControl);

void setTimeForOneSnapshotRateOfProgressControl1D (ClassRateOfProgressControl1D
ropControl,
                    double timeForOneSnapshot);

void modifyToleranceIfNecessaryRateOfProgressControl1D
(ClassRateOfProgressControl1D ropControl,
                                          double executionTime, double
lastStepExecutionTime);

void destroyRateOfProgressControl1D (ClassRateOfProgressControl1D ropControl);
```



```
struct ClassMethodOfLines1D_rep
{
   ClassSolutionOfMethodOfLines1D userSolution;
   ClassInternalSolutionOfMethodOfLines1D internalSolution;
   ClassRateOfProgressControl1D  ropControl;
   ClassErrorControl1D errorControl;
};// end struct

typedef struct ClassMethodOfLines1D_rep *ClassMethodOfLines1D;


ClassMethodOfLines1D createMethodOfLines1D (ClassFunction2D solutionAtT0,
            ClassFunction2D solutionAtY0,
            ClassFunction2D solutionAtYLast,
            ClassFunctionDiscretised2D functionFirstDerivate,
            double t0,
            double tLast,
            double y0,
            double yLast,
            double userDeltaTime,
            double userDeltaY,
            double tolerance,
            double initialDeltaTime,
            double initialDeltaY,
            double maxTolerance,
            double maxDeltaTime,
            double maxDeltaY,
            double rateOfProgressPerSecond,
            ClassInterpolationMethod1D method,
            ClassSolverOfInitialValueProblem1D registeredSolvers[],
            int numberOfRegisteredSolvers,
            int keyInitialSolver,
            ClassFunction2D functionLinesDiscretisationError,
            ClassFunction3D functionSolverError,
            char *solutionFileName);
```

```
void setSolver (ClassMethodOfLines1D mol,
                ClassSolverOfInitialValueProblem1D newSolver);

ClassSolverOfInitialValueProblem1D getSolver (ClassMethodOfLines1D mol);


void setSolution (ClassMethodOfLines1D mol, ClassSolutionOfMethodOfLines1D
solution);

ClassSolutionOfMethodOfLines1D getSolution (ClassMethodOfLines1D mol);


void setErrorControl (ClassMethodOfLines1D mol,  ClassErrorControl1D
controler);

ClassErrorControl1D getErrorControl (ClassMethodOfLines1D mol);

void solveNext (ClassMethodOfLines1D mol);

void solveMethodOfLines1D (ClassMethodOfLines1D mol);

void destroyMethodOfLines1D(ClassMethodOfLines1D mol);
```