

AMICI

Generated by Doxygen 1.8.10

Wed Nov 11 2015 13:21:53

Contents

1	AMICI 0.1 General Documentation	1
1.1	Introduction	2
1.2	Availability	2
1.3	Installation	3
2	Model Definition & Simulation	3
2.1	Model Definition	3
2.2	Model Compilation	6
2.3	Model Simulation	7
3	Examples	9
3.1	Example 1	9
3.2	Example 2	14
3.3	Example 3	19
3.4	Example 4	23
3.5	Example 5	26
3.6	Example 6	29
4	Code Organization	33
5	Class Index	33
5.1	Class List	33
6	Class Documentation	34
6.1	amimodel Class Reference	34
6.2	ExpData Struct Reference	46
6.3	ReturnData Struct Reference	47
6.4	TempData Struct Reference	50
6.5	UserData Struct Reference	55
7	File Documentation	59
7.1	amiwrap.c File Reference	59
7.2	amiwrap.m File Reference	61
7.3	src/amici.c File Reference	62
7.4	src/spline.c File Reference	71
7.5	src/symbolic_functions.c File Reference	74
	Index	99

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://www.mathworks.com/help/matlab/creating_compilers.html)

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 `xdot` 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](#) on [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_t`

```
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 Examples

In this section we include multiple examples on defining and simulating models.

[Example 1](#) : Forward Sensitivities for model with events and discontinuities.

[Example 2](#) : Forward Sensitivities for mRNA transfection model with bolus injection.

[Example 3](#) : Steady State Sensitivities.

[Example 4](#) : Adjoint Sensitivities for JAK/STAT model with parametric standard deviation.

[Example 5](#) : Adjoint Sensitivities for mRNA transfection model with bolus injection.

[Example 6](#) : Adjoint Sensitivities for simple model with analytic solution.

3.1 Example 1

3.1.1 Model Definition

```
function [model] = example_model_1_syms()
```

CVODES OPTIONS

```
% set the default absolute tolerance
model.atol = 1e-8;
% set the default relative tolerance
model.rtol = 1e-8;
% set the default maximum number of integration steps
model.maxsteps = 1e4;
% set the parametrisation of the problem options are 'log', 'log10' and
% 'lin' (default).
model.param = 'log10';
```

STATES

```
% create state syms
syms x1 x2 x3

% create state vector
x = [
x1 x2 x3
];
```

PARAMETERS (for these sensitivities will be computed)

```
% create parameter syms
syms p1 p2 p3 p4

% create parameter vector
p = [p1,p2,p3,p4];
```

CONSTANTS (for these no sensitivities will be computed) this part is optional and can be ommited

```
% create parameter syms
syms k1 k2 k3 k4

% create parameter vector
k = [k1 k2 k3 k4];
```

SYSTEM EQUATIONS

```
% create symbolic variable for time
syms t

xdot = sym(zeros(size(x)));

% piecewise defined function
xdot(1) = -p1*heaviside(t-p4)*x1;
% inhomogeneous
xdot(2) = +p2*x1*exp(-0.1*t)-p3*x2 ;
xdot(3) = -1.5*x3;
```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));

x0(1) = k1;
x0(2) = k2;
x0(3) = k3;
```

OBSERVALES

```
y = sym(zeros(1,1));

y(1) = p4 * (x1+x2+x3);
```

EVENTS this part is optional and can be ommited

```
% events fire when there is a zero crossing of the root function
root = sym(zeros(2,1));
% x3 == x2
root(1) = x3 - x2;
% x3 == x1
root(2) = x3 - x1;
```

SYSTEM STRUCT

```

model.sym.x = x;
model.sym.k = k;
model.sym.root = root;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;

end

ans =
    atol: 1e-08
    rtol: 1e-08
    maxsteps: 10000
    param: 'log10'
    sym: [1x1 struct]

```

3.1.2 Simulation

```

clear
close all
clc

```

COMPILATION

```

[exdir,~,~]=fileparts(which('example_model_1.m'));
% compile the model
amiwrap('model_example_1','example_model_1_syms',exdir)
% add the model to the path
addpath(genpath([strrep(which('amiwrap.m'),'amiwrap.m','') 'models/model_example_1']))

Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.

```

SIMULATION

```

% time vector
t = linspace(0,10,20);
p = [0.5;2;0.5;0.5];
k = [4,8,10,4];

options.sensi = 0;
options.ccode_maxsteps = 1e6;
% load mex into memory
sol = simulate_model_example_1(t,log10(p),k,[],options);

tic
sol = simulate_model_example_1(t,log10(p),k,[],options);
disp(['Time elapsed with ccodes: ' num2str(toc) ])

```

Time elapsed with ccodes: 0.00195

ODE15S

```

ode_system = @(t,x,p,k) [-p(1)*heaviside(t-p(4))*x(1);
    +p(2)*x(1)*exp(-0.1*t)-p(3)*x(2);
    -1.5*x(3)];
% event_fn = @(t,x) [x(3) - x(2);
%     x(3) - x(1)];
% 'Events',event_fn
options_ode15s = odeset('RelTol',1e-8,'AbsTol',1e-8,'MaxStep',1e4);

tic
[~, X_ode15s] = ode15s(@(t,x) ode_system(t,x,p,k),t,k(1:3),options_ode15s);
disp(['Time elapsed with ode15s: ' num2str(toc) ])

```

Time elapsed with ode15s: 0.19678

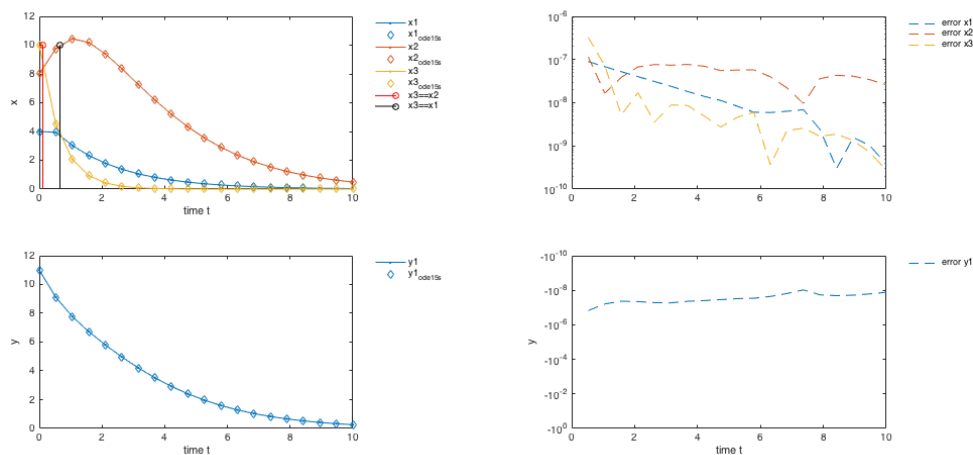
PLOTTING

```
figure
c_x = get(gca,'ColorOrder');
subplot(2,2,1)
for ix = 1:size(sol.x,2)
    plot(t,sol.x(:,ix),'.-','Color',c_x(ix,:))
    hold on
    plot(t,X_ode15s(:,ix),'d','Color',c_x(ix,:))
end
stem(sol.root(:,1),sol.root(:,1)*0+10,'r')
stem(sol.root(:,2),sol.root(:,2)*0+10,'k')
legend('x1','x1_ode15s','x2','x2_ode15s','x3','x3_ode15s','x3==x2','x3==x1','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('x')
box on
subplot(2,2,2)
plot(t,abs(sol.x-X_ode15s),'--')
set(gca,'YScale','log')
legend('error x1','error x2','error x3','Location','NorthEastOutside')
legend boxoff

subplot(2,2,3)
plot(t,sol.y,'.-','Color',c_x(1,:))
hold on
plot(t,p(4)*sum(X_ode15s,2),'d','Color',c_x(1,:))
legend('y1','y1_ode15s','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('y')
box on

subplot(2,2,4)
plot(t,sol.y-p(4)*sum(X_ode15s,2),'--')
set(gca,'YScale','log')
legend('error y1','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('y')
box on

set(gcf,'Position',[100 300 1200 500])
```



FORWARD SENSITIVITY ANALYSIS

```
options.sensi = 1;
sol = simulate_model_example_1(t,log10(p),k,[],options);
```

FINITE DIFFERENCES

```

eps = 1e-4;
xi = log10(p);
for ip = 1:4;
    xip = xi;
    xip(ip) = xip(ip) + eps;
    solp = simulate_model_example_1(t,xip,k,[],options);
    sx_fd(:, :, ip) = (solp.x - sol.x)/eps;
    sy_fd(:, :, ip) = (solp.y - sol.y)/eps;
    sroot_fd(:, :, ip) = (solp.root - sol.root)/eps;
end

```

PLOTTING

```

figure
for ip = 1:4
    subplot(4,2,ip*2-1)
    hold on
    for ix = 1:size(sol.x,2)
        plot(t,sol.sx(:,ix,ip),'.-','Color',c_x(ix,:))
        plot(t,sx_fd(:,ix,ip),'d','Color',c_x(ix,:))
    end
    legend('x1','x1_fd','x2','x2_fd','x3','x3_fd','Location','NorthEastOutside')
    legend boxoff
    title(['state sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('x')
    box on

    subplot(4,2,ip*2)
    plot(t,abs(sol.sx(:, :, ip)-sx_fd(:, :, ip)),'--')
    legend('error x1','error x2','error x3','Location','NorthEastOutside')
    legend boxoff
    title(['state sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('error')
    set(gca,'YScale','log')
    box on
end
set(gcf,'Position',[100 300 1200 500])

figure
for ip = 1:4
    subplot(4,2,ip*2-1)
    hold on
    for iy = 1:size(sol.y,2)
        plot(t,sol.sy(:,iy,ip),'.-','Color',c_x(iy,:))
        plot(t,sy_fd(:,iy,ip),'d','Color',c_x(iy,:))
    end
    legend('y1','y1_fd','Location','NorthEastOutside')
    legend boxoff
    title(['observable sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('y')
    box on

    subplot(4,2,ip*2)
    plot(t,abs(sol.sy(:, :, ip)-sy_fd(:, :, ip)),'--')
    legend('error y1','Location','NorthEastOutside')
    legend boxoff
    title(['error observable sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('error')
    set(gca,'YScale','log')
    box on
end
set(gcf,'Position',[100 300 1200 500])

figure
for ip = 1:4
    subplot(4,2,2*ip-1)
    bar(1:6,sol.sroot(1:6,:,ip),0.8)
    hold on
    bar(1:6,sroot_fd(1:6,:,ip),0.4)
    legend('x3==x2','x3==x1','x3==x2 fd','x3==x1 fd','Location','NorthEastOutside')
    legend boxoff
    title(['event sensitivity for p' num2str(ip)])
    xlabel('event #')
    ylabel('y')
    box on

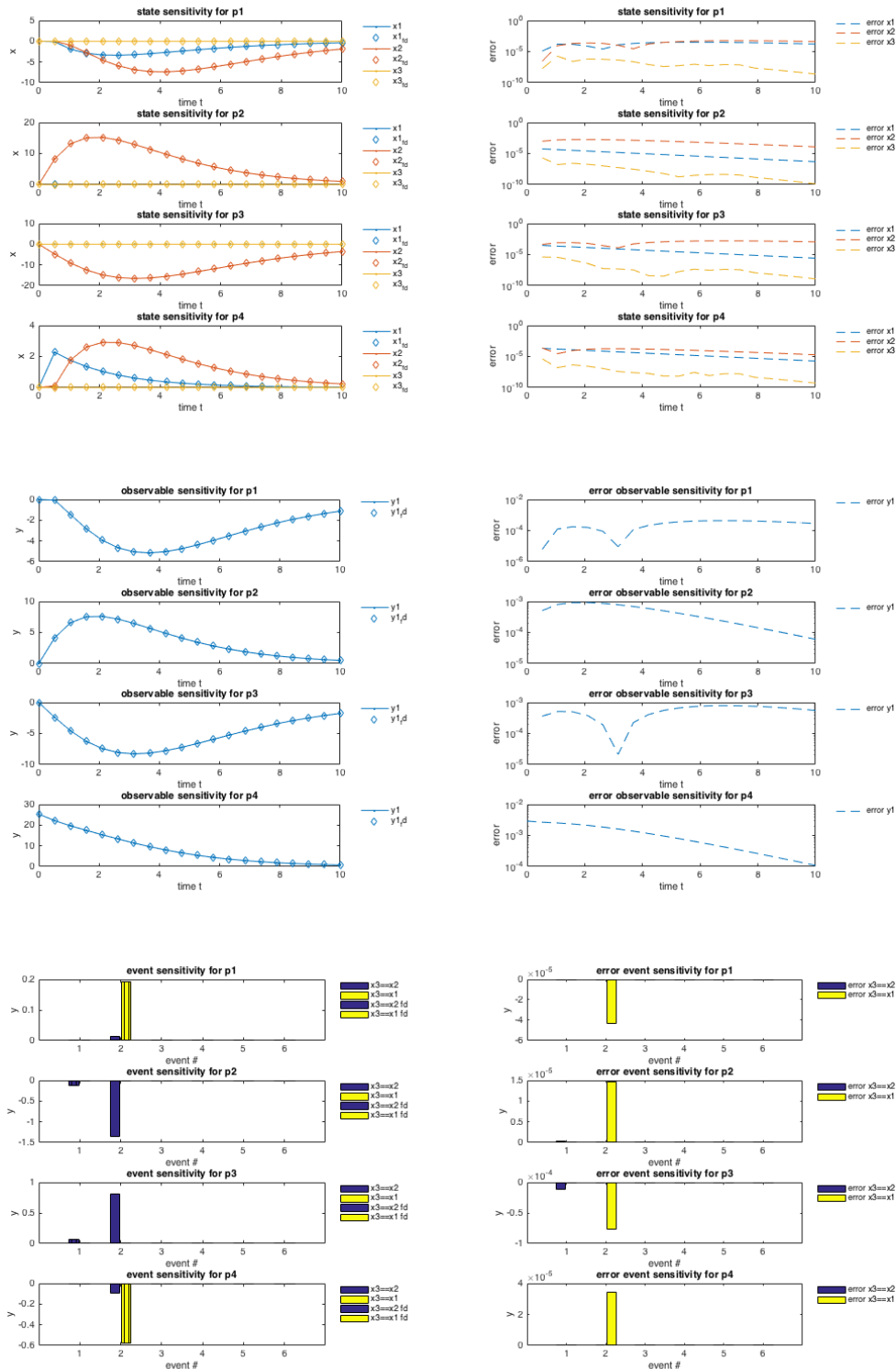
    subplot(4,2,2*ip)
    bar(1:6,sol.sroot(1:6,:,ip)-sroot_fd(1:6,:,ip),0.8)
    legend('error x3==x2','error x3==x1','Location','NorthEastOutside')
    legend boxoff

```

```

title(['error event sensitivity for p' num2str(ip)])
xlabel('event #')
ylabel('y')
box on
end
set(gcf,'Position',[100 300 1200 500])

```



3.2 Example 2

3.2.1 Model Definition

```
function [model] = example_model_2_syms()
```

CVODES OPTIONS

```
% set the default absolute tolerance
model.atol = 1e-8;
% set the default relative tolerance
model.rtol = 1e-8;
% set the default maximum number of integration steps
model.maxsteps = 1e4;
% set the parametrisation of the problem options are 'log', 'log10' and
% 'lin' (default).
model.param = 'log10';
```

STATES

```
% create state syms
syms x1 x2

% create state vector
x = [ x1 x2 ];
```

PARAMETERS (for these sensitivities will be computed)

```
% create parameter syms
syms p1 p2 p3 p4

% create parameter vector
p = [p1,p2,p3,p4];
```

SYSTEM EQUATIONS

```
% create symbolic variable for time
syms t

xdot = sym(zeros(size(x)));

% piecewise defined function
xdot(1) = -p1*x1 + dirac(t-p2);
% inhomogeneous
xdot(2) = p3*x1 - p4*x2 ;
```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));

x0(1) = 0;
x0(2) = 0;
```

OBSERVALES

```
y = sym(zeros(1,1));

y(1) = x2;
```

SYSTEM STRUCT

```
model.sym.x = x;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;
```



```

end

ans =
    atol: 1e-08
    rtol: 1e-08
    maxsteps: 10000
    param: 'log10'
    sym: [1x1 struct]

```

3.2.2 Simulation

```
clear
```

COMPILATION

```

[exdir,~,~]=fileparts(which('example_model_2.m'));
% compile the model
amiwrap('model_example_2','example_model_2_syms',exdir)

```

```

Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.

```

SIMULATION

```

% time vector
t = linspace(0,3,1001);
p = [1;0.5;2;3];
k = [];

options.sensi = 0;
options.ccode_maxsteps = 1e6;
% load mex into memory
[msg] = which('simulate_model_example_2'); % fix for inaccessability problems
sol = simulate_model_example_2(t,log10(p),k,[],options);

tic
sol = simulate_model_example_2(t,log10(p),k,[],options);
disp(['Time elapsed with amiwrap: ' num2str(toc) ])

```

Time elapsed with amiwrap: 0.0019205

ODE15S

```

sig = 1e-2;
delta_num = @(tau) exp(-1/2*(tau/sig).^2)/(sqrt(2*pi)*sig);

ode_system = @(t,x,p,k) [-p(1)*x(1)+delta_num(t-p(2));
    +p(3)*x(1) - p(4)*x(2)];

options_ode45 = odeset('RelTol',1e-8,'AbsTol',1e-8,'MaxStep',1e4);

tic
[~, X_ode45] = ode45(@(t,x) ode_system(t,x,p,k),t,[0;0],options_ode45);
disp(['Time elapsed with ode45: ' num2str(toc) ])

```

Time elapsed with ode45: 0.042852

PLOTTING

```

figure
c_x = get(gca,'ColorOrder');
subplot(2,2,1)
for ix = 1:size(sol.x,2)

```

```

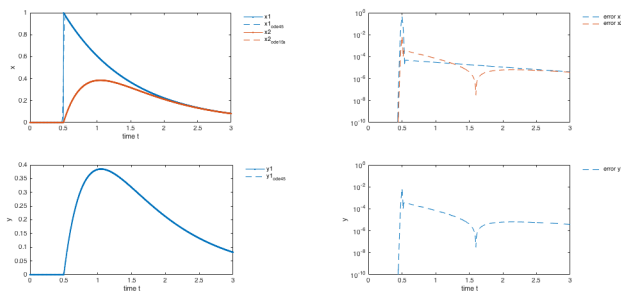
plot(t,sol.x(:,ix),'.-','Color',c_x(ix,:))
hold on
plot(t,X_ode45(:,ix),'--','Color',c_x(ix,:))
end

legend('x1','x1_ode45','x2','x2_ode15s','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('x')
box on
subplot(2,2,2)
plot(t,abs(sol.x-X_ode45(:,2)), '--','Color',c_x(1,:))
set(gca,'YScale','log')
ylim([1e-10,1e0])
legend('error x1','error x2','Location','NorthEastOutside')
legend boxoff

subplot(2,2,3)
plot(t,sol.y, '.-','Color',c_x(1,:))
hold on
plot(t,X_ode45(:,2), '--','Color',c_x(1,:))
legend('y1','y1_ode45','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('y')
box on

subplot(2,2,4)
plot(t,abs(sol.y-X_ode45(:,2)), '--','Color',c_x(1,:))
set(gca,'YScale','log')
ylim([1e-10,1e0])
legend('error y1','error y2','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('y')
box on
set(gcf,'Position',[100 300 1200 500])

```



FORWARD SENSITIVITY ANALYSIS

```

options.sensi = 1;

sol = simulate_model_example_2(t,log10(p),k,[],options);

```

FINITE DIFFERENCES

```

eps = 1e-4;
xi = log10(p);
for ip = 1:4;
    xip = xi;
    xip(ip) = xip(ip) + eps;
    solp = simulate_model_example_2(t,xip,k,[],options);
    sx_fd(:,ip) = (solp.x - sol.x)/eps;
    sy_fd(:,ip) = (solp.y - sol.y)/eps;
end

```

PLOTTING

```

figure
for ip = 1:4
    subplot(4,2,ip*2-1)

```

```

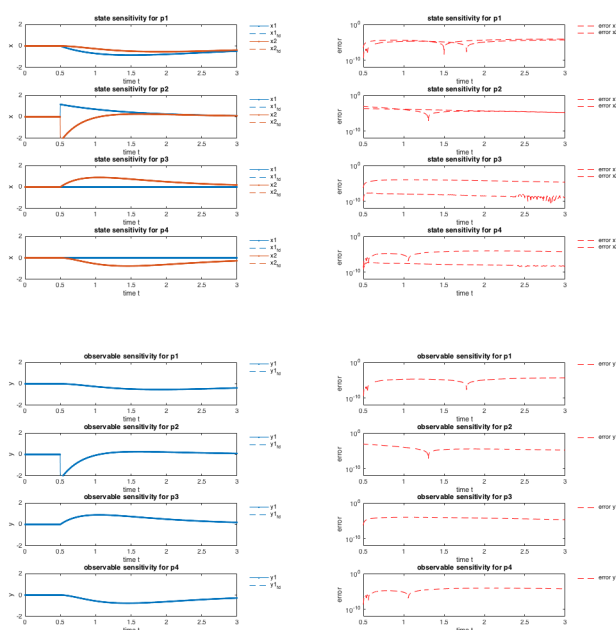
hold on
for ix = 1:size(sol.x,2)
    plot(t,sol.sx(:,ix,ip),'.-','Color',c_x(ix,:))
    plot(t,sx_fd(:,ix,ip),'--','Color',c_x(ix,:))
end
ylim([-2,2])
legend('x1','x1_fd','x2','x2_fd','Location','NorthEastOutside')
legend boxoff
title(['state sensitivity for p' num2str(ip)])
xlabel('time t')
ylabel('x')
box on

subplot(4,2,ip*2)
plot(t,abs(sol.sx(:,ix,ip)-sx_fd(:,ix,ip)),'r--')
legend('error x1','error x2','Location','NorthEastOutside')
legend boxoff
title(['state sensitivity for p' num2str(ip)])
xlabel('time t')
ylabel('error')
ylim([1e-12,1e0])
set(gca,'YScale','log')
box on
end
set(gcf,'Position',[100 300 1200 500])

figure
for ip = 1:4
    subplot(4,2,ip*2-1)
    hold on
    for iy = 1:size(sol.y,2)
        plot(t,sol.sy(:,iy,ip),'.-','Color',c_x(iy,:))
        plot(t,sy_fd(:,iy,ip),'--','Color',c_x(iy,:))
    end
    ylim([-2,2])
    legend('y1','y1_fd','Location','NorthEastOutside')
    legend boxoff
    title(['observable sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('y')
    box on

    subplot(4,2,ip*2)
    plot(t,abs(sol.sy(:,iy,ip)-sy_fd(:,iy,ip)),'r--')
    legend('error y1','Location','NorthEastOutside')
    legend boxoff
    title(['observable sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('error')
    ylim([1e-12,1e0])
    set(gca,'YScale','log')
    box on
end
set(gcf,'Position',[100 300 1200 500])

```



3.3 Example 3

3.3.1 Model Definition

```
function [model] = example_model_3_syms()
```

CVODES OPTIONS

```
% set the default absolute tolerance
model.atol = 1e-8;
% set the default relative tolerance
model.rtol = 1e-8;
% set the default maximum number of integration steps
model.maxsteps = 1e4;
% set the parametrisation of the problem options are 'log', 'log10' and
% 'lin' (default).
model.param = 'log10';
```

STATES

```
% create state syms
syms x1 x2 x3

% create state vector
x = [
x1 x2 x3
];
```

PARAMETERS (for these sensitivities will be computed)

```
% create parameter syms
syms p1 p2 p3 p4 p5

% create parameter vector
p = [p1,p2,p3,p4,p5];
```

CONSTANTS (for these no sensitivities will be computed) this part is optional and can be omitted

```
% create parameter syms
syms k1 k2 k3 k4

% create parameter vector
k = [k1 k2 k3 k4];
```

SYSTEM EQUATIONS

```
% create symbolic variable for time
syms t

xdot = sym(zeros(size(x)));

% piecewise defined function
xdot(1) = -2*p1*x1^2 - p2*x1*x2 + 2*p3*x2 + p4*x3 + p5;
% inhomogeneous
xdot(2) = +p1*x1^2 - p2*x1*x2 - p3*x2 + p4*x3;
xdot(3) = p2*x1*x2 - p4*x(3) - k4*x(3);
```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));

x0(1) = k1;
x0(2) = k2;
x0(3) = k3;
```

OBSERVALES

```
y = sym(zeros(1,1));
y = x;
```

SYSTEM STRUCT

```
model.sym.x = x;
model.sym.k = k;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;

end

ans =
    atol: 1e-08
    rtol: 1e-08
    maxsteps: 10000
    param: 'log10'
    sym: [1x1 struct]
```

3.3.2 Simulation

```
clear
```

COMPILATION

```
[exdir,~,~]=fileparts(which('example_model_3.m'));
% compile the model
amiwrap('model_example_3','example_model_3_syms',exdir)
% add the model to the path
addpath(genpath([strrep(which('amiwrap.m'),'amiwrap.m','') 'models/model_example_3']))

Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.
```

SIMULATION

```
% time vector
t = linspace(0,300,20);
p = [1;0.5;0.4;2;0.1];
k = [0.1,0.4,0.7,1];

options.sensi = 0;
options.ccode_maxsteps = 1e6;
% load mex into memory
sol = simulate_model_example_3(t,log10(p),k,[],options);

tic
sol = simulate_model_example_3(t,log10(p),k,[],options);
disp(['Time elapsed with ccodes: ' num2str(toc) ])
```

```
Time elapsed with ccodes: 0.002146
```

ODE15S

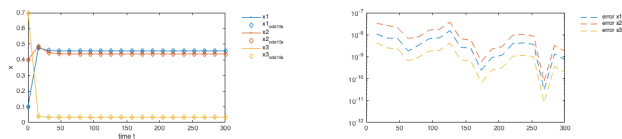
```
ode_system = @(t,x,p,k) [-2*p(1)*x(1)^2 - p(2)*x(1)*x(2) + 2*p(3)*x(2) + p(4)*x(3) + p(5);
    + p(1)*x(1)^2 - p(2)*x(1)*x(2) - p(3)*x(2) + p(4)*x(3);
    + p(2)*x(1)*x(2) - p(4)*x(3) - k(4)*x(3)];
options_ode15s = odeset('RelTol',1e-8,'AbsTol',1e-8,'MaxStep',1e4);

tic
[~, X_ode15s] = ode15s(@(t,x) ode_system(t,x,p,k),t,k(1:3),options_ode15s);
disp(['Time elapsed with ode15s: ' num2str(toc) ])
```

Time elapsed with ode15s: 0.18018

PLOTTING

```
figure
c_x = get(gca,'ColorOrder');
subplot(2,2,1)
for ix = 1:size(sol.x,2)
    plot(t,sol.x(:,ix),'.-','Color',c_x(ix,:))
    hold on
    plot(t,X_ode15s(:,ix),'d','Color',c_x(ix,:))
end
legend('x1','x1_ode15s','x2','x2_ode15s','x3','x3_ode15s','Location','NorthEastOutside')
legend boxoff
xlabel('time t')
ylabel('x')
box on
subplot(2,2,2)
plot(t,abs(sol.x-X_ode15s),'--')
set(gca,'YScale','log')
legend('error x1','error x2','error x3','Location','NorthEastOutside')
legend boxoff
set(gcf,'Position',[100 300 1200 500])
```



FORWARD SENSITIVITY ANALYSIS

```
options.sensi = 1;
options.sens_ind = [3,1,2,4];

sol = simulate_model_example_3(t,log10(p),k,[],options);
```

FINITE DIFFERENCES

```
eps = 1e-3;

xi = log10(p);
for ip = 1:4;
    xip = xi;
    xip(ip) = xip(ip) + eps;
    solp = simulate_model_example_3(t,xip,k,[],options);
    sx_fd(:,ip) = (solp.x - sol.x)/eps;
    sy_fd(:,ip) = (solp.y - sol.y)/eps;
end
```

PLOTTING

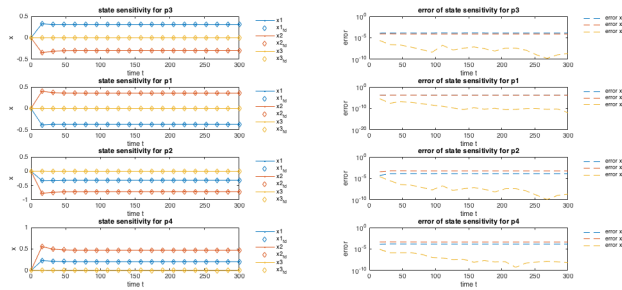
```
figure
for ip = 1:4
```

```

subplot(4,2,ip*2-1)
hold on
for ix = 1:size(sol.x,2)
    plot(t,sol.sx(:,ix,ip),'.-','Color',c_x(ix,:))
    plot(t,sx_fd(:,ix,options.sens_ind(ip)),'d','Color',c_x(ix,:))
end
legend('x1','x1_fd','x2','x2_fd','x3','x3_fd','Location','NorthEastOutside')
legend boxoff
title(['state sensitivity for p' num2str(options.sens_ind(ip))])
xlabel('time t')
ylabel('x')
box on

subplot(4,2,ip*2)
plot(t,abs(sol.sx(:, :, ip)-sx_fd(:, :, options.sens_ind(ip))), '--')
legend('error x1','error x2','error x3','Location','NorthEastOutside')
legend boxoff
title(['error of state sensitivity for p' num2str(options.sens_ind(ip))])
xlabel('time t')
ylabel('error')
set(gca,'YScale','log')
box on
end
set(gcf,'Position',[100 300 1200 500])

```



STEADY STATE SENSITIVITY

```

sssens = NaN(size(sol.sx));
for it = 2:length(t)
    tt = [0,t(it)];
    options.sensi_meth = 'ss';
    solss = simulate_model_example_3(tt,log10(p),k,[],options);
    sssens(it,:) = solss.sx;
    ssxdot(it,:) = solss.xdot;
end

```

PLOTTING

```

figure
for ip = 1:4
    subplot(4,2,ip*2-1)
    hold on
    for ix = 1:size(sol.x,2)
        plot(t,sol.sx(:,ix,ip),'.-','Color',c_x(ix,:))
        plot(t,sssens(:,ix,ip),'d-','Color',c_x(ix,:))
    end
    legend('x1','x1_ss','x2','x2_ss','x3','x3_ss','Location','NorthEastOutside')
    legend boxoff
    title(['state steady sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('x')
    box on

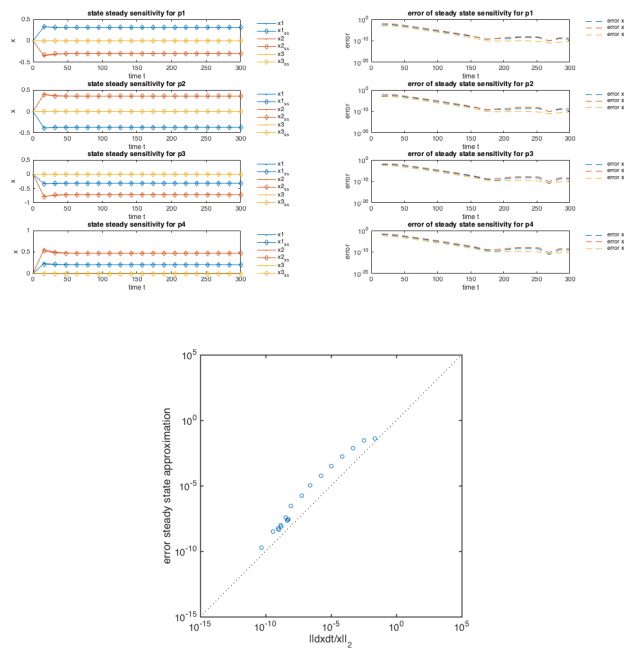
    subplot(4,2,ip*2)
    plot(t,abs(sol.sx(:, :, ip)-sssens(:, :, ip)), '--')
    legend('error x1','error x2','error x3','Location','NorthEastOutside')
    legend boxoff
    title(['error of steady state sensitivity for p' num2str(ip)])
    xlabel('time t')
    ylabel('error')
    set(gca,'YScale','log')
    box on
end
set(gcf,'Position',[100 300 1200 500])

```

```

figure
scatter(sqrt(sum((ssxdot./sol.x).^2,2)),sqrt(sum(sum((sol.sx-sssens).^2,2),3)))
hold on
plot([1e-15,1e5],[1e-15,1e5],'k:')
set(gca,'YScale','log')
set(gca,'XScale','log')
box on
axis square
xlabel('||dxdt/x||_2')
ylabel('error steady state approximation')
set(gca,'FontSize',15)
set(gca,'LineWidth',1.5)
set(gcf,'Position',[100 300 1200 500])

```



3.4 Example 4

3.4.1 Model Definition

```
function [model] = example_model_4_syms()
```

CVODES OPTIONS

```

model.atol = 1e-12;
model.rtol = 1e-8;
model.maxsteps = 1e4;
model.param = 'log10';

```

STATES

```

syms STAT pSTAT pSTAT_pSTAT npSTAT_npSTAT nSTAT1 nSTAT2 nSTAT3 nSTAT4 nSTAT5

x = [
STAT, pSTAT, pSTAT_pSTAT, npSTAT_npSTAT, nSTAT1, nSTAT2, nSTAT3, nSTAT4, nSTAT5 ...
];

```

PARAMETERS

```

syms p1 p2 p3 p4 init_STAT Omega_cyt Omega_nuc sp1 sp2 sp3 sp4 sp5 offset_tSTAT offset_pSTAT scale_tSTAT scale_pSTAT sigma_pSTAT sigma_tSTAT sigma_nuc

p = [p1,p2,p3,p4,init_STAT,sp1,sp2,sp3,sp4,sp5,offset_tSTAT,offset_pSTAT,scale_tSTAT,scale_pSTAT,sigma_pSTAT,sigma_tSTAT,sigma_nuc];

k = [Omega_cyt,Omega_nuc];

```


INPUT

```
syms t
u(1) = spline_pos5(t, 0.0, sp1, 5.0, sp2, 10.0, sp3, 20.0, sp4, 60.0, sp5, 0, 0.0);
```

SYSTEM EQUATIONS

```
xdot = sym(zeros(size(x)));

xdot(1) = (Omega_nuc*p4*nSTAT5 - Omega_cyt*STAT*p1*u(1))/Omega_cyt;
xdot(2) = STAT*p1*u(1) - 2*p2*pSTAT^2;
xdot(3) = p2*pSTAT^2 - p3*pSTAT_pSTAT;
xdot(4) = -(Omega_nuc*p4*npSTAT_npSTAT - Omega_cyt*p3*pSTAT_pSTAT)/Omega_nuc;
xdot(5) = -p4*(nSTAT1 - 2*npSTAT_npSTAT);
xdot(6) = p4*(nSTAT1 - nSTAT2);
xdot(7) = p4*(nSTAT2 - nSTAT3);
xdot(8) = p4*(nSTAT3 - nSTAT4);
xdot(9) = p4*(nSTAT4 - nSTAT5);
```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));
x0(1) = init_STAT;
```

OBSERVABLES

```
y = sym(zeros(3,1));

y(1) = offset_pSTAT + scale_pSTAT/init_STAT*(pSTAT + 2*pSTAT_pSTAT);
y(2) = offset_tSTAT + scale_tSTAT/init_STAT*(STAT + pSTAT + 2*(pSTAT_pSTAT));
y(3) = u(1);
```

SIGMA

```
sigma_y = sym(size(y));

sigma_y(1) = sigma_pSTAT;
sigma_y(2) = sigma_tSTAT;
sigma_y(3) = sigma_pEpoR;
```

SYSTEM STRUCT

```
model.sym.x = x;
model.sym.u = u;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.k = k;
model.sym.x0 = x0;
model.sym.y = y;
model.sym.sigma_y = sigma_y;
```

```
end
```

```
ans =
    atol: 1e-12
    rtol: 1e-08
  maxsteps: 10000
    param: 'log10'
      sym: [1x1 struct]
```

3.4.2 Simulation

```

clear
% compile the model
[exdir,~,~]=fileparts(which('example_model_4.m'));
amiwrap('model_example_4','example_model_4_syms',exdir)

num = xlsread(fullfile(exdir,'pnas_data_original.xls'));

t = num(:,1);

D.Y = num(:,[2,4,6]);
D.Sigma_Y = NaN(size(D.Y));

kappa = [1.4,0.45];

xi = [0.595102743982229
      2.999999999999997
      -0.948930681736172
      -0.00751433662124028
      0
      -2.78593598707493
      -0.256066441623149
      -0.07511250551843
      -0.411247187909784
      -4.999999999999546
      -0.735327875726678
      -0.64146041506584
      -0.107897525629158
      0.0272647740863191
      -0.5
      0
      -0.5];

options.sensi = 0;
sol = simulate_model_example_4(t,xi,kappa,D,options);

figure
for iy = 1:3
    subplot(2,2,iy)
    plot(t,D.Y(:,iy),'rx')
    hold on
    plot(t,sol.y(:,iy),'.-')
    xlim([0,60])
    xlabel('t')
    switch(iy)
        case 1
            ylabel('pStat')
        case 2
            ylabel('tStat')
        case 3
            ylabel('pEpoR')
    end
    ylim([0,1.2])
end
set(gcf,'Position',[100 300 1200 500])

% generate new
xi_rand = xi + 0.1;
options.sensi = 1;
options.sensi_meth = 'adjoint';
sol = simulate_model_example_4(t,xi_rand,kappa,D,options);

options.sensi = 0;
eps = 1e-4;
fd_grad = NaN(length(xi),1);
for ip = 1:length(xi)
    xip = xi_rand;
    xip(ip) = xip(ip) + eps;
    psol = simulate_model_example_4(t,xip,kappa,D,options);
    fd_grad(ip) = (psol.llh-sol.llh)/eps;
end

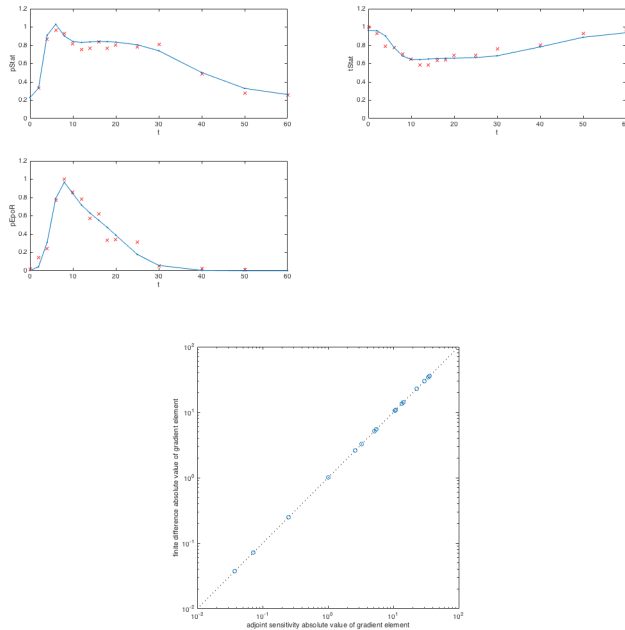
figure
scatter(abs(sol.sllh),abs(fd_grad))
set(gca,'XScale','log')
set(gca,'YScale','log')
xlim([1e-2,1e2])
ylim([1e-2,1e2])
box on
hold on
axis square
plot([1e-2,1e2],[1e-2,1e2],'k:')
xlabel('adjoint sensitivity absolute value of gradient element')
ylabel('finite difference absolute value of gradient element')
set(gcf,'Position',[100 300 1200 500])

```

```

Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.

```



3.5 Example 5

3.5.1 Model Definition

```
function [model] = example_model_5_syms()
```

CVODES OPTIONS

```

% set the default absolute tolerance
model.atol = 1e-8;
% set the default relative tolerance
model.rtol = 1e-8;
% set the default maximum number of integration steps
model.maxsteps = 1e4;
% set the parametrisation of the problem options are 'log', 'log10' and
% 'lin' (default).
model.param = 'log10';

```

STATES

```

% create state syms
syms x1 x2

% create state vector
x = [ x1 x2 ];

```

PARAMETERS (for these sensitivities will be computed)

```

% create parameter syms
syms p1 p2 p3 p4

% create parameter vector
p = [p1,p2,p3,p4];

```

SYSTEM EQUATIONS

```
% create symbolic variable for time
syms t

xdot = sym(zeros(size(x)));

% piecewise defined function
xdot(1) = -p1*x1 + dirac(t-p2);
% inhomogeneous
xdot(2) = p3*x1 - p4*x2 ;
```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));

x0(1) = 0;
x0(2) = 0;
```

OBSERVALES

```
y = sym(zeros(1,1));

y(1) = x2;
```

SYSTEM STRUCT

```
model.sym.x = x;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;

end

ans =
    atol: 1e-08
    rtol: 1e-08
  maxsteps: 10000
    param: 'log10'
     sym: [1x1 struct]
```

3.5.2 Simulation

```
clear
```

COMPILATION

```
[exdir,~,~]=fileparts(which('example_model_5.m'));
% compile the model
amiwrap('model_example_5','example_model_5_syms',exdir)
```

```
Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.
```

SIMULATION

```

% time vector
tout = linspace(0,4,9);
tfine = linspace(0,4,10001);
p = [1;0.4;2;3];
k = [];

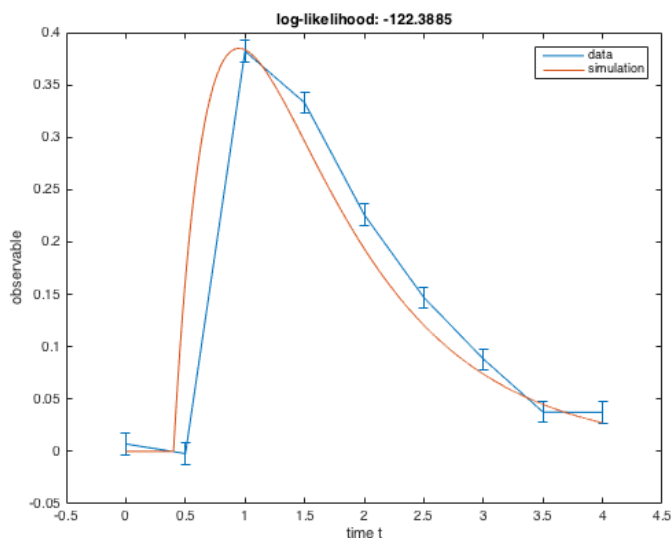
D.Y = [ 0.00714742903826096
        -0.00204966058299775
         0.382159034587845
         0.33298932672138
         0.226111476113441
         0.147028440865854
         0.0882468698791813
         0.0375887796628869
         0.0373422340295005];

D.Sigma_Y = 0.01*ones(size(D.Y));

options.sensi = 1;
options.sensi_meth = 'adjoint';
options.cvode_maxsteps = 1e4;
sol = simulate_model_example_5(tout,log10(p),k,D,options);
options.sensi = 0;
solfine = simulate_model_example_5(tfine,log10(p),k,[],options);

figure
errorbar(tout,D.Y,D.Sigma_Y)
hold on
plot(tfine,solfine.y)
legend('data','simulation')
xlabel('time t')
ylabel('observable')
title(['log-likelihood: ' num2str(sol.llh) ])

```



FD

```

eps = 1e-4;
xi = log10(p);
grad_fd_f = NaN(4,1);
grad_fd_b = NaN(4,1);
for ip = 1:4;
    options.sensi = 0;
    xip = xi;
    xip(ip) = xip(ip) + eps;
    solpf = simulate_model_example_5(tout,xip,k,D,options);
    grad_fd_f(ip,1) = (solpf.llh-sol.llh)/eps;
    xip = xi;
    xip(ip) = xip(ip) - eps;
    solpb = simulate_model_example_5(tout,xip,k,D,options);
    grad_fd_b(ip,1) = -(solpb.llh-sol.llh)/eps;
end

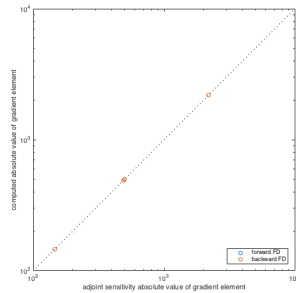
figure
plot(abs(grad_fd_f),abs(sol.sllh),'o')
hold on

```

```

plot(abs(grad_fd_b),abs(sol.sllh),'o')
set(gca,'XScale','log')
set(gca,'YScale','log')
hold on
axis square
plot([1e2,1e4],[1e2,1e4],'k:')
xlim([1e2,1e4])
ylim([1e2,1e4])
legend('forward FD','backward FD','Location','SouthEast')
xlabel('adjoint sensitivity absolute value of gradient element')
ylabel('computed absolute value of gradient element')
set(gcf,'Position',[100 300 1200 500])

```



3.6 Example 6

3.6.1 Model Definition

```
function [model] = example_model_6_syms()
```

CVODES OPTIONS

```

% set the default absolute tolerance
model.atol = 1e-8;
% set the default relative tolerance
model.rtol = 1e-8;
% set the default maximum number of integration steps
model.maxsteps = 1e4;
% set the parametrisation of the problem options are 'log', 'log10' and
% 'lin' (default).
model.param = 'log10';

```

STATES

```

% create state syms
syms x1

% create state vector
x = [ x1];

```

PARAMETERS (for these sensitivities will be computed)

```

% create parameter syms
syms p1 p2 p3

% create parameter vector
p = [p1 p2 p3];

```

SYSTEM EQUATIONS

```

% create symbolic variable for time
syms t

xdot = sym(zeros(size(x)));

% piecewise defined function
xdot(1) = -p1*x1*heaviside(t-2) + p2;

```

INITIAL CONDITIONS

```
x0 = sym(zeros(size(x)));
x0(1) = p3;
```

OBSERVALES

```
y = sym(zeros(1,1));
y(1) = x1;
```

SYSTEM STRUCT

```
model.sym.x = x;
model.sym.xdot = xdot;
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;

end

ans =
    atol: 1e-08
    rtol: 1e-08
    maxsteps: 10000
    param: 'log10'
    sym: [1x1 struct]
```

3.6.2 Simulation

```
clear
```

COMPILATION

```
[exdir,~,~]=fileparts(which('example_model_6.m'));
% compile the model
amiwrap('model_example_6','example_model_6_syms',exdir)
```

```
Generating model struct ...
Parsing model struct ...
Generating C code ...
headers | wrapfunctions |
Compiling mex file ...
Building with 'Xcode with Clang'.
MEX completed successfully.
```

SIMULATION

```
% time vector
t = [linspace(0,4,5)];
p = [1.1,0.3,1];
k = [];

% D.Y = [      1.0171
%      1.1761
%      1.1680
%      1.1359
%      1.1778
%      1.3423
%      1.3079
%      1.2784
%      1.4976
%      1.5903
%      1.6585
%      1.4688
%      1.0999
```

```
%      1.0128
%      0.7198
%      0.9814
%      0.6755
%      0.5091
%      0.4471
%      0.5249
%      0.3288];

D.Y = [      1.0171
      1.3423
      1.6585
      0.9814
      0.3288];

D.Sigma_Y = 0.1*ones(size(D.Y));

options.sensi = 1;
options.sensi_meth = 'adjoint';
options.cvode_maxsteps = 1e6;
options.cvode_rtol = 1e-12;
options.cvode_atol = 1e-12;
% load mex into memory
[msg] = which('simulate_model_example_6'); % fix for inaccessability problems
sol = simulate_model_example_6(t, log10(p), k, D, options);
```

Plot

```
figure
subplot(3,1,1)
errorbar(t,D.Y,D.Sigma_Y)
hold on
% plot(t,sol.y)

xlabel('time t')
ylabel('observable')
title(['log-likelihood: ' num2str(sol.llh) ])

y = (p(2)*t + p(3)).*(t<2) + ( (2*p(2)+p(3)-p(2)/p(1))*exp(-p(1)*(t-2))+p(2)/p(1) ).*(t>=2);

tfine = linspace(0,4,100001);
xfine = (p(2)*tfine + 1).*(tfine<2) + ( (2*p(2)+p(3)-p(2)/p(1))*exp(-p(1)*(tfine-2))+p(2)/p(1) ).*(tfine>=2);

mu = zeros(1,length(tfine));
for it = 1:length(t)
if (t(it)<=2)
mu = mu + ((y(it)-D.Y(it))/(D.Sigma_Y(it)^2))*(tfine<=t(it));
else
mu = mu + ((y(it)-D.Y(it))/(D.Sigma_Y(it)^2))*exp(p(1)*(tfine-t(it))).*(tfine<=t(it)).*(tfine>2) + ((y(it)-D.Y(it))/(D.Sigma_Y(it)^2))*exp(p(1)*(tfine-t(it))).*(tfine>2);
end
end
plot(tfine,xfine)
legend('data','simulation')
xlim([min(t)-0.5,max(t)+0.5])
subplot(3,1,2)
plot(tfine,mu)
ylabel('adjoint')
xlabel('time t')
xlim([min(t)-0.5,max(t)+0.5])

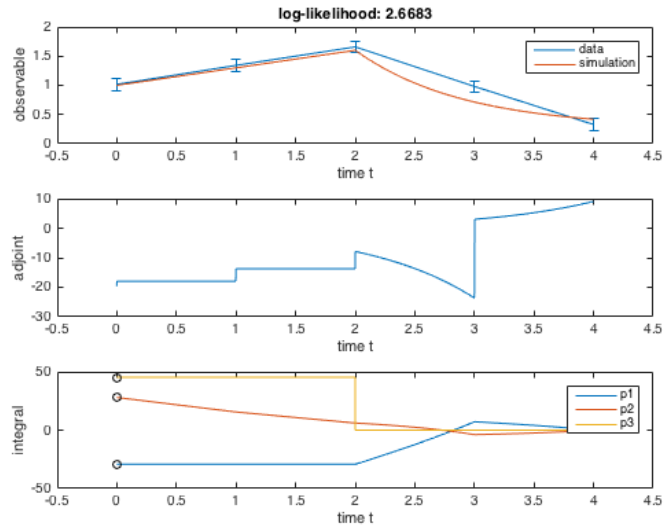
subplot(3,1,3)

plot(fliplr(tfine),-cumsum(fliplr(-mu.*xfine.*(tfine>2)))*p(1)*log(10)*(t(end)/numel(tfine)))
hold on
plot(fliplr(tfine),-cumsum(fliplr(mu))*p(2)*log(10)*(t(end)/numel(tfine)))
plot(tfine,-mu(1)*p(3)*log(10)*(tfine<2))
xlim([min(t)-0.5,max(t)+0.5])
ylabel('integral')
xlabel('time t')

legend('p1','p2','p3')

grad(1,1) = -trapz(tfine,-mu.*xfine.*(tfine>2))*p(1)*log(10);
grad(2,1) = -trapz(tfine,mu)*p(2)*log(10);
grad(3,1) = -mu(1)*p(3)*log(10);

plot(zeros(3,1),grad,'ko')
```

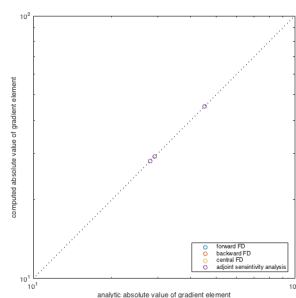
FD

```

eps = 1e-5;
xi = log10(p);
grad_fd_f = NaN(3,1);
grad_fd_b = NaN(3,1);
for ip = 1:3;
    options.sensi = 0;
    xip = xi;
    xip(ip) = xip(ip) + eps;
    solp = simulate_model_example_6(t,xip,k,D,options);
    grad_fd_f(ip,1) = (solp.llh-sol.llh)/eps;
    xip = xi;
    xip(ip) = xip(ip) - eps;
    solp = simulate_model_example_6(t,xip,k,D,options);
    grad_fd_b(ip,1) = -(solp.llh-sol.llh)/eps;
end

figure
plot(abs(grad),abs(grad_fd_f),'o')
hold on
plot(abs(grad),abs(grad_fd_b),'o')
plot(abs(grad),mean([abs(grad_fd_b),abs(grad_fd_f)],2),'o')
plot(abs(grad),abs(sol.sllh),'o')
plot([1e1,1e2],[1e1,1e2],'k:')
set(gca,'XScale','log')
set(gca,'YScale','log')
axis square
legend('forward FD','backward FD','central FD','adjoint sensitivity analysis','Location','SouthEast')
xlabel('analytic absolute value of gradient element')
ylabel('computed absolute value of gradient element')
set(gcf,'Position',[100 300 1200 500])

```



4 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.

5 Class Index

5.1 Class List

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

<code>amimodel</code>	
Amimodel is the object in which all model definitions are stored	34
<code>ExpData</code>	
Struct that carries all information about experimental data	46
<code>ReturnData</code>	
Struct that stores all data which is later returned by the mex function	47
<code>TempData</code>	
Struct that provides temporary storage for different variables	50
<code>UserData</code>	
Struct that stores all user provided data	55

6 Class Documentation

6.1 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*
- `matlabtypesubstitute wrap_path`
 - path to wrapper*

6.1.1 Detailed Description

Definition at line 17 of file amimodel.m.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `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 285 of file amimodel.m.

6.1.3 Member Function Documentation

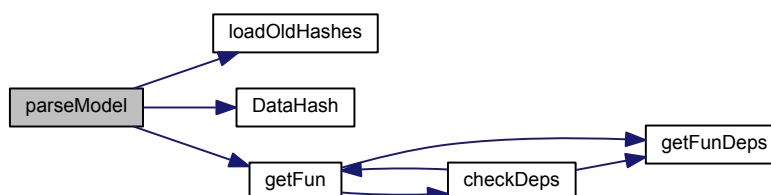
6.1.3.1 `mlhsInnerSubst<::amimodel> parseModel ()`

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:



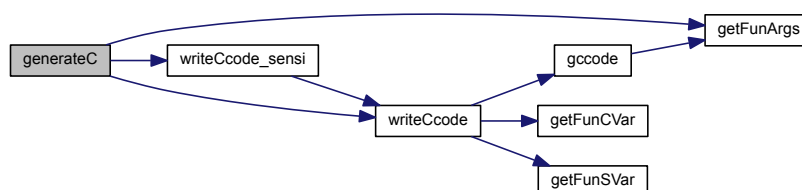
6.1.3.2 mlhsInnerSubst<::amimodel> generateC ()

Return values

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

Definition at line 18 of file generateC.m.

Here is the call graph for this function:



6.1.3.3 mlhsInnerSubst<::amimodel> compileC ()

Return values

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

Definition at line 18 of file compileC.m.

6.1.3.4 mlhsInnerSubst<::amimodel> writeCcode_sensi (::symbolic svar, ::fileid fid)

Parameters

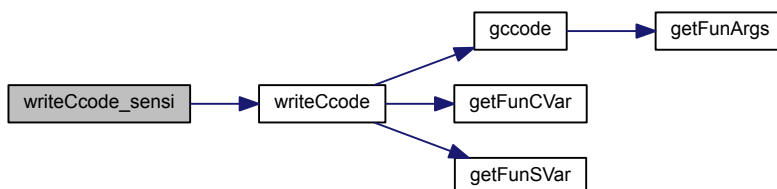
<i>svar</i>	symbolic variable which is later on passed to gccode
<i>fid</i>	file id in which the final expression is written

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:



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

Parameters

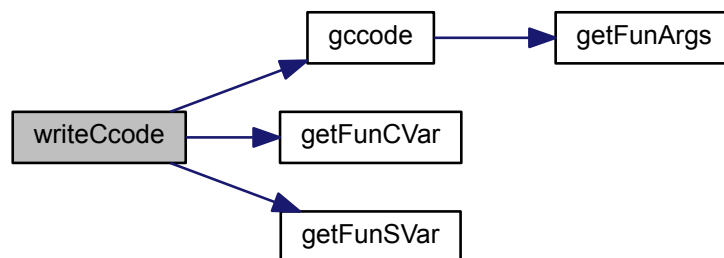
<i>funstr</i>	function for which C code is to be generated
<i>fid</i>	file id in which the final expression is written
<i>ip</i>	index for symbolic variable, if applicable svar(:,ip) will be passed to gccode
<i>jp</i>	index for symbolic variable, if applicable svar(:,ip,jp) will be passed to gccode

Return values

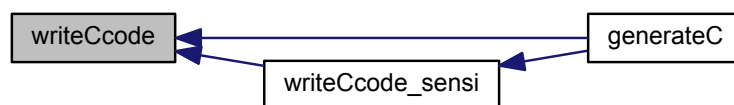
<i>this</i>	model definition object
-------------	-------------------------

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.6 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

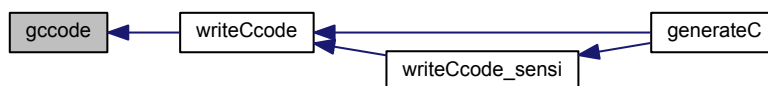
<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file gccode.m.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.7 `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`.

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

Parameters

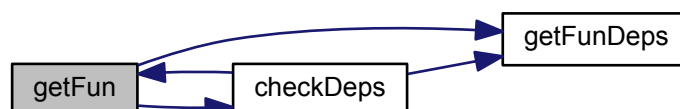
<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation
<i>funstr</i>	function for which symbolic expressions should be computed

Return values

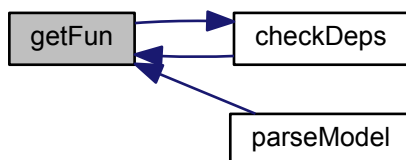
<i>this</i>	updated model definition object
-------------	---------------------------------

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.9 mlhsInnerSubst<::cell> getFunDeps (::string *funstr*)

Parameters

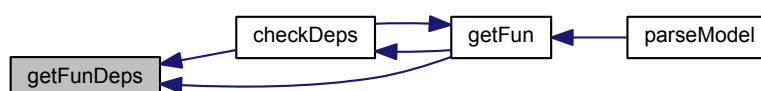
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

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:



6.1.3.10 mlhsInnerSubst< matlabtypesubstitute > getFunArgs (::string *funstr*)

Parameters

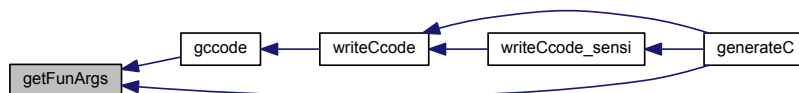
<i>funstr</i>	function name
---------------	---------------

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:



6.1.3.11 mlhsInnerSubst<::string> getFunCVar (::string funstr)

Parameters

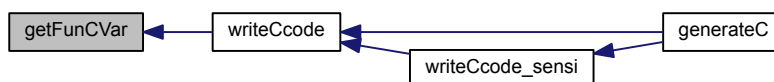
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

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:



6.1.3.12 mlhsInnerSubst< matlabtypesubstitute > getFunSVar (::string funstr)

Parameters

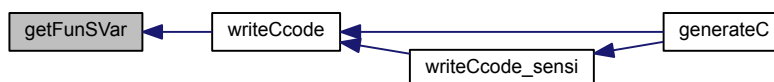
<i>funstr</i>	function or expression for which the dependencies should be returned
---------------	--

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:



6.1.3.13 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

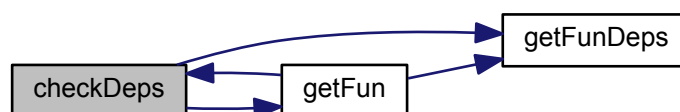
Return values

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

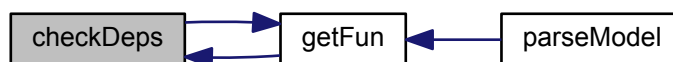
changed with respect to the hashes stored in HTable

Definition at line 18 of file checkDeps.m.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.14 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:



6.1.3.15 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.

6.1.4 Member Data Documentation

6.1.4.1 atol = 1e-8

Default: 1e-8

Definition at line 61 of file amimodel.m.

6.1.4.2 rtol = 1e-8

Default: 1e-8

Definition at line 69 of file amimodel.m.

6.1.4.3 maxsteps = 1e4

Default: 1e4

Definition at line 77 of file amimodel.m.

6.1.4.4 debug = false

Default: false

Definition at line 85 of file amimodel.m.

6.1.4.5 adjoint = true

Default: true

Definition at line 93 of file amimodel.m.

6.1.4.6 forward = true

Default: true

Definition at line 101 of file amimodel.m.

6.1.4.7 t0 = 0

Default: 0

Definition at line 109 of file amimodel.m.

6.1.4.8 nxtrue = 0

Default: 0

Definition at line 131 of file amimodel.m.

6.1.4.9 nytrue = 0

Default: 0

Definition at line 146 of file amimodel.m.

6.1.4.10 coptim = "-O3"

Default: "-O3"

Definition at line 259 of file amimodel.m.

6.1.4.11 param = "lin"

Default: "lin"

Definition at line 267 of file amimodel.m.

6.2 ExpData Struct Reference

struct that carries all information about experimental data

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

Public Attributes

- double * [am_my](#)
- double * [am_ysigma](#)
- double * [am_mt](#)
- double * [am_tsigma](#)

6.2.1 Detailed Description

Definition at line 18 of file edata.h.

6.2.2 Member Data Documentation

6.2.2.1 double* am_my

observed data

Definition at line 20 of file edata.h.

6.2.2.2 double* am_ysigma

standard deviation of observed data

Definition at line 22 of file edata.h.

6.2.2.3 double* am_mt

observed events

Definition at line 25 of file edata.h.

6.2.2.4 double* am_tsigma

standard deviation of observed events

Definition at line 27 of file edata.h.

6.3 ReturnData Struct Reference

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

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

Public Attributes

- double * [am_tsdata](#)
- double * [am_xdotdata](#)
- double * [am_dxdotdpdata](#)
- double * [am_dydxdata](#)
- double * [am_dydpdata](#)
- double * [am_Jdata](#)
- double * [am_rootdata](#)
- double * [am_rootSdata](#)
- double * [am_rootS2data](#)
- double * [am_rootvaldata](#)
- double * [am_rootvalSdata](#)
- double * [am_rootvalS2data](#)
- double * [am_xdata](#)
- double * [am_xSdata](#)
- double * [am_ydata](#)
- double * [am_ySdata](#)
- double * [am_numstepsdata](#)
- double * [am_numstepsSdata](#)
- double * [am_numrhsevalsdata](#)
- double * [am_numrhsevalsSdata](#)
- double * [am_orderdata](#)
- double * [am_llhdata](#)
- double * [am_chi2data](#)
- double * [am_llhSdata](#)
- double * [am_llhS2data](#)

6.3.1 Detailed Description

Definition at line 42 of file rdata.h.

6.3.2 Member Data Documentation

6.3.2.1 double* am_tsdata

timepoints

Definition at line 45 of file rdata.h.

6.3.2.2 double* am_xdotdata

time derivative

Definition at line 47 of file rdata.h.

6.3.2.3 double* am_dxdotpdata

parameter derivative of time derivative

Definition at line 49 of file rdata.h.

6.3.2.4 double* am_dydxdata

state derivative of observables

Definition at line 51 of file rdata.h.

6.3.2.5 double* am_dydpdata

parameter derivative of observables

Definition at line 53 of file rdata.h.

6.3.2.6 double* am_Jdata

Jacobian of differential equation right hand side

Definition at line 55 of file rdata.h.

6.3.2.7 double* am_rootdata

events

Definition at line 57 of file rdata.h.

6.3.2.8 double* am_rootSdata

parameter derivative of events

Definition at line 59 of file rdata.h.

6.3.2.9 double* am_rootS2data

second order parameter derivative of events

Definition at line 61 of file rdata.h.

6.3.2.10 double* am_rootvaldata

value of event function

Definition at line 63 of file rdata.h.

6.3.2.11 double* am_rootvalSdata

parameter derivative of event function

Definition at line 65 of file rdata.h.

6.3.2.12 double* am_rootvalS2data

second order parameter derivative of event function

Definition at line 67 of file rdata.h.

6.3.2.13 double* am_xdata

returned vector state

Definition at line 69 of file rdata.h.

6.3.2.14 double* am_xSdata

parameter derivative of state

Definition at line 71 of file rdata.h.

6.3.2.15 double* am_ydata

observable

Definition at line 73 of file rdata.h.

6.3.2.16 double* am_ySdata

parameter derivative of observable

Definition at line 75 of file rdata.h.

6.3.2.17 double* am_numstepsdata

number of integration steps forward problem

Definition at line 78 of file rdata.h.

6.3.2.18 double* am_numstepsSdata

number of integration steps backward problem

Definition at line 80 of file rdata.h.

6.3.2.19 double* am_numrhsevalsdata

number of right hand side evaluations forward problem

Definition at line 82 of file rdata.h.

6.3.2.20 double* am_numrhsevalsSdata

number of right hand side evaluations backward problem

Definition at line 84 of file rdata.h.

6.3.2.21 double* am_orderdata

employed order forward problem

Definition at line 86 of file rdata.h.

6.3.2.22 double* am_llhdata

likelihood value

Definition at line 89 of file rdata.h.

6.3.2.23 double* am_chi2data

chi2 value

Definition at line 91 of file rdata.h.

6.3.2.24 double* am_llhSdata

parameter derivative of likelihood

Definition at line 93 of file rdata.h.

6.3.2.25 double* am_llhS2data

second order parameter derivative of likelihood

Definition at line 95 of file rdata.h.

6.4 TempData Struct Reference

struct that provides temporary storage for different variables

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

Public Attributes

- reatype [am_t](#)
- N_Vector [am_x](#)
- N_Vector [am_dx](#)
- N_Vector [am_xdot](#)
- N_Vector [am_xB](#)
- N_Vector [am_dxB](#)
- N_Vector [am_xQB](#)
- N_Vector * [am_sx](#)
- N_Vector * [am_sdx](#)
- N_Vector [am_id](#)
- DlsMat [am_Jtmp](#)
- double * [am_llhS0](#)
- double [am_g](#)
- double * [am_dgdp](#)
- double * [am_dgdx](#)
- double [am_r](#)
- double * [am_drdp](#)
- double * [am_drdx](#)
- double [am_rval](#)
- double * [am_drvaldp](#)
- double * [am_drvaldx](#)
- double * [am_dtdx](#)
- double * [am_dtdp](#)
- double * [am_dydp](#)
- double * [am_dydx](#)
- double * [am_yS0](#)
- double * [am_sigma_y](#)
- double * [am_dsigma_ydp](#)
- double * [am_sigma_t](#)
- double * [am_dsigma_tdp](#)
- double * [am_x_tmp](#)
- double * [am_sx_tmp](#)
- double * [am_dx_tmp](#)
- double * [am_sdx_tmp](#)
- double * [am_xdot_tmp](#)
- double * [am_xB_tmp](#)
- double * [am_xQB_tmp](#)
- double * [am_dxB_tmp](#)
- double * [am_id_tmp](#)
- int * [am_rootsfound](#)
- double * [am_rootvaltmp](#)

- int * [am_rootidx](#)
- int [am_which](#)
- double * [am_discs](#)
- double * [am_irdiscs](#)

6.4.1 Detailed Description

Definition at line 64 of file tdata.h.

6.4.2 Member Data Documentation

6.4.2.1 realtype am_t

current time

Definition at line 66 of file tdata.h.

6.4.2.2 N_Vector am_x

state vector

Definition at line 69 of file tdata.h.

6.4.2.3 N_Vector am_dx

differential state vector

Definition at line 71 of file tdata.h.

6.4.2.4 N_Vector am_xdot

time derivative state vector

Definition at line 73 of file tdata.h.

6.4.2.5 N_Vector am_xB

adjoint state vector

Definition at line 75 of file tdata.h.

6.4.2.6 N_Vector am_dxB

differential adjoint state vector

Definition at line 77 of file tdata.h.

6.4.2.7 N_Vector am_xQB

quadrature state vector

Definition at line 79 of file tdata.h.

6.4.2.8 N_Vector* am_sx

sensitivity state vector array

Definition at line 81 of file tdata.h.

6.4.2.9 N_Vector* am_sdx

differential sensitivity state vector array

Definition at line 83 of file tdata.h.

6.4.2.10 N_Vector am_id

index indicating DAE equations vector

Definition at line 85 of file tdata.h.

6.4.2.11 DlsMat am_Jtmp

Jacobian

Definition at line 87 of file tdata.h.

6.4.2.12 double* am_llhS0

parameter derivative of likelihood array

Definition at line 90 of file tdata.h.

6.4.2.13 double am_g

data likelihood

Definition at line 92 of file tdata.h.

6.4.2.14 double* am_dgdp

parameter derivative of data likelihood

Definition at line 94 of file tdata.h.

6.4.2.15 double* am_dgdx

state derivative of data likelihood

Definition at line 96 of file tdata.h.

6.4.2.16 double am_r

event likelihood

Definition at line 98 of file tdata.h.

6.4.2.17 double* am_drdp

parameter derivative of event likelihood

Definition at line 100 of file tdata.h.

6.4.2.18 double* am_drdx

state derivative of event likelihood

Definition at line 102 of file tdata.h.

6.4.2.19 double am_rval

root function likelihood

Definition at line 104 of file tdata.h.

6.4.2.20 double* am_drvaldp

parameter derivative of root function likelihood

Definition at line 106 of file tdata.h.

6.4.2.21 double* am_drvaldx

state derivative of root function likelihood

Definition at line 108 of file tdata.h.

6.4.2.22 double* am_dtdx

state derivative of event

Definition at line 110 of file tdata.h.

6.4.2.23 double* am_dtdp

parameter derivative of event

Definition at line 112 of file tdata.h.

6.4.2.24 double* am_dydp

parameter derivative of observable

Definition at line 114 of file tdata.h.

6.4.2.25 double* am_dydx

state derivative of observable

Definition at line 116 of file tdata.h.

6.4.2.26 double* am_yS0

initial sensitivity of observable

Definition at line 118 of file tdata.h.

6.4.2.27 double* am_sigma_y

data standard deviation

Definition at line 120 of file tdata.h.

6.4.2.28 double* am_dsigma_ydp

parameter derivative of data standard deviation

Definition at line 122 of file tdata.h.

6.4.2.29 double* am_sigma_t

event standard deviation

Definition at line 124 of file tdata.h.

6.4.2.30 double* am_dsigma_tdp

parameter derivative of event standard deviation

Definition at line 126 of file tdata.h.

6.4.2.31 double* am_x_tmp

state array

Definition at line 129 of file tdata.h.

6.4.2.32 double* am_sx_tmp

sensitivity state array

Definition at line 131 of file tdata.h.

6.4.2.33 double* am_dx_tmp

differential state array

Definition at line 133 of file tdata.h.

6.4.2.34 double* am_sdx_tmp

differential sensitivity state array

Definition at line 135 of file tdata.h.

6.4.2.35 double* am_xdot_tmp

time derivative state array

Definition at line 137 of file tdata.h.

6.4.2.36 double* am_xB_tmp

differential adjoint state array

Definition at line 139 of file tdata.h.

6.4.2.37 double* am_xQB_tmp

quadrature state array

Definition at line 141 of file tdata.h.

6.4.2.38 double* am_dxB_tmp

differential adjoint state array

Definition at line 143 of file tdata.h.

6.4.2.39 double* am_id_tmp

index indicating DAE equations array

Definition at line 145 of file tdata.h.

6.4.2.40 int* am_rootsfound

array of flags indicating which root has been found

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 = 1 if not.

Definition at line 151 of file tdata.h.

6.4.2.41 double* am_rootvaltmp

array of values of the root function

Definition at line 153 of file tdata.h.

6.4.2.42 int* am_rootidx

array of index which root has been found

Definition at line 155 of file tdata.h.

6.4.2.43 int am_which

integer for indexing of backwards problems

Definition at line 159 of file tdata.h.

6.4.2.44 double* am_discs

array containing the time-points of discontinuities

Definition at line 162 of file tdata.h.

6.4.2.45 double* am_irdiscs

array containing the index of discontinuities

Definition at line 164 of file tdata.h.

6.5 UserData Struct Reference

struct that stores all user provided data

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

Public Attributes

- int * [am_plist](#)
- int [am_np](#)
- int [am_ny](#)
- int [am_nx](#)
- int [am_nr](#)
- int [am_nt](#)
- int [am_ndisc](#)
- int [am_nnz](#)
- int [am_nmaxroot](#)
- int [am_nmaxdisc](#)
- double * [am_p](#)
- double * [am_k](#)
- double [am_tstart](#)
- double * [am_ts](#)
- double * [am_pbar](#)
- double * [am_xbar](#)
- double * [am_idlist](#)
- int [am_sensi](#)
- double [am_atol](#)
- double [am_rtol](#)
- int [am_maxsteps](#)
- int [am_ism](#)
- int [am_sensi_meth](#)
- int [am_linsol](#)
- int [am_interpType](#)
- int [am_lmm](#)
- int [am_iter](#)
- booleanType [am_stldet](#)
- int [am_ubw](#)
- int [am_lbw](#)

- boolean type [am_bsx0](#)
- double * [am_sx0data](#)
- int [am_event_model](#)
- int [am_data_model](#)
- int [am_ordering](#)

6.5.1 Detailed Description

Definition at line 61 of file udata.h.

6.5.2 Member Data Documentation

6.5.2.1 int* am_plist

parameter reordering

Definition at line 64 of file udata.h.

6.5.2.2 int am_np

number of parameters

Definition at line 66 of file udata.h.

6.5.2.3 int am_ny

number of observables

Definition at line 68 of file udata.h.

6.5.2.4 int am_nx

number of states

Definition at line 70 of file udata.h.

6.5.2.5 int am_nr

number of roots

Definition at line 72 of file udata.h.

6.5.2.6 int am_nt

number of timepoints

Definition at line 74 of file udata.h.

6.5.2.7 int am_ndisc

number of discontinuities

Definition at line 76 of file udata.h.

6.5.2.8 int am_nnz

number of nonzero entries in jacobian

Definition at line 78 of file udata.h.

6.5.2.9 int am_nmaxroot

maximal number of roots to collect

Definition at line 80 of file udata.h.

6.5.2.10 int am_nmaxdisc

maximal number of discontinuities to track

Definition at line 82 of file udata.h.

6.5.2.11 double* am_p

parameter array

Definition at line 85 of file udata.h.

6.5.2.12 double* am_k

constants array

Definition at line 87 of file udata.h.

6.5.2.13 double am_tstart

starting time

Definition at line 90 of file udata.h.

6.5.2.14 double* am_ts

timepoints

Definition at line 92 of file udata.h.

6.5.2.15 double* am_pbar

scaling of parameters

Definition at line 95 of file udata.h.

6.5.2.16 double* am_xbar

scaling of states

Definition at line 97 of file udata.h.

6.5.2.17 double* am_idlist

flag array for DAE equations

Definition at line 100 of file udata.h.

6.5.2.18 int am_sensi

flag indicating whether sensitivities are supposed to be computed

Definition at line 103 of file udata.h.

6.5.2.19 double am_atol

absolute tolerances for integration

Definition at line 105 of file udata.h.

6.5.2.20 double am_rtol

relative tolerances for integration

Definition at line 107 of file udata.h.

6.5.2.21 int am_maxsteps

maximum number of allowed integration steps

Definition at line 109 of file udata.h.

6.5.2.22 int am_ism

internal sensitivity method

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.

6.5.2.23 int am_sensi_meth

method for sensitivity computation

CW_FSA for forward sensitivity analysis, CW_ASA for adjoint sensitivity analysis

Definition at line 121 of file udata.h.

6.5.2.24 int am_linsol

linear solver specification

Definition at line 123 of file udata.h.

6.5.2.25 int am_interpType

interpolation type

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.

6.5.2.26 int am_lmm

linear multistep method

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.

6.5.2.27 int am_iter

nonlinear solver

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

Definition at line 140 of file udata.h.

6.5.2.28 booleantype am_stldet

flag controlling stability limit detection

Definition at line 143 of file udata.h.

6.5.2.29 int am_ubw

upper bandwidth of the jacobian

Definition at line 146 of file udata.h.

6.5.2.30 int am_lbw

lower bandwidth of the jacobian

Definition at line 148 of file udata.h.

6.5.2.31 boolean_t am_bsx0

flag for sensitivity initialisation

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

Definition at line 154 of file udata.h.

6.5.2.32 double* am_sx0data

sensitivity initialisation

Definition at line 157 of file udata.h.

6.5.2.33 int am_event_model

error model for events

Definition at line 160 of file udata.h.

6.5.2.34 int am_data_model

error model for udata

Definition at line 162 of file udata.h.

6.5.2.35 int am_ordering

state ordering

Definition at line 165 of file udata.h.

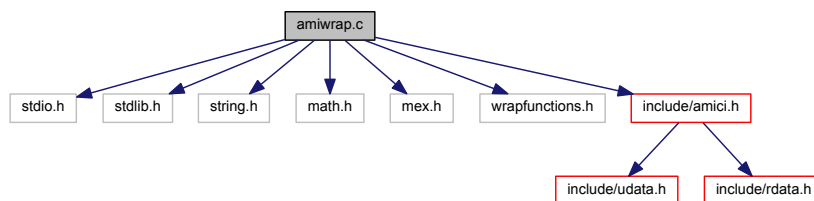
7 File Documentation

7.1 amiwrap.c File Reference

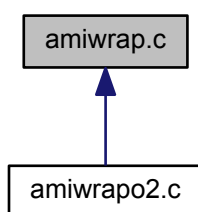
core routines for mex interface

```
#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*[])

7.1.1 Detailed Description

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

7.1.2 Function Documentation

7.1.2.1 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 Type: int
out	<i>plhs</i>	pointer to the array of output arguments Type: mxArray

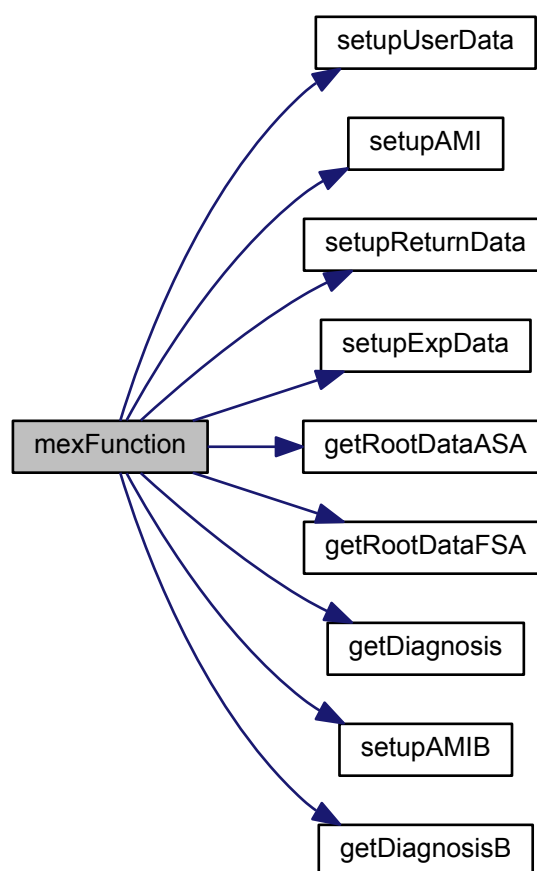
in	<i>nrhs</i>	number of input arguments of the matlab call Type: int
in	<i>prhs</i>	pointer to the array of input arguments Type: mxArray

Returns

void

Definition at line 25 of file amiwrap.c.

Here is the call graph for this function:

**7.2 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.

7.2.1 Function Documentation

7.2.1.1 noret::substitute amiwrap (matlabtypesubstitute *varargin*)

Parameters

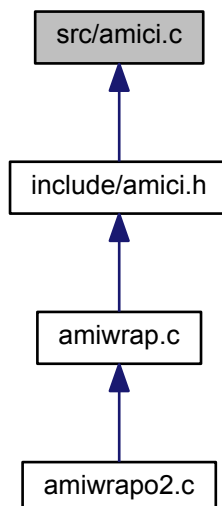
<i>varargin</i>	<pre>1 amiwrap (modelname, symfun, tdir, o2flag)</pre> <p><i>Required Parameters for varargin:</i></p> <ul style="list-style-type: none"> • modelname specifies the name of the model which will be later used for the naming of the simulation file • symfun specifies a function which executes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

Definition at line 17 of file amiwrap.m.

7.3 src/amici.c File Reference

core routines for integration

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



Macros

- #define [amici_c](#)
- #define [AMI_SUCCESS](#) 0

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)

7.3.1 Macro Definition Documentation

7.3.1.1 #define amici_c

include guard

Definition at line 8 of file amici.c.

7.3.1.2 #define AMI_SUCCESS 0

return value indicating successful execution

Definition at line 10 of file amici.c.

7.3.2 Function Documentation

7.3.2.1 [UserData](#) `setupUserData` (const mxArray * *prhs* [])

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

Parameters

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

Returns

udata: struct containing all provided user data

Definition at line 12 of file amici.c.

Here is the caller graph for this function:



7.3.2.2 void* setupAMI (int * *status*, void * *user_data*, void * *temp_data*)

setupAMIs initialises the ami memory object

Parameters

out	<i>status</i>	flag indicating success of execution Type: *int
in	<i>user_data</i>	pointer to the user data struct Type: UserData
in	<i>temp_data</i>	pointer to the temporary data struct Type: TempData

Returns

void

Definition at line 151 of file amici.c.

Here is the caller graph for this function:



7.3.2.3 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 Type: *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 Type: UserData
in	<i>temp_data</i>	pointer to the temporary data struct Type: TempData

Returns

void

Definition at line 433 of file amici.c.

Here is the caller graph for this function:



7.3.2.4 ReturnData setupReturnData (const mxArray * *prhs*[], void * *user_data*)

setupReturnData initialises the return data struct

Parameters

in	<i>prhs</i>	user input Type: *mxArray
in	<i>user_data</i>	pointer to the user data struct Type: UserData

Returns

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

user udata

Definition at line 624 of file amici.c.

Here is the caller graph for this function:



7.3.2.5 ExpData setupExpData (const mxArray * *prhs*[], void * *user_data*)

setupExpData initialises the experimental data struct

Parameters

in	<i>prhs</i>	user input Type: *mxArray
in	<i>user_data</i>	pointer to the user data struct Type: UserData

Returns

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

user udata

Definition at line 689 of file amici.c.

Here is the caller graph for this function:



7.3.2.6 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 Type: int
out	<i>nroots</i>	counter for the number of found roots Type: int
in	<i>ami_mem</i>	pointer to the solver memory block Type: void
in	<i>user_data</i>	pointer to the user data struct Type: UserData
out	<i>return_data</i>	pointer to the return data struct Type: ReturnData
out	<i>temp_data</i>	pointer to the temporary data struct Type: TempData

Returns

void

Definition at line 788 of file amici.c.

Here is the caller graph for this function:



7.3.2.7 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 Type: *int
out	<i>nroots</i>	counter for the number of found roots Type: *int
out	<i>idisc</i>	counter for the number of found discontinuities Type: *int
in	<i>ami_mem</i>	pointer to the solver memory block Type: *void
in	<i>user_data</i>	pointer to the user data struct Type: UserData
out	<i>return_data</i>	pointer to the return data struct Type: ReturnData
in	<i>exp_data</i>	pointer to the experimental data struct Type: ExpData

out	<i>temp_data</i>	pointer to the temporary data struct Type: TempData
-----	------------------	---

Returns

void

Definition at line 872 of file amici.c.

Here is the caller graph for this function:

**7.3.2.8 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 Type: *int
in	<i>it</i>	time-point index Type: int
in	<i>ami_mem</i>	pointer to the solver memory block Type: *void
in	<i>user_data</i>	pointer to the user data struct Type: UserData
out	<i>return_data</i>	pointer to the return data struct Type: ReturnData

Returns

void

Definition at line 980 of file amici.c.

Here is the caller graph for this function:

**7.3.2.9 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 Type: *int
in	<i>it</i>	time-point index Type: int
in	<i>ami_mem</i>	pointer to the solver memory block Type: *void
in	<i>user_data</i>	pointer to the user data struct Type: UserData
out	<i>return_data</i>	pointer to the return data struct Type: ReturnData
out	<i>temp_data</i>	pointer to the temporary data struct Type: TempData

Returns

void

Definition at line 1014 of file amici.c.

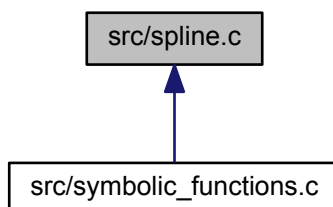
Here is the caller graph for this function:



7.4 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[])

7.4.1 Detailed Description

Author

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

7.4.2 Function Documentation

7.4.2.1 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.

7.4.2.2 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 spline() .
in	<i>c</i>	array of spline coefficients computed by spline() .
in	<i>d</i>	array of spline coefficients computed by spline() .

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.

7.4.2.3 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 spline()
in	<i>c</i>	array of spline coefficients computed by spline()
in	<i>d</i>	array of spline coefficients computed by spline()

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.

7.4.2.4 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 spline() .
in	c	array of spline coefficients computed by spline() .
in	d	array of spline coefficients computed by spline() .

Returns

the value of the spline function at u

Notes

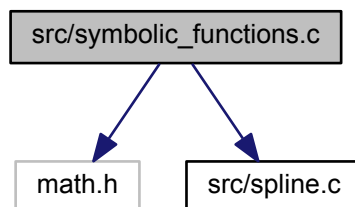
- If u is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

Definition at line 324 of file spline.c.

7.5 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`
- `#define TRUE 1`
- `#define FALSE 0`

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)

7.5.1 Detailed Description

This file contains definitions of various symbolic functions which

7.5.2 Macro Definition Documentation

7.5.2.1 `#define amici_symbolic_functions_c`

include guard

Definition at line 10 of file symbolic_functions.c.

7.5.2.2 `#define TRUE 1`

bool return value true

Definition at line 16 of file symbolic_functions.c.

7.5.2.3 `#define FALSE 0`

bool return value false

Definition at line 18 of file symbolic_functions.c.

7.5.3 Function Documentation

7.5.3.1 `int am_ge (double a, double b)`

c implementation of matlab function ge

Parameters

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

Returns

if(*a* >= 0) then 1 else 0

Type: int

Definition at line 28 of file symbolic_functions.c.

7.5.3.2 `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 Type: double
<i>b</i>	value2 Type: double

Returns

0

Definition at line 45 of file symbolic_functions.c.

7.5.3.3 `int am_gt (double a, double b)`

c implementation of matlab function gt

Parameters

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

Returns

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

Type: int

Definition at line 57 of file symbolic_functions.c.

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

Returns

0

Definition at line 74 of file symbolic_functions.c.

7.5.3.5 int am_le (double *a*, double *b*)

c implementation of matlab function le

Parameters

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

Returns

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

Type: int

Definition at line 86 of file symbolic_functions.c.

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

Returns

0

Definition at line 103 of file symbolic_functions.c.

7.5.3.7 int am_lt (double *a*, double *b*)

c implementation of matlab function lt

Parameters

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

Returnsif($a < 0$) then 1 else 0**Type:** int

Definition at line 115 of file symbolic_functions.c.

7.5.3.8 int Dam_lt (int *id*, double *a*, double *b*)

parameter derivative of c implementation of matlab function lt

Parameters

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

Returns

0

Definition at line 132 of file symbolic_functions.c.

7.5.3.9 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 Type: double

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

Returns

if(condition) then truepart else falsepart

Type: int

Definition at line 145 of file symbolic_functions.c.

7.5.3.10 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 Type: double
<i>falsepart</i>	returnvalue if condition is false Type: 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 166 of file symbolic_functions.c.

7.5.3.11 int am_and (int *a*, int *b*)

c implementation of matlab function and

Parameters

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

Returns

if(a & b) then 1 else 0

Type: int

Definition at line 200 of file symbolic_functions.c.

7.5.3.12 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 Type: int
<i>b</i>	bool2 Type: int

Returns

0
Type: double

Definition at line 217 of file symbolic_functions.c.

7.5.3.13 int am_or (int *a*, int *b*)

c implementation of matlab function or

Parameters

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

Returns

if(*a* | *b*) then 1 else 0
Type: int

Definition at line 229 of file symbolic_functions.c.

7.5.3.14 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 Type: int
<i>b</i>	bool2 Type: int

Returns

0
Type: double

Definition at line 246 of file symbolic_functions.c.

7.5.3.15 double am_min (double *a*, double *b*)

c implementation of matlab function min

Parameters

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

Returns

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

Definition at line 258 of file symbolic_functions.c.

7.5.3.16 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 Type: double
<i>b</i>	bool2 Type: 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 276 of file symbolic_functions.c.

7.5.3.17 double am_max (double *a*, double *b*)

c implementation of matlab function min

Parameters

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

Returns

if($a > b$) then a else b
Type: double

Definition at line 300 of file symbolic_functions.c.

7.5.3.18 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 Type: double
<i>b</i>	bool2 Type: 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 318 of file symbolic_functions.c.

7.5.3.19 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 341 of file symbolic_functions.c.

7.5.3.20 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 356 of file symbolic_functions.c.

7.5.3.21 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 372 of file symbolic_functions.c.

7.5.3.22 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

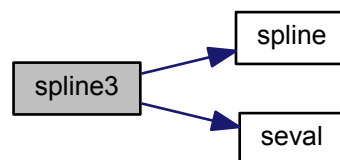
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should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

Returns

spline(*t*)

Definition at line 396 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.23 `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

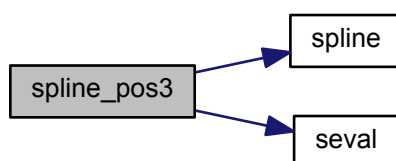
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag indicating whether slope at first node should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

Returns

spline(t)

Definition at line 437 of file symbolic_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.24 `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

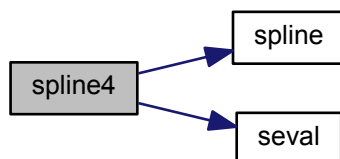
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 485 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.25 `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

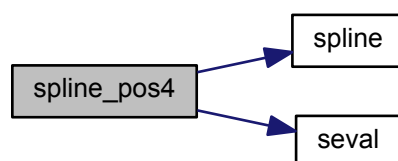
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 529 of file `symbolic_functions.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.26 `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

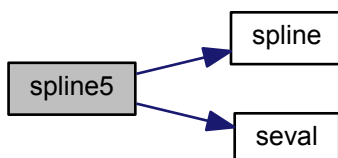
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 581 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.27 `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

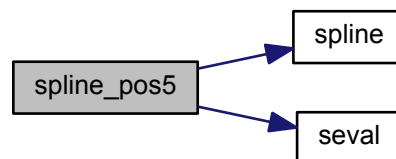
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 629 of file symbolic_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.28 `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

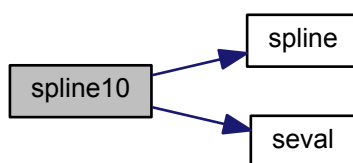
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 693 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.29 `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

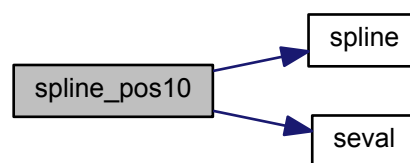
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 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 761 of file symbolic_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.30 `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

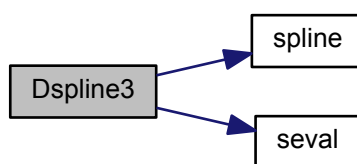
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag 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 822 of file `symbolic_functions.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.31 `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

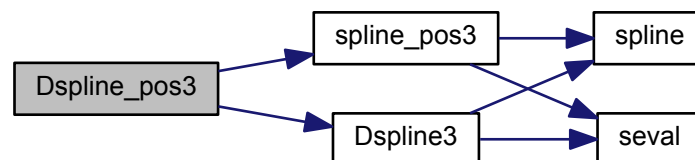
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>ss</i>	flag 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 867 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.32 `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

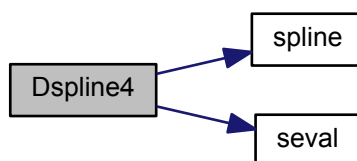
<i>dudt</i>	user defined slope at first node
-------------	----------------------------------

Returns

dspline(t)dp(id)

Definition at line 910 of file symbolic_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.33 `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

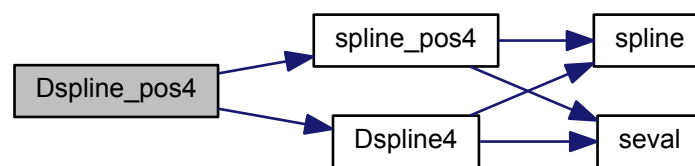
<i>p4</i>	spline value at node 4
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

Returns

dspline(t)dp(id)

Definition at line 959 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.34 `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

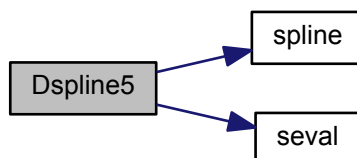
<i>id</i>	argument index for differentiation
<i>t</i>	point at which the spline should be evaluated
<i>t1</i>	location of node 1
<i>p1</i>	spline value at node 1
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location 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 1004 of file `symbolic_functions.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.35 `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

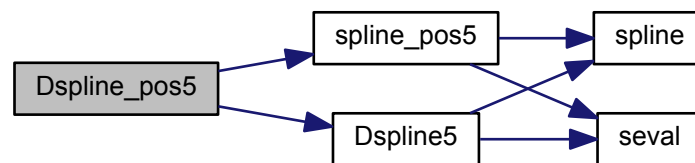
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

Returns

dspline(t)dp(id)

Definition at line 1057 of file symbolic_functions.c.

Here is the call graph for this function:



7.5.3.36 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

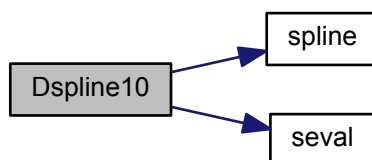
<i>t9</i>	location of node 9
<i>p9</i>	spline value at node 9
<i>t10</i>	location of node 10
<i>p10</i>	spline value at node 10
<i>ss</i>	flag indicating whether slope at first node should be user defined
<i>dudt</i>	user defined slope at first node

Returns

dspline(t)dp(id)

Definition at line 1113 of file symbolic_functions.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.37 `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

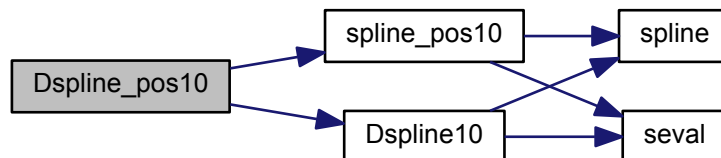
<i>t2</i>	location of node 2
<i>p2</i>	spline value at node 2
<i>t3</i>	location of node 3
<i>p3</i>	spline value at node 3
<i>t4</i>	location of node 4
<i>p4</i>	spline value at node 4
<i>t5</i>	location of node 5
<i>p5</i>	spline value at node 5
<i>t6</i>	location of node 6
<i>p6</i>	spline value at 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 1186 of file symbolic_functions.c.

Here is the call graph for this function:



Index

AMI_SUCCESS
 amici.c, 63
adjoint
 amimodel, 45
am_Jdata
 ReturnData, 48
am_Jtmp
 TempData, 52
am_and
 symbolic_functions.c, 79
am_atol
 UserData, 57
am_bsx0
 UserData, 59
am_chi2data
 ReturnData, 49
am_data_model
 UserData, 59
am_dgdp
 TempData, 52
am_dgdx
 TempData, 52
am_discs
 TempData, 55
am_drdp
 TempData, 52
am_drdx
 TempData, 52
am_drvaldp
 TempData, 52
am_drvaldx
 TempData, 52
am_dsigma_tdp
 TempData, 53
am_dsigma_ydp
 TempData, 53
am_dtdp
 TempData, 53
am_dtdx
 TempData, 53
am_dx
 TempData, 51
am_dx_tmp
 TempData, 54
am_dxB
 TempData, 51
am_dxB_tmp
 TempData, 54
am_dxdotdpdata
 ReturnData, 47
am_dydp
 TempData, 53
am_dydpdata
 ReturnData, 48
am_dydx
 TempData, 53
am_dydxdata
 ReturnData, 48
am_event_model
 UserData, 59
am_g
 TempData, 52
am_ge
 symbolic_functions.c, 76
am_gt
 symbolic_functions.c, 76
am_id
 TempData, 51
am_id_tmp
 TempData, 54
am_idlist
 UserData, 57
am_if
 symbolic_functions.c, 78
am_interpType
 UserData, 58
am_irdiscs
 TempData, 55
am_ism
 UserData, 58
am_iter
 UserData, 58
am_k
 UserData, 57
am_lbw
 UserData, 58
am_le
 symbolic_functions.c, 77
am_linsol
 UserData, 58
am_llhS0
 TempData, 52
am_llhS2data
 ReturnData, 49
am_llhSdata
 ReturnData, 49
am_llhdata
 ReturnData, 49
am_lmm
 UserData, 58
am_lt
 symbolic_functions.c, 78
am_max
 symbolic_functions.c, 81
am_maxsteps
 UserData, 57
am_min
 symbolic_functions.c, 80
am_mt
 ExpData, 46

- am_my
 - ExpData, [46](#)
- am_ndisc
 - UserData, [56](#)
- am_nmaxdisc
 - UserData, [57](#)
- am_nmaxroot
 - UserData, [56](#)
- am_nnz
 - UserData, [56](#)
- am_np
 - UserData, [56](#)
- am_nr
 - UserData, [56](#)
- am_nt
 - UserData, [56](#)
- am_numrhsevalsSdata
 - ReturnData, [49](#)
- am_numrhsevalsddata
 - ReturnData, [49](#)
- am_numstepsSdata
 - ReturnData, [49](#)
- am_numstepsdata
 - ReturnData, [49](#)
- am_nx
 - UserData, [56](#)
- am_ny
 - UserData, [56](#)
- am_or
 - symbolic_functions.c, [80](#)
- am_orderdata
 - ReturnData, [49](#)
- am_ordering
 - UserData, [59](#)
- am_p
 - UserData, [57](#)
- am_pbar
 - UserData, [57](#)
- am_plist
 - UserData, [56](#)
- am_r
 - TempData, [52](#)
- am_rootS2data
 - ReturnData, [48](#)
- am_rootSdata
 - ReturnData, [48](#)
- am_rootdata
 - ReturnData, [48](#)
- am_rootidx
 - TempData, [54](#)
- am_rootsfound
 - TempData, [54](#)
- am_rootvalS2data
 - ReturnData, [48](#)
- am_rootvalSdata
 - ReturnData, [48](#)
- am_rootvaldata
 - ReturnData, [48](#)
- am_rootvaltmp
 - TempData, [54](#)
- am_rtol
 - UserData, [57](#)
- am_rval
 - TempData, [52](#)
- am_sdx
 - TempData, [51](#)
- am_sdx_tmp
 - TempData, [54](#)
- am_sensi
 - UserData, [57](#)
- am_sensi_meth
 - UserData, [58](#)
- am_sigma_t
 - TempData, [53](#)
- am_sigma_y
 - TempData, [53](#)
- am_stldet
 - UserData, [58](#)
- am_sx
 - TempData, [51](#)
- am_sx0data
 - UserData, [59](#)
- am_sx_tmp
 - TempData, [53](#)
- am_t
 - TempData, [51](#)
- am_ts
 - UserData, [57](#)
- am_tsdata
 - ReturnData, [47](#)
- am_tsigma
 - ExpData, [46](#)
- am_tstart
 - UserData, [57](#)
- am_ubw
 - UserData, [58](#)
- am_which
 - TempData, [55](#)
- am_x
 - TempData, [51](#)
- am_x_tmp
 - TempData, [53](#)
- am_xB
 - TempData, [51](#)
- am_xB_tmp
 - TempData, [54](#)
- am_xQB
 - TempData, [51](#)
- am_xQB_tmp
 - TempData, [54](#)
- am_xSdata
 - ReturnData, [48](#)
- am_xbar
 - UserData, [57](#)
- am_xdata
 - ReturnData, [48](#)

- am_xdot
 - TempData, 51
- am_xdot_tmp
 - TempData, 54
- am_xdotdata
 - ReturnData, 47
- am_yS0
 - TempData, 53
- am_ySdata
 - ReturnData, 49
- am_ydata
 - ReturnData, 49
- am_ysigma
 - ExpData, 46
- amici.c
 - AMI_SUCCESS, 63
 - amici_c, 63
 - getDiagnosis, 70
 - getDiagnosisB, 70
 - getRootDataASA, 69
 - getRootDataFSA, 67
 - setupAMI, 63
 - setupAMIB, 65
 - setupExpData, 67
 - setupReturnData, 65
 - setupUserData, 63
- amici_c
 - amici.c, 63
- amici_symbolic_functions_c
 - symbolic_functions.c, 76
- amimodel, 34
 - adjoint, 45
 - amimodel, 36
 - atol, 45
 - augmento2, 43
 - checkDeps, 42
 - compileC, 37
 - coptim, 46
 - debug, 45
 - forward, 45
 - gccode, 39
 - generateC, 37
 - generateM, 40
 - getFun, 40
 - getFunArgs, 41
 - getFunCVar, 42
 - getFunDeps, 41
 - getFunSVar, 42
 - loadOldHashes, 43
 - maxsteps, 45
 - nxtrue, 45
 - nytrue, 45
 - param, 46
 - parseModel, 36
 - rtol, 45
 - t0, 45
 - writeCcode, 38
 - writeCcode_sensi, 37
- amiwrap
 - amiwrap.m, 62
- amiwrap.c, 59
 - mexFunction, 60
- amiwrap.m, 61
 - amiwrap, 62
- atol
 - amimodel, 45
- augmento2
 - amimodel, 43
- checkDeps
 - amimodel, 42
- compileC
 - amimodel, 37
- coptim
 - amimodel, 46
- Dam_and
 - symbolic_functions.c, 79
- Dam_ge
 - symbolic_functions.c, 76
- Dam_gt
 - symbolic_functions.c, 77
- Dam_if
 - symbolic_functions.c, 79
- Dam_le
 - symbolic_functions.c, 77
- Dam_lt
 - symbolic_functions.c, 78
- Dam_max
 - symbolic_functions.c, 81
- Dam_min
 - symbolic_functions.c, 81
- Dam_or
 - symbolic_functions.c, 80
- Ddirac
 - symbolic_functions.c, 82
- debug
 - amimodel, 45
- deriv
 - spline.c, 73
- dirac
 - symbolic_functions.c, 82
- Dspline10
 - symbolic_functions.c, 95
- Dspline3
 - symbolic_functions.c, 90
- Dspline4
 - symbolic_functions.c, 91
- Dspline5
 - symbolic_functions.c, 93
- Dspline_pos10
 - symbolic_functions.c, 96
- Dspline_pos3
 - symbolic_functions.c, 90
- Dspline_pos4
 - symbolic_functions.c, 92
- Dspline_pos5

- symbolic_functions.c, 94
- ExpData, 46
 - am_mt, 46
 - am_my, 46
 - am_tsigma, 46
 - am_ysigma, 46
- FALSE
 - symbolic_functions.c, 76
- forward
 - amimodel, 45
- gccode
 - amimodel, 39
- generateC
 - amimodel, 37
- generateM
 - amimodel, 40
- getDiagnosis
 - amici.c, 70
- getDiagnosisB
 - amici.c, 70
- getFun
 - amimodel, 40
- getFunArgs
 - amimodel, 41
- getFunCVar
 - amimodel, 42
- getFunDeps
 - amimodel, 41
- getFunSVar
 - amimodel, 42
- getRootDataASA
 - amici.c, 69
- getRootDataFSA
 - amici.c, 67
- heaviside
 - symbolic_functions.c, 82
- loadOldHashes
 - amimodel, 43
- maxsteps
 - amimodel, 45
- mexFunction
 - amiwrap.c, 60
- nxtrue
 - amimodel, 45
- nytrue
 - amimodel, 45
- param
 - amimodel, 46
- parseModel
 - amimodel, 36
- ReturnData, 47
 - am_Jdata, 48
 - am_chi2data, 49
 - am_dxdotdpdata, 47
 - am_dydpdata, 48
 - am_dydxdata, 48
 - am_llhS2data, 49
 - am_llhSdata, 49
 - am_llhdata, 49
 - am_numrhsevalsSdata, 49
 - am_numrhsevalsddata, 49
 - am_numstepsSdata, 49
 - am_numstepsdata, 49
 - am_orderdata, 49
 - am_rootS2data, 48
 - am_rootSdata, 48
 - am_rootdata, 48
 - am_rootvalS2data, 48
 - am_rootvalSdata, 48
 - am_rootvaldata, 48
 - am_tsdata, 47
 - am_xSdata, 48
 - am_xdata, 48
 - am_xdotdata, 47
 - am_ySdata, 49
 - am_ydata, 49
- rtol
 - amimodel, 45
- setupAMI
 - amici.c, 63
- setupAMIB
 - amici.c, 65
- setupExpData
 - amici.c, 67
- setupReturnData
 - amici.c, 65
- setupUserData
 - amici.c, 63
- seval
 - spline.c, 72
- sinteg
 - spline.c, 73
- spline
 - spline.c, 72
- spline.c
 - deriv, 73
 - seval, 72
 - sinteg, 73
 - spline, 72
- spline10
 - symbolic_functions.c, 87
- spline3
 - symbolic_functions.c, 82
- spline4
 - symbolic_functions.c, 84
- spline5
 - symbolic_functions.c, 86
- spline_pos10
 - symbolic_functions.c, 88

- spline_pos3
 - symbolic_functions.c, 83
- spline_pos4
 - symbolic_functions.c, 85
- spline_pos5
 - symbolic_functions.c, 86
- src/amici.c, 62
- src/spline.c, 71
- src/symbolic_functions.c, 74
- symbolic_functions.c
 - am_and, 79
 - am_ge, 76
 - am_gt, 76
 - am_if, 78
 - am_le, 77
 - am_lt, 78
 - am_max, 81
 - am_min, 80
 - am_or, 80
 - amici_symbolic_functions_c, 76
 - Dam_and, 79
 - Dam_ge, 76
 - Dam_gt, 77
 - Dam_if, 79
 - Dam_le, 77
 - Dam_lt, 78
 - Dam_max, 81
 - Dam_min, 81
 - Dam_or, 80
 - Ddirac, 82
 - dirac, 82
 - Dspline10, 95
 - Dspline3, 90
 - Dspline4, 91
 - Dspline5, 93
 - Dspline_pos10, 96
 - Dspline_pos3, 90
 - Dspline_pos4, 92
 - Dspline_pos5, 94
 - FALSE, 76
 - heaviside, 82
 - spline10, 87
 - spline3, 82
 - spline4, 84
 - spline5, 86
 - spline_pos10, 88
 - spline_pos3, 83
 - spline_pos4, 85
 - spline_pos5, 86
 - TRUE, 76
- t0
 - amimodel, 45
- TRUE
 - symbolic_functions.c, 76
- TempData, 50
 - am_Jtmp, 52
 - am_dgdp, 52
 - am_dgdx, 52
 - am_discs, 55
 - am_drdp, 52
 - am_dr dx, 52
 - am_drvaldp, 52
 - am_drvaldx, 52
 - am_dsigma_tdp, 53
 - am_dsigma_ydp, 53
 - am_dtdp, 53
 - am_dtdx, 53
 - am_dx, 51
 - am_dx_tmp, 54
 - am_dxB, 51
 - am_dxB_tmp, 54
 - am_dydp, 53
 - am_dydx, 53
 - am_g, 52
 - am_id, 51
 - am_id_tmp, 54
 - am_irdiscs, 55
 - am_IIhS0, 52
 - am_r, 52
 - am_rootidx, 54
 - am_rootsfound, 54
 - am_rootvaltmp, 54
 - am_rval, 52
 - am_sdx, 51
 - am_sdx_tmp, 54
 - am_sigma_t, 53
 - am_sigma_y, 53
 - am_sx, 51
 - am_sx_tmp, 53
 - am_t, 51
 - am_which, 55
 - am_x, 51
 - am_x_tmp, 53
 - am_xB, 51
 - am_xB_tmp, 54
 - am_xQB, 51
 - am_xQB_tmp, 54
 - am_xdot, 51
 - am_xdot_tmp, 54
 - am_yS0, 53
- UserData, 55
 - am_atol, 57
 - am_bsx0, 59
 - am_data_model, 59
 - am_event_model, 59
 - am_idlist, 57
 - am_interpType, 58
 - am_ism, 58
 - am_iter, 58
 - am_k, 57
 - am_lb, 58
 - am_linsol, 58
 - am_lmm, 58
 - am_maxsteps, 57
 - am_ndisc, 56
 - am_nmaxdisc, 57

- am_nmaxroot, [56](#)
- am_nnz, [56](#)
- am_np, [56](#)
- am_nr, [56](#)
- am_nt, [56](#)
- am_nx, [56](#)
- am_ny, [56](#)
- am_ordering, [59](#)
- am_p, [57](#)
- am_pbar, [57](#)
- am_plist, [56](#)
- am_rtol, [57](#)
- am_sensi, [57](#)
- am_sensi_meth, [58](#)
- am_stldet, [58](#)
- am_sx0data, [59](#)
- am_ts, [57](#)
- am_tstart, [57](#)
- am_ubw, [58](#)
- am_xbar, [57](#)
- writeCcode
 - amimodel, [38](#)
- writeCcode_sensi
 - amimodel, [37](#)