

AMICI

Generated by Doxygen 1.8.10

Thu Nov 5 2015 15:37:48

## Contents

<b>1</b>	<b>AMICI 0.1 General Documentation</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Availability . . . . .	2
1.3	Installation . . . . .	2
<b>2</b>	<b>Model Definition &amp; Simulation</b>	<b>3</b>
2.1	Model Definition . . . . .	3
2.2	Model Compilation . . . . .	6
2.3	Model Simulation . . . . .	6
<b>3</b>	<b>Code Organization</b>	<b>9</b>
<b>4</b>	<b>Class Index</b>	<b>9</b>
4.1	Class List . . . . .	9
<b>5</b>	<b>Class Documentation</b>	<b>10</b>
5.1	amimodel Class Reference . . . . .	10
5.2	ExpData Struct Reference . . . . .	22
5.3	ReturnData Struct Reference . . . . .	22
5.4	TempData Struct Reference . . . . .	24
5.5	UserData Struct Reference . . . . .	26
<b>6</b>	<b>File Documentation</b>	<b>28</b>
6.1	amiwrap.c File Reference . . . . .	28
6.2	amiwrap.m File Reference . . . . .	30
6.3	src/amici.c File Reference . . . . .	31
6.4	src/spline.c File Reference . . . . .	38
6.5	src/symbolic_functions.c File Reference . . . . .	41
	<b>Index</b>	<b>67</b>

## 1 AMICI 0.1 General Documentation

### 1.1 Introduction

AMICI is a MATLAB interface for the **SUNDIALS** solvers CVODES (for ordinary differential equations) and IDAS (for algebraic differential equations). AMICI allows the user to specify differential equation models in terms of symbolic variables in MATLAB and automatically compiles such models as .mex simulation files. In contrast to the SUNDIALSTB interface, all necessary functions are transformed into native C code, which allows for a significantly faster numerical integration. Beyond forward integration, the compiled simulation file also allows for first and second order forward sensitivity analysis, steady state sensitivity analysis and adjoint sensitivity analysis for likelihood based output functions.

The interface was designed to provide routines for efficient gradient computation in parameter estimation of biochemical reaction models but is also applicable to a wider range of differential equation constrained optimization

problems.

## 1.2 Availability

The sources for AMICI are accessible as

- Source [tarball](#)
- Source [zipball](#)
- GIT repository on [github](#)

Once you've obtained your copy check out the [Installation](#)

### 1.2.1 Obtaining AMICI via the GIT versioning system

In order to always stay up-to-date with the latest AMICI versions, simply pull it from our GIT repository and recompile it when a new release is available. For more information about GIT checkout their [website](#)

The GIT repository can currently be found at <https://github.com/FFroehlich/AMICI> and a direct clone is possible via

```
git clone https://github.com/FFroehlich/AMICI.git AMICI
```

### 1.2.2 License Conditions

This software is available under the [BSD license](#)

Copyright (c) 2015, Fabian Fröhlich and Jan Hasenauer All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 1.3 Installation

If AMICI was downloaded as a zip, it needs to be unpacked in a convenient directory. If AMICI was obtained via cloning of the git repository, no further unpacking is necessary.

To use AMICI, start MATLAB and add the AMICI directory to the MATLAB path. To add all toolbox directories to the MATLAB path, execute the matlab script

```
installToolbox.m
```

To store the installation for further MATLAB session, the path can be saved via

```
savepath
```

For the compilation of .mex files, MATLAB needs to be configured with a working C compiler. The C compiler needs to be installed and configured via:

```
mex -setup c
```

For a list of supported compilers we refer to the mathworks documentation: [mathworks.de](https://mathworks.de)

The tools SUNDIALS and SuiteSparse shipped with AMICI do **not** require further installation.

AMICI uses the following packages from SUNDIALS:

**CVODES:** the sensitivity-enabled ODE solver in SUNDIALS. Radu Serban and Alan C. Hindmarsh. *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2005. [PDF](#)

### IDAS

AMICI uses the following packages from SuiteSparse:

**Algorithm 907: KLU**, A Direct Sparse Solver for Circuit Simulation Problems. Timothy A. Davis, Ekanathan Palamadai Natarajan, *ACM Transactions on Mathematical Software*, Vol 37, Issue 6, 2010, pp 36:1 - 36:17. [PDF](#)

**Algorithm 837: AMD**, an approximate minimum degree ordering algorithm, Patrick R. Amestoy, Timothy A. Davis, Iain S. Duff, *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 381 - 388. [PDF](#)

**Algorithm 836: COLAMD**, a column approximate minimum degree ordering algorithm, Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 377 - 380. [PDF](#)

## 2 Model Definition & Simulation

In the following we will give a detailed overview how to specify models in AMIWRAP and how to call the generated simulation files.

### 2.1 Model Definition

This guide will guide the user on how to specify models in MATLAB. For example implementations see the examples in the example directory.

#### 2.1.1 Header

The model definition needs to be defined as a function which returns a struct with all symbolic definitions and options.

```
function [model] = example_model_syms()
```

#### 2.1.2 Options

Set the options by specifying the respective field of the modelstruct

```
model.(fieldname) = (value)
```

The options specify default options for simulation, parametrisation and compilation. All of these options are optional.

field	description	default
.atol	absolute integration tolerance	1e-8
.rtol	relative integration tolerance	1e-8
.maxsteps	maximal number integration steps	1e-8
.param	parametrisation 'log'/'log10'/'lin'	'lin'
.debug	flag to compile with debug symbols	false
.forward	flag to activate forward sensitivities	true
.adjoint	flag to activate adjoint sensitivities	true

When set to true, the fields 'noforward' and 'noadjoint' will speed up the time required to compile the model but also disable the respective sensitivity computation.

### 2.1.3 States

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily.

```
syms state1 state2 state3
```

Create the state vector containing all states:

```
x = [ state1 state2 state3 ];
```

### 2.1.4 Parameters

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities **will be derived** for all parameters.

```
syms param1 param2 param3 param4 param5 param6
```

Create the parameters vector

```
p = [ param1 param2 param3 param4 param5 param6 ];
```

### 2.1.5 Constants

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities with respect to constants **will not be derived**.

```
syms const1 const2
```

Create the parameters vector

```
k = [ const1 const2 ];
```

### 2.1.6 Differential Equation

For time-dependent differential equations you can specify a symbolic variable for time. This **needs** to be denoted by t.

```
syms t
```

Specify the right hand side of the differential equation f or xdot

```
xdot(1) = [ const1 - param1*state1 ];
xdot(2) = [ +param2*x(1) + dirac(t-param3) - const2*x(2) ];
xdot(3) = [ param4*x(2) ];
```

or

```
f(1) = [ const1 - param1*state1 ];
f(2) = [ +param2*x(1) + dirac(t-param3) - const2*x(2) ];
f(3) = [ 2*x(2) ];
```

The specification of  $f$  or  $\dot{x}$  may depend on [States](#), [Parameters](#) and [Constants](#).

For DAEs also specify the mass matrix.

```
M = [1, 0, 0;...
      0, 1, 0;...
      0, 0, 0];
```

The specification of  $M$  may depend on parameters and constants.

For ODEs the integrator will solve the equation  $\dot{x} = f$  and for DAEs the equations  $M \cdot \dot{x} = f$ . AMICI will decide whether to use CVODES (for ODEs) or IDAS (for DAEs) based on whether the mass matrix is defined or not.

In the definition of the differential equation you can use certain symbolic functions. For a full list of available functions see [symbolic\\_functions.c](#).

Dirac functions can be used to cause a jump in the respective states at the specified time-point. This is typically used to model injections, or other external stimuli. Spline functions can be used to model time/state dependent response with unknown time/state dependence.

### 2.1.7 Initial Conditions

Specify the initial conditions. These may depend on [Parameters](#) or [Constants](#) and must have the same size as  $x$ .

```
x0 = [ param4, 0, 0 ];
```

### 2.1.8 Observables

Specify the observables. These may depend on [Parameters](#) and [Constants](#).

```
y(1) = state1 + state2;
y(2) = state3 - state2;
```

In the definition of the observable you can use certain symbolic functions. For a full list of available functions see [symbolic\\_functions.c](#). Dirac functions in observables will have no effect.

### 2.1.9 Events

Specifying events is optional. Events are specified in terms of a root function. Events are defined to happen at the roots of the root function.

```
root(1) = state1 - state2;
```

Events may depend on [States](#), [Parameters](#) and [Constants](#) but **not** on [Observables](#)

### 2.1.10 Standard Deviation

Specifying of standard deviations is optional. It only has an effect when computing adjoint sensitivities. It allows the user to specify standard deviations of experimental data for [Observables](#) and [Events](#).

Standard deviation for observable data is denoted by `sigma_y`

```
sigma_y(1) = param5;
```

Standard deviation for event data is denoted by `sigma_y`

```
sigma_t(1) = param6;
```

Both `sigma_y` and `sigma_t` can either be a scalar or of the same dimension as the [Observables](#) / [Events](#) function. They can depend on time and [Parameters](#) but must not depend on the [States](#) or [Observables](#). The values provided in `sigma_y` and `sigma_t` will only be used if the value in `Sigma_Y` or `Sigma_T` in the user-provided data struct is NaN. See [Model Simulation](#) for details.

### 2.1.11 Attach to Model Struct

Eventually all symbolic expressions need to be attached to the model struct.

```
model.sym.x = x;
model.sym.k = k;
model.sym.root = root;
model.sym.xdot = xdot;
% or
model.sym.f = f;
model.sym.M = M; %only for DAEs
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;
model.sym.sigma_y = sigma_y;
model.sym.sigma_t = sigma_t;
```

## 2.2 Model Compilation

The model can then be compiled by calling `amiwrap`:

```
amiwrap(modelname, 'example_model_syms', dir, o2flag)
```

Here `modelname` should be a string defining the modelname, `dir` should be a string containing the path to the directory in which simulation files should be placed and `o2flag` is a flag indicating whether second order sensitivities should also be compiled. The user should make sure that the previously defined function 'example\_model\_syms' is in the user path. Alternatively, the user can also call the function 'example\_model\_syms'

```
[model] = example_model_syms()
```

and subsequently provide the generated struct to `amiwrap()`, instead of providing the symbolic function:

```
amiwrap(modelname, model, dir, o2flag)
```

In a similar fashion, the user could also generate multiple model and pass them directly to `amiwrap()` without generating respective model definition scripts.

See also

[amiwrap\(\)](#)

## 2.3 Model Simulation

After the call to `amiwrap()` two files will be placed in the specified directory. One is a `am_modelname.mex` and the other is `simulate_modelname.m`. The mex file should never be called directly. Instead the MATLAB script, which acts as a wrapper around the .mex simulation file should be used.

The `simulate_modelname.m` itself carries extensive documentation on how to call the function, what it returns and what additional options can be specified. In the following we will give a short overview of possible function calls.

### 2.3.1 Integration

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `sol.status` flag. Negative values indicated failed integration. The states will then be available as `sol.x`. The observables will then be available as `sol.y`. The events will then be available as `sol.root`. If no event occurred there will be an event at the end of the considered interval with the final value of the root function stored in `sol.rval`.

Alternatively the integration call also be called via

```
[status,t,x,y] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `status` flag. Negative values indicated failed integration. The states will then be available as `x`. The observables will then be available as `y`. No event output will be given.

### 2.3.2 Forward Sensitivities

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Set the sensitivity computation to forward sensitivities and Integrate:

```
options.sensi = 1;  
options.forward = true;  
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `sol.status` flag. Negative values indicated failed integration. The states will then be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will then be available as `sol.y`, with the derivative with respect to the parameters in `sol.sy`. The events will then be available as `sol.root`, with the derivative with respect to the parameters in `sol.sroot`. If no event occurred there will be an event at the end of the considered interval with the final value of the root function stored in `sol.rootval`, with the derivative with respect to the parameters in `sol.srootval`.

Alternatively the integration call also be called via

```
[status,t,x,y,sx,sy] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `status` flag. Negative values indicated failed integration. The states will then be available as `x`, with derivative with respect to the parameters in `sx`. The observables will then be available as `y`, with derivative with respect to the parameters in `sy`. No event output will be given.



### 2.3.3 Adjoint Sensitivities

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Set the sensitivity computation to adjoint sensitivities:

```
options.sensi = 1;
options.adjoint = true;
```

Define Experimental Data:

```
D.Y = [NaN(1,2)],ones(length(t)-1,2)];
D.Sigma_Y = [0.1*ones(length(t)-1,2),NaN(1,2)];
D.T = ones(1,1);
D.Sigma_T = NaN;
```

The NaN values in Sigma\_Y and Sigma\_T will be replaced by the specification in [Standard Deviation](#). Data points with NaN value will be completely ignored.

Generate a constants vector:

```
kappa = ones(2,1);
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The integration status will be indicated by the sol.status flag. Negative values indicated failed integration. The log-likelihood will then be available as sol.llh and the derivative with respect to the parameters in sol.sllh. Notice that for adjoint sensitivities no state, observable and event sensitivities will be available. Yet this approach can be expected to be significantly faster for systems with a large number of parameters.

### 2.3.4 Steady State Sensitivities

This will compute state sensitivities according to the formula  $s_k^x = -\left(\frac{\partial f}{\partial x}\right)^{-1} \frac{\partial f}{\partial \theta_k}$

In the current implementation this formulation does not allow for conservation laws as this would result in a singular Jacobian.

Define a final timepoint t:

```
t = 100
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Set the sensitivity computation to steady state sensitivities:

```
options.sensi = 1;
options.ss = 1;
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The states will then be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will then be available as `sol.y`, with the derivative with respect to the parameters in `sol.sy`. Notice that for steady state sensitivities no event sensitivities will be available. For the accuracy of the computed derivatives it is essential that the system is sufficiently close to a steady state. This can be checked by examining the right hand side of the system at the final time-point via `sol.xdot`.

## 3 Code Organization

In the following we will briefly outline what happens when a model is compiled. For a more detailed description we refer the reader to the documentation of the individual functions.

After specifying a model (see [Model Definition](#)) the user will typically compile the model by invoking `amiwrap()`. `amiwrap()` first instantiates an object of the class `amimodel`. The properties of this object are initialised based on the user-defined model. If the `o2flag` is active, all subsequent computations will also be carried out on the augmented system, which also includes the equations for forward sensitivities. This allows the computation of second order sensitivities in a forward-forward approach. A forward-adjoint approach will be implemented in the future.

The `sym` and `fun` fields of this object will then be populated by `amimodel::parseModel()`. The `amimodel::sym` field contains all necessary symbolic expression while the `amimodel::fun` field will contain flags for all functions whether the C code for a certain function needs to be regenerated or not. The set of functions to be considered will depend on the user specification of the model fields `amimodel::adjoint` and `amimodel::forward` (see [Options](#)) as well as the employed solver (CVODES or IDAS, see [Differential Equation](#)). For all considered functions `amimodel::parseModel()` will check their dependencies via `amimodel::checkDeps()`. These dependencies are a subset of the user-specified fields of `amimodel::sym` (see [Attach to Model Struct](#)). `amimodel::parseModel()` compares the hashes of all dependencies against the `amimodel::HTable` of possible previous compilations and will only compute necessary symbolic expressions if changes in these fields occurred. If changes in the dependencies occurred, also the respective subfield in `amimodel::fun` be set to 1, indicating the necessity of regeneration of respective C code.

For all functions for which `amimodel::fun` is set to 1, `amimodel::generateC()` will generate C files. These files together with their respective header files will be placed in `$AMICIDIR/models/modelname`. `amimodel::generateC()` will also generate `wrapfunctions.h` and `wrapfunctions.c`. These files define and declare model unspecific wrapper functions around model specific functions. This construction allows us to use to build multiple different models against the same simulation routines by linking different realisations of these wrapper functions.

All the generated C functions are subsequently compiled by `amimodel::compileC()`. For all functions individual object files are created to reduce the computation cost of code optimization. Moreover necessary code from sundials and SuiteSparse is compiled as object files and placed in `/models/mexext`, where `mexext` stands for the string returned by matlab to the command `mexext`. The mex simulation file is compiled from `amiwrap.c`, linked against all object necessary of sundials, SuiteSparse and model specific functions. Depending on the required solver, the compilation will either include `cvodewrap.h` or `idawrap.h`. These files implement solver specific realisations of the AMI... functions used in `amiwrap.c` and `amici.c`. This allows the use of the same simulation routines for both CVODES and IDAS.

## 4 Class Index

### 4.1 Class List

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

#### `amimodel`

**Amimodel is the object in which all model definitions are stored**

**10**

<b>ExpData</b>	
Struct that carries all information about experimental data	22
<b>ReturnData</b>	
Struct that stores all data which is later returned by the mex function	22
<b>TempData</b>	
Struct that provides temporary storage for different variables	24
<b>UserData</b>	
Struct that stores all user provided data	26

## 5 Class Documentation

### 5.1 amimodel Class Reference

amimodel is the object in which all model definitions are stored

#### Public Member Functions

- **amimodel** (::string symfun,::string **modelName**)  
*constructor of the amimodel class. this function initializes the model object based on the provided symfun and modelName*
- mlhsInnerSubst< matlabtypesubstitute > **parseModel** ()  
*parseModel parses the this and computes all necessary symbolic expressions.*
- mlhsInnerSubst< matlabtypesubstitute > **generateC** ()  
*generateC generates the c files which will be used in the compilation.*
- mlhsInnerSubst< matlabtypesubstitute > **compileC** ()  
*compileC compiles the mex simulation file*
- mlhsInnerSubst< matlabtypesubstitute > **writeCcode\_sensi** (::symbolic svar,::fileid fid)  
*writeCcode\_sensi is a wrapper for writeCcode which loops over parameters and reduces overhead by check nonzero values*
- mlhsInnerSubst< matlabtypesubstitute > **writeCcode** (::string funstr,::fileid fid, matlabtypesubstitute ip, matlabtypesubstitute jp)  
*writeCcode is a wrapper for this.gccode which initialises data and reduces overhead by check nonzero values*
- mlhsInnerSubst< matlabtypesubstitute > **gccode** (::symbolic csym,::string funstr,::string cvar,::fileid fid)  
*gccode transforms symbolic expressions into c code and writes the respective expression into a specified file*
- mlhsInnerSubst< matlabtypesubstitute > **generateM** (::amimodel amimodelo2)  
*generateM generates the matlab wrapper for the compiled C files.*
- mlhsInnerSubst< matlabtypesubstitute > **getFun** (::struct HTable,::string funstr)  
*getFun generates symbolic expressions for the requested function.*
- mlhsInnerSubst< matlabtypesubstitute > **getFunDeps** (::string funstr)  
*getDeps returns the dependencies for the requested function/expression.*
- mlhsInnerSubst< matlabtypesubstitute > **getFunArgs** (::string funstr)  
*getfunargs returns a string which contains all input arguments for the function fun.*
- mlhsInnerSubst< matlabtypesubstitute > **getFunCVar** (::string funstr)  
*getDeps returns the c variable for the requested function.*
- mlhsInnerSubst< matlabtypesubstitute > **getFunSVar** (::string funstr)  
*getDeps returns the symbolic variable for the requested function.*
- mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >,mlhsInnerSubst< matlabtypesubstitute > > **checkDeps** (::struct HTable,::cell deps)  
*checkDeps checks the dependencies of functions and populates sym fields if necessary*

- `mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >,mlhsInnerSubst< matlabtypesubstitute > > loadOldHashes ()`  
*loadOldHashes loads information from a previous compilation of the model.*
- `mlhsInnerSubst< matlabtypesubstitute > augmento2 ()`  
*augmento2 augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward approach later on.*

#### Public Attributes

- `::struct sym`  
*symbolic definition struct*
- `::struct fun`  
*struct which stores information for which functions c code needs to be generated*
- `::struct strsym`  
*short names for symbolic variables*
- `::string modelname`  
*name of the model*
- `::struct HTable`  
*struct that contains hash values for the symbolic model definitions*
- `::double atol = 1e-8`  
*default absolute tolerance*
- `::double rtol = 1e-8`  
*default relative tolerance*
- `::int maxsteps = 1e4`  
*default maximal number of integration steps*
- `::bool debug = false`  
*flag indicating whether debugging symbols should be compiled*
- `::bool adjoint = true`  
*flag indicating whether adjoint sensitivities should be enabled*
- `::bool forward = true`  
*flag indicating whether forward sensitivities should be enabled*
- `::double t0 = 0`  
*default initial time*
- `::string wtype`  
*type of wrapper (cvodes/idas)*
- `::int nx`  
*number of states*
- `::int nxtrue = 0`  
*number of original states for second order sensitivities*
- `::int ny`  
*number of observables*
- `::int nytrue = 0`  
*number of original observables for second order sensitivities*
- `::int nr`  
*number of events*
- `::int ndisc`  
*number of discontinuities*
- `::int np`  
*number of parameters*
- `::int nk`  
*number of constants*

- `::*int id`  
*flag for DAEs*
- `::int ubw`  
*upper Jacobian bandwidth*
- `::int lbw`  
*lower Jacobian bandwidth*
- `::int nnz`  
*number of nonzero entries in Jacobian*
- `::*int sparseidx`  
*dataindexes of sparse Jacobian*
- `::*int rowvals`  
*rowindexes of sparse Jacobian*
- `::*int colptrs`  
*columnindexes of sparse Jacobian*
- `::*int sparseidxB`  
*dataindexes of sparse Jacobian*
- `::*int rowvalsB`  
*rowindexes of sparse Jacobian*
- `::*int colptrsB`  
*columnindexes of sparse Jacobian*
- `::*cell funs`  
*cell array of functions to be compiled*
- `::string coptim = "-O3"`  
*optimisation flag for compilation*
- `::string param = "lin"`  
*default parametrisation*

### 5.1.1 Detailed Description

Definition at line 17 of file amimodel.m.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 `amimodel ( ::string symfun, ::string modelname )`

##### Parameters

<i>symfun</i>	this is the string to the function which generates the modelstruct. You can also directly pass the struct here
<i>modelname</i>	name of the model

Definition at line 279 of file amimodel.m.

### 5.1.3 Member Function Documentation

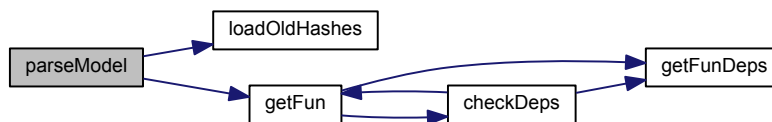
#### 5.1.3.1 `mlhsInnerSubst<::amimodel> parseModel ( )`

##### Return values

<i>this</i>	updated model definition object
-------------	---------------------------------

Definition at line 18 of file parseModel.m.

Here is the call graph for this function:



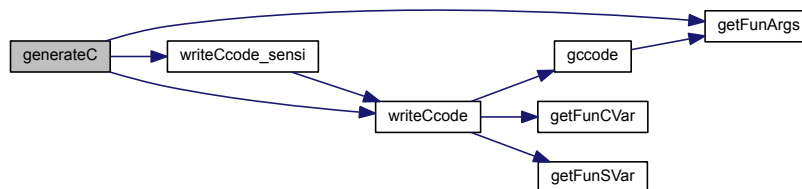
### 5.1.3.2 mlhsInnerSubst<::amimodel> generateC ( )

Return values

<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file generateC.m.

Here is the call graph for this function:



### 5.1.3.3 mlhsInnerSubst<::amimodel> compileC ( )

Return values

<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file compileC.m.

### 5.1.3.4 mlhsInnerSubst<::amimodel> writeCcode\_sensi ( ::symbolic svar, ::fileid fid )

Parameters

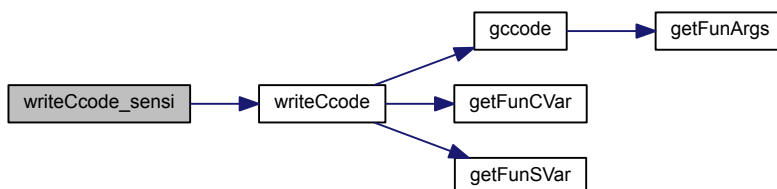
<i>svar</i>	symbolic variable which is later on passed to gcode
<i>fid</i>	file id in which the final expression is written

Return values

<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file writeCcode\_sensi.m.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.5 `mlhsInnerSubst<::amimodel> writeCcode ( ::string funstr, ::fileid fid, matlabtypesubstitute ip, matlabtypesubstitute jp )`

#### Parameters

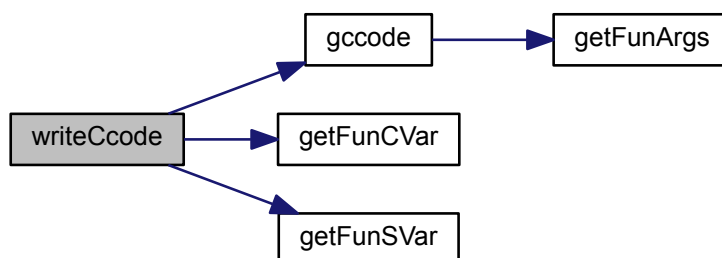
<i>funstr</i>	function for which C code is to be generated
<i>fid</i>	file id in which the final expression is written
<i>ip</i>	index for symbolic variable, if applicable <code>svar(:,ip)</code> will be passed to <code>gccode</code>
<i>jp</i>	index for symbolic variable, if applicable <code>svar(:,ip,jp)</code> will be passed to <code>gccode</code>

#### Return values

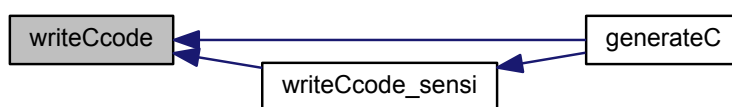
<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file `writeCcode.m`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.3.6 mlhsInnerSubst<::amimodel> gccode ( ::symbolic csym, ::string funstr, ::string cvar, ::fileid fid )

##### Parameters

<i>csym</i>	symbolic expression to be transform
<i>funstr</i>	function for which C code is to be generated
<i>cvar</i>	name of the assigned variable in C
<i>fid</i>	file id in which the expression should be written

##### Return values

<i>this</i>	model definition object
-------------	-------------------------

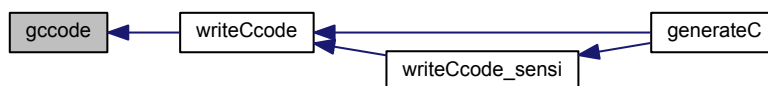
Definition at line 18 of file gccode.m.

Here is the call graph for this function:





Here is the caller graph for this function:



#### 5.1.3.7 `mlhsInnerSubst<::amimodel > generateM ( ::amimodel amimodelo2 )`

##### Parameters

<i>amimodelo2</i>	this struct must contain all necessary symbolic definitions for second order sensitivities
-------------------	--

##### Return values

<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file `generateM.m`.

#### 5.1.3.8 `mlhsInnerSubst<::amimodel > getFun ( ::struct HTable, ::string funstr )`

##### Parameters

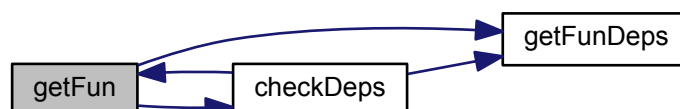
<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation
<i>funstr</i>	function for which symbolic expressions should be computed

##### Return values

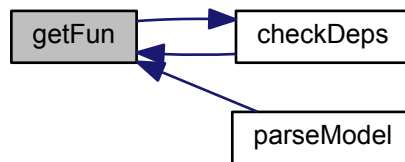
<i>this</i>	updated model definition object
-------------	---------------------------------

Definition at line 18 of file `getFun.m`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.3.9 mlhsInnerSubst<::cell> getFunDeps ( ::string funstr )

##### Parameters

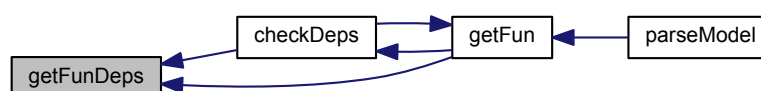
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

##### Return values

<i>deps</i>	cell array of dependencies
-------------	----------------------------

Definition at line 18 of file `getFunDeps.m`.

Here is the caller graph for this function:



#### 5.1.3.10 mlhsInnerSubst< matlabtypesubstitute > getFunArgs ( ::string funstr )

##### Parameters

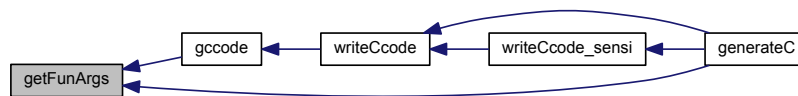
<i>funstr</i>	function name
---------------	---------------

##### Return values

<i>str</i>	string containing function arguments
------------	--------------------------------------

Definition at line 18 of file `getFunArgs.m`.

Here is the caller graph for this function:



#### 5.1.3.11 mlhsInnerSubst<::string> getFunCVar ( ::string funstr )

##### Parameters

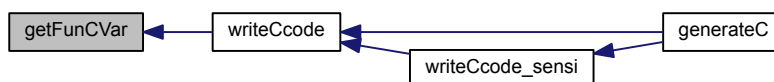
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

##### Return values

<i>cvar</i>	cell array of dependencies
-------------	----------------------------

Definition at line 18 of file getFunCVar.m.

Here is the caller graph for this function:



#### 5.1.3.12 mlhsInnerSubst< matlabtypesubstitute > getFunSVar ( ::string funstr )

##### Parameters

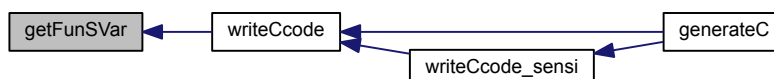
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

##### Return values

<i>cvar</i>	cell array of dependencies
-------------	----------------------------

Definition at line 18 of file getFunSVar.m.

Here is the caller graph for this function:



#### 5.1.3.13 mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >, mlhsInnerSubst<::HTable> > checkDeps ( ::struct HTable, ::cell deps )

## Parameters

<i>HTable</i>	struct with reference hashes of functions in its fields
<i>deps</i>	cell array with containing a list of dependencies

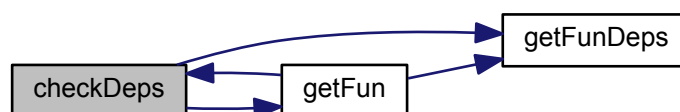
## Return values

<i>cflag</i>	boolean indicating whether any of the dependencies have
--------------	---

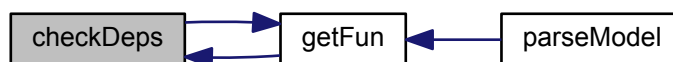
changed with respect to the hashes stored in HTable

Definition at line 18 of file checkDeps.m.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.3.14 mlhsSubst< mlhsInnerSubst<::amimodel >,mlhsInnerSubst<::struct > > loadOldHashes ( )

## Return values

<i>this</i>	updated model definition object
<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation

Definition at line 18 of file loadOldHashes.m.

Here is the caller graph for this function:



5.1.3.15 mlhsInnerSubst<::amimodel > augmento2 ( )

## Return values

<i>this</i>	augmented system which contains symbolic definition of the original system and its sensitivities
-------------	--

Definition at line 18 of file augmento2.m.

## 5.1.4 Member Data Documentation

## 5.1.4.1 atol = 1e-8

**Default:** 1e-8

Definition at line 61 of file amimodel.m.

## 5.1.4.2 rtol = 1e-8

**Default:** 1e-8

Definition at line 69 of file amimodel.m.

## 5.1.4.3 maxsteps = 1e4

**Default:** 1e4

Definition at line 77 of file amimodel.m.

## 5.1.4.4 debug = false

**Default:** false

Definition at line 85 of file amimodel.m.

## 5.1.4.5 adjoint = true

**Default:** true

Definition at line 93 of file amimodel.m.

## 5.1.4.6 forward = true

**Default:** true

Definition at line 101 of file amimodel.m.

## 5.1.4.7 t0 = 0

**Default:** 0

Definition at line 109 of file amimodel.m.

## 5.1.4.8 nxtrue = 0

**Default:** 0

Definition at line 131 of file amimodel.m.

#### 5.1.4.9 nytrue = 0

**Default:** 0

Definition at line 146 of file amimodel.m.

#### 5.1.4.10 coptim = "-O3"

**Default:** "-O3"

Definition at line 259 of file amimodel.m.

#### 5.1.4.11 param = "lin"

**Default:** "lin"

Definition at line 267 of file amimodel.m.

## 5.2 ExpData Struct Reference

struct that carries all information about experimental data

```
#include <edata.h>
```

### Public Attributes

- double \* [am\\_my](#)  
*observed data*
- double \* [am\\_ysigma](#)  
*standard deviation of observed data*
- double \* [am\\_mt](#)  
*observed events*
- double \* [am\\_tsigma](#)  
*standard deviation of observed events*

#### 5.2.1 Detailed Description

Definition at line 18 of file edata.h.

## 5.3 ReturnData Struct Reference

struct that stores all data which is later returned by the mex function

```
#include <rdata.h>
```

### Public Attributes

- double \* [am\\_tsdata](#)  
*timepoints*
- double \* [am\\_xdotdata](#)  
*time derivative*
- double \* [am\\_dxdotdpdata](#)  
*parameter derivative of time derivative*

- double \* [am\\_dydxdata](#)  
*state derivative of observables*
- double \* [am\\_dydpdata](#)  
*parameter derivative of observables*
- double \* [am\\_Jdata](#)  
*Jacobian of differential equation right hand side.*
- double \* [am\\_rootdata](#)  
*events*
- double \* [am\\_rootSdata](#)  
*parameter derivative of events*
- double \* [am\\_rootS2data](#)  
*second order parameter derivative of events*
- double \* [am\\_rootvaldata](#)  
*value of event function*
- double \* [am\\_rootvalSdata](#)  
*parameter derivative of event function*
- double \* [am\\_rootvalS2data](#)  
*second order parameter derivative of event function*
- double \* [am\\_xdata](#)  
*state*
- double \* [am\\_xSdata](#)  
*parameter derivative of state*
- double \* [am\\_ydata](#)  
*observable*
- double \* [am\\_ySdata](#)  
*parameter derivative of observable*
- double \* [am\\_numstepsdata](#)  
*number of integration steps forward problem*
- double \* [am\\_numstepsSdata](#)  
*number of integration steps backward problem*
- double \* [am\\_numrhsevalsddata](#)  
*number of right hand side evaluations forward problem*
- double \* [am\\_numrhsevalSdata](#)  
*number of right hand side evaluations backward problem*
- double \* [am\\_orderdata](#)  
*employed order forward problem*
- double \* [am\\_llhdata](#)  
*likelihood value*
- double \* [am\\_chi2data](#)  
*chi2 value*
- double \* [am\\_llhSdata](#)  
*parameter derivative of likelihood*
- double \* [am\\_llhS2data](#)  
*second order parameter derivative of likelihood*

### 5.3.1 Detailed Description

Definition at line 42 of file rdata.h.



### 5.3.2 Member Data Documentation

#### 5.3.2.1 double\* am\_xdata

returned vector

Definition at line 69 of file rdata.h.

## 5.4 TempData Struct Reference

struct that provides temporary storage for different variables

```
#include <tdata.h>
```

### Public Attributes

- realtype [am\\_t](#)  
*current time*
- N\_Vector [am\\_x](#)  
*state vector*
- N\_Vector [am\\_dx](#)  
*differential state vector*
- N\_Vector [am\\_xdot](#)  
*time derivative state vector*
- N\_Vector [am\\_xB](#)  
*adjoint state vector*
- N\_Vector [am\\_dxB](#)  
*differential adjoint state vector*
- N\_Vector [am\\_xQB](#)  
*quadrature state vector*
- N\_Vector \* [am\\_sx](#)  
*sensitivity state vector array*
- N\_Vector \* [am\\_sdx](#)  
*differential sensitivity state vector array*
- N\_Vector [am\\_id](#)  
*index indicating DAE equations vector*
- DlsMat [am\\_Jtmp](#)  
*Jacobian.*
- double \* [am\\_llhS0](#)  
*parameter derivative of likelihood array*
- double [am\\_g](#)  
*data likelihood*
- double \* [am\\_dgdp](#)  
*parameter derivative of data likelihood*
- double \* [am\\_dgdx](#)  
*state derivative of data likelihood*
- double [am\\_r](#)  
*event likelihood*
- double \* [am\\_drdp](#)  
*parameter derivative of event likelihood*
- double \* [am\\_dr dx](#)  
*state derivative of event likelihood*

- double [am\\_rval](#)  
*root function likelihood*
- double \* [am\\_drvaldp](#)  
*parameter derivative of root function likelihood*
- double \* [am\\_drvaldx](#)  
*state derivative of root function likelihood*
- double \* [am\\_dtdx](#)  
*state derivative of event*
- double \* [am\\_dtdp](#)  
*parameter derivative of event*
- double \* [am\\_dydp](#)  
*parameter derivative of observable*
- double \* [am\\_dydx](#)  
*state derivative of observable*
- double \* [am\\_yS0](#)  
*initial sensitivity of observable*
- double \* [am\\_sigma\\_y](#)  
*data standard deviation*
- double \* [am\\_dsigma\\_ydp](#)  
*parameter derivative of data standard deviation*
- double \* [am\\_sigma\\_t](#)  
*event standard deviation*
- double \* [am\\_dsigma\\_tdp](#)  
*parameter derivative of event standard deviation*
- double \* [am\\_x\\_tmp](#)  
*state array*
- double \* [am\\_sx\\_tmp](#)  
*sensitivity state array*
- double \* [am\\_dx\\_tmp](#)  
*differential state array*
- double \* [am\\_sdx\\_tmp](#)  
*differential sensitivity state array*
- double \* [am\\_xdot\\_tmp](#)  
*time derivative state array*
- double \* [am\\_xB\\_tmp](#)  
*differential adjoint state array*
- double \* [am\\_xQB\\_tmp](#)  
*quadrature state array*
- double \* [am\\_dxB\\_tmp](#)  
*differential adjoint state array*
- double \* [am\\_id\\_tmp](#)  
*index indicating DAE equations array*
- int \* [am\\_rootsfound](#)  
*array of flags indicating which root has been found*
- double \* [am\\_rootvaltmp](#)  
*array of values of the root function*
- int \* [am\\_rootidx](#)  
*array of index which root has been found*
- int [am\\_which](#)  
*integer for indexing of backwards problems*
- double \* [am\\_discs](#)  
*array containing the time-points of discontinuities*
- double \* [am\\_irdiscs](#)  
*array containing the index of discontinuities*

### 5.4.1 Detailed Description

Definition at line 64 of file tdata.h.

### 5.4.2 Member Data Documentation

#### 5.4.2.1 int\* am\_rootsfound

array of length nr with the indices of the user functions gi found to have a root. For  $i = 0, \dots, nr-1$ , rootsfound[i] = 0 if gi has a root, and = 0 if not.

Definition at line 151 of file tdata.h.

## 5.5 UserData Struct Reference

struct that stores all user provided data

```
#include <udata.h>
```

### Public Attributes

- int \* [am\\_plist](#)  
*parameter reordering*
- int [am\\_np](#)  
*number of parameters*
- int [am\\_ny](#)  
*number of observables*
- int [am\\_nx](#)  
*number of states*
- int [am\\_nr](#)  
*number of roots*
- int [am\\_nt](#)  
*number of timepoints*
- int [am\\_ndisc](#)  
*number of discontinuities*
- int [am\\_nnz](#)  
*number of nonzero entries in jacobian*
- int [am\\_nmaxroot](#)  
*maximal number of roots to collect*
- int [am\\_nmaxdisc](#)  
*maximal number of discontinuities to track*
- double \* [am\\_p](#)  
*parameter array*
- double \* [am\\_k](#)  
*constants array*
- double [am\\_tstart](#)  
*starting time*
- double \* [am\\_ts](#)  
*timepoints*
- double \* [am\\_pbar](#)  
*scaling of parameters*
- double \* [am\\_xbar](#)

- scaling of states*
- double \* [am\\_idlist](#)  
*flag array for DAE equations*
- int [am\\_sensi](#)  
*flag indicating whether sensitivities are supposed to be computed*
- double [am\\_atol](#)  
*absolute tolerances for integration*
- double [am\\_rtol](#)  
*relative tolerances for integration*
- int [am\\_maxsteps](#)  
*maximum number of allowed integration steps*
- int [am\\_ism](#)  
*internal sensitivity method*
- int [am\\_sensi\\_meth](#)  
*method for sensitivity computation*
- int [am\\_linsol](#)  
*linear solver specification*
- int [am\\_interpType](#)  
*interpolation type*
- int [am\\_lmm](#)  
*linear multistep method*
- int [am\\_iter](#)  
*nonlinear solver*
- boolean [am\\_stldet](#)  
*flag controlling stability limit detection*
- int [am\\_ubw](#)  
*upper bandwidth of the jacobian*
- int [am\\_lbw](#)  
*lower bandwidth of the jacobian*
- boolean [am\\_bsx0](#)  
*flag for sensitivity initialisation*
- double \* [am\\_sx0data](#)  
*sensitivity initialisation*
- int [am\\_event\\_model](#)  
*error model for events*
- int [am\\_data\\_model](#)  
*error model for udata*
- int [am\\_ordering](#)  
*state ordering*

### 5.5.1 Detailed Description

Definition at line 61 of file udata.h.

### 5.5.2 Member Data Documentation

#### 5.5.2.1 int am\_ism

a flag used to select the sensitivity solution method. Its value can be CV SIMULTANEOUS or CV STAGGERED. Only applies for Forward Sensitivities.

Definition at line 115 of file udata.h.

### 5.5.2.2 int am\_sensi\_meth

CW\_FSA for forward sensitivity analysis, CW\_ASA for adjoint sensitivity analysis

Definition at line 121 of file udata.h.

### 5.5.2.3 int am\_interpType

specifies the interpolation type for the forward problem solution which is then used for the backwards problem. can be either CV\_POLYNOMIAL or CV\_HERMITE

Definition at line 128 of file udata.h.

### 5.5.2.4 int am\_lmm

specifies the linear multistep method and may be one of two possible values: CV ADAMS or CV BDF.

Definition at line 134 of file udata.h.

### 5.5.2.5 int am\_iter

specifies the type of nonlinear solver iteration and may be either CV NEWTON or CV FUNCTIONAL.

Definition at line 140 of file udata.h.

### 5.5.2.6 booleantype am\_bsx0

flag which determines whether analytic sensitivities initialisation or provided initialisation should be used

Definition at line 154 of file udata.h.

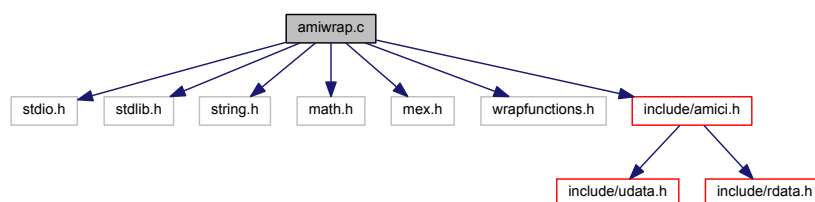
## 6 File Documentation

### 6.1 amiwrap.c File Reference

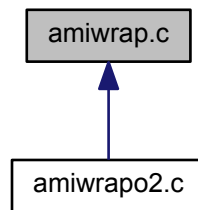
core routines for mex interface

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mex.h>
#include "wrapfunctions.h"
#include <include/amici.h>
```

Include dependency graph for amiwrap.c:



This graph shows which files directly or indirectly include this file:



## Functions

- void [mexFunction](#) (int *nlhs*, mxArray \**plhs*[], int *nrhs*, const mxArray \**prhs*[])

### 6.1.1 Detailed Description

This file defines the fuction `mexFunction` which is executed upon calling the mex file from matlab

### 6.1.2 Function Documentation

#### 6.1.2.1 void mexFunction ( int *nlhs*, mxArray \* *plhs*[], int *nrhs*, const mxArray \* *prhs*[] )

`mexFunction` is the main function of the mex simulation file this function carries out all numerical integration and writes results into the `sol` struct.

#### Parameters

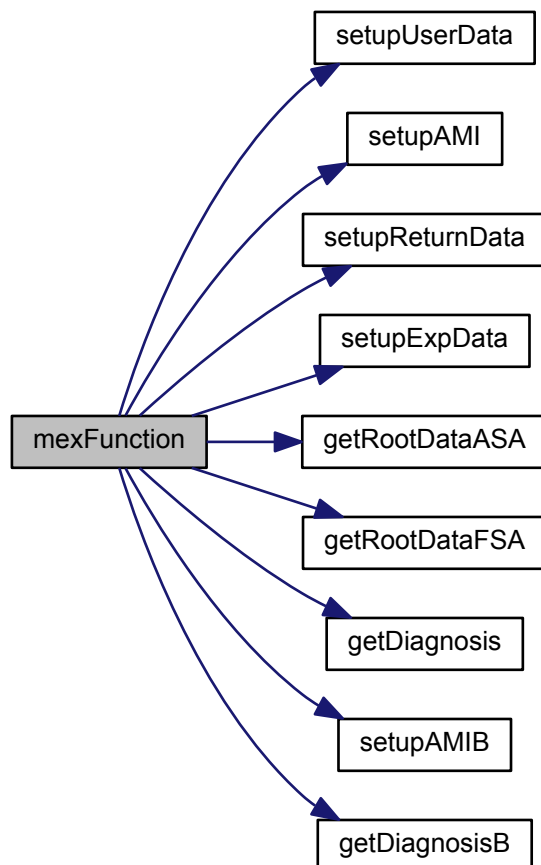
in	<i>nlhs</i>	number of output arguments of the matlab call <b>Type:</b> int
out	<i>plhs</i>	pointer to the array of output arguments <b>Type:</b> mxArray
in	<i>nrhs</i>	number of input arguments of the matlab call <b>Type:</b> int
in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> mxArray

## Returns

void

Definition at line 25 of file amiwrap.c.

Here is the call graph for this function:



## 6.2 amiwrap.m File Reference

AMIWRAP generates c mex files for the simulation of systems of differential equations via CVODES and IDAS.

## Functions

- noret::substitute [amiwrap](#) (matlabtypesubstitute varargin)  
*AMIWRAP generates c mex files for the simulation of systems of differential equations via CVODES and IDAS.*

### 6.2.1 Function Documentation

#### 6.2.1.1 noret::substitute amiwrap ( matlabtypesubstitute varargin )

## Parameters

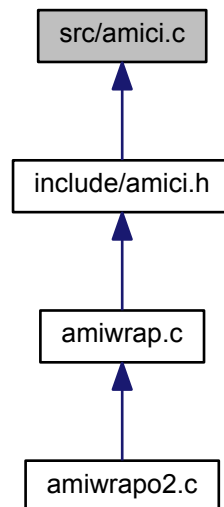
<i>varargin</i>	<pre>1 amiwrap ( modelname, symfun, tdir, o2flag )</pre> <p><i>Required Parameters for varargin:</i></p> <ul style="list-style-type: none"> <li>• modelname specifies the name of the model which will be later used for the naming of the simulation file</li> <li>• symfun specifies a function which executes model definition see <a href="#">Model Definition</a> for details</li> <li>• tdir target directory where the simulation file should be placed <b>Default:</b> \$AMI-CIDIR/models/modelname</li> <li>• o2flag boolean whether second order sensitivities should be enabled <b>Default:</b> false</li> </ul>
-----------------	---

Definition at line 17 of file amiwrap.m.

## 6.3 src/amici.c File Reference

core routines for integration

This graph shows which files directly or indirectly include this file:



## Macros

- `#define amici_c`  
*include guard*
- `#define AMI_SUCCESS 0`  
*return value indicating successful execution*



## Functions

- [UserData](#) [setupUserData](#) (const mxArray \*prhs[])
- void \* [setupAMI](#) (int \*status, void \*user\_data, void \*temp\_data)
- void [setupAMIB](#) (int \*status, void \*ami\_mem, void \*user\_data, void \*temp\_data)
- [ReturnData](#) [setupReturnData](#) (const mxArray \*prhs[], void \*user\_data)
- [ExpData](#) [setupExpData](#) (const mxArray \*prhs[], void \*user\_data)
- void [getRootDataFSA](#) (int \*status, int \*nroots, void \*ami\_mem, void \*user\_data, void \*return\_data, void \*temp\_data)
- void [getRootDataASA](#) (int \*status, int \*nroots, int \*idisc, void \*ami\_mem, void \*user\_data, void \*return\_data, void \*exp\_data, void \*temp\_data)
- void [getDiagnosis](#) (int \*status, int it, void \*ami\_mem, void \*user\_data, void \*return\_data)
- void [getDiagnosisB](#) (int \*status, int it, void \*ami\_mem, void \*user\_data, void \*return\_data, void \*temp\_data)

### 6.3.1 Function Documentation

#### 6.3.1.1 [UserData](#) [setupUserData](#) ( const mxArray \* *prhs*[ ] )

[setupUserData](#) extracts information from the matlab call and returns the corresponding [UserData](#) struct

##### Parameters

in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> mxArray
----	-------------	---

##### Returns

udata: struct containing all provided user data

Definition at line 10 of file amici.c.

Here is the caller graph for this function:



#### 6.3.1.2 void\* [setupAMI](#) ( int \* *status*, void \* *user\_data*, void \* *temp\_data* )

[setupAMI](#)s initialises the ami memory object

##### Parameters

out	<i>status</i>	flag indicating success of execution <b>Type:</b> *int
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>

in	<i>temp_data</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
----	------------------	---

**Returns**

void

Definition at line 149 of file amici.c.

Here is the caller graph for this function:



### 6.3.1.3 void setupAMIB ( int \* *status*, void \* *ami\_mem*, void \* *user\_data*, void \* *temp\_data* )

setupAMIB initialises the AMI memory object for the backwards problem

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> *int
in	<i>ami_mem</i>	pointer to the solver memory object of the forward problem
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>temp_data</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

**Returns**

void

Definition at line 434 of file amici.c.

Here is the caller graph for this function:



### 6.3.1.4 ReturnData setupReturnData ( const mxArray \* *prhs*[], void \* *user\_data* )

setupReturnData initialises the return data struct

**Parameters**

in	<i>prhs</i>	user input <b>Type:</b> *mxArray
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>

**Returns**

rdata: return data struct  
**Type:** [ReturnData](#)

user udata

Definition at line 625 of file amici.c.

Here is the caller graph for this function:



### 6.3.1.5 ExpData setupExpData ( const mxArray \* *prhs*[], void \* *user\_data* )

setupExpData initialises the experimental data struct

**Parameters**

in	<i>prhs</i>	user input <b>Type:</b> *mxArray
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>

**Returns**

edata: experimental data struct  
**Type:** [ExpData](#)

user udata

Definition at line 690 of file amici.c.

Here is the caller graph for this function:



6.3.1.6 void getRootDataFSA ( int \* *status*, int \* *nroots*, void \* *ami\_mem*, void \* *user\_data*, void \* *return\_data*, void \* *temp\_data* )

getRootDataFSA extracts root information for forward sensitivity analysis

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> int
out	<i>nroots</i>	counter for the number of found roots <b>Type:</b> int
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> void
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>return_data</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
out	<i>temp_data</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

**Returns**

void

Definition at line 789 of file amici.c.

Here is the caller graph for this function:



**6.3.1.7** void `getRootDataASA` ( int \* *status*, int \* *nroots*, int \* *idisc*, void \* *ami\_mem*, void \* *user\_data*, void \* *return\_data*, void \* *exp\_data*, void \* *temp\_data* )

`getRootDataASA` extracts root information for adjoint sensitivity analysis

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> *int
out	<i>nroots</i>	counter for the number of found roots <b>Type:</b> *int
out	<i>idisc</i>	counter for the number of found discontinuities <b>Type:</b> *int
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> *void
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>return_data</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>exp_data</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>

out	<i>temp_data</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
-----	------------------	---

**Returns**

void

Definition at line 873 of file amici.c.

Here is the caller graph for this function:



### 6.3.1.8 void getDiagnosis ( int \* *status*, int *it*, void \* *ami\_mem*, void \* *user\_data*, void \* *return\_data* )

getDiagnosis extracts diagnosis information from solver memory block and writes them into the return data struct

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> *int
in	<i>it</i>	time-point index <b>Type:</b> int
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> *void
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>return_data</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>

**Returns**

void

Definition at line 981 of file amici.c.

Here is the caller graph for this function:



### 6.3.1.9 void getDiagnosisB ( int \* *status*, int *it*, void \* *ami\_mem*, void \* *user\_data*, void \* *return\_data*, void \* *temp\_data* )

getDiagnosis extracts diagnosis information from solver memory block and writes them into the return data struct

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> *int
in	<i>it</i>	time-point index <b>Type:</b> int
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> *void
in	<i>user_data</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>return_data</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
out	<i>temp_data</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

**Returns**

void

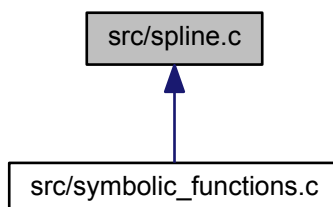
Definition at line 1015 of file amici.c.

Here is the caller graph for this function:

**6.4 src/spline.c File Reference**

definition of spline functions

This graph shows which files directly or indirectly include this file:

**Functions**

- int [spline](#) (int n, int end1, int end2, double slope1, double slope2, double x[], double y[], double b[], double c[], double d[])

- double [seval](#) (int n, double u, double x[], double y[], double b[], double c[], double d[])
- double [deriv](#) (int n, double u, double x[], double b[], double c[], double d[])
- double [sinteg](#) (int n, double u, double x[], double y[], double b[], double c[], double d[])

#### 6.4.1 Detailed Description

##### Author

Peter & Nigel, Design Software, 42 Gubberley St, Kenmore, 4069, Australia.

#### 6.4.2 Function Documentation

**6.4.2.1** int spline ( int n, int end1, int end2, double slope1, double slope2, double x[], double y[], double b[], double c[], double d[] )

Evaluate the coefficients b[i], c[i], d[i], i = 0, 1, .. n-1 for a cubic interpolating spline

$S(xx) = Y[i] + b[i] * w + c[i] * w**2 + d[i] * w**3$  where  $w = xx - x[i]$  and  $x[i] \leq xx \leq x[i+1]$

The n supplied data points are x[i], y[i], i = 0 ... n-1.

##### Parameters

in	n	The number of data points or knots (n >= 2)
in	end1	0: default condition 1: specify the slopes at x[0]
in	end2	0: default condition 1: specify the slopes at x[n-1]
in	slope1	slope at x[0]
in	slope2	slope at x[n-1]
in	x[]	the abscissas of the knots in strictly increasing order
in	y[]	the ordinates of the knots
out	b[]	array of spline coefficients
out	c[]	array of spline coefficients
out	d[]	array of spline coefficients

##### Return values

0	normal return
1	less than two data points; cannot interpolate
2	x[] are not in ascending order

##### Notes

- The accompanying function [seval\(\)](#) may be used to evaluate the spline while deriv will provide the first derivative.
- Using p to denote differentiation  $y[i] = S(X[i])$   $b[i] = S_p(X[i])$   $c[i] = S_{pp}(X[i])/2$   $d[i] = S_{ppp}(X[i])/6$  ( Derivative from the right )
- Since the zero elements of the arrays ARE NOW used here, all arrays to be passed from the main program should be dimensioned at least [n]. These routines will use elements [0 .. n-1].
- Adapted from the text Forsythe, G.E., Malcolm, M.A. and Moler, C.B. (1977) "Computer Methods for Mathematical Computations" Prentice Hall
- Note that although there are only n-1 polynomial segments, n elements are required in b, c, d. The elements b[n-1], c[n-1] and d[n-1] are set to continue the last segment past x[n-1].

Definition at line 66 of file spline.c.



#### 6.4.2.2 double seval ( int *n*, double *u*, double *x*[], double *y*[], double *b*[], double *c*[], double *d*[] )

Evaluate the cubic spline function

$S(x) = y[i] + b[i] * w + c[i] * w^2 + d[i] * w^3$  where  $w = u - x[i]$  and  $x[i] \leq u \leq x[i+1]$  Note that Horner's rule is used. If  $u < x[0]$  then  $i = 0$  is used. If  $u > x[n-1]$  then  $i = n-1$  is used.

##### Parameters

in	<i>n</i>	The number of data points or knots ( $n \geq 2$ )
in	<i>u</i>	the abscissa at which the spline is to be evaluated
in	<i>x</i> []	the abscissas of the knots in strictly increasing order
in	<i>y</i> []	the ordinates of the knots
in	<i>b</i>	array of spline coefficients computed by <a href="#">spline()</a> .
in	<i>c</i>	array of spline coefficients computed by <a href="#">spline()</a> .
in	<i>d</i>	array of spline coefficients computed by <a href="#">spline()</a> .

##### Returns

the value of the spline function at *u*

##### Notes

- If *u* is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

Definition at line 208 of file spline.c.

#### 6.4.2.3 double deriv ( int *n*, double *u*, double *x*[], double *b*[], double *c*[], double *d*[] )

Evaluate the derivative of the cubic spline function

$S(x) = B[i] + 2.0 * C[i] * w + 3.0 * D[i] * w^2$  where  $w = u - X[i]$  and  $X[i] \leq u \leq X[i+1]$  Note that Horner's rule is used. If  $U < X[0]$  then  $i = 0$  is used. If  $U > X[n-1]$  then  $i = n-1$  is used.

##### Parameters

in	<i>n</i>	the number of data points or knots ( $n \geq 2$ )
in	<i>u</i>	the abscissa at which the derivative is to be evaluated
in	<i>x</i>	the abscissas of the knots in strictly increasing order
in	<i>b</i>	array of spline coefficients computed by <a href="#">spline()</a>
in	<i>c</i>	array of spline coefficients computed by <a href="#">spline()</a>
in	<i>d</i>	array of spline coefficients computed by <a href="#">spline()</a>

##### Returns

the value of the derivative of the spline function at *u*

##### Notes

- If *u* is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

Definition at line 264 of file spline.c.

#### 6.4.2.4 double sinteg ( int *n*, double *u*, double *x*[], double *y*[], double *b*[], double *c*[], double *d*[] )

Integrate the cubic spline function

$S(x) = y[i] + b[i] * w + c[i] * w^2 + d[i] * w^3$  where  $w = u - x[i]$  and  $x[i] \leq u \leq x[i+1]$

The integral is zero at  $u = x[0]$ .

If  $u < x[0]$  then  $i = 0$  segment is extrapolated. If  $u > x[n-1]$  then  $i = n-1$  segment is extrapolated.

## Parameters

in	$n$	the number of data points or knots ( $n \geq 2$ )
in	$u$	the abscissa at which the spline is to be evaluated
in	$x[]$	the abscissas of the knots in strictly increasing order
in	$y[]$	the ordinates of the knots
in	$b$	array of spline coefficients computed by <a href="#">spline()</a> .
in	$c$	array of spline coefficients computed by <a href="#">spline()</a> .
in	$d$	array of spline coefficients computed by <a href="#">spline()</a> .

## Returns

the value of the spline function at  $u$

## Notes

- If  $u$  is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

Definition at line 324 of file spline.c.

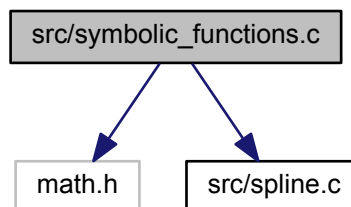
## 6.5 src/symbolic\_functions.c File Reference

definition of symbolic functions

```
#include <math.h>
```

```
#include <src/spline.c>
```

Include dependency graph for symbolic\_functions.c:



## Macros

- `#define amici\_symbolic\_functions\_c`  
*include guard*
- `#define TRUE 1`  
*bool return value true*
- `#define FALSE 0`  
*bool return value false*

## Functions

- int [am\\_ge](#) (double a, double b)

- int [Dam\\_ge](#) (int id, double a, double b)
- int [am\\_gt](#) (double a, double b)
- int [Dam\\_gt](#) (int id, double a, double b)
- int [am\\_le](#) (double a, double b)
- int [Dam\\_le](#) (int id, double a, double b)
- int [am\\_lt](#) (double a, double b)
- int [Dam\\_lt](#) (int id, double a, double b)
- double [am\\_if](#) (int condition, double truepart, double falsepart)
- double [Dam\\_if](#) (int id, int condition, double truepart, double falsepart)
- int [am\\_and](#) (int a, int b)
- double [Dam\\_and](#) (int id, int a, int b)
- int [am\\_or](#) (int a, int b)
- double [Dam\\_or](#) (int id, int a, int b)
- double [am\\_min](#) (double a, double b)
- double [Dam\\_min](#) (int id, double a, double b)
- double [am\\_max](#) (double a, double b)
- double [Dam\\_max](#) (int id, double a, double b)
- double [heaviside](#) (double x)
- double [dirac](#) (double x)
- double [Ddirac](#) (int id, double x)
- double [spline3](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt)
- double [spline\\_pos3](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt)
- double [spline4](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt)
- double [spline\\_pos4](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt)
- double [spline5](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt)
- double [spline\\_pos5](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt)
- double [spline10](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, double t6, double p6, double t7, double p7, double t8, double p8, double t9, double p9, double t10, double p10, int ss, double dudt)
- double [spline\\_pos10](#) (double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, double t6, double p6, double t7, double p7, double t8, double p8, double t9, double p9, double t10, double p10, int ss, double dudt)
- double [Dspline3](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt)
- double [Dspline\\_pos3](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt)
- double [Dspline4](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt)
- double [Dspline\\_pos4](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt)
- double [Dspline5](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt)
- double [Dspline\\_pos5](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt)
- double [Dspline10](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, double t6, double p6, double t7, double p7, double t8, double p8, double t9, double p9, double t10, double p10, int ss, double dudt)
- double [Dspline\\_pos10](#) (int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, double t6, double p6, double t7, double p7, double t8, double p8, double t9, double p9, double t10, double p10, int ss, double dudt)

## 6.5.1 Detailed Description

This file contains definitions of various symbolic functions which

## 6.5.2 Function Documentation

6.5.2.1 int am\_ge ( double *a*, double *b* )

c implementation of matlab function ge

## Parameters

<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

if( $a \geq 0$ ) then 1 else 0

**Type:** int

Definition at line 27 of file symbolic\_functions.c.

6.5.2.2 int Dam\_ge ( int *id*, double *a*, double *b* )

parameter derivative of c implementation of matlab function ge

## Parameters

<i>id</i>	argument index for differentiation
<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

0

Definition at line 44 of file symbolic\_functions.c.

6.5.2.3 int am\_gt ( double *a*, double *b* )

c implementation of matlab function gt

## Parameters

<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

if( $a > 0$ ) then 1 else 0

**Type:** int

Definition at line 56 of file symbolic\_functions.c.

6.5.2.4 `int Dam_gt ( int id, double a, double b )`

parameter derivative of c implementation of matlab function gt

## Parameters

<i>id</i>	argument index for differentiation
<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

0

Definition at line 73 of file symbolic\_functions.c.

6.5.2.5 int am\_le ( double *a*, double *b* )

c implementation of matlab function le

## Parameters

<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

if(*a* <= 0) then 1 else 0**Type:** int

Definition at line 85 of file symbolic\_functions.c.

6.5.2.6 int Dam\_le ( int *id*, double *a*, double *b* )

parameter derivative of c implementation of matlab function le

## Parameters

<i>id</i>	argument index for differentiation
<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

## Returns

0

Definition at line 102 of file symbolic\_functions.c.

6.5.2.7 int am\_lt ( double *a*, double *b* )

c implementation of matlab function lt

## Parameters

<i>a</i>	value1 <b>Type:</b> double
----------	-------------------------------

<i>b</i>	value2 <b>Type:</b> double
----------	-------------------------------

**Returns**

if( $a < 0$ ) then 1 else 0

**Type:** int

Definition at line 114 of file symbolic\_functions.c.

#### 6.5.2.8 int Dam\_It ( int *id*, double *a*, double *b* )

parameter derivative of c implementation of matlab function It

**Parameters**

<i>id</i>	argument index for differentiation
<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

**Returns**

0

Definition at line 131 of file symbolic\_functions.c.

#### 6.5.2.9 double am\_if ( int *condition*, double *truepart*, double *falsepart* )

c implementation of if function

**Parameters**

<i>condition</i>	condition that decides on the output
<i>truepart</i>	returnvalue if condition is true <b>Type:</b> double
<i>falsepart</i>	returnvalue if condition is false <b>Type:</b> double

**Returns**

if(condition) then truepart else falsepart

**Type:** int

Definition at line 144 of file symbolic\_functions.c.

#### 6.5.2.10 double Dam\_if ( int *id*, int *condition*, double *truepart*, double *falsepart* )

parameter derivative of c implementation of if function

**Parameters**

<i>id</i>	argument index for differentiation
<i>condition</i>	condition that decides on the output
<i>truepart</i>	returnvalue if condition is true <b>Type:</b> double

<i>falsepart</i>	returnvalue if condition is false <b>Type:</b> double
------------------	--

**Returns**

id==1: 0

**Type:** double

id==2: if(condition) then 1 else 0

**Type:** double

id==3: if(condition) then 0 else 1

**Type:** double

Definition at line 165 of file symbolic\_functions.c.

**6.5.2.11 int am\_and ( int *a*, int *b* )**

c implementation of matlab function and

**Parameters**

<i>a</i>	bool1 <b>Type:</b> int
<i>b</i>	bool2 <b>Type:</b> int

**Returns**

if(a &amp; b) then 1 else 0

**Type:** int

Definition at line 199 of file symbolic\_functions.c.

**6.5.2.12 double Dam\_and ( int *id*, int *a*, int *b* )**

parameter derivative of c implementation of matlab function and

**Parameters**

<i>id</i>	argument index for differentiation
<i>a</i>	bool1 <b>Type:</b> int
<i>b</i>	bool2 <b>Type:</b> int

**Returns**

0

**Type:** double

Definition at line 216 of file symbolic\_functions.c.

**6.5.2.13 int am\_or ( int *a*, int *b* )**

c implementation of matlab function or

**Parameters**



<i>a</i>	bool1 <b>Type:</b> int
<i>b</i>	bool2 <b>Type:</b> int

**Returns**

if( $a \mid b$ ) then 1 else 0  
**Type:** int

Definition at line 228 of file symbolic\_functions.c.

#### 6.5.2.14 double Dam\_or ( int *id*, int *a*, int *b* )

parameter derivative of c implementation of matlab function or

**Parameters**

<i>id</i>	argument index for differentiation
<i>a</i>	bool1 <b>Type:</b> int
<i>b</i>	bool2 <b>Type:</b> int

**Returns**

0  
**Type:** double

Definition at line 245 of file symbolic\_functions.c.

#### 6.5.2.15 double am\_min ( double *a*, double *b* )

c implementation of matlab function min

**Parameters**

<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

**Returns**

if( $a < b$ ) then *a* else *b*  
**Type:** double

Definition at line 257 of file symbolic\_functions.c.

#### 6.5.2.16 double Dam\_min ( int *id*, double *a*, double *b* )

parameter derivative of c implementation of matlab function min

**Parameters**

<i>id</i>	argument index for differentiation
-----------	------------------------------------

<i>a</i>	bool1 <b>Type:</b> double
<i>b</i>	bool2 <b>Type:</b> double

**Returns**

id == 1: if( $a < b$ ) then 1 else 0

**Type:** double

id == 2: if( $a < b$ ) then 0 else 1

**Type:** double

Definition at line 275 of file symbolic\_functions.c.

**6.5.2.17 double am\_max ( double *a*, double *b* )**

c implementation of matlab function min

**Parameters**

<i>a</i>	value1 <b>Type:</b> double
<i>b</i>	value2 <b>Type:</b> double

**Returns**

if( $a > b$ ) then *a* else *b*

**Type:** double

Definition at line 299 of file symbolic\_functions.c.

**6.5.2.18 double Dam\_max ( int *id*, double *a*, double *b* )**

parameter derivative of c implementation of matlab function min

**Parameters**

<i>id</i>	argument index for differentiation
<i>a</i>	bool1 <b>Type:</b> double
<i>b</i>	bool2 <b>Type:</b> double

**Returns**

id == 1: if( $a > b$ ) then 1 else 0

**Type:** double

id == 2: if( $a > b$ ) then 0 else 1

**Type:** double

Definition at line 317 of file symbolic\_functions.c.

**6.5.2.19 double heaviside ( double *x* )**

c implementation of matlab function heaviside

**Parameters**

<i>x</i>	argument
----------	----------

**Returns**

if( $x > 0$ ) then 1 else 0

Definition at line 340 of file symbolic\_functions.c.

**6.5.2.20 double dirac ( double *x* )**

c implementation of matlab function dirac

**Parameters**

<i>x</i>	argument
----------	----------

**Returns**

if( $x == 0$ ) then 1 else 0

Definition at line 355 of file symbolic\_functions.c.

**6.5.2.21 double Ddirac ( int *id*, double *x* )**

derivatives of the c implementation of matlab function dirac

**Parameters**

<i>id</i>	argument index for differentiation
<i>x</i>	argument

**Returns**

0

**Type:** double

Definition at line 371 of file symbolic\_functions.c.

**6.5.2.22 double spline3 ( double *t*, double *t1*, double *p1*, double *t2*, double *p2*, double *t3*, double *p3*, int *ss*, double *dudt* )**

spline function with 3 nodes

**Parameters**

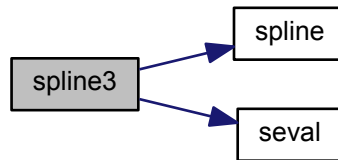
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

spline(*t*)

Definition at line 395 of file symbolic\_functions.c.

Here is the call graph for this function:



6.5.2.23 `double spline_pos3 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt )`

positive spline function with 3 nodes

#### Parameters

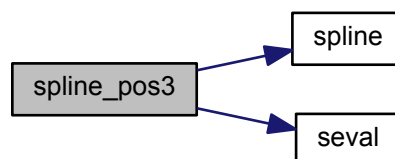
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

#### Returns

`spline(t)`

Definition at line 436 of file `symbolic_functions.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.24 `double spline4 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt )`

spline function with 4 nodes

#### Parameters

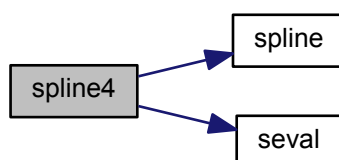
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

#### Returns

spline(*t*)

Definition at line 484 of file `symbolic_functions.c`.

Here is the call graph for this function:



6.5.2.25 `double spline_pos4 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt )`

positive spline function with 4 nodes

## Parameters

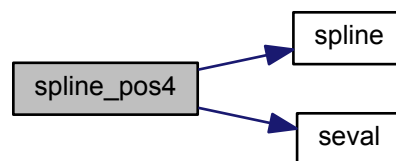
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

## Returns

spline(t)

Definition at line 528 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



**6.5.2.26** `double spline5 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt )`

spline function with 5 nodes

## Parameters

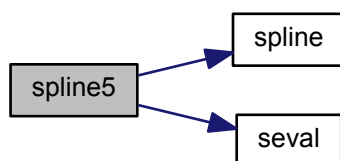
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

### Returns

spline(*t*)

Definition at line 580 of file symbolic\_functions.c.

Here is the call graph for this function:



**6.5.2.27** `double spline_pos5 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt )`

positive spline function with 5 nodes

### Parameters

<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5

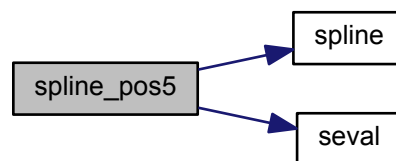
<i>p5</i>	spline value at node 5
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

spline(t)

Definition at line 628 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.28 `double spline10 ( double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, double t6, double p6, double t7, double p7, double t8, double p8, double t9, double p9, double t10, double p10, int ss, double dudt )`

spline function with 10 nodes

**Parameters**

<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3



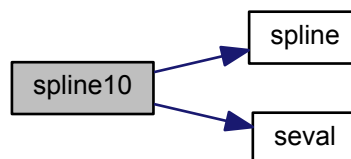
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>t6</i>	location of node 6
<i>p6</i>	spline value at node 6
<i>t7</i>	location of node 7
<i>p7</i>	spline value at node 7
<i>t8</i>	location of node 8
<i>p8</i>	spline value at node 8
<i>t9</i>	location of node 9
<i>p9</i>	spline value at node 9
<i>t10</i>	location of node 10
<i>p10</i>	spline value at node 10
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

#### Returns

spline(t)

Definition at line 692 of file symbolic\_functions.c.

Here is the call graph for this function:



6.5.2.29 double spline\_pos10 ( double *t*, double *t1*, double *p1*, double *t2*, double *p2*, double *t3*, double *p3*, double *t4*, double *p4*, double *t5*, double *p5*, double *t6*, double *p6*, double *t7*, double *p7*, double *t8*, double *p8*, double *t9*, double *p9*, double *t10*, double *p10*, int *ss*, double *dudt* )

positive spline function with 10 nodes

#### Parameters

<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3

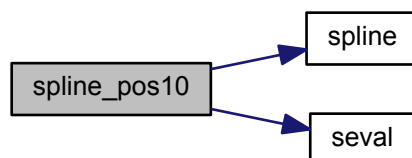
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>t6</i>	location of node 6
<i>p6</i>	spline value at node 6
<i>t7</i>	location of node 7
<i>p7</i>	spline value at node 7
<i>t8</i>	location of node 8
<i>p8</i>	spline value at node 8
<i>t9</i>	location of node 9
<i>p9</i>	spline value at node 9
<i>t10</i>	location of node 10
<i>p10</i>	spline value at node 10
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

spline(t)

Definition at line 760 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



**6.5.2.30** `double Dspline3 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss, double dudt )`

parameter derivative of spline function with 3 nodes

## Parameters

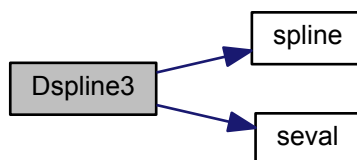
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

## Returns

dspline(t)dp(id)

Definition at line 821 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



```
6.5.2.31 double Dspline_pos3 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, int ss,
double dudt )
```

parameter derivative of positive spline function with 3 nodes

## Parameters

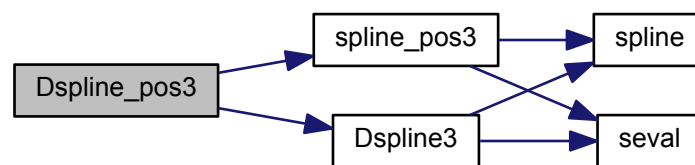
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

dspline(t)dp(id)

Definition at line 866 of file symbolic\_functions.c.

Here is the call graph for this function:



6.5.2.32 `double Dspline4 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt )`

parameter derivative of spline function with 4 nodes

**Parameters**

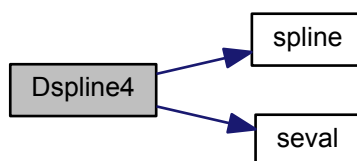
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

dspline(t)dp(id)

Definition at line 909 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.33 `double Dspline_pos4 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, int ss, double dudt )`

parameter derivative of positive spline function with 4 nodes

#### Parameters

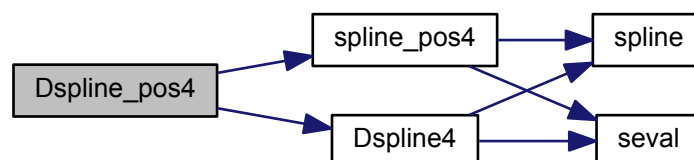
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

## Returns

dspline(t)dp(id)

Definition at line 958 of file symbolic\_functions.c.

Here is the call graph for this function:



**6.5.2.34** `double Dspline5 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt )`

parameter derivative of spline function with 5 nodes

## Parameters

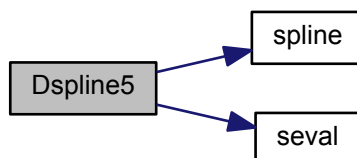
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

`dspline(t)dp(id)`

Definition at line 1003 of file `symbolic_functions.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



**6.5.2.35** `double Dspline_pos5 ( int id, double t, double t1, double p1, double t2, double p2, double t3, double p3, double t4, double p4, double t5, double p5, int ss, double dudt )`

parameter derivative of positive spline function with 5 nodes

**Parameters**

<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5

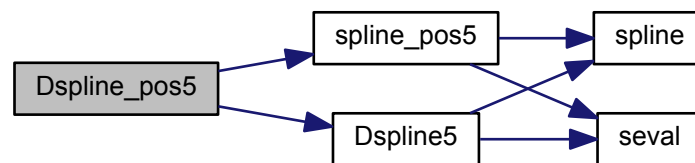
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

dspline(t)dp(id)

Definition at line 1056 of file symbolic\_functions.c.

Here is the call graph for this function:



**6.5.2.36** double Dspline10 ( int *id*, double *t*, double *t1*, double *p1*, double *t2*, double *p2*, double *t3*, double *p3*, double *t4*, double *p4*, double *t5*, double *p5*, double *t6*, double *p6*, double *t7*, double *p7*, double *t8*, double *p8*, double *t9*, double *p9*, double *t10*, double *p10*, int *ss*, double *dudt* )

parameter derivative of spline function with 10 nodes

**Parameters**

<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>t6</i>	location of node 6
<i>p6</i>	spline value at node 6
<i>t7</i>	location of node 7
<i>p7</i>	spline value at node 7
<i>t8</i>	location of node 8
<i>p8</i>	spline value at node 8



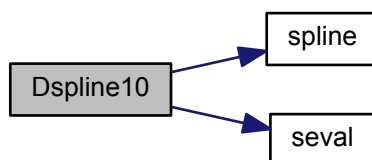
<i>t9</i>	location of node 9
<i>p9</i>	spline value at node 9
<i>t10</i>	location of node 10
<i>p10</i>	spline value at node 10
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

#### Returns

dspline(t)dp(id)

Definition at line 1112 of file symbolic\_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.37 double Dspline\_pos10 ( int *id*, double *t*, double *t1*, double *p1*, double *t2*, double *p2*, double *t3*, double *p3*, double *t4*, double *p4*, double *t5*, double *p5*, double *t6*, double *p6*, double *t7*, double *p7*, double *t8*, double *p8*, double *t9*, double *p9*, double *t10*, double *p10*, int *ss*, double *dudt* )

parameter derivative of positive spline function with 10 nodes

#### Parameters

<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1

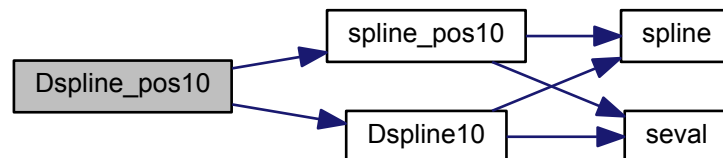
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>t6</i>	location of node 6
<i>p6</i>	spline value at node 6
<i>t7</i>	location of node 7
<i>p7</i>	spline value at node 7
<i>t8</i>	location of node 8
<i>p8</i>	spline value at node 8
<i>t9</i>	location of node 9
<i>p9</i>	spline value at node 9
<i>t10</i>	location of node 10
<i>p10</i>	spline value at node 10
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

**Returns**

dspline(t)dp(id)

Definition at line 1185 of file symbolic\_functions.c.

Here is the call graph for this function:





## Index

adjoint  
    amimodel, 21  
am\_and  
    symbolic\_functions.c, 47  
am\_bsx0  
    UserData, 28  
am\_ge  
    symbolic\_functions.c, 43  
am\_gt  
    symbolic\_functions.c, 43  
am\_if  
    symbolic\_functions.c, 46  
am\_interpType  
    UserData, 28  
am\_ism  
    UserData, 27  
am\_iter  
    UserData, 28  
am\_le  
    symbolic\_functions.c, 45  
am\_lmm  
    UserData, 28  
am\_lt  
    symbolic\_functions.c, 45  
am\_max  
    symbolic\_functions.c, 49  
am\_min  
    symbolic\_functions.c, 48  
am\_or  
    symbolic\_functions.c, 47  
am\_rootsfound  
    TempData, 26  
am\_sensi\_meth  
    UserData, 27  
am\_xdata  
    ReturnData, 24  
amici.c  
    getDiagnosis, 37  
    getDiagnosisB, 37  
    getRootDataASA, 36  
    getRootDataFSA, 34  
    setupAMI, 32  
    setupAMIB, 33  
    setupExpData, 34  
    setupReturnData, 33  
    setupUserData, 32  
amimodel, 10  
    adjoint, 21  
    amimodel, 12  
    atol, 21  
    augmento2, 19  
    checkDeps, 18  
    compileC, 13  
    coptim, 22  
    debug, 21  
    forward, 21  
    gccode, 15  
    generateC, 13  
    generateM, 16  
    getFun, 16  
    getFunArgs, 17  
    getFunCVar, 18  
    getFunDeps, 17  
    getFunSVar, 18  
    loadOldHashes, 19  
    maxsteps, 21  
    nxtrue, 21  
    nytrue, 21  
    param, 22  
    parseModel, 12  
    rtol, 21  
    t0, 21  
    writeCcode, 14  
    writeCcode\_sensi, 13  
amiwrap  
    amiwrap.m, 30  
amiwrap.c, 28  
    mexFunction, 29  
amiwrap.m, 30  
    amiwrap, 30  
atol  
    amimodel, 21  
augmento2  
    amimodel, 19  
  
checkDeps  
    amimodel, 18  
compileC  
    amimodel, 13  
coptim  
    amimodel, 22  
  
Dam\_and  
    symbolic\_functions.c, 47  
Dam\_ge  
    symbolic\_functions.c, 43  
Dam\_gt  
    symbolic\_functions.c, 43  
Dam\_if  
    symbolic\_functions.c, 46  
Dam\_le  
    symbolic\_functions.c, 45  
Dam\_lt  
    symbolic\_functions.c, 46  
Dam\_max  
    symbolic\_functions.c, 49  
Dam\_min  
    symbolic\_functions.c, 48  
Dam\_or  
    symbolic\_functions.c, 48  
Ddirac

- symbolic\_functions.c, [50](#)
- debug
  - amimodel, [21](#)
- deriv
  - spline.c, [40](#)
- dirac
  - symbolic\_functions.c, [50](#)
- Dspline10
  - symbolic\_functions.c, [63](#)
- Dspline3
  - symbolic\_functions.c, [57](#)
- Dspline4
  - symbolic\_functions.c, [59](#)
- Dspline5
  - symbolic\_functions.c, [61](#)
- Dspline\_pos10
  - symbolic\_functions.c, [64](#)
- Dspline\_pos3
  - symbolic\_functions.c, [58](#)
- Dspline\_pos4
  - symbolic\_functions.c, [60](#)
- Dspline\_pos5
  - symbolic\_functions.c, [62](#)
- ExpData, [22](#)
- forward
  - amimodel, [21](#)
- gccode
  - amimodel, [15](#)
- generateC
  - amimodel, [13](#)
- generateM
  - amimodel, [16](#)
- getDiagnosis
  - amici.c, [37](#)
- getDiagnosisB
  - amici.c, [37](#)
- getFun
  - amimodel, [16](#)
- getFunArgs
  - amimodel, [17](#)
- getFunCVar
  - amimodel, [18](#)
- getFunDeps
  - amimodel, [17](#)
- getFunSVar
  - amimodel, [18](#)
- getRootDataASA
  - amici.c, [36](#)
- getRootDataFSA
  - amici.c, [34](#)
- heaviside
  - symbolic\_functions.c, [49](#)
- loadOldHashes
  - amimodel, [19](#)
- maxsteps
  - amimodel, [21](#)
- mexFunction
  - amiwrap.c, [29](#)
- nxtrue
  - amimodel, [21](#)
- nytrue
  - amimodel, [21](#)
- param
  - amimodel, [22](#)
- parseModel
  - amimodel, [12](#)
- ReturnData, [22](#)
  - am\_xdata, [24](#)
- rtol
  - amimodel, [21](#)
- setupAMI
  - amici.c, [32](#)
- setupAMIB
  - amici.c, [33](#)
- setupExpData
  - amici.c, [34](#)
- setupReturnData
  - amici.c, [33](#)
- setupUserData
  - amici.c, [32](#)
- seval
  - spline.c, [39](#)
- sinteg
  - spline.c, [40](#)
- spline
  - spline.c, [39](#)
- spline.c
  - deriv, [40](#)
  - seval, [39](#)
  - sinteg, [40](#)
  - spline, [39](#)
- spline10
  - symbolic\_functions.c, [55](#)
- spline3
  - symbolic\_functions.c, [50](#)
- spline4
  - symbolic\_functions.c, [52](#)
- spline5
  - symbolic\_functions.c, [53](#)
- spline\_pos10
  - symbolic\_functions.c, [56](#)
- spline\_pos3
  - symbolic\_functions.c, [51](#)
- spline\_pos4
  - symbolic\_functions.c, [52](#)
- spline\_pos5
  - symbolic\_functions.c, [54](#)
- src/amici.c, [31](#)
- src/spline.c, [38](#)

src/symbolic\_functions.c, [41](#)

symbolic\_functions.c

am\_and, [47](#)

am\_ge, [43](#)

am\_gt, [43](#)

am\_if, [46](#)

am\_le, [45](#)

am\_lt, [45](#)

am\_max, [49](#)

am\_min, [48](#)

am\_or, [47](#)

Dam\_and, [47](#)

Dam\_ge, [43](#)

Dam\_gt, [43](#)

Dam\_if, [46](#)

Dam\_le, [45](#)

Dam\_lt, [46](#)

Dam\_max, [49](#)

Dam\_min, [48](#)

Dam\_or, [48](#)

Ddirac, [50](#)

dirac, [50](#)

Dspline10, [63](#)

Dspline3, [57](#)

Dspline4, [59](#)

Dspline5, [61](#)

Dspline\_pos10, [64](#)

Dspline\_pos3, [58](#)

Dspline\_pos4, [60](#)

Dspline\_pos5, [62](#)

heaviside, [49](#)

spline10, [55](#)

spline3, [50](#)

spline4, [52](#)

spline5, [53](#)

spline\_pos10, [56](#)

spline\_pos3, [51](#)

spline\_pos4, [52](#)

spline\_pos5, [54](#)

t0

amimodel, [21](#)

TempData, [24](#)

am\_rootsfound, [26](#)

UserData, [26](#)

am\_bsx0, [28](#)

am\_interpType, [28](#)

am\_ism, [27](#)

am\_iter, [28](#)

am\_lmm, [28](#)

am\_sensi\_meth, [27](#)

writeCcode

amimodel, [14](#)

writeCcode\_sensi

amimodel, [13](#)