sundialsTB v2.4.0, a matlab Interface to sundials

Radu Serban
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

July 30, 2015

UCRL-SM-212121

DISCLAIMER

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Contents

1 Introduction

SUNDIALS [2], SUite of Nonlinear and DIfferential/ALgebraic equation Solvers, is a family of software tools for integration of ODE and DAE initial value problems and for the solution of nonlinear systems of equations. It consists of CVODE, IDA, and KINSOL, and variants of these with sensitivity analysis capabilities.

SUNDIALSTB is a collection of MATLAB functions which provide interfaces to the SUNDIALS solvers.

The core of each MATLAB interface in SUNDIALSTB is a single MEX file which interfaces to the various user-callable functions for that solver. However, this MEX file should not be called directly, but rather through the user-callable functions provided for each MATLAB interface.

A major design principle for SUNDIALSTB was to provide an interface that is, as much as possible, equally familiar to both SUNDIALS users and MATLAB users. Moreover, we tried to keep the number of user-callable functions to a minimum. For example, the CVODES MATLAB interface contains only 12 such functions, 2 of which relate to forward sensitivity analysis and 4 more interface solely to the adjoint sensitivity module in CVODES. A user who is only interested in integration of ODEs and not in sensitivity analysis therefore needs to call at most 6 functions. In tune with the MATLAB ODESET function, optional solver inputs in SUNDIALSTB are specified through a single function; e.g. `CvodeSetOptions` for CVODES (a similar function is used to specify optional inputs for forward sensitivity analysis). However, unlike the ODE solvers in MATLAB, we have kept the more flexible SUNDIALS model in which a separate "solve" function (`CVode` for CVODES) must be called to return the solution at a desired output time. Solver statistics, as well as optional outputs (such as solution and solution derivatives at additional times) can be obtained at any time with calls to separate functions (`CVodeGetStats` and `CVodeGet` for CVODES).

This document provides a complete documentation for the SUNDIALSTB functions. For additional details on the methods and underlying SUNDIALS software consult also the coresponding SUNDIALS user guides [3, 4, 1].

Requirements. For parallel support, SUNDIALSTB depends on MPITB with LAM v $>$ 7.1.1 (for MPI-2 spawning feature). The required software packages can be obtained from the following addresses.

SUNDIALS	`http://www.llnl.gov/CASC/sundials`
MPITB	`http://atc.ugr.es/javier-bin/mpitb_eng`
LAM	`http://www.lam-mpi.org/`

2 Installation

The following steps are required to install and setup SUNDIALSTB:

2.1 Compilation and installation of sundialsTB

As of version 2.3.0, SUNDIALSTB is distributed only with the complete SUNDIALS package.

In the sequel, we assume that the SUNDIALS package was unpacked under the directory *srcdir*. The SUNDIALSTB files are therefore in *srcdir*/`sundialsTB`.

Compilation and installation of the SUNDIALSTB toolbox is done by running the MATLAB script `install_STB.m` which is present in the SUNDIALSTB top directory.

1. Launch MATLAB in sundialsTB

```
% cd srcdir/sundialsTB
% matlab
```

2. Run the MATLAB script install_STB

   Note that parallel support will be compiled into the MEX files only if $LAMHOME is defined **and** $MPITB_ROOT is defined **and** *srcdir*/src/nvec_par exists.

After the MEX files are generated, you will be asked if you wish to install the SUNDIALSTB toolbox. If you answer yes, you will be then asked for the installation directory (called in the sequel *instdir*). To install SUNDIALSTB for all MATLAB users (not usual), assuming MATLAB is installed under `/usr/local/matlab7`, specify *instdir* = `/usr/local/matlab7/toolbox`. To install SUNDIALSTB for just one user (usual configuration), install SUNDIALSTB under a directory of your choice (typically under your `matlab` working directory). In other words, specify *instdir* = `/home/user/matlab`.

2.2 Configuring Matlab's startup

After a successful installation, a SUNDIALSTB.m startup script is generated in *instdir*/`sundialsTB`. This file must be called by MATLAB at initialization.

If SUNDIALSTB was installed for all MATLAB users (not usual), add the SUNDIALSTB startup to the system-wide startup file (by linking or copying):

```
% cd /usr/local/matlab7/toolbox/local
% ln -s ../sundialsTB/startup_STB.m .
```

and add these lines to your original local startup.m

```
% SUNDIALS Toolbox startup M-file, if it exists.
if exist('startup_STB','file')
   startup_STB
end
```

If SUNDIALSTB was installed for just one user (usual configuration) and assuming you do not need to keep any previously existing startup.m, link or copy the startup_STB.m script to your working 'matlab' directory:

```
% cd ~/matlab
% ln -s sundialsTB/startup_STB.m startup.m
```

If you already have a startup.m, use the method described above, first linking (or copying) `startup_STB.m` to the destination subdirectory and then editing the file `/matlab/startup.m` to run `startup_STB.m`.

2.3 Testing the installation

If everything went fine, you should now be able to try one of the CVODES, IDAS, or KINSOL examples (in MATLAB, type 'help cvodes', 'help idas', or 'help kinsol' to see a list of all examples available). For example, go to the CVODES serial example directory:

```
% cd instdir/sundialsTB/cvode/examples_ser
```

and then launch MATLAB and execute mcvsRoberts_dns.

3 MATLAB Interface to CVODES

The MATLAB interface to CVODES provides access to all functionality of the CVODES solver, including IVP simulation and sensitvity analysis (both forward and adjoint).

The interface consists of several user-callable functions. In addition, the user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions are listed in Tables 1, 2, and 3 for IVP solution, forward sensitivity analysis (FSA), and adjoint sensitivity analysis (ASA), respectively. For completness, some functions appear in more than one table. The types of user-supplied functions are listed in Table 4. All these functions are fully documented later in this section. For more in depth details, consult also the CVODES user guide [3].

To illustrate the use of the CVODES MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with CVODES.

Table 1: CVODES MATLAB interface functions for ODE integration

CVodeSetOptions	create an options structure for an ODE problem.	4
CVodeQuadSetOptions	create an options structure for quadrature integration.	9
CVodeInit	allocate and initialize memory for CVODES.	11
CVodeQuadInit	allocate and initialize memory for quadrature integration.	12
CVodeReInit	reinitialize memory for CVODES.	14
CVodeQuadReInit	reinitialize memory for quadrature integration.	15
CVode	integrate the ODE problem.	17
CVodeGetStats	return statistics for the CVODES solver.	19
CVodeGet	extract data from CVODES memory.	22
CVodeFree	deallocate memory for the CVODES solver.	24
CVodeMonitor	monitoring function.	104

Table 2: CVODES MATLAB interface functions for FSA

CVodeSetOptions	create an options structure for an ODE problem.	4
CVodeQuadSetOptions	create an options structure for quadrature integration.	9
CVodeSensSetOptions	create an options structure for FSA.	10
CVodeInit	allocate and initialize memory for CVODES.	11
CVodeQuadInit	allocate and initialize memory for quadrature integration.	12
CVodeSensInit	allocate and initialize memory for FSA.	12
CVodeReInit	reinitialize memory for CVODES.	14
CVodeQuadReInit	reinitialize memory for quadrature integration.	15
CVodeSensReInit	reinitialize memory for FSA.	15
CVodeSensToggleOff	temporarily deactivates FSA.	19
CVode	integrate the ODE problem.	17
CVodeGetStats	return statistics for the CVODES solver.	19
CVodeGet	extract data from CVODES memory.	22
CVodeFree	deallocate memory for the CVODES solver.	24
CVodeMonitor	monitoring function.	104

Table 3: CVODES MATLAB interface functions for ASA

CVodeSetOptions CVodeQuadSetOptions	create an options structure for an ODE problem. create an options structure for quadrature integration.	4 9
CVodeInit CVodeQuadInit CVodeQuadReInit CVodeReInit	allocate and initialize memory for the forward problem. allocate and initialize memory for forward quadrature integration. reinitialize memory for forward quadrature integration. reinitialize memory for the forward problem.	11 12 15 14
CVodeAdjInit CVodeInitB CVodeAdjReInit CVodeReInitB	allocate and initialize memory for ASA. allocate and initialize a backward problem. reinitialize memory for ASA. reinitialize a backward problem.	13 13 16 16
CVode CVodeB	integrate the forward ODE problem. integrate the backward problems.	17 18
CVodeGetStats CVodeGetStatsB CVodeGet	return statistics for the integration of the forward problem. return statistics for the integration of a backward problem. extract data from CVODES memory.	19 21 22
CVodeFree	deallocate memory for the CVODES solver.	24
CVodeMonitor CVodeMonitorB	monitoring function for forward problem. monitoring function for backward problems.	104 119

3.1 Interface functions

CVodeSetOptions

PURPOSE

```
CVodeSetOptions creates an options structure for CVODES.
```

SYNOPSIS

```
function options = CVodeSetOptions(varargin)
```

DESCRIPTION

```
CVodeSetOptions creates an options structure for CVODES.

   Usage: OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a CVODES options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.

   CVodeSetOptions with no input arguments displays all property names
```

Table 4: CVODES MATLAB function types

Forward problems	CVRhsFn	RHS function	25
	CVRootFn	root-finding function	26
	CVQuadRhsFn	quadrature RHS function	26
	CVSensRhsFn	sensitivity RHS function	25
	CVDenseJacFn	dense Jacobian function	27
	CVBandJacFn	banded Jacobian function	28
	CVJacTimesVecFn	Jacobian times vector function	28
	CVPrecSetupFn	preconditioner setup function	29
	CVPrecSolveFn	preconditioner solve function	30
	CVGlocalFn	RHS approximation function (BBDPre)	32
	CVGcommFn	communication function (BBDPre)	31
	CVMonitorFn	monitoring function	33
Backward problems	CVRhsFnB	RHS function	34
	CVQuadRhsFnB	quadrature RHS function	34
	CVDenseJacFnB	dense Jacobian function	35
	CVBandJacFnB	banded Jacobian function	35
	CVJacTimesVecFnB	Jacobian times vector function	36
	CVPrecSetupFnB	preconditioner setup function	37
	CVPrecSolveFnB	preconditioner solve function	38
	CVGlocalFnB	RHS approximation function (BBDPre)	39
	CVGcommFnB	communication function (BBDPre)	38
	CVMonitorFnB	monitoring function	40

```
   and their possible values.

CVodeSetOptions properties
(See also the CVODES User Guide)

UserData - User data passed unmodified to all functions [ empty ]
   If UserData is not empty, all user provided functions will be
   passed the problem data as their last input argument. For example,
   the RHS function must be defined as YD = ODEFUN(T,Y,DATA).

LMM - Linear Multistep Method [ 'Adams' | 'BDF' ]
   This property specifies whether the Adams method is to be used instead
   of the default Backward Differentiation Formulas (BDF) method.
   The Adams method is recommended for non-stiff problems, while BDF is
   recommended for stiff problems.
NonlinearSolver - Type of nonlinear solver used [ Functional | Newton ]
   The 'Functional' nonlinear solver is best suited for non-stiff
   problems, in conjunction with the 'Adams' linear multistep method,
   while 'Newton' is better suited for stiff problems, using the 'BDF'
   method.
RelTol - Relative tolerance [ positive scalar | 1e-4 ]
   RelTol defaults to 1e-4 and is applied to all components of the solution
   vector. See AbsTol.
AbsTol - Absolute tolerance [ positive scalar or vector | 1e-6 ]
   The relative and absolute tolerances define a vector of error weights
   with components
     ewt(i) = 1/(RelTol*|y(i)| + AbsTol)    if AbsTol is a scalar
```

```
    ewt(i) = 1/(RelTol*|y(i)| + AbsTol(i)) if AbsTol is a vector
   This vector is used in all error and convergence tests, which
   use a weighted RMS norm on all error-like vectors v:
     WRMSnorm(v) = sqrt( (1/N) sum(i=1..N) (v(i)*ewt(i))^2 ),
   where N is the problem dimension.
MaxNumSteps - Maximum number of steps [positive integer | 500]
   CVode will return with an error after taking MaxNumSteps internal steps
   in its attempt to reach the next output time.
InitialStep - Suggested initial stepsize [ positive scalar ]
   By default, CVode estimates an initial stepsize h0 at the initial time
   t0 as the solution of
     WRMSnorm(h0^2 ydd / 2) = 1
   where ydd is an estimated second derivative of y(t0).
MaxStep - Maximum stepsize [ positive scalar | inf ]
   Defines an upper bound on the integration step size.
MinStep - Minimum stepsize [ positive scalar | 0.0 ]
   Defines a lower bound on the integration step size.
MaxOrder - Maximum method order [ 1-12 for Adams, 1-5 for BDF | 5 ]
   Defines an upper bound on the linear multistep method order.
StopTime - Stopping time [ scalar ]
   Defines a value for the independent variable past which the solution
   is not to proceed.
RootsFn - Rootfinding function [ function ]
   To detect events (roots of functions), set this property to the event
   function. See CVRootFn.
NumRoots - Number of root functions [ integer | 0 ]
   Set NumRoots to the number of functions for which roots are monitored.
   If NumRoots is 0, rootfinding is disabled.
StabilityLimDet - Stability limit detection algorithm [ false | true ]
   Flag used to turn on or off the stability limit detection algorithm
   within CVODES. This property can be used only with the BDF method.
   In this case, if the order is 3 or greater and if the stability limit
   is detected, the method order is reduced.

LinearSolver - Linear solver type [Dense|Diag|Band|GMRES|BiCGStab|TFQMR]
   Specifies the type of linear solver to be used for the Newton nonlinear
   solver (see NonlinearSolver). Valid choices are: Dense (direct, dense
   Jacobian), Band (direct, banded Jacobian), Diag (direct, diagonal Jacobian),
   GMRES (iterative, scaled preconditioned GMRES), BiCGStab (iterative, scaled
   preconditioned stabilized BiCG), TFQMR (iterative, scaled transpose-free QMR).
   The GMRES, BiCGStab, and TFQMR are matrix-free linear solvers.
JacobianFn - Jacobian function [ function ]
   This propeerty is overloaded. Set this value to a function that returns
   Jacobian information consistent with the linear solver used (see Linsolver).
   If not specified, CVODES uses difference quotient approximations.
   For the Dense linear solver, JacobianFn must be of type CVDenseJacFn and
   must return a dense Jacobian matrix. For the Band linear solver, JacobianFn
   must be of type CVBandJacFn and must return a banded Jacobian matrix.
   For the iterative linear solvers, GMRES, BiCGStab, and TFQMR, JacobianFn must
   be of type CVJacTimesVecFn and must return a Jacobian-vector product. This
   property is not used for the Diag linear solver.
   If these options are for a backward problem, the corresponding funciton types
   are CVDenseJacFnB for the Dense linear solver, CVBandJacFnB for he band linear
   solver, and CVJacTimesVecFnB for the iterative linear solvers.
```

KrylovMaxDim - Maximum number of Krylov subspace vectors [integer | 5]
Specifies the maximum number of vectors in the Krylov subspace. This property is used only if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver).

GramSchmidtType - Gram-Schmidt orthogonalization [Classical | Modified]
Specifies the type of Gram-Schmidt orthogonalization (classical or modified). This property is used only if the GMRES linear solver is used (see LinSolver).

PrecType - Preconditioner type [Left | Right | Both | None]
Specifies the type of user preconditioning to be done if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver). PrecType must be one of the following: 'None', 'Left', 'Right', or 'Both', corresponding to no preconditioning, left preconditioning only, right preconditioning only, and both left and right preconditioning, respectively.

PrecModule - Preconditioner module [BandPre | BBDPre | UserDefined]
If PrecModule = 'UserDefined', then the user must provide at least a preconditioner solve function (see PrecSolveFn)
CVODES provides the following two general-purpose preconditioner modules:
BandPre provide a band matrix preconditioner based on difference quotients of the ODE right-hand side function. The user must specify the lower and upper half-bandwidths through the properties LowerBwidth and UpperBwidth, respectively.
BBDPre can be only used with parallel vectors. It provide a preconditioner matrix that is block-diagonal with banded blocks. The blocking corresponds to the distribution of the dependent variable vector y among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor) of a given function g(t,y) approximating f(t,y) (see GlocalFn). The blocks are generated by a difference quotient scheme on each processor independently. This scheme utilizes an assumed banded structure with given half-bandwidths, mldq and mudq (specified through LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian block kept by the scheme has half-bandwiths ml and mu (specified through LowerBwidth and UpperBwidth), which may be smaller.

PrecSetupFn - Preconditioner setup function [function]
If PrecType is not 'None', PrecSetupFn specifies an optional function which, together with PrecSolve, defines left and right preconditioner matrices (either of which can be trivial), such that the product P1*P2 is an aproximation to the Newton matrix. PrecSetupFn must be of type CVPrecSetupFn or CVPrecSetupFnB for forward and backward problems, respectively.

PrecSolveFn - Preconditioner solve function [function]
If PrecType is not 'None', PrecSolveFn specifies a required function which must solve a linear system Pz = r, for given r. PrecSolveFn must be of type CVPrecSolveFn or CVPrecSolveFnB for forward and backward problems, respectively.

GlocalFn - Local right-hand side approximation funciton for BBDPre [function]
If PrecModule is BBDPre, GlocalFn specifies a required function that evaluates a local approximation to the ODE right-hand side. GlocalFn must be of type CVGlocFn or CVGlocFnB for forward and backward problems, respectively.

GcommFn - Inter-process communication function for BBDPre [function]
If PrecModule is BBDPre, GcommFn specifies an optional function to perform any inter-process communication required for the evaluation of GlocalFn. GcommFn must be of type CVGcommFn or CVGcommFnB for forward and backward problems, respectively.

LowerBwidth - Jacobian/preconditioner lower bandwidth [integer | 0]
This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the lower half-bandwidth of the band Jacobian approximation.

```
   If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver) and if the BBDPre preconditioner module in CVODES is used
   (see PrecModule), it specifies the lower half-bandwidth of the retained
   banded approximation of the local Jacobian block. If the BandPre preconditioner
   module (see PrecModule) is used, it specifies the lower half-bandwidth of
   the band preconditioner matrix. LowerBwidth defaults to 0 (no sub-diagonals).
UpperBwidth - Jacobian/preconditioner upper bandwidth [ integer | 0 ]
   This property is overloaded. If the Band linear solver is used (see LinSolver),
   it specifies the upper half-bandwidth of the band Jacobian approximation.
   If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver) and if the BBDPre preconditioner module in CVODES is used
   (see PrecModule), it specifies the upper half-bandwidth of the retained
   banded approximation of the local Jacobian block. If the BandPre
   preconditioner module (see PrecModule) is used, it specifies the upper
   half-bandwidth of the band preconditioner matrix. UpperBwidth defaults to
   0 (no super-diagonals).
LowerBwidthDQ - BBDPre preconditioner DQ lower bandwidth [ integer | 0 ]
   Specifies the lower half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).
UpperBwidthDQ - BBDPre preconditioner DQ upper bandwidth [ integer | 0 ]
   Specifies the upper half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).

MonitorFn - User-provied monitoring function [ function ]
   Specifies a function that is called after each successful integration step.
   This function must have type CVMonitorFn or CVMonitorFnB, depending on
   whether these options are for a forward or a backward problem, respectively.
   Sample monitoring functions CVodeMonitor and CvodeMonitorB are provided
   with CVODES.
MonitorData - User-provied data for the monitoring function [ struct ]
   Specifies a data structure that is passed to the MonitorFn function every
   time it is called.

SensDependent - Backward problem depending on sensitivities [ false | true ]
   Specifies whether the backward problem right-hand side depends on
   forward sensitivites. If TRUE, the right-hand side function provided for
   this backward problem must have the appropriate type (see CVRhsFnB).

ErrorMessages - Post error/warning messages [ true | false ]
   Note that any errors in CVodeInit will result in a Matlab error, thus
   stoping execution. Only subsequent calls to CVODES functions will respect
   the value specified for 'ErrorMessages'.

NOTES:

   The properties listed above that can only be used for forward problems
   are: StopTime, RootsFn, and NumRoots.

   The property SensDependent is relevant only for backward problems.


   See also
        CVodeInit, CVodeReInit, CVodeInitB, CVodeReInitB
        CVRhsFn, CVRootFn,
```

```
CVDenseJacFn, CVBandJacFn, CVJacTimesVecFn
CVPrecSetupFn, CVPrecSolveFn
CVGlocalFn, CVGcommFn
CVMonitorFn
CVRhsFnB,
CVDenseJacFnB, CVBandJacFnB, CVJacTimesVecFnB
CVPrecSetupFnB, CVPrecSolveFnB
CVGlocalFnB, CVGcommFnB
CVMonitorFnB
```

CVodeQuadSetOptions

Purpose

CVodeQuadSetOptions creates an options structure for quadrature integration with CVODES.

Synopsis

```
function options = CVodeQuadSetOptions(varargin)
```

Description

```
CVodeQuadSetOptions creates an options structure for quadrature integration with CVODES.

   Usage: OPTIONS = CVodeQuadSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = CVodeQuadSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = CVodeQuadSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a CVODES options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = CVodeQuadSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.

   CVodeQuadSetOptions with no input arguments displays all property names
   and their possible values.

CVodeQuadSetOptions properties
(See also the CVODES User Guide)

ErrControl - Error control strategy for quadrature variables [ false | true ]
   Specifies whether quadrature variables are included in the error test.
RelTol - Relative tolerance for quadrature variables [ scalar 1e-4 ]
   Specifies the relative tolerance for quadrature variables. This parameter is
   used only if ErrControl = true.
AbsTol - Absolute tolerance for quadrature variables [ scalar or vector 1e-6 ]
   Specifies the absolute tolerance for quadrature variables. This parameter is
   used only if ErrControl = true.

SensDependent - Backward problem depending on sensitivities [ false | true ]
   Specifies whether the backward problem quadrature right-hand side depends
   on forward sensitivites. If TRUE, the right-hand side function provided for
```

```
   this backward problem must have the appropriate type (see CVQuadRhsFnB).


   See also
        CVodeQuadInit, CVodeQuadReInit.
        CVodeQuadInitB, CVodeQuadReInitB
```

CVodeSensSetOptions

Purpose

CVodeSensSetOptions creates an options structure for FSA with CVODES.

Synopsis

```
function options = CVodeSensSetOptions(varargin)
```

Description

```
CVodeSensSetOptions creates an options structure for FSA with CVODES.

   Usage: OPTIONS = CVodeSensSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = CVodeSensSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = CVodeSensSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a CVODES options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = CVodeSensSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.

   CVodeSensSetOptions with no input arguments displays all property names
   and their possible values.

CVodeSensSetOptions properties
(See also the CVODES User Guide)

method - FSA solution method [ 'Simultaneous' | 'Staggered' ]
   Specifies the FSA method for treating the nonlinear system solution for
   sensitivity variables. In the simultaneous case, the nonlinear systems
   for states and all sensitivities are solved simultaneously. In the
   Staggered case, the nonlinear system for states is solved first and then
   the nonlinear systems for all sensitivities are solved at the same time.
ParamField - Problem parameters  [ string ]
   Specifies the name of the field in the user data structure (specified through
   the 'UserData' field with CVodeSetOptions) in which the nominal values of the problem
   parameters are stored. This property is used only if  CVODES will use difference
   quotient approximations to the sensitivity right-hand sides (see CVSensRhsFn).
ParamList - Parameters with respect to which FSA is performed [ integer vector ]
   Specifies a list of Ns parameters with respect to which sensitivities are to
   be computed. This property is used only if CVODES will use difference-quotient
   approximations to the sensitivity right-hand sides. Its length must be Ns,
```

consistent with the number of columns of yS0 (see CVodeSensInit).
ParamScales - Order of magnitude for problem parameters [vector]
Provides order of magnitude information for the parameters with respect to which sensitivities are computed. This information is used if CVODES approximates the sensitivity right-hand sides or if CVODES estimates integration tolerances for the sensitivity variables (see RelTol and AbsTol).
RelTol - Relative tolerance for sensitivity variables [positive scalar]
Specifies the scalar relative tolerance for the sensitivity variables. See also AbsTol.
AbsTol - Absolute tolerance for sensitivity variables [row-vector or matrix]
Specifies the absolute tolerance for sensitivity variables. AbsTol must be either a row vector of dimension Ns, in which case each of its components is used as a scalar absolute tolerance for the coresponding sensitivity vector, or a N x Ns matrix, in which case each of its columns is used as a vector of absolute tolerances for the corresponding sensitivity vector.
By default, CVODES estimates the integration tolerances for sensitivity variables, based on those for the states and on the order of magnitude information for the problem parameters specified through ParamScales.
ErrControl - Error control strategy for sensitivity variables [false | true]
Specifies whether sensitivity variables are included in the error control test. Note that sensitivity variables are always included in the nonlinear system convergence test.
DQtype - Type of DQ approx. of the sensi. RHS [Centered | Forward]
Specifies whether to use centered (second-order) or forward (first-order) difference quotient approximations of the sensitivity eqation right-hand sides. This property is used only if a user-defined sensitivity right-hand side function was not provided.
DQparam - Cut-off parameter for the DQ approx. of the sensi. RHS [scalar | 0.0]
Specifies the value which controls the selection of the difference-quotient scheme used in evaluating the sensitivity right-hand sides (switch between simultaneous or separate evaluations of the two components in the sensitivity right-hand side). The default value 0.0 indicates the use of simultaenous approximation exclusively (centered or forward, depending on the value of DQtype.
For DQparam >= 1, CVODES uses a simultaneous approximation if the estimated DQ perturbations for states and parameters are within a factor of DQparam, and separate approximations otherwise. Note that a value DQparam < 1 will inhibit switching! This property is used only if a user-defined sensitivity right-hand side function was not provided.

See also
CVodeSensInit, CVodeSensReInit

CVodeInit

Purpose

CVodeInit allocates and initializes memory for CVODES.

Synopsis

function status = CVodeInit(fct, lmm, nls, t0, y0, options)

Description

```
CVodeInit allocates and initializes memory for CVODES.

   Usage: CVodeInit ( ODEFUN, LMM, NLS, T0, Y0 [, OPTIONS ] )

   ODEFUN   is a function defining the ODE right-hand side: y' = f(t,y).
            This function must return a vector containing the current
            value of the righ-hand side.
   LMM      is the Linear Multistep Method ('Adams' or 'BDF')
   NLS      is the type of nonlinear solver used ('Functional' or 'Newton')
   T0       is the initial value of t.
   Y0       is the initial condition vector y(t0).
   OPTIONS  is an (optional) set of integration options, created with
            the CVodeSetOptions function.

   See also: CVodeSetOptions, CVRhsFn

   NOTES:
    1) The 'Functional' nonlinear solver is best suited for non-stiff
       problems, in conjunction with the 'Adams' linear multistep method,
       while 'Newton' is better suited for stiff problems, using the 'BDF'
       method.
    2) When using the 'Newton' nonlinear solver, a linear solver is also
       required. The default one is 'Dense', indicating the use of direct
       dense linear algebra (LU factorization). A different linear solver
       can be specified through the option 'LinearSolver' to CVodeSetOptions.
```

CVodeQuadInit

Purpose

CVodeQuadInit allocates and initializes memory for quadrature integration.

Synopsis

```
function status = CVodeQuadInit(fctQ, yQ0, options)
```

Description

```
CVodeQuadInit allocates and initializes memory for quadrature integration.

   Usage: CVodeQuadInit ( QFUN, YQ0 [, OPTIONS ] )

   QFUN     is a function defining the righ-hand sides of the quadrature
            ODEs yQ' = fQ(t,y).
   YQ0      is the initial conditions vector yQ(t0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the CVodeSetQuadOptions function.

   See also: CVodeSetQuadOptions, CVQuadRhsFn
```

CVodeSensInit

Purpose

CVodeSensInit allocates and initializes memory for FSA with CVODES.

Synopsis

function status = CVodeSensInit(Ns,fctS,yS0,options)

Description

CVodeSensInit allocates and initializes memory for FSA with CVODES.

```
   Usage: CVodeSensInit ( NS, SFUN, YS0 [, OPTIONS ] )

   NS       is the number of parameters with respect to which sensitivities
            are desired
   SFUN     is a function defining the righ-hand sides of the sensitivity
            ODEs yS' = fS(t,y,yS).
   YS0      Initial conditions for sensitivity variables.
            YS0 must be a matrix with N rows and Ns columns, where N is the problem
            dimension and Ns the number of sensitivity systems.
   OPTIONS  is an (optional) set of FSA options, created with
            the CVodeSetFSAOptions function.

   See also CVodeSensSetOptions, CVodeInit, CVSensRhsFn
```

CVodeAdjInit

Purpose

CVodeAdjInit allocates and initializes memory for ASA with CVODES.

Synopsis

function status = CVodeAdjInit(steps, interp)

Description

CVodeAdjInit allocates and initializes memory for ASA with CVODES.

```
   Usage: CVodeAdjInit(STEPS, INTEPR)

   STEPS    specifies the (maximum) number of integration steps between two
            consecutive check points.
   INTERP   Specifies the type of interpolation used for estimating the forward
            solution during the backward integration phase. INTERP should be
            'Hermite', indicating cubic Hermite interpolation, or 'Polynomial',
            indicating variable order polynomial interpolation.
```

CVodeInitB

Purpose

CVodeInitB allocates and initializes backward memory for CVODES.

Synopsis

function [idxB, status] = CVodeInitB(fctB, lmmB, nlsB, tB0, yB0, optionsB)

Description

```
CVodeInitB allocates and initializes backward memory for CVODES.

   Usage:   IDXB = CVodeInitB ( FCTB, LMMB, NLSB, TB0, YB0 [, OPTIONSB] )

   FCTB     is a function defining the adjoint ODE right-hand side.
            This function must return a vector containing the current
            value of the adjoint ODE righ-hand side.
   LMMB     is the Linear Multistep Method ('Adams' or 'BDF')
   NLSB     is the type of nonlinear solver used ('Functional' or 'Newton')
   TB0      is the final value of t.
   YB0      is the final condition vector yB(tB0).
   OPTIONSB is an (optional) set of integration options, created with
            the CVodeSetOptions function.

   CVodeInitB returns the index IDXB associated with this backward
   problem. This index must be passed as an argument to any subsequent
   functions related to this backward problem.

   See also: CVodeSetOptions, CVodeInit, CVRhsFnB
```

CVodeQuadInitB

Purpose

CVodeQuadInitB allocates and initializes memory for backward quadrature integration.

Synopsis

```
function status = CVodeQuadInitB(idxB, fctQB, yQB0, optionsB)
```

Description

```
CVodeQuadInitB allocates and initializes memory for backward quadrature integration.

   Usage: CVodeQuadInitB ( IDXB, QBFUN, YQB0 [, OPTIONS ] )

   IDXB     is the index of the backward problem, returned by
            CVodeInitB.
   QBFUN    is a function defining the righ-hand sides of the
            backward ODEs yQB' = fQB(t,y,yB).
   YQB0     is the final conditions vector yQB(tB0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the CVodeSetQuadOptions function.

   See also: CVodeInitB, CVodeSetQuadOptions, CVQuadRhsFnB
```

CVodeReInit

Purpose

CVodeReInit reinitializes memory for CVODES

Synopsis

```
function status = CVodeReInit(t0, y0, options)
```

Description

```
CVodeReInit reinitializes memory for CVODES
   where a prior call to CVodeInit has been made with the same
   problem size N. CVodeReInit performs the same input checking
   and initializations that CVodeInit does, but it does no
   memory allocation, assuming that the existing internal memory
   is sufficient for the new problem.

   Usage: CVodeReInit ( T0, Y0 [, OPTIONS ] )

   T0       is the initial value of t.
   Y0       is the initial condition vector y(t0).
   OPTIONS  is an (optional) set of integration options, created with
            the CVodeSetOptions function.

   See also: CVodeSetOptions, CVodeInit
```

CVodeQuadReInit

Purpose

```
CVodeQuadReInit reinitializes CVODES's quadrature-related memory
```

Synopsis

```
function status = CVodeQuadReInit(yQ0, options)
```

Description

```
CVodeQuadReInit reinitializes CVODES's quadrature-related memory
   assuming it has already been allocated in prior calls to CVodeInit
   and CVodeQuadInit.

   Usage: CVodeQuadReInit ( YQ0 [, OPTIONS ] )

   YQ0      Initial conditions for quadrature variables yQ(t0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the CVodeSetQuadOptions function.

   See also: CVodeSetQuadOptions, CVodeQuadInit
```

CVodeSensReInit

Purpose

```
CVodeSensReInit reinitializes CVODES's FSA-related memory
```

Synopsis

```
function status = CVodeSensReInit(yS0, options)
```

Description

```
CVodeSensReInit reinitializes CVODES's FSA-related memory
   assuming it has already been allocated in prior calls to CVodeInit
   and CVodeSensInit.
   The number of sensitivities Ns is assumed to be unchanged since the
   previous call to CVodeSensInit.

   Usage: CVodeSensReInit ( YS0 [, OPTIONS ] )

   YS0      Initial conditions for sensitivity variables.
            YS0 must be a matrix with N rows and Ns columns, where N is the problem
            dimension and Ns the number of sensitivity systems.
   OPTIONS  is an (optional) set of FSA options, created with
            the CVodeSensSetOptions function.

   See also: CVodeSensSetOptions, CVodeReInit, CVodeSensInit
```

CVodeAdjReInit

Purpose

CVodeAdjReInit re-initializes memory for ASA with CVODES.

Synopsis

```
function status = CVodeAdjReInit()
```

Description

```
CVodeAdjReInit re-initializes memory for ASA with CVODES.

   Usage: CVodeAdjReInit
```

CVodeReInitB

Purpose

CVodeReInitB re-initializes backward memory for CVODES.

Synopsis

```
function status = CVodeReInitB(idxB, tB0, yB0, optionsB)
```

Description

```
CVodeReInitB re-initializes backward memory for CVODES.
   where a prior call to CVodeInitB has been made with the same
   problem size NB. CVodeReInitB performs the same input checking
   and initializations that CVodeInitB does, but it does no
   memory allocation, assuming that the existing internal memory
   is sufficient for the new problem.

   Usage:   CVodeReInitB ( IDXB, TB0, YB0 [, OPTIONSB] )

   IDXB     is the index of the backward problem, returned by
```

```
          CVodeInitB.
TB0       is the final value of t.
YB0       is the final condition vector yB(tB0).
OPTIONSB is an (optional) set of integration options, created with
          the CVodeSetOptions function.

See also: CVodeSetOptions, CVodeInitB
```

CVodeQuadReInitB

Purpose

CVodeQuadReInitB reinitializes memory for backward quadrature integration.

Synopsis

```
function status = CVodeQuadReInitB(idxB, yQB0, optionsB)
```

Description

CVodeQuadReInitB reinitializes memory for backward quadrature integration.

```
Usage: CVodeQuadReInitB ( IDXB, YS0 [, OPTIONS ] )

IDXB      is the index of the backward problem, returned by
          CVodeInitB.
YQB0      is the final conditions vector yQB(tB0).
OPTIONS   is an (optional) set of QUAD options, created with
          the CVodeSetQuadOptions function.

See also: CVodeSetQuadOptions, CVodeReInitB, CVodeQuadInitB
```

CVode

Purpose

CVode integrates the ODE.

Synopsis

```
function [varargout] = CVode(tout, itask)
```

Description

CVode integrates the ODE.

```
Usage: [STATUS, T, Y] = CVode ( TOUT, ITASK )
       [STATUS, T, Y, YS] = CVode ( TOUT, ITASK )
       [STATUS, T, Y, YQ] = CVode  (TOUT, ITASK )
       [STATUS, T, Y, YQ, YS] = CVode ( TOUT, ITASK )

If ITASK is 'Normal', then the solver integrates from its current internal
T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
Y(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
```

```
and returns in Y the solution at the new internal time. In this case, TOUT
is used only during the first call to CVode to determine the direction of
integration and the rough scale of the problem. In either case, the time
reached by the solver is returned in T.

If quadratures were computed (see CVodeQuadInit), CVode will return their
values at T in the vector YQ.

If sensitivity calculations were enabled (see CVodeSensInit), CVode will
return their values at T in the matrix YS. Each row in the matrix YS
represents the sensitivity vector with respect to one of the problem parameters.

In ITASK =' Normal' mode, to obtain solutions at specific times T0,T1,...,TFINAL
(all increasing or all decreasing) use TOUT = [T0 T1  ... TFINAL]. In this case
the output arguments Y and YQ are matrices, each column representing the solution
vector at the corresponding time returned in the vector T. If computed, the
sensitivities are eturned in the 3-dimensional array YS, with YS(:,:,I) representing
the sensitivity vectors at the time T(I).

On return, STATUS is one of the following:
  0: successful CVode return.
  1: CVode succeded and returned at tstop.
  2: CVode succeeded and found one or more roots.
 -1: an error occurred (see printed message).

See also CVodeSetOptions, CVodeGetStats
```

CVodeB

PURPOSE

CVodeB integrates all backwards ODEs currently defined.

SYNOPSIS

function [varargout] = CVodeB(tout,itask)

DESCRIPTION

CVodeB integrates all backwards ODEs currently defined.

```
Usage:  [STATUS, T, YB] = CVodeB ( TOUT, ITASK )
        [STATUS, T, YB, YQB] = CVodeB ( TOUT, ITASK )

If ITASK is 'Normal', then the solver integrates from its current internal
T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
YB(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
and returns in YB the solution at the new internal time. In this case, TOUT
is used only during the first call to CVodeB to determine the direction of
integration and the rough scale of the problem. In either case, the time
reached by the solver is returned in T.

If quadratures were computed (see CVodeQuadInitB), CVodeB will return their
values at T in the vector YQB.
```

```
In ITASK =' Normal' mode, to obtain solutions at specific times T0,T1,...,TFINAL
(all increasing or all decreasing) use TOUT = [T0 T1  ... TFINAL]. In this case
the output arguments YB and YQB are matrices, each column representing the solution
vector at the corresponding time returned in the vector T.

If more than one backward problem was defined, the return arguments are cell
arrays, with TIDXB, YBIDXB, and YQBIDXB corresponding to the backward
problem with index IDXB (as returned by CVodeInitB).

On return, STATUS is one of the following:
  0: successful CVodeB return.
  1: CVodeB succeded and return at a tstop value (internally set).
 -1: an error occurred (see printed message).

See also CVodeSetOptions, CVodeGetStatsB
```

CVodeSensToggleOff

PURPOSE

CVodeSensToggleOff deactivates sensitivity calculations.

SYNOPSIS

function status = CVodeSensToggleOff()

DESCRIPTION

```
CVodeSensToggleOff deactivates sensitivity calculations.
  It does NOT deallocate sensitivity-related memory so that
  sensitivity computations can be later toggled ON (through
  CVodeSensReInit).

  Usage: CVodeSensToggleOff

  See also: CVodeSensInit, CVodeSensReInit
```

CVodeGetStats

PURPOSE

CVodeGetStats returns run statistics for the CVODES solver.

SYNOPSIS

function [si, status] = CVodeGetStats()

DESCRIPTION

```
CVodeGetStats returns run statistics for the CVODES solver.

  Usage: STATS = CVodeGetStats
```

```
Fields in the structure STATS

o nst      - number of integration steps
o nfe      - number of right-hand side function evaluations
o nsetups  - number of linear solver setup calls
o netf     - number of error test failures
o nni      - number of nonlinear solver iterations
o ncfn     - number of convergence test failures
o qlast    - last method order used
o qcur     - current method order
o h0used   - actual initial step size used
o hlast    - last step size used
o hcur     - current step size
o tcur     - current time reached by the integrator
o RootInfo - strucutre with rootfinding information
o QuadInfo - structure with quadrature integration statistics
o LSInfo   - structure with linear solver statistics
o FSAInfo  - structure with forward sensitivity solver statistics

If rootfinding was requested, the structure RootInfo has the following fields

o nge   - number of calls to the rootfinding function
o roots - array of integers (a value of 1 in the i-th component means that the
          i-th rootfinding function has a root (upon a return with status=2 from
          CVode).

If quadratures were present, the structure QuadInfo has the following fields

o nfQe  - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

The structure LSinfo has different fields, depending on the linear solver used.

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'Diag' linear solver

o name  - 'Diag'
o nfeDI - number of right-hand side function evaluations for difference-quotient
          Jacobian approximation

  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nfeB - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers
```

```
o name  - 'GMRES' or 'BiCGStab'
o nli   - number of linear solver iterations
o npe   - number of preconditioner setups
o nps   - number of preconditioner solve function calls
o ncfl  - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nfeSG - number of right-hand side function evaluations for difference-quotient
          Jacobian-vector product approximation

If forward sensitivities were computed, the structure FSAInfo has the
following fields

o nfSe      - number of sensitivity right-hand side evaluations
o nfeS      - number of right-hand side evaluations for difference-quotient
              sensitivity right-hand side approximation
o nsetupsS  - number of linear solver setups triggered by sensitivity variables
o netfS     - number of error test failures for sensitivity variables
o nniS      - number of nonlinear solver iterations for sensitivity variables
o ncfnS     - number of convergence test failures due to sensitivity variables
```

CVodeGetStatsB

PURPOSE

CVodeGetStatsB returns run statistics for the backward CVODES solver.

SYNOPSIS

```
function [si, status] = CVodeGetStatsB(idxB)
```

DESCRIPTION

```
CVodeGetStatsB returns run statistics for the backward CVODES solver.

   Usage: STATS = CVodeGetStatsB( IDXB )

   IDXB     is the index of the backward problem, returned by
            CVodeInitB.

Fields in the structure STATS

o nst - number of integration steps
o nfe - number of right-hand side function evaluations
o nsetups - number of linear solver setup calls
o netf - number of error test failures
o nni - number of nonlinear solver iterations
o ncfn - number of convergence test failures
o qlast - last method order used
o qcur - current method order
o h0used - actual initial step size used
o hlast - last step size used
o hcur - current step size
o tcur - current time reached by the integrator
```

```
o QuadInfo - structure with quadrature integration statistics
o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

If quadratures were present, the structure QuadInfo has the following fields

o nfQe - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'Diag' linear solver

o name - 'Diag'
o nfeDI - number of right-hand side function evaluations for difference-quotient
          Jacobian approximation

  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nfeB - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

o name - 'GMRES' or 'BiCGStab'
o nli - number of linear solver iterations
o npe - number of preconditioner setups
o nps - number of preconditioner solve function calls
o ncfl - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nfeSG -  number of right-hand side function evaluations for difference-quotient
          Jacobian-vector product approximation
```

CVodeGet

Purpose

CVodeGet extracts data from the CVODES solver memory.

Synopsis

```
function [output, status] = CVodeGet(key, varargin)
```

Description

```
CVodeGet extracts data from the CVODES solver memory.

   Usage: RET = CVodeGet ( KEY [, P1 [, P2] ... ])

   CVodeGet returns internal CVODES information based on KEY. For some values
   of KEY, additional arguments may be required and/or more than one output is
   returned.

   KEY is a string and should be one of:
    o DerivSolution - Returns a vector containing the K-th order derivative
      of the solution at time T. The time T and order K must be passed through
      the input arguments P1 and P2, respectively:
      DKY = CVodeGet('DerivSolution', T, K)
    o ErrorWeights - Returns a vector containing the current error weights.
      EWT = CVodeGet('ErrorWeights')
    o CheckPointsInfo - Returns an array of structures with check point information.
      CK = CVodeGet('CheckPointInfo)
```

CVodeSet

Purpose

CVodeSet changes optional input values during the integration.

Synopsis

```
function status = CVodeSet(varargin)
```

Description

```
CVodeSet changes optional input values during the integration.

   Usage: CVodeSet('NAME1',VALUE1,'NAME2',VALUE2,...)

   CVodeSet can be used to change some of the optional inputs during
   the integration, i.e., without need for a solver reinitialization.
   The property names accepted by CVodeSet are a subset of those valid
   for CVodeSetOptions. Any unspecified properties are left unchanged.

   CVodeSet with no input arguments displays all property names.

CVodeSet properties
(See also the CVODES User Guide)

UserData - problem data passed unmodified to all user functions.
  Set VALUE to be the new user data.
RelTol - Relative tolerance
  Set VALUE to the new relative tolerance
AbsTol - absolute tolerance
  Set VALUE to be either the new scalar absolute tolerance or
  a vector of absolute tolerances, one for each solution component.
StopTime - Stopping time
  Set VALUE to be a new value for the independent variable past which
  the solution is not to proceed.
```

CVodeSetB

Purpose

CVodeSetB changes optional input values during the integration.

Synopsis

```
function status = CVodeSetB(idxB, varargin)
```

Description

CVodeSetB changes optional input values during the integration.

```
   Usage: CVodeSetB( IDXB, 'NAME1',VALUE1,'NAME2',VALUE2,... )

   CVodeSetB can be used to change some of the optional inputs for
   the backward problem identified by IDXB during the backward
   integration, i.e., without need for a solver reinitialization.
   The property names accepted by CVodeSet are a subset of those valid
   for CVodeSetOptions. Any unspecified properties are left unchanged.

   CVodeSetB with no input arguments displays all property names.

CVodeSetB properties
(See also the CVODES User Guide)

UserData - problem data passed unmodified to all user functions.
  Set VALUE to be the new user data.
RelTol - Relative tolerance
  Set VALUE to the new relative tolerance
AbsTol - absolute tolerance
  Set VALUE to be either the new scalar absolute tolerance or
  a vector of absolute tolerances, one for each solution component.
```

CVodeFree

Purpose

CVodeFree deallocates memory for the CVODES solver.

Synopsis

```
function CVodeFree()
```

Description

CVodeFree deallocates memory for the CVODES solver.

```
   Usage:  CVodeFree
```

3.2 Function types

CVRhsFn

Purpose

CVRhsFn - type for user provided RHS function

Synopsis

This is a script file.

Description

CVRhsFn - type for user provided RHS function

```
   The function ODEFUN must be defined as
        FUNCTION [YD, FLAG] = ODEFUN(T,Y)
   and must return a vector YD corresponding to f(t,y).
   If a user data structure DATA was specified in CVodeInit, then
   ODEFUN must be defined as
        FUNCTION [YD, FLAG, NEW_DATA] = ODEFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector YD,
   the ODEFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function ODEFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also CVodeInit
```

CVSensRhsFn

Purpose

CVSensRhsFn - type for user provided sensitivity RHS function.

Synopsis

This is a script file.

Description

CVSensRhsFn - type for user provided sensitivity RHS function.

```
   The function ODESFUN must be defined as
        FUNCTION [YSD, FLAG] = ODESFUN(T,Y,YD,YS)
   and must return a matrix YSD corresponding to fS(t,y,yS).
   If a user data structure DATA was specified in CVodeInit, then
   ODESFUN must be defined as
        FUNCTION [YSD, FLAG, NEW_DATA] = ODESFUN(T,Y,YD,YS,DATA)
```

```
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the matrix YSD,
the ODESFUN function must also set NEW_DATA. Otherwise, it should
set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

The function ODESFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVodeSetFSAOptions

NOTE: ODESFUN is specified through the property FSARhsFn to
      CVodeSetFSAOptions.
```

CVQuadRhsFn

Purpose

CVQuadRhsFn - type for user provided quadrature RHS function.

Synopsis

This is a script file.

Description

CVQuadRhsFn - type for user provided quadrature RHS function.

```
The function ODEQFUN must be defined as
     FUNCTION [YQD, FLAG] = ODEQFUN(T,Y)
and must return a vector YQD corresponding to fQ(t,y), the integrand
for the integral to be evaluated.
If a user data structure DATA was specified in CVodeInit, then
ODEQFUN must be defined as
     FUNCTION [YQD, FLAG, NEW_DATA] = ODEQFUN(T,Y,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the vector YQD,
the ODEQFUN function must also set NEW_DATA. Otherwise, it should set
NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

The function ODEQFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVodeQuadInit
```

CVRootFn

Purpose

CVRootFn - type for user provided root-finding function.

Synopsis

This is a script file.

Description

```
CVRootFn - type for user provided root-finding function.

   The function ROOTFUN must be defined as
        FUNCTION [G, FLAG] = ROOTFUN(T,Y)
   and must return a vector G corresponding to g(t,y).
   If a user data structure DATA was specified in CVodeInit, then
   ROOTFUN must be defined as
        FUNCTION [G, FLAG, NEW_DATA] = ROOTFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector G,
   the ROOTFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function ROOTFUN must set FLAG=0 if successful, or FLAG~=0 if
   a failure occurred.

   See also CVodeSetOptions

   NOTE: ROOTFUN is specified through the RootsFn property in
   CVodeSetOptions and is used only if the property NumRoots is a
   positive integer.
```

CVDenseJacFn

Purpose

```
CVDenseJacFn - type for user provided dense Jacobian function.
```

Synopsis

```
This is a script file.
```

Description

```
CVDenseJacFn - type for user provided dense Jacobian function.

   The function DJACFUN must be defined as
        FUNCTION [J, FLAG] = DJACFUN(T, Y, FY)
   and must return a matrix J corresponding to the Jacobian of f(t,y).
   The input argument FY contains the current value of f(t,y).
   If a user data structure DATA was specified in CVodeInit, then
   DJACFUN must be defined as
        FUNCTION [J, FLAG, NEW_DATA] = DJACFUN(T, Y, FY, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J,
   the DJACFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function DJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.
   See also CVodeSetOptions
```

```
NOTE: DJACFUN is specified through the property JacobianFn to
CVodeSetOptions and is used only if the property LinearSolver
was set to 'Dense'.
```

CVBandJacFn

Purpose

CVBandJacFn - type for user provided banded Jacobian function.

Synopsis

This is a script file.

Description

CVBandJacFn - type for user provided banded Jacobian function.

```
The function BJACFUN must be defined as
     FUNCTION [J, FLAG] = BJACFUN(T, Y, FY)
and must return a matrix J corresponding to the banded Jacobian of f(t,y).
The input argument FY contains the current value of f(t,y).
If a user data structure DATA was specified in CVodeInit, then
BJACFUN must be defined as
     FUNCTION [J, FLAG, NEW_DATA] = BJACFUN(T, Y, FY, DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the matrix J,
the BJACFUN function must also set NEW_DATA. Otherwise, it should
set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

The function BJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVodeSetOptions

See the CVODES user guide for more informaiton on the structure of
a banded Jacobian.

NOTE: BJACFUN is specified through the property JacobianFn to
CVodeSetOptions and is used only if the property LinearSolver
was set to 'Band'.
```

CVJacTimesVecFn

Purpose

CVJacTimesVecFn - type for user provided Jacobian times vector function.

Synopsis

This is a script file.

Description

```
CVJacTimesVecFn - type for user provided Jacobian times vector function.

   The function JTVFUN must be defined as
        FUNCTION [JV, FLAG] = JTVFUN(T,Y,FY,V)
   and must return a vector JV corresponding to the product of the
   Jacobian of f(t,y) with the vector v.
   The input argument FY contains the current value of f(t,y).
   If a user data structure DATA was specified in CVodeInit, then
   JTVFUN must be defined as
        FUNCTION [JV, FLAG, NEW_DATA] = JTVFUN(T,Y,FY,V,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector JV,
   the JTVFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function JTVFUN must set FLAG=0 if successful, or FLAG~=0 if
   a failure occurred.

   See also CVodeSetOptions

   NOTE: JTVFUN is specified through the property JacobianFn to
   CVodeSetOptions and is used only if the property LinearSolver
   was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

CVPrecSetupFn

Purpose

CVPrecSetupFn - type for user provided preconditioner setup function.

Synopsis

This is a script file.

Description

```
CVPrecSetupFn - type for user provided preconditioner setup function.

   The user-supplied preconditioner setup function PSETFUN and
   the user-supplied preconditioner solve function PSOLFUN
   together must define left and right preconditoner matrices
   P1 and P2 (either of which may be trivial), such that the
   product P1*P2 is an approximation to the Newton matrix
   M = I - gamma*J.  Here J is the system Jacobian J = df/dy,
   and gamma is a scalar proportional to the integration step
   size h.  The solution of systems P z = r, with P = P1 or P2,
   is to be carried out by the PrecSolve function, and PSETFUN
   is to do any necessary setup operations.

   The user-supplied preconditioner setup function PSETFUN
   is to evaluate and preprocess any Jacobian-related data
   needed by the preconditioner solve function PSOLFUN.
   This might include forming a crude approximate Jacobian,
```

```
and performing an LU factorization on the resulting
approximation to M.  This function will not be called in
advance of every call to PSOLFUN, but instead will be called
only as often as necessary to achieve convergence within the
Newton iteration.  If the PSOLFUN function needs no
preparation, the PSETFUN function need not be provided.

For greater efficiency, the PSETFUN function may save
Jacobian-related data and reuse it, rather than generating it
from scratch.  In this case, it should use the input flag JOK
to decide whether to recompute the data, and set the output
flag JCUR accordingly.

Each call to the PSETFUN function is preceded by a call to
ODEFUN with the same (t,y) arguments.  Thus the PSETFUN
function can use any auxiliary data that is computed and
saved by the ODEFUN function and made accessible to PSETFUN.

The function PSETFUN must be defined as
     FUNCTION [JCUR, FLAG] = PSETFUN(T,Y,FY,JOK,GAMMA)
and must return a logical flag JCUR (true if Jacobian information
was recomputed and false if saved data was reused). If PSETFUN
was successful, it must return FLAG=0. For a recoverable error (in
which case the setup will be retried) it must set FLAG to a positive
integer value. If an unrecoverable error occurs, it must set FLAG
to a negative value, in which case the integration will be halted.
The input argument FY contains the current value of f(t,y).
If the input logical flag JOK is false, it means that
Jacobian-related data must be recomputed from scratch. If it is true,
it means that Jacobian data, if saved from the previous PSETFUN call
can be reused (with the current value of GAMMA).

If a user data structure DATA was specified in CVodeInit, then
PSETFUN must be defined as
     FUNCTION [JCUR, FLAG, NEW_DATA] = PSETFUN(T,Y,FY,JOK,GAMMA,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the flags JCUR
and FLAG, the PSETFUN function must also set NEW_DATA. Otherwise, it
should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
to unnecessary copying).

See also CVPrecSolveFn, CVodeSetOptions

NOTE: PSETFUN is specified through the property PrecSetupFn to
CVodeSetOptions and is used only if the property LinearSolver was
set to 'GMRES', 'BiCGStab', or 'TFQMR' and if the property PrecType
is not 'None'.
```

CVPrecSolveFn

Purpose

CVPrecSolveFn - type for user provided preconditioner solve function.

Synopsis

This is a script file.

Description

CVPrecSolveFn - type for user provided preconditioner solve function.

```
The user-supplied preconditioner solve function PSOLFN
is to solve a linear system P z = r in which the matrix P is
one of the preconditioner matrices P1 or P2, depending on the
type of preconditioning chosen.

The function PSOLFUN must be defined as
     FUNCTION [Z, FLAG] = PSOLFUN(T,Y,FY,R)
and must return a vector Z containing the solution of Pz=r.
If PSOLFUN was successful, it must return FLAG=0. For a recoverable
error (in which case the step will be retried) it must set FLAG to a
positive value. If an unrecoverable error occurs, it must set FLAG
to a negative value, in which case the integration will be halted.
The input argument FY contains the current value of f(t,y).

If a user data structure DATA was specified in CVodeInit, then
PSOLFUN must be defined as
     FUNCTION [Z, FLAG, NEW_DATA] = PSOLFUN(T,Y,FY,R,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the vector Z and
the flag FLAG, the PSOLFUN function must also set NEW_DATA. Otherwise,
it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would
lead to unnecessary copying).

See also CVPrecSetupFn, CVodeSetOptions

NOTE: PSOLFUN is specified through the property PrecSolveFn to
CVodeSetOptions and is used only if the property LinearSolver was
set to 'GMRES', 'BiCGStab', or 'TFQMR' and if the property PrecType
is not 'None'.
```

CVGcommFn

Purpose

CVGcommFn - type for user provided communication function (BBDPre).

Synopsis

This is a script file.

Description

CVGcommFn - type for user provided communication function (BBDPre).

```
   The function GCOMFUN must be defined as
        FUNCTION FLAG = GCOMFUN(T, Y)
   and can be used to perform all interprocess communication necessary
   to evaluate the approximate right-hand side function for the BBDPre
   preconditioner module.
   If a user data structure DATA was specified in CVodeInit, then
   GCOMFUN must be defined as
        FUNCTION [FLAG, NEW_DATA] = GCOMFUN(T, Y, DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then the GCOMFUN function must also
   set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set
   NEW_DATA = DATA as it would lead to unnecessary copying).

   The function GCOMFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also CVGlocalFn, CVodeSetOptions

   NOTES:
     GCOMFUN is specified through the GcommFn property in CVodeSetOptions
     and is used only if the property PrecModule is set to 'BBDPre'.

     Each call to GCOMFUN is preceded by a call to the RHS function
     ODEFUN with the same arguments T and Y. Thus GCOMFUN can omit
     any communication done by ODEFUN if relevant to the evaluation
     of G by GLOCFUN. If all necessary communication was done by ODEFUN,
     GCOMFUN need not be provided.
```

CVGlocalFn

Purpose

CVGlocalFn - type for user provided RHS approximation function (BBDPre).

Synopsis

This is a script file.

Description

CVGlocalFn - type for user provided RHS approximation function (BBDPre).

```
   The function GLOCFUN must be defined as
        FUNCTION [GLOC, FLAG] = GLOCFUN(T,Y)
   and must return a vector GLOC corresponding to an approximation to f(t,y)
   which will be used in the BBDPRE preconditioner module. The case where
   G is mathematically identical to F is allowed.
   If a user data structure DATA was specified in CVodeInit, then
   GLOCFUN must be defined as
        FUNCTION [GLOC, FLAG, NEW_DATA] = GLOCFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed
```

```
in other user-provided functions then, besides setting the vector G,
the GLOCFUN function must also set NEW_DATA. Otherwise, it should set
NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

The function GLOCFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVGcommFn, CVodeSetOptions

NOTE: GLOCFUN is specified through the GlocalFn property in CVodeSetOptions
and is used only if the property PrecModule is set to 'BBDPre'.
```

CVMonitorFn

Purpose

CVMonitorFn - type for user provided monitoring function for forward problems.

Synopsis

This is a script file.

Description

CVMonitorFn - type for user provided monitoring function for forward problems.

```
The function MONFUN must be defined as
    FUNCTION [] = MONFUN(CALL, T, Y, YQ, YS)
It is called after every internal CVode step and can be used to
monitor the progress of the solver. MONFUN is called with CALL=0
from CVodeInit at which time it should initialize itself and it
is called with CALL=2 from CVodeFree. Otherwise, CALL=1.

It receives as arguments the current time T, solution vector Y,
and, if they were computed, quadrature vector YQ, and forward
sensitivity matrix YS. If YQ and/or YS were not computed they
are empty here.

If additional data is needed inside MONFUN, it must be defined
as
   FUNCTION NEW_MONDATA = MONFUN(CALL, T, Y, YQ, YS, MONDATA)
If the local modifications to the user data structure need to be
saved (e.g. for future calls to MONFUN), then MONFUN must set
NEW_MONDATA. Otherwise, it should set NEW_MONDATA=[]
(do not set NEW_MONDATA = DATA as it would lead to unnecessary copying).

A sample monitoring function, CVodeMonitor, is provided with CVODES.

See also CVodeSetOptions, CVodeMonitor

NOTES:
```

```
   MONFUN is specified through the MonitorFn property in CVodeSetOptions.
   If this property is not set, or if it is empty, MONFUN is not used.
   MONDATA is specified through the MonitorData property in CVodeSetOptions.

   See CVodeMonitor for an implementation example.
```

CVRhsFnB

Purpose

CVRhsFnB - type for user provided RHS function for backward problems.

Synopsis

This is a script file.

Description

```
CVRhsFnB - type for user provided RHS function for backward problems.

   The function ODEFUNB must be defined either as
        FUNCTION [YBD, FLAG] = ODEFUNB(T,Y,YB)
   or as
        FUNCTION [YBD, FLAG, NEW_DATA] = ODEFUNB(T,Y,YB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeInit. In either case, it must return the vector YBD
   corresponding to fB(t,y,yB).

   The function ODEFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also CVodeInitB
```

CVQuadRhsFnB

Purpose

CVQuadRhsFnB - type for user provided quadrature RHS function for backward problems

Synopsis

This is a script file.

Description

```
CVQuadRhsFnB - type for user provided quadrature RHS function for backward problems

   The function ODEQFUNB must be defined either as
        FUNCTION [YQBD, FLAG] = ODEQFUNB(T,Y,YB)
   or as
        FUNCTION [YQBD, FLAG, NEW_DATA] = ODEQFUNB(T,Y,YB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeInit. In either case, it must return the vector YQBD
```

corresponding to fQB(t,y,yB), the integrand for the integral to be
evaluated on the backward phase.

The function ODEQFUNB must set FLAG=0 if successful, FLAG<0 if an
unrecoverable failure occurred, or FLAG>0 if a recoverable error
occurred.

See also CVodeQuadInitB

CVDenseJacFnB

Purpose

CVDenseJacFnB - type for user provided dense Jacobian function for backward problems.

Synopsis

This is a script file.

Description

CVDenseJacFnB - type for user provided dense Jacobian function for backward problems.

The function DJACFUNB must be defined either as
    FUNCTION [JB, FLAG] = DJACFUNB(T, Y, YB, FYB)
or as
    FUNCTION [JB, FLAG, NEW_DATA] = DJACFUNB(T, Y, YB, FYB, DATA)
depending on whether a user data structure DATA was specified in
CVodeInit. In either case, it must return the matrix JB, the
Jacobian of fB(t,y,yB), with respect to yB. The input argument
FYB contains the current value of f(t,y,yB).

The function DJACFUNB must set FLAG=0 if successful, FLAG<0 if an
unrecoverable failure occurred, or FLAG>0 if a recoverable error
occurred.

See also CVodeSetOptions

NOTE: DJACFUNB is specified through the property JacobianFn to
CVodeSetOptions and is used only if the property LinearSolver
was set to 'Dense'.

CVBandJacFnB

Purpose

CVBandJacFnB - type for user provided banded Jacobian function for backward problems.

Synopsis

This is a script file.

Description

CVBandJacFnB - type for user provided banded Jacobian function for backward problems.

```
The function BJACFUNB must be defined either as
     FUNCTION [JB, FLAG] = BJACFUNB(T, Y, YB, FYB)
or as
     FUNCTION [JB, FLAG, NEW_DATA] = BJACFUNB(T, Y, YB, FYB, DATA)
depending on whether a user data structure DATA was specified in
CVodeInit. In either case, it must return the matrix JB, the
Jacobian of fB(t,y,yB), with respect to yB. The input argument
FYB contains the current value of f(t,y,yB).

The function BJACFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVodeSetOptions

See the CVODES user guide for more informaiton on the structure of
a banded Jacobian.

NOTE: BJACFUNB is specified through the property JacobianFn to
CVodeSetOptions and is used only if the property LinearSolver
was set to 'Band'.
```

CVJacTimesVecFnB

Purpose

CVJacTimesVecFnB - type for user provided Jacobian times vector function for backward problems.

Synopsis

This is a script file.

Description

CVJacTimesVecFnB - type for user provided Jacobian times vector function for backward problems.

```
The function JTVFUNB must be defined either as
     FUNCTION [JVB, FLAG] = JTVFUNB(T,Y,YB,FYB,VB)
or as
     FUNCTION [JVB, FLAG, NEW_DATA] = JTVFUNB(T,Y,YB,FYB,VB,DATA)
depending on whether a user data structure DATA was specified in
CVodeInit. In either case, it must return the vector JVB, the
product of the Jacobian of fB(t,y,yB) with respect to yB and a vector
vB. The input argument FYB contains the current value of f(t,y,yB).

The function JTVFUNB must set FLAG=0 if successful, or FLAG~=0 if
a failure occurred.

See also CVodeSetOptions

NOTE: JTVFUNB is specified through the property JacobianFn to
CVodeSetOptions and is used only if the property LinearSolver
was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

CVPrecSetupFnB

Purpose

CVPrecSetupFnB - type for user provided preconditioner setup function for backward problems.

Synopsis

This is a script file.

Description

CVPrecSetupFnB - type for user provided preconditioner setup function for backward problems.

```
   The user-supplied preconditioner setup function PSETFUN and
   the user-supplied preconditioner solve function PSOLFUN
   together must define left and right preconditoner matrices
   P1 and P2 (either of which may be trivial), such that the
   product P1*P2 is an approximation to the Newton matrix
   M = I - gamma*J.  Here J is the system Jacobian J = df/dy,
   and gamma is a scalar proportional to the integration step
   size h.  The solution of systems P z = r, with P = P1 or P2,
   is to be carried out by the PrecSolve function, and PSETFUN
   is to do any necessary setup operations.

   The user-supplied preconditioner setup function PSETFUN
   is to evaluate and preprocess any Jacobian-related data
   needed by the preconditioner solve function PSOLFUN.
   This might include forming a crude approximate Jacobian,
   and performing an LU factorization on the resulting
   approximation to M.  This function will not be called in
   advance of every call to PSOLFUN, but instead will be called
   only as often as necessary to achieve convergence within the
   Newton iteration.  If the PSOLFUN function needs no
   preparation, the PSETFUN function need not be provided.

   For greater efficiency, the PSETFUN function may save
   Jacobian-related data and reuse it, rather than generating it
   from scratch.  In this case, it should use the input flag JOK
   to decide whether to recompute the data, and set the output
   flag JCUR accordingly.

   Each call to the PSETFUN function is preceded by a call to
   ODEFUN with the same (t,y) arguments.  Thus the PSETFUN
   function can use any auxiliary data that is computed and
   saved by the ODEFUN function and made accessible to PSETFUN.


   The function PSETFUNB must be defined either as
        FUNCTION [JCURB, FLAG] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB)
   or as
        FUNCTION [JCURB, FLAG, NEW_DATA] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeInit. In either case, it must return the flags JCURB and FLAG.
```

```
See also CVPrecSolveFnB, CVodeSetOptions

NOTE: PSETFUNB is specified through the property PrecSetupFn to
CVodeSetOptions and is used only if the property LinearSolver was
set to 'GMRES', 'BiCGStab', or 'TFQMR' and if the property PrecType
is not 'None'.
```

CVPrecSolveFnB

Purpose

CVPrecSolveFnB - type for user provided preconditioner solve function for backward problems.

Synopsis

This is a script file.

Description

CVPrecSolveFnB - type for user provided preconditioner solve function for backward problems.

```
The user-supplied preconditioner solve function PSOLFN
is to solve a linear system P z = r in which the matrix P is
one of the preconditioner matrices P1 or P2, depending on the
type of preconditioning chosen.

The function PSOLFUNB must be defined either as
     FUNCTION [ZB, FLAG] = PSOLFUNB(T,Y,YB,FYB,RB)
or as
     FUNCTION [ZB, FLAG, NEW_DATA] = PSOLFUNB(T,Y,YB,FYB,RB,DATA)
depending on whether a user data structure DATA was specified in
CVodeInit. In either case, it must return the vector ZB and the
flag FLAG.

See also CVPrecSetupFnB, CVodeSetOptions

NOTE: PSOLFUNB is specified through the property PrecSolveFn to
CVodeSetOptions and is used only if the property LinearSolver was
set to 'GMRES', 'BiCGStab', or 'TFQMR' and if the property PrecType
is not 'None'.
```

CVGcommFnB

Purpose

CVGcommFn - type for user provided communication function (BBDPre) for backward problems.

Synopsis

This is a script file.

Description

CVGcommFn - type for user provided communication function (BBDPre) for backward problems.

```
The function GCOMFUNB must be defined either as
     FUNCTION FLAG = GCOMFUNB(T, Y, YB)
or as
     FUNCTION [FLAG, NEW_DATA] = GCOMFUNB(T, Y, YB, DATA)
depending on whether a user data structure DATA was specified in
CVodeInit.

The function GCOMFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVGlocalFnB, CVodeSetOptions

NOTES:
  GCOMFUNB is specified through the GcommFn property in CVodeSetOptions
  and is used only if the property PrecModule is set to 'BBDPre'.

  Each call to GCOMFUNB is preceded by a call to the RHS function
  ODEFUNB with the same arguments T, Y, and YB. Thus GCOMFUNB can
  omit any communication done by ODEFUNB if relevant to the evaluation
  of G by GLOCFUNB. If all necessary communication was done by ODEFUNB,
  GCOMFUNB need not be provided.
```

CVGlocalFnB

Purpose

CVGlocalFnB - type for user provided RHS approximation function (BBDPre) for backward problems.

Synopsis

This is a script file.

Description

CVGlocalFnB - type for user provided RHS approximation function (BBDPre) for backward problems.

```
The function GLOCFUNB must be defined either as
     FUNCTION [GLOCB, FLAG] = GLOCFUNB(T,Y,YB)
or as
     FUNCTION [GLOCB, FLAG, NEW_DATA] = GLOCFUNB(T,Y,YB,DATA)
depending on whether a user data structure DATA was specified in
CVodeInit. In either case, it must return the vector GLOCB
corresponding to an approximation to fB(t,y,yB).

The function GLOCFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also CVGcommFnB, CVodeSetOptions

NOTE: GLOCFUNB is specified through the GlocalFn property in CVodeSetOptions
and is used only if the property PrecModule is set to 'BBDPre'.
```

CVMonitorFnB

Purpose

CVMonitorFnB - type of user provided monitoring function for backward problems.

Synopsis

This is a script file.

Description

CVMonitorFnB - type of user provided monitoring function for backward problems.

```
   The function MONFUNB must be defined as
       FUNCTION [] = MONFUNB(CALL, IDXB, T, Y, YQ)
   It is called after every internal CVodeB step and can be used to
   monitor the progress of the solver. MONFUNB is called with CALL=0
   from CVodeInitB at which time it should initialize itself and it
   is called with CALL=2 from CVodeFree. Otherwise, CALL=1.

   It receives as arguments the index of the backward problem (as
   returned by CVodeInitB), the current time T, solution vector Y,
   and, if it was computed, the quadrature vector YQ. If quadratures
   were not computed for this backward problem, YQ is empty here.

   If additional data is needed inside MONFUNB, it must be defined
   as
      FUNCTION NEW_MONDATA = MONFUNB(CALL, IDXB, T, Y, YQ, MONDATA)
   If the local modifications to the user data structure need to be
   saved (e.g. for future calls to MONFUNB), then MONFUNB must set
   NEW_MONDATA. Otherwise, it should set NEW_MONDATA=[]
   (do not set NEW_MONDATA = DATA as it would lead to unnecessary copying).

   A sample monitoring function, CVodeMonitorB, is provided with CVODES.

   See also CVodeSetOptions, CVodeMonitorB

   NOTES:

   MONFUNB is specified through the MonitorFn property in CVodeSetOptions.
   If this property is not set, or if it is empty, MONFUNB is not used.
   MONDATA is specified through the MonitorData property in CVodeSetOptions.

   See CVodeMonitorB for an implementation example.
```

4 MATLAB Interface to IDAS

The MATLAB interface to IDAS provides access to all functionality of the IDAS solver, including DAE simulation and sensitvity analysis (both forward and adjoint).

The interface consists of 9 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions are listed in Tables 5, 6, and 7 for IVP solution, forward sensitivity analysis (FSA), and adjoint sensitivity analysis (ASA), respectively. For completness, some functions appear in more than one table. The types of user-supplied functions are listed in Table 8. All these functions are fully documented later in this section. For more in depth details, consult also the IDAS user guide [4].

To illustrate the use of the IDAS MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with IDAS.

Table 5: IDAS MATLAB interface functions for DAE integration

IDASetOptions	create an options structure for an DAE problem.	42
IDAQuadSetOptions	create an options structure for quadrature integration.	46
IDAInit	allocate and initialize memory for IDAS.	49
IDAQuadInit	allocate and initialize memory for quadrature integration.	49
IDAReInit	reinitialize memory for IDAS.	52
IDAQuadReInit	reinitialize memory for quadrature integration.	52
IDACalcIC	compute consistent initial conditions.	54
IDASolve	integrate the DAE problem.	56
IDAGetStats	return statistics for the IDAS solver.	58
IDAGet	extract data from IDAS memory.	61
IDAFree	deallocate memory for the IDAS solver.	63
IDAMonitor	monitoring function.	120

Table 6: IDAS MATLAB interface functions for FSA

IDASetOptions	create an options structure for an DAE problem.	42
IDAQuadSetOptions	create an options structure for quadrature integration.	46
IDASensSetOptions	create an options structure for FSA.	47
IDAInit	allocate and initialize memory for IDAS.	49
IDAQuadInit	allocate and initialize memory for quadrature integration.	49
IDASensInit	allocate and initialize memory for FSA.	50
IDAReInit	reinitialize memory for IDAS.	52
IDAQuadReInit	reinitialize memory for quadrature integration.	52
IDASensReInit	reinitialize memory for FSA.	53
IDASensToggleOff	temporarily deactivates FSA.	58
IDACalcIC	compute consistent initial conditions.	54
IDASolve	integrate the DAE problem.	56
IDAGetStats	return statistics for the IDAS solver.	58
IDAGet	extract data from IDAS memory.	61
IDAFree	deallocate memory for the IDAS solver.	63
IDAMonitor	monitoring function.	120

Table 7: IDAS MATLAB interface functions for ASA

IDASetOptions	create an options structure for an DAE problem.	42
IDAQuadSetOptions	create an options structure for quadrature integration.	46
IDAInit	allocate and initialize memory for the forward problem.	49
IDAQuadInit	allocate and initialize memory for forward quadrature integration.	49
IDAQuadReInit	reinitialize memory for forward quadrature integration.	52
IDAReInit	reinitialize memory for the forward problem.	52
IDAAdjInit	allocate and initialize memory for ASA.	50
IDAInitB	allocate and initialize a backward problem.	51
IDAAdjReInit	reinitialize memory for ASA.	53
IDAReInitB	reinitialize a backward problem.	53
IDACalcIC	compute consistent initial conditions.	54
IDACalcICB	compute consistent initial conditions for the backward problem.	56
IDASolve	integrate the forward DAE problem.	56
IDASolveB	integrate the backward problems.	57
IDAGetStats	return statistics for the integration of the forward problem.	58
IDAGetStatsB	return statistics for the integration of a backward problem.	60
IDAGet	extract data from IDAS memory.	61
IDAFree	deallocate memory for the IDAS solver.	63
IDAMonitor	monitoring function for forward problem.	120
IDAMonitorB	monitoring function for backward problems.	135

4.1 Interface functions

IDASetOptions

PURPOSE

`IDASetOptions creates an options structure for IDAS.`

SYNOPSIS

`function options = IDASetOptions(varargin)`

DESCRIPTION

```
IDASetOptions creates an options structure for IDAS.

   Usage: OPTIONS = IDASetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = IDASetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = IDASetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a IDAS options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = IDASetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.
```

Table 8: IDAS MATLAB function types

Forward problems	IDARhsFn	residual function	64
	IDARootFn	root-finding function	65
	IDAQuadRhsFn	quadrature RHS function	65
	IDASensRhsFn	sensitivity RHS function	64
	IDADenseJacFn	dense Jacobian function	66
	IDABandJacFn	banded Jacobian function	67
	IDAJacTimesVecFn	Jacobian times vector function	67
	IDAPrecSetupFn	preconditioner setup function	68
	IDAPrecSolveFn	preconditioner solve function	69
	IDAGlocalFn	residual approximation function (BBDPre)	71
	IDAGcommFn	communication function (BBDPre)	70
	IDAMonitorFn	monitoring function	71
Backward problems	IDARhsFnB	residual function	73
	IDAQuadRhsFnB	quadrature RHS function	73
	IDADenseJacFnB	dense Jacobian function	74
	IDABandJacFnB	banded Jacobian function	74
	IDAJacTimesVecFnB	Jacobian times vector function	75
	IDAPrecSetupFnB	preconditioner setup function	76
	IDAPrecSolveFnB	preconditioner solve function	76
	IDAGlocalFnB	residual approximation function (BBDPre)	77
	IDAGcommFnB	communication function (BBDPre)	77
	IDAMonitorFnB	monitoring function	78

```
   IDASetOptions with no input arguments displays all property names
   and their possible values.

IDASetOptions properties
(See also the IDAS User Guide)

UserData - User data passed unmodified to all functions [ empty ]
   If UserData is not empty, all user provided functions will be
   passed the problem data as their last input argument. For example,
   the RES function must be defined as R = DAEFUN(T,YY,TP,DATA).

RelTol - Relative tolerance [ positive scalar | 1e-4 ]
   RelTol defaults to 1e-4 and is applied to all components of the solution
   vector. See AbsTol.
AbsTol - Absolute tolerance [ positive scalar or vector | 1e-6 ]
   The relative and absolute tolerances define a vector of error weights
   with components
     ewt(i) = 1/(RelTol*|y(i)| + AbsTol)    if AbsTol is a scalar
     ewt(i) = 1/(RelTol*|y(i)| + AbsTol(i)) if AbsTol is a vector
   This vector is used in all error and convergence tests, which
   use a weighted RMS norm on all error-like vectors v:
     WRMSnorm(v) = sqrt( (1/N) sum(i=1..N) (v(i)*ewt(i))^2 ),
   where N is the problem dimension.
MaxNumSteps - Maximum number of steps [positive integer | 500]
   IDASolve will return with an error after taking MaxNumSteps internal steps
   in its attempt to reach the next output time.
```

```
InitialStep - Suggested initial stepsize [ positive scalar ]
   By default, IDASolve estimates an initial stepsize h0 at the initial time
   t0 as the solution of
     WRMSnorm(h0^2 ydd / 2) = 1
   where ydd is an estimated second derivative of y(t0).
MaxStep - Maximum stepsize [ positive scalar | inf ]
   Defines an upper bound on the integration step size.
MaxOrder - Maximum method order [ 1-5 for BDF | 5 ]
   Defines an upper bound on the linear multistep method order.
StopTime - Stopping time [ scalar ]
   Defines a value for the independent variable past which the solution
   is not to proceed.
RootsFn - Rootfinding function [ function ]
   To detect events (roots of functions), set this property to the event
   function. See IDARootFn.
NumRoots - Number of root functions [ integer | 0 ]
   Set NumRoots to the number of functions for which roots are monitored.
   If NumRoots is 0, rootfinding is disabled.

SuppressAlgVars - Suppres algebraic vars. from error test [ on | off ]
VariableTypes - Alg./diff. variables [ vector ]
ConstraintTypes - Simple bound constraints [ vector ]

LinearSolver - Linear solver type [Dense|Band|GMRES|BiCGStab|TFQMR]
   Specifies the type of linear solver to be used for the Newton nonlinear
   solver. Valid choices are: Dense (direct, dense Jacobian), Band (direct,
   banded Jacobian), GMRES (iterative, scaled preconditioned GMRES),
   BiCGStab (iterative, scaled preconditioned stabilized BiCG), TFQMR
   (iterative, scaled transpose-free QMR).
   The GMRES, BiCGStab, and TFQMR are matrix-free linear solvers.
JacobianFn - Jacobian function [ function ]
   This propeerty is overloaded. Set this value to a function that returns
   Jacobian information consistent with the linear solver used (see Linsolver).
   If not specified, IDAS uses difference quotient approximations.
   For the Dense linear solver, JacobianFn must be of type IDADenseJacFn and
   must return a dense Jacobian matrix. For the Band linear solver, JacobianFn
   must be of type IDABandJacFn and must return a banded Jacobian matrix.
   For the iterative linear solvers, GMRES, BiCGStab, and TFQMR, JacobianFn must
   be of type IDAJacTimesVecFn and must return a Jacobian-vector product.
KrylovMaxDim - Maximum number of Krylov subspace vectors [ integer | 5 ]
   Specifies the maximum number of vectors in the Krylov subspace. This property
   is used only if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver).
GramSchmidtType - Gram-Schmidt orthogonalization [ Classical | Modified ]
   Specifies the type of Gram-Schmidt orthogonalization (classical or modified).
   This property is used only if the GMRES linear solver is used (see LinSolver).
PrecModule - Preconditioner module [ BBDPre | UserDefined ]
   If PrecModule = 'UserDefined', then the user must provide at least a
   preconditioner solve function (see PrecSolveFn)
   IDAS provides one general-purpose preconditioner module, BBDPre, which can
   be only used with parallel vectors. It provide a preconditioner matrix that
   is block-diagonal with banded blocks. The blocking corresponds to the
   distribution of the dependent variable vector y among the processors.
   Each preconditioner block is generated from the Jacobian of the local part
```

```
   (on the current processor) of a given function g(t,y,yp) approximating
   f(t,y,yp) (see GlocalFn). The blocks are generated by a difference quotient
   scheme on each processor independently. This scheme utilizes an assumed
   banded structure with given half-bandwidths, mldq and mudq (specified through
   LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian
   block kept by the scheme has half-bandwiths ml and mu (specified through
   LowerBwidth and UpperBwidth), which may be smaller.
PrecSetupFn - Preconditioner setup function [ function ]
   If PrecType is not 'None', PrecSetupFn specifies an optional function which,
   together with PrecSolve, defines the preconditioner matrix, which must be an
   aproximation to the Newton matrix. PrecSetupFn must be of type IDAPrecSetupFn.
PrecSolveFn - Preconditioner solve function [ function ]
   If PrecType is not 'None', PrecSolveFn specifies a required function which
   must solve a linear system Pz = r, for given r. PrecSolveFn must be of type
   IDAPrecSolveFn.
GlocalFn - Local residual approximation function for BBDPre [ function ]
   If PrecModule is BBDPre, GlocalFn specifies a required function that
   evaluates a local approximation to the DAE residual. GlocalFn must
   be of type IDAGlocFn.
GcommFn - Inter-process communication function for BBDPre [ function ]
   If PrecModule is BBDPre, GcommFn specifies an optional function
   to perform any inter-process communication required for the evaluation of
   GlocalFn. GcommFn must be of type IDAGcommFn.
LowerBwidth - Jacobian/preconditioner lower bandwidth [ integer | 0 ]
   This property is overloaded. If the Band linear solver is used (see LinSolver),
   it specifies the lower half-bandwidth of the band Jacobian approximation.
   If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver) and if the BBDPre preconditioner module in IDAS is used
   (see PrecModule), it specifies the lower half-bandwidth of the retained
   banded approximation of the local Jacobian block.
   LowerBwidth defaults to 0 (no sub-diagonals).
UpperBwidth - Jacobian/preconditioner upper bandwidth [ integer | 0 ]
   This property is overloaded. If the Band linear solver is used (see LinSolver),
   it specifies the upper half-bandwidth of the band Jacobian approximation.
   If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver) and if the BBDPre preconditioner module in IDAS is used
   (see PrecModule), it specifies the upper half-bandwidth of the retained
   banded approximation of the local Jacobian block.
   UpperBwidth defaults to 0 (no super-diagonals).
LowerBwidthDQ - BBDPre preconditioner DQ lower bandwidth [ integer | 0 ]
   Specifies the lower half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).
UpperBwidthDQ - BBDPre preconditioner DQ upper bandwidth [ integer | 0 ]
   Specifies the upper half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).

MonitorFn - User-provied monitoring function [ function ]
   Specifies a function that is called after each successful integration step.
   This function must have type IDAMonitorFn or IDAMonitorFnB, depending on
   whether these options are for a forward or a backward problem, respectively.
   Sample monitoring functions IDAMonitor and IDAMonitorB are provided
   with IDAS.
MonitorData - User-provied data for the monitoring function [ struct ]
   Specifies a data structure that is passed to the MonitorFn function every time
```

```
   it is called.

SensDependent - Backward problem depending on sensitivities [ false | true ]
   Specifies whether the backward problem right-hand side depends on
   forward sensitivites. If TRUE, the residual function provided for
   this backward problem must have the appropriate type (see IDAResFnB).

ErrorMessages - Post error/warning messages [ true | false ]
   Note that any errors in IDAInit will result in a Matlab error, thus
   stoping execution. Only subsequent calls to IDAS functions will respect
   the value specified for 'ErrorMessages'.

NOTES:

   The properties listed above that can only be used for forward problems
   are: ConstraintTypes, StopTime, RootsFn, and NumRoots.

   The property SensDependent is relevant only for backward problems.

   See also
        IDAInit, IDAReInit, IDAInitB, IDAReInitB
        IDAResFn, IDARootFn
        IDADenseJacFn, IDABandJacFn, IDAJacTimesVecFn
        IDAPrecSetupFn, IDAPrecSolveFn
        IDAGlocalFn, IDAGcommFn
        IDAMonitorFn
        IDAResFnB
        IDADenseJacFnB, IDABandJacFnB, IDAJacTimesVecFnB
        IDAPrecSetupFnB, IDAPrecSolveFnB
        IDAGlocalFnB, IDAGcommFnB
        IDAMonitorFnB
```

IDAQuadSetOptions

Purpose

IDAQuadSetOptions creates an options structure for IDAS.

Synopsis

```
function options = IDAQuadSetOptions(varargin)
```

Description

IDAQuadSetOptions creates an options structure for IDAS.

```
   Usage: OPTIONS = IDAQuadSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = IDAQuadSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = IDAQuadSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   an IDAS options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.
```

```
   OPTIONS = IDAQuadSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.

   IDAQuadSetOptions with no input arguments displays all property names
   and their possible values.

IDAQuadSetOptions properties
(See also the IDAS User Guide)

ErrControl - Error control strategy for quadrature variables [ on | off ]
   Specifies whether quadrature variables are included in the error test.
RelTol - Relative tolerance for quadrature variables [ scalar 1e-4 ]
   Specifies the relative tolerance for quadrature variables. This parameter is
   used only if QuadErrCon=on.
AbsTol - Absolute tolerance for quadrature variables [ scalar or vector 1e-6 ]
   Specifies the absolute tolerance for quadrature variables. This parameter is
   used only if QuadErrCon=on.

SensDependent - Backward problem depending on sensitivities [ false | true ]
   Specifies whether the backward problem quadrature right-hand side depends
   on forward sensitivites. If TRUE, the right-hand side function provided for
   this backward problem must have the appropriate type (see IDAQuadRhsFnB).

   See also
        IDAQuadInit, IDAQuadReInit.
        IDAQuadInitB, IDAQuadReInitB
```

IDASensSetOptions

Purpose

IDASensSetOptions creates an options structure for FSA with IDAS.

Synopsis

```
function options = IDASensSetOptions(varargin)
```

Description

```
IDASensSetOptions creates an options structure for FSA with IDAS.

   Usage: OPTIONS = IDASensSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = IDASensSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)

   OPTIONS = IDASensSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a IDAS options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = IDASensSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.
```

```
   IDASensSetOptions with no input arguments displays all property names
   and their possible values.

IDASensSetOptions properties
(See also the IDAS User Guide)

method - FSA solution method [ 'Simultaneous' | 'Staggered' ]
   Specifies the FSA method for treating the nonlinear system solution for
   sensitivity variables. In the simultaneous case, the nonlinear systems
   for states and all sensitivities are solved simultaneously. In the
   Staggered case, the nonlinear system for states is solved first and then
   the nonlinear systems for all sensitivities are solved at the same time.
ParamField - Problem parameters  [ string ]
   Specifies the name of the field in the user data structure (specified through
   the 'UserData' field with IDASetOptions) in which the nominal values of the problem
   parameters are stored. This property is used only if  IDAS will use difference
   quotient approximations to the sensitivity residuals (see IDASensResFn).
ParamList - Parameters with respect to which FSA is performed [ integer vector ]
   Specifies a list of Ns parameters with respect to which sensitivities are to
   be computed. This property is used only if IDAS will use difference-quotient
   approximations to the sensitivity residuals. Its length must be Ns,
   consistent with the number of columns of yS0 (see IDASensInit).
ParamScales - Order of magnitude for problem parameters [ vector ]
   Provides order of magnitude information for the parameters with respect to
   which sensitivities are computed. This information is used if IDAS
   approximates the sensitivity residuals or if IDAS estimates integration
   tolerances for the sensitivity variables (see RelTol and AbsTol).
RelTol - Relative tolerance for sensitivity variables [ positive scalar ]
   Specifies the scalar relative tolerance for the sensitivity variables.
   See also AbsTol.
AbsTol - Absolute tolerance for sensitivity variables [ row-vector or matrix ]
   Specifies the absolute tolerance for sensitivity variables. AbsTol must be
   either a row vector of dimension Ns, in which case each of its components is
   used as a scalar absolute tolerance for the coresponding sensitivity vector,
   or a N x Ns matrix, in which case each of its columns is used as a vector
   of absolute tolerances for the corresponding sensitivity vector.
   By default, IDAS estimates the integration tolerances for sensitivity
   variables, based on those for the states and on the order of magnitude
   information for the problem parameters specified through ParamScales.
ErrControl - Error control strategy for sensitivity variables [ false | true ]
   Specifies whether sensitivity variables are included in the error control test.
   Note that sensitivity variables are always included in the nonlinear system
   convergence test.
DQtype - Type of DQ approx. of the sensi. RHS [Centered | Forward ]
   Specifies whether to use centered (second-order) or forward (first-order)
   difference quotient approximations of the sensitivity eqation residuals.
   This property is used only if a user-defined sensitivity residual function
   was not provided.
DQparam - Cut-off parameter for the DQ approx. of the sensi. RES [ scalar | 0.0 ]
   Specifies the value which controls the selection of the difference-quotient
   scheme used in evaluating the sensitivity residuals (switch between
   simultaneous or separate evaluations of the two components in the sensitivity
   right-hand side). The default value 0.0 indicates the use of simultaenous approximation
   exclusively (centered or forward, depending on the value of DQtype.
```

```
For DQparam &gt;= 1, IDAS uses a simultaneous approximation if the estimated
DQ perturbations for states and parameters are within a factor of DQparam,
and separate approximations otherwise. Note that a value DQparam &lt; 1
will inhibit switching! This property is used only if a user-defined sensitivity
residual function was not provided.

See also
     IDASensInit, IDASensReInit
```

IDAInit

Purpose

IDAInit allocates and initializes memory for IDAS.

Synopsis

function status = IDAInit(fct,t0,yy0,yp0,options)

Description

IDAInit allocates and initializes memory for IDAS.

```
Usage: IDAInit ( DAEFUN, T0, YY0, YP0 [, OPTIONS ] )

DAEFUN   is a function defining the DAE residual: f(t,yy,yp).
         This function must return a vector containing the current
         value of the residual.
T0       is the initial value of t.
YY0      is the initial condition vector y(t0).
YP0      is the initial condition vector y'(t0).
OPTIONS  is an (optional) set of integration options, created with
         the IDASetOptions function.

See also: IDASetOptions, IDAResFn
```

IDAQuadInit

Purpose

IDAQuadInit allocates and initializes memory for quadrature integration.

Synopsis

function status = IDAQuadInit(fctQ, yQ0, options)

Description

IDAQuadInit allocates and initializes memory for quadrature integration.

```
Usage: IDAQuadInit ( QFUN, YQ0 [, OPTIONS ] )

QFUN     is a function defining the righ-hand sides of the quadrature
         ODEs yQ' = fQ(t,y).
YQ0      is the initial conditions vector yQ(t0).
OPTIONS  is an (optional) set of QUAD options, created with
         the IDASetQuadOptions function.

See also: IDASetQuadOptions, IDAQuadRhsFn
```

IDASensInit

Purpose

IDASensInit allocates and initializes memory for FSA with IDAS.

Synopsis

```
function status = IDASensInit(Ns,fctS,yyS0,ypS0,options)
```

Description

IDASensInit allocates and initializes memory for FSA with IDAS.

```
Usage: IDASensInit ( NS, SFUN, YYS0, YPS0 [, OPTIONS ] )

NS       is the number of parameters with respect to which sensitivities
         are desired
SFUN     is a function defining the residual of the sensitivity DAEs
         fS(t,y,yp,yS,ypS).
YYS0, YPS0   Initial conditions for sensitivity variables.
         YYS0 and YPS0 must be matrices with N rows and Ns columns, where N is
         the problem dimension and Ns the number of sensitivity systems.
OPTIONS  is an (optional) set of FSA options, created with
         the IDASetFSAOptions function.

See also IDASensSetOptions, IDAInit, IDASensResFn
```

IDAAdjInit

Purpose

IDAAdjInit allocates and initializes memory for ASA with IDAS.

Synopsis

```
function status = IDAAdjInit(steps, interp)
```

Description

IDAAdjInit allocates and initializes memory for ASA with IDAS.

```
Usage: IDAAdjInit(STEPS, INTEPR)

STEPS    specifies the (maximum) number of integration steps between two
         consecutive check points.
INTERP   Specifies the type of interpolation used for estimating the forward
         solution during the backward integration phase. INTERP should be
         'Hermite', indicating cubic Hermite interpolation, or 'Polynomial',
         indicating variable order polynomial interpolation.
```

IDAInitB

PURPOSE

IDAInitB allocates and initializes backward memory for CVODES.

SYNOPSIS

function [idxB, status] = IDAInitB(fctB, tB0, yyB0, ypB0, optionsB)

DESCRIPTION

IDAInitB allocates and initializes backward memory for CVODES.

```
   Usage:   IDXB = IDAInitB ( DAEFUNB, TB0, YYB0, YPB0 [, OPTIONSB] )

   DAEFUNB  is a function defining the adjoint DAE: F(t,y,y',yB,yB')=0
            This function must return a vector containing the current
            value of the adjoint DAE residual.
   TB0      is the final value of t.
   YYB0     is the final condition vector yB(tB0).
   YPB0     is the final condition vector yB'(tB0).
   OPTIONSB is an (optional) set of integration options, created with
            the IDASetOptions function.

   IDAInitB returns the index IDXB associated with this backward
   problem. This index must be passed as an argument to any subsequent
   functions related to this backward problem.

   See also: IDASetOptions, IDAResFnB
```

IDAQuadInitB

PURPOSE

IDAQuadInitB allocates and initializes memory for backward quadrature integration.

SYNOPSIS

function status = IDAQuadInitB(idxB, fctQB, yQB0, optionsB)

DESCRIPTION

IDAQuadInitB allocates and initializes memory for backward quadrature integration.

```
   Usage: IDAQuadInitB ( IDXB, QBFUN, YQB0 [, OPTIONS ] )

   IDXB     is the index of the backward problem, returned by
            IDAInitB.
   QBFUN    is a function defining the righ-hand sides of the
            backward ODEs yQB' = fQB(t,y,yB).
   YQB0     is the final conditions vector yQB(tB0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the IDASetQuadOptions function.

   See also: IDAInitB, IDASetQuadOptions, IDAQuadRhsFnB
```

IDAReInit

PURPOSE

IDAReInit reinitializes memory for IDAS.

SYNOPSIS

```
function status = IDAReInit(t0,yy0,yp0,options)
```

DESCRIPTION

```
IDAReInit reinitializes memory for IDAS.
   where a prior call to IDAInit has been made with the same
   problem size N. IDAReInit performs the same input checking
   and initializations that IDAInit does, but it does no
   memory allocation, assuming that the existing internal memory
   is sufficient for the new problem.

   Usage: IDAReInit ( T0, YY0, YP0 [, OPTIONS ] )

   T0       is the initial value of t.
   YY0      is the initial condition vector y(t0).
   YP0      is the initial condition vector y'(t0).
   OPTIONS  is an (optional) set of integration options, created with
            the IDASetOptions function.

  See also: IDASetOptions, IDAInit
```

IDAQuadReInit

PURPOSE

IDAQuadReInit reinitializes IDAS's quadrature-related memory

SYNOPSIS

```
function status = IDAQuadReInit(yQ0, options)
```

DESCRIPTION

```
IDAQuadReInit reinitializes IDAS's quadrature-related memory
   assuming it has already been allocated in prior calls to IDAInit
   and IDAQuadInit.

   Usage: IDAQuadReInit ( YQ0 [, OPTIONS ] )

   YQ0      Initial conditions for quadrature variables yQ(t0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the IDASetQuadOptions function.

   See also: IDASetQuadOptions, IDAQuadInit
```

IDASensReInit

Purpose

IDASensReInit reinitializes IDAS's FSA-related memory

Synopsis

function status = IDASensReInit(yyS0,ypS0,options)

Description

```
IDASensReInit reinitializes IDAS's FSA-related memory
   assuming it has already been allocated in prior calls to IDAInit
   and IDASensInit.
   The number of sensitivities Ns is assumed to be unchanged since the
   previous call to IDASensInit.

   Usage: IDASensReInit ( YYS0, YPS0 [, OPTIONS ] )

   YYS0, YPS0   Initial conditions for sensitivity variables.
            YYS0 and YPS0 must be matrices with N rows and Ns columns, where N is
            the problem dimension and Ns the number of sensitivity systems.
   OPTIONS  is an (optional) set of FSA options, created with
            the IDASetFSAOptions function.

   See also: IDASensSetOptions, IDAReInit, IDASensInit
```

IDAAdjReInit

Purpose

IDAAdjReInit re-initializes memory for ASA with CVODES.

Synopsis

function status = IDAAdjReInit()

Description

```
IDAAdjReInit re-initializes memory for ASA with CVODES.

   Usage: IDAAdjReInit
```

IDAReInitB

Purpose

IDAReInitB allocates and initializes backward memory for IDAS.

Synopsis

function status = IDAReInitB(idxB,tB0,yyB0,ypB0,optionsB)

Description

```
IDAReInitB allocates and initializes backward memory for IDAS.
   where a prior call to IDAInitB has been made with the same
   problem size NB. IDAReInitB performs the same input checking
   and initializations that IDAInitB does, but it does no
   memory allocation, assuming that the existing internal memory
   is sufficient for the new problem.

   Usage:   IDAReInitB ( IDXB, TB0, YYB0, YPB0 [, OPTIONSB] )

   IDXB     is the index of the backward problem, returned by
            IDAInitB.
   TB0      is the final value of t.
   YYB0     is the final condition vector yB(tB0).
   YPB0     is the final condition vector yB'(tB0).
   OPTIONSB is an (optional) set of integration options, created with
            the IDASetOptions function.

   See also: IDASetOptions, IDAInitB
```

IDAQuadReInitB

PURPOSE

IDAQuadReInitB reinitializes memory for backward quadrature integration.

SYNOPSIS

```
function status = IDAQuadReInitB(idxB, yQB0, optionsB)
```

DESCRIPTION

```
IDAQuadReInitB reinitializes memory for backward quadrature integration.

   Usage: IDAQuadReInitB ( IDXB, YS0 [, OPTIONS ] )

   IDXB     is the index of the backward problem, returned by
            IDAInitB.
   YQB0     is the final conditions vector yQB(tB0).
   OPTIONS  is an (optional) set of QUAD options, created with
            the IDASetQuadOptions function.

   See also: IDASetQuadOptions, IDAReInitB, IDAQuadInitB
```

IDACalcIC

PURPOSE

IDACalcIC computes consistent initial conditions

SYNOPSIS

```
function [status, varargout] = IDACalcIC(tout,icmeth)
```

DESCRIPTION

```
IDACalcIC computes consistent initial conditions

   Usage: STATUS = IDACalcIC ( TOUT, ICMETH )
          [STATUS, YY0, YP0] = IDACalcIC ( TOUT, ICMETH )

   IDACalcIC corrects the guess for initial conditions passed
   to IDAInit or IDAReInit so that the algebraic constraints
   are satisfied.

   The argument TOUT is the first value of t at which a soluton will be
   requested (from IDASolve). This is needed here to determine the
   direction of integration and rough scale in the independent variable.

   If ICMETH is 'FindAlgebraic', then IDACalcIC attempts to compute
   the algebraic components of y and differential components of y',
   given the differential components of y.
   This option requires that the vector id was set through IDASetOptions
   specifying the differential and algebraic components.
   If ICMETH is 'FindAll', then IDACalcIC attempts to compute all
   components of y, given y'.  In this case, id is not required.

   On return, STATUS is one of the following:
 SUCCESS              IDACalcIC was successful.  The corrected
                      initial value vectors are in y0 and yp0.
 IDA_MEM_NULL         The argument ida_mem was NULL.
 IDA_ILL_INPUT        One of the input arguments was illegal.
                      See printed message.
 IDA_LINIT_FAIL       The linear solver's init routine failed.
 IDA_BAD_EWT          Some component of the error weight vector
                      is zero (illegal), either for the input
                      value of y0 or a corrected value.
 IDA_RES_FAIL         The user's residual routine returned
                      a non-recoverable error flag.
 IDA_FIRST_RES_FAIL   The user's residual routine returned
                      a recoverable error flag on the first call,
                      but IDACalcIC was unable to recover.
 IDA_LSETUP_FAIL      The linear solver's setup routine had a
                      non-recoverable error.
 IDA_LSOLVE_FAIL      The linear solver's solve routine had a
                      non-recoverable error.
 IDA_NO_RECOVERY      The user's residual routine, or the linear
                      solver's setup or solve routine had a
                      recoverable error, but IDACalcIC was
                      unable to recover.
 IDA_CONSTR_FAIL      IDACalcIC was unable to find a solution
                      satisfying the inequality constraints.
 IDA_LINESEARCH_FAIL The Linesearch algorithm failed to find a
                      solution with a step larger than steptol
                      in weighted RMS norm.
 IDA_CONV_FAIL        IDACalcIC failed to get convergence of the
                      Newton iterations.

   If the output arguments YY0 and YP0 are present, they will
   contain the consistent initial conditions.
```

```
  See also: IDASetOptions, IDAInit, IDAReInit
```

IDACalcICB

Purpose

IDACalcICB computes consistent initial conditions for the backward phase.

Synopsis

function [status, varargout] = IDACalcICB(tout,icmeth)

Description

IDACalcICB computes consistent initial conditions for the backward phase.

```
   Usage: STATUS = IDACalcICB ( TOUTB, ICMETHB )
          [STATUS, YY0B, YP0B] = IDACalcIC ( TOUTB, ICMETHB )

  See also: IDASetOptions, IDAInitB, IDAReInitB
```

IDASolve

Purpose

IDASolve integrates the DAE.

Synopsis

function [varargout] = IDASolve(tout,itask)

Description

IDASolve integrates the DAE.

```
   Usage: [STATUS, T, Y] = IDASolve ( TOUT, ITASK )
          [STATUS, T, Y, YQ] = IDASolve  (TOUT, ITASK )
          [STATUS, T, Y, YS] = IDASolve ( TOUT, ITASK )
          [STATUS, T, Y, YQ, YS] = IDASolve ( TOUT, ITASK )

   If ITASK is 'Normal', then the solver integrates from its current internal
   T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
   Y(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
   and returns in Y the solution at the new internal time. In this case, TOUT
   is used only during the first call to IDASolve to determine the direction of
   integration and the rough scale of the problem. In either case, the time
   reached by the solver is returned in T.

   If quadratures were computed (see IDAQuadInit), IDASolve will return their
   values at T in the vector YQ.

   If sensitivity calculations were enabled (see IDASensInit), IDASolve will
   return their values at T in the matrix YS. Each row in the matrix YS
```

```
represents the sensitivity vector with respect to one of the problem parameters.

In ITASK =' Normal' mode, to obtain solutions at specific times T0,T1,...,TFINAL
(all increasing or all decreasing) use TOUT = [T0 T1  ... TFINAL]. In this case
the output arguments Y and YQ are matrices, each column representing the solution
vector at the corresponding time returned in the vector T. If computed, the
sensitivities are eturned in the 3-dimensional array YS, with YS(:,:,I) representing
the sensitivity vectors at the time T(I).

On return, STATUS is one of the following:
  0: IDASolve succeeded and no roots were found.
  1: IDASolve succeded and returned at tstop.
  2: IDASolve succeeded, and found one or more roots.
 -1: An error occurred (see printed message).

See also IDASetOptions, IDAGetStats
```

IDASolveB

Purpose

IDASolveB integrates the backward DAE.

Synopsis

```
function [varargout] = IDASolveB(tout,itask)
```

Description

IDASolveB integrates the backward DAE.

```
Usage:  [STATUS, T, YB] = IDASolveB ( TOUT, ITASK )
        [STATUS, T, YB, YQB] = IDASolveB ( TOUT, ITASK )

If ITASK is 'Normal', then the solver integrates from its current internal
T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
YB(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
and returns in YB the solution at the new internal time. In this case, TOUT
is used only during the first call to IDASolveB to determine the direction of
integration and the rough scale of the problem. In either case, the time
reached by the solver is returned in T.

If quadratures were computed (see IDAQuadInitB), IDASolveB will return their
values at T in the vector YQB.

In ITASK =' Normal' mode, to obtain solutions at specific times T0,T1,...,TFINAL
(all increasing or all decreasing) use TOUT = [T0 T1  ... TFINAL]. In this case
the output arguments YB and YQB are matrices, each column representing the solution
vector at the corresponding time returned in the vector T.

If more than one backward problem was defined, the return arguments are cell
arrays, with TIDXB, YBIDXB, and YQBIDXB corresponding to the backward
problem with index IDXB (as returned by IDAInitB).
```

```
   On return, STATUS is one of the following:
     0: IDASolveB succeeded.
     1: IDASolveB succeded and return at a tstop value (internally set).
    -1: An error occurred (see printed message).

   See also IDASetOptions, IDAGetStatsB
```

IDASensToggleOff

PURPOSE

IDASensToggleOff deactivates sensitivity calculations.

SYNOPSIS

```
function status = IDASensToggleOff()
```

DESCRIPTION

```
 IDASensToggleOff deactivates sensitivity calculations.
   It does NOT deallocate sensitivity-related memory so that
   sensitivity computations can be later toggled ON (through
   IDASensReInit).

   Usage: IDASensToggleOff

   See also: IDASensInit, IDASensReInit
```

IDAGetStats

PURPOSE

IDAGetStats returns run statistics for the IDAS solver.

SYNOPSIS

```
function [si, status] = IDAGetStats()
```

DESCRIPTION

```
IDAGetStats returns run statistics for the IDAS solver.

   Usage: STATS = IDAGetStats

Fields in the structure STATS

o nst      - number of integration steps
o nre      - number of residual function evaluations
o nsetups  - number of linear solver setup calls
o netf     - number of error test failures
o nni      - number of nonlinear solver iterations
o ncfn     - number of convergence test failures
o qlast    - last method order used
o qcur     - current method order
```

```
o h0used   - actual initial step size used
o hlast    - last step size used
o hcur     - current step size
o tcur     - current time reached by the integrator
o RootInfo - strucutre with rootfinding information
o QuadInfo - structure with quadrature integration statistics
o LSInfo   - structure with linear solver statistics
o FSAInfo  - structure with forward sensitivity solver statistics

If rootfinding was requested, the structure RootInfo has the following fields

o nge   - number of calls to the rootfinding function
o roots - array of integers (a value of 1 in the i-th component means that the
          i-th rootfinding function has a root (upon a return with status=2 from
          IDASolve).

If quadratures were present, the structure QuadInfo has the following fields

o nfQe  - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

The structure LSinfo has different fields, depending on the linear solver used.

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nreD - number of residual function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nreB - number of residual function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

o name  - 'GMRES' or 'BiCGStab'
o nli   - number of linear solver iterations
o npe   - number of preconditioner setups
o nps   - number of preconditioner solve function calls
o ncfl  - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nreSG - number of residual function evaluations for difference-quotient
          Jacobian-vector product approximation

If forward sensitivities were computed, the structure FSAInfo has the
following fields

o nrSe      - number of sensitivity residual evaluations
o nreS      - number of residual evaluations for difference-quotient
              sensitivity residual approximation
```

```
o nsetupsS  - number of linear solver setups triggered by sensitivity variables
o netfS     - number of error test failures for sensitivity variables
o nniS      - number of nonlinear solver iterations for sensitivity variables
o ncfnS     - number of convergence test failures due to sensitivity variables
```

IDAGetStatsB

Purpose

IDAGetStatsB returns run statistics for the backward IDAS solver.

Synopsis

```
function [si, status] = IDAGetStatsB(idxB)
```

Description

```
IDAGetStatsB returns run statistics for the backward IDAS solver.

   Usage: STATS = IDAGetStatsB(IDXB)

   IDXB is the index of the backward problem, returned by IDAInitB.

Fields in the structure STATS

o nst - number of integration steps
o nre - number of residual function evaluations
o nsetups - number of linear solver setup calls
o netf - number of error test failures
o nni - number of nonlinear solver iterations
o ncfn - number of convergence test failures
o qlast - last method order used
o qcur - current method order
o h0used - actual initial step size used
o hlast - last step size used
o hcur - current step size
o tcur - current time reached by the integrator
o QuadInfo - structure with quadrature integration statistics
o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

If quadratures were present, the structure QuadInfo has the following fields

o nfQe - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nreD - number of residual function evaluations for difference-quotient
         Jacobian approximation
```

```
  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nreB - number of residual function evaluations for difference-quotient
       Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

o name - 'GMRES' or 'BiCGStab'
o nli - number of linear solver iterations
o npe - number of preconditioner setups
o nps - number of preconditioner solve function calls
o ncfl - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nreSG -  number of residual function evaluations for difference-quotient
        Jacobian-vector product approximation
```

IDAGet

PURPOSE

IDAGet extracts data from the IDAS solver memory.

SYNOPSIS

```
function [output, status] = IDAGet(key, varargin)
```

DESCRIPTION

IDAGet extracts data from the IDAS solver memory.

```
   Usage: RET = IDAGet ( KEY [, P1 [, P2] ... ])

   IDAGet returns internal IDAS information based on KEY. For some values
   of KEY, additional arguments may be required and/or more than one output is
   returned.

   KEY is a string and should be one of:
    o DerivSolution - Returns a vector containing the K-th order derivative
       of the solution at time T. The time T and order K must be passed through
       the input arguments P1 and P2, respectively:
       DKY = IDAGet('DerivSolution', T, K)
    o ErrorWeights - Returns a vector containing the current error weights.
       EWT = IDAGet('ErrorWeights')
    o CheckPointsInfo - Returns an array of structures with check point information.
       CK = IDAGet('CheckPointInfo)
```

IDASet

PURPOSE

IDASet changes optional input values during the integration.

SYNOPSIS

```
function status = IDASet(varargin)
```

DESCRIPTION

```
IDASet changes optional input values during the integration.

   Usage: IDASet('NAME1',VALUE1,'NAME2',VALUE2,...)

   IDASet can be used to change some of the optional inputs during
   the integration, i.e., without need for a solver reinitialization.
   The property names accepted by IDASet are a subset of those valid
   for IDASetOptions. Any unspecified properties are left unchanged.

   IDASet with no input arguments displays all property names.

IDASet properties
(See also the IDAS User Guide)

UserData - problem data passed unmodified to all user functions.
  Set VALUE to be the new user data.
RelTol - Relative tolerance
  Set VALUE to the new relative tolerance
AbsTol - absolute tolerance
  Set VALUE to be either the new scalar absolute tolerance or
  a vector of absolute tolerances, one for each solution component.
StopTime - Stopping time
  Set VALUE to be a new value for the independent variable past which
  the solution is not to proceed.
```

IDASetB

Purpose

IDASetB changes optional input values during the integration.

Synopsis

```
function status = IDASetB(idxB, varargin)
```

Description

```
IDASetB changes optional input values during the integration.

   Usage: IDASetB( IDXB, 'NAME1',VALUE1,'NAME2',VALUE2,... )

   IDASetB can be used to change some of the optional inputs for
   the backward problem identified by IDXB during the backward
   integration, i.e., without need for a solver reinitialization.
   The property names accepted by IDASet are a subset of those valid
   for IDASetOptions. Any unspecified properties are left unchanged.

   IDASetB with no input arguments displays all property names.

IDASetB properties
(See also the IDAS User Guide)

UserData - problem data passed unmodified to all user functions.
  Set VALUE to be the new user data.
```

```
RelTol - Relative tolerance
  Set VALUE to the new relative tolerance
AbsTol - absolute tolerance
  Set VALUE to be either the new scalar absolute tolerance or
  a vector of absolute tolerances, one for each solution component.
```

IDAFree

Purpose

IDAFree deallocates memory for the IDAS solver.

Synopsis

```
function [] = IDAFree()
```

Description

IDAFree deallocates memory for the IDAS solver.

```
Usage:  IDAFree
```

4.2 Function types

IDAResFn

Purpose

IDAResFn - type for residual function

Synopsis

This is a script file.

Description

IDAResFn - type for residual function

```
   The function DAEFUN must be defined as
        FUNCTION [R, FLAG] = DAEFUN(T, YY, YP)
   and must return a vector R corresponding to f(t,yy,yp).
   If a user data structure DATA was specified in IDAInit, then
   DAEFUN must be defined as
        FUNCTION [R, FLAG, NEW_DATA] = DAEFUN(T, YY, YP, DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector YD,
   the DAEFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function DAEFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDAInit
```

IDASensResFn

Purpose

IDASensRhsFn - type for user provided sensitivity RHS function.

Synopsis

This is a script file.

Description

IDASensRhsFn - type for user provided sensitivity RHS function.

```
   The function DAESFUN must be defined as
        FUNCTION [RS, FLAG] = DAESFUN(T,YY,YP,YYS,YPS)
   and must return a matrix RS corresponding to fS(t,yy,yp,yyS,ypS).
   If a user data structure DATA was specified in IDAInit, then
   DAESFUN must be defined as
        FUNCTION [RS, FLAG, NEW_DATA] = DAESFUN(T,YY,YP,YYS,YPS,DATA)
```

```
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix YSD,
   the ODESFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function DAESFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDASetFSAOptions

   NOTE: DAESFUN is specified through the property FSAResFn to
         IDASetFSAOptions.
```

IDAQuadRhsFn

Purpose

IDAQuadRhsFn - type for user provided quadrature RHS function.

Synopsis

This is a script file.

Description

IDAQuadRhsFn - type for user provided quadrature RHS function.

```
   The function QFUN must be defined as
        FUNCTION [YQD, FLAG] = QFUN(T, YY, YP)
   and must return a vector YQD corresponding to fQ(t,yy,yp), the
   integrand for the integral to be evaluated.
   If a user data structure DATA was specified in IDAInit, then
   QFUN must be defined as
        FUNCTION [YQD, FLAG, NEW_DATA] = QFUN(T, YY, YP, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector YQD,
   the QFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function QFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDAQuadInit
```

IDARootFn

Purpose

IDARootFn - type for user provided root-finding function.

Synopsis

This is a script file.

Description

IDARootFn - type for user provided root-finding function.

```
   The function ROOTFUN must be defined as
        FUNCTION [G, FLAG] = ROOTFUN(T,YY,YP)
   and must return a vector G corresponding to g(t,yy,yp).
   If a user data structure DATA was specified in IDAInit, then
   ROOTFUN must be defined as
        FUNCTION [G, FLAG, NEW_DATA] = ROOTFUN(T,YY,YP,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector G,
   the ROOTFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function ROOTFUN must set FLAG=0 if successful, or FLAG~=0 if
   a failure occurred.

   See also IDASetOptions

   NOTE: ROOTFUN is specified through the RootsFn property in
   IDASetOptions and is used only if the property NumRoots is a
   positive integer.
```

IDADenseJacFn

Purpose

IDADenseJacFn - type for dense Jacobian function.

Synopsis

This is a script file.

Description

IDADenseJacFn - type for dense Jacobian function.

```
   The function DJACFUN must be defined as
        FUNCTION [J, FLAG] = DJACFUN(T, YY, YP, RR, CJ)
   and must return a matrix J corresponding to the Jacobian
   (df/dyy + cj*df/dyp).
   The input argument RR contains the current value of f(t,yy,yp).
   If a user data structure DATA was specified in IDAInit, then
   DJACFUN must be defined as
        FUNCTION [J, FLAG, NEW_DATA] = DJACFUN(T, YY, YP, RR, CJ, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J,
   the DJACFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function DJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.
```

```
   See also IDASetOptions

   NOTE: DJACFUN is specified through the property JacobianFn to
   IDASetOptions and is used only if the property LinearSolver
   was set to 'Dense'.
```

IDABandJacFn

Purpose

IDABandJacFn - type for banded Jacobian function.

Synopsis

This is a script file.

Description

IDABandJacFn - type for banded Jacobian function.

```
   The function BJACFUN must be defined as
        FUNCTION [J, FLAG] = BJACFUN(T, YY, YP, RR, CJ)
   and must return a matrix J corresponding to the banded Jacobian
   (df/dyy + cj*df/dyp).
   The input argument RR contains the current value of f(t,yy,yp).
   If a user data structure DATA was specified in IDAInit, then
   BJACFUN must be defined as
        FUNCTION [J, FLAG, NEW_DATA] = BJACFUN(T, YY, YP, RR, CJ, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J,
   the BJACFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function BJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDASetOptions

   See the IDAS user guide for more information on the structure of
   a banded Jacobian.

   NOTE: BJACFUN is specified through the property JacobianFn to
   IDASetOptions and is used only if the property LinearSolver
   was set to 'Band'.
```

IDAJacTimesVecFn

Purpose

IDAJacTimesVecFn - type for Jacobian times vector function.

Synopsis

This is a script file.

Description

```
IDAJacTimesVecFn - type for Jacobian times vector function.

   The function JTVFUN must be defined as
        FUNCTION [JV, FLAG] = JTVFUN(T,YY,YP,RR,V,CJ)
   and must return a vector JV corresponding to the product of the
   Jacobian ( df/dyy + cj * df/dyp ) with the vector v.
   The input argument RR contains the current value of f(t,yy,yp).
   If a user data structure DATA was specified in IDAInit, then
   JTVFUN must be defined as
        FUNCTION [JV, FLAG, NEW_DATA] = JTVFUN(T,YY,YP,RR,V,CJ,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector JV,
   the JTVFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function JTVFUN must set FLAG=0 if successful, or FLAG~=0 if
   a failure occurred.

   See also IDASetOptions

   NOTE: JTVFUN is specified through the property JacobianFn to
   IDASetOptions and is used only if the property LinearSolver
   was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

IDAPrecSetupFn

Purpose

IDAPrecSetupFn - type for preconditioner setup function.

Synopsis

This is a script file.

Description

```
IDAPrecSetupFn - type for preconditioner setup function.

   The user-supplied preconditioner setup function PSETFUN and
   the user-supplied preconditioner solve function PSOLFUN
   together must define a preconditoner matrix P which is an
   approximation to the Newton matrix M = J_yy - cj*J_yp.
   Here J_yy = df/dyy, J_yp = df/dyp, and cj is a scalar proportional
   to the integration step size h.  The solution of systems P z = r,
   is to be carried out by the PrecSolve function, and PSETFUN
   is to do any necessary setup operations.

   The user-supplied preconditioner setup function PSETFUN
   is to evaluate and preprocess any Jacobian-related data
   needed by the preconditioner solve function PSOLFUN.
   This might include forming a crude approximate Jacobian,
   and performing an LU factorization on the resulting
   approximation to M.  This function will not be called in
```

```
advance of every call to PSOLFUN, but instead will be called
only as often as necessary to achieve convergence within the
Newton iteration.  If the PSOLFUN function needs no
preparation, the PSETFUN function need not be provided.

Each call to the PSETFUN function is preceded by a call to
DAEFUN with the same (t,yy,yp) arguments.  Thus the PSETFUN
function can use any auxiliary data that is computed and
saved by the DAEFUN function and made accessible to PSETFUN.

The function PSETFUN must be defined as
     FUNCTION FLAG = PSETFUN(T,YY,YP,RR,CJ)
If successful, it must return FLAG=0. For a recoverable error (in
which case the setup will be retried) it must set FLAG to a positive
integer value. If an unrecoverable error occurs, it must set FLAG
to a negative value, in which case the integration will be halted.
The input argument RR contains the current value of f(t,yy,yp).

If a user data structure DATA was specified in IDASetUserData, then
PSETFUN must be defined as
     FUNCTION [FLAG,NEW_DATA] = PSETFUN(T,YY,YP,RR,CJ,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the flag
FLAG, the PSETFUN function must also set NEW_DATA. Otherwise, it
should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
to unnecessary copying).

See also IDAPrecSolveFn, IDASetOptions

NOTE: PSETFUN and PSETFUNB are specified through the property
PrecSetupFn to IDASetOptions and are used only if the property
LinearSolver was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

IDAPrecSolveFn

Purpose

IDAPrecSolveFn - type for preconditioner solve function.

Synopsis

This is a script file.

Description

IDAPrecSolveFn - type for preconditioner solve function.

```
The user-supplied preconditioner solve function PSOLFUN
is to solve a linear system P z = r, where P is the
preconditioner matrix.

The function PSOLFUN must be defined as
     FUNCTION [Z, FLAG] = PSOLFUN(T,YY,YP,RR,R)
and must return a vector Z containing the solution of Pz=r.
```

If PSOLFUN was successful, it must return FLAG=0. For a recoverable error (in which case the step will be retried) it must set FLAG to a positive value. If an unrecoverable error occurs, it must set FLAG to a negative value, in which case the integration will be halted. The input argument RR contains the current value of f(t,yy,yp).

If a user data structure DATA was specified in IDAInit, then PSOLFUN must be defined as

```
     FUNCTION [Z, FLAG, NEW_DATA] = PSOLFUN(T,YY,YP,RR,R,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector Z and the flag FLAG, the PSOLFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

See also IDAPrecSetupFn, IDASetOptions

NOTE: PSOLFUN and PSOLFUNB are specified through the property PrecSolveFn to IDASetOptions and are used only if the property LinearSolver was set to 'GMRES', 'BiCGStab', or 'TFQMR'.

IDAGcommFn

Purpose

IDAGcommFn - type for communication function (BBDPre).

Synopsis

This is a script file.

Description

IDAGcommFn - type for communication function (BBDPre).

The function GCOMFUN must be defined as

```
     FUNCTION FLAG = GCOMFUN(T, YY, YP)
```

and can be used to perform all interprocess communication necessary to evaluate the approximate residual function for the BBDPre preconditioner module.
If a user data structure DATA was specified in IDAInit, then GCOMFUN must be defined as

```
     FUNCTION [FLAG, NEW_DATA] = GCOMFUN(T, YY, YP, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then the GCOMFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function GCOMFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also IDAGlocalFn, IDASetOptions

```
NOTES:
  GCOMFUN is specified through the GcommFn property in IDASetOptions
  and is used only if the property PrecModule is set to 'BBDPre'.

  Each call to GCOMFUN is preceded by a call to the residual function
  DAEFUN with the same arguments T, YY, and YP.
  Thus GCOMFUN can omit any communication done by DAEFUN if relevant
  to the evaluation of G by GLOCFUN. If all necessary communication
  was done by DAEFUN, GCOMFUN need not be provided.
```

IDAGlocalFn

PURPOSE

IDAGlocalFn - type for RES approximation function (BBDPre).

SYNOPSIS

This is a script file.

DESCRIPTION

IDAGlocalFn - type for RES approximation function (BBDPre).

```
The function GLOCFUN must be defined as
     FUNCTION [GLOC, FLAG] = GLOCFUN(T,YY,YP)
and must return a vector GLOC corresponding to an approximation to f(t,yy,yp)
which will be used in the BBDPRE preconditioner module. The case where
G is mathematically identical to F is allowed.
If a user data structure DATA was specified in IDAInit, then
GLOCFUN must be defined as
     FUNCTION [GLOC, FLAG, NEW_DATA] = GLOCFUN(T,YY,YP,DATA)
If the local modifications to the user data structure are needed
in other user-provided functions then, besides setting the vector G,
the GLOCFUN function must also set NEW_DATA. Otherwise, it should set
NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

The function GLOCFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also IDAGcommFn, IDASetOptions

NOTE: GLOCFUN and GLOCFUNB are specified through the GlocalFn property
in IDASetOptions and are used only if the property PrecModule
is set to 'BBDPre'.
```

IDAMonitorFn

PURPOSE

IDAMonitorFn - type for monitoring function.

SYNOPSIS

This is a script file.

DESCRIPTION

```
IDAMonitorFn - type for monitoring function.

   The function MONFUN must be defined as
      FUNCTION [] = MONFUN(CALL, T, YY, YP, YQ, YYS, YPS)

   To enable monitoring using a given monitor function MONFUN,
   use IDASetOptions to set the property ’MonitorFn&quot; to ’MONFUN’
   (or to @MONFUN).

   MONFUN is called with the following input arguments:

   o CALL indicates the phase during the integration process at which
     MONFUN is called:
     CALL=1 : MONFUN was called at the initial time; this can be either
              after IDAInit or after IDAReInit.
              (typically, MONFUN should perform its own initialization)
     CALL=2 : MONFUN was called right before a solver reinitializtion.
              (typically, MONFUN should decide whether to initialize
              itself or else to continue monitoring)
     CALL=3 : MONFUN was called during solver finalization.
              (typically, MONFUN should finalize monitoring)
     CALL=0 : MONFUN was called after the solver took a successful
              internal step.
              (typically, MONFUN should collect and/or display data)

   o T is the current integration time

   o YY and YP are vectors containing the solution and solution
     derivative at time T

   o YQ is a vector containing the quadrature variables at time T

   o YYS and YPS are matrices containing the forward sensitivities
     and their derivatives, respectively, at time T.

   If additional data is needed inside a MONFUN function, then it must
   be defined as
      FUNCTION NEW_MONDATA = MONFUN(CALL, T, YY, YP, YQ, YYS, YPS, MONDATA)

   In this case, the MONFUN function is passed the additional argument
   MONDATA, the same as that specified through the property ’MonitorData’
   in IDASetOptions. If the local modifications to the monitor data structure
   need to be saved (e.g. for future calls to MONFUN), then MONFUN must set
   NEW_MONDATA. Otherwise, it should set NEW_MONDATA=[] (do not set
   NEW_MONDATA = DATA as it would lead to unnecessary copying).

   NOTES:

   1. MONFUN is specified through the MonitorFn property in IDASetOptions.
      If this property is not set, or if it is empty, MONFUN is not used.
      MONDATA is specified through the MonitorData property in IDASetOptions.

   2. If quadrature integration is not enabled, YQ is empty. Similarly, if
      forward sensitivity analysis is not enabled, YYS and YPS are empty.
```

3. When CALL = 2 or 3, all arguments YY, YP, YQ, YYS, and YPS are empty. Moreover, when CALL = 3, T = 0.0

4. If MONFUN is used on the backward integration phase, YYS and YPS are always empty.

See also IDASetOptions, IDAMonitor

IDAResFnB

Purpose

IDAResFnb - type for residual function for backward problems

Synopsis

This is a script file.

Description

IDAResFnb - type for residual function for backward problems

The function DAEFUNB must be defined either as

```
     FUNCTION [RB, FLAG] = DAEFUNB(T, YY, YP, YYB, YPB)
```

or as

```
     FUNCTION [RB, FLAG, NEW_DATA] = DAEFUNB(T, YY, YP, YYB, YPB, DATA)
```

depending on whether a user data structure DATA was specified in IDAInit. In either case, it must return the vector RB corresponding to fB(t,yy,yp,yyB,ypB).

The function DAEFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also IDAInitB, IDARhsFn

IDAQuadRhsFnB

Purpose

IDAQuadRhsFnB - type for quadrature RHS function for backward problems

Synopsis

This is a script file.

Description

IDAQuadRhsFnB - type for quadrature RHS function for backward problems

The function QFUNB must be defined either as

```
     FUNCTION [YQBD, FLAG] = QFUNB(T, YY, YP, YYB, YPB)
```

or as

```
     FUNCTION [YQBD, FLAG, NEW_DATA] = QFUNB(T, YY, YP, YYB, YPB, DATA)
  depending on whether a user data structure DATA was specified in
  IDAInit. In either case, it must return the vector YQBD
  corresponding to fQB(t,yy,yp,yyB,ypB), the integrand for the integral to be
  evaluated on the backward phase.

  The function QFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
  unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
  occurred.

  See also IDAQuadInitB
```

IDADenseJacFnB

Purpose

IDADenseJacFnb - type for dense Jacobian function for backward problems.

Synopsis

This is a script file.

Description

IDADenseJacFnb - type for dense Jacobian function for backward problems.

```
  The function DJACFUNB must be defined either as
     FUNCTION [JB, FLAG] = DJACFUNB(T, YY, YP, YYB, YPB, RRB, CJB)
  or as
     FUNCTION [JB,FLAG,NEW_DATA] = DJACFUNB(T,YY,YP,YYB,YPB,RRB,CJB,DATA)
  depending on whether a user data structure DATA was specified in
  IDAInit. In either case, it must return the matrix JB, the
  Jacobian (dfB/dyyB + cjb*dfB/dypB). The input argument RRB contains
  the current value of f(t,yy,yp,yyB,ypB).

  The function DJACFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
  unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
  occurred.

  See also IDADenseJacFn, IDASetOptions

  NOTE: DJACFUNB is specified through the property JacobianFn to
  IDASetOptions and is used only if the property LinearSolver was
  set to 'Dense'.
```

IDABandJacFnB

Purpose

IDABandJacFnB - type for banded Jacobian function for backward problems.

Synopsis

This is a script file.

Description

```
IDABandJacFnB - type for banded Jacobian function for backward problems.

   The function BJACFUNB must be defined either as
        FUNCTION [JB, FLAG] = BJACFUNB(T, YY, YP, YYB, YPB, RRB, CJB)
   or as
        FUNCTION [JB,FLAG,NEW_DATA] = BJACFUNB(T,YY,YP,YYB,YPB,RRB,CJB)
   depending on whether a user data structure DATA was specified in
   IDAInit. In either case, it must return the matrix JB, the
   Jacobian (dfB/dyyB + cjB*dfB/dypB)of fB(t,y,yB). The input argument
   RRB contains the current value of f(t,yy,yp,yyB,ypB).

   The function BJACFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDASetOptions

   See the IDAS user guide for more information on the structure of
   a banded Jacobian.

   NOTE: BJACFUNB is specified through the property JacobianFn to
   IDASetOptions and is used only if the property LinearSolver
   was set to 'Band'.
```

IDAJacTimesVecFnB

Purpose

```
IDAJacTimesVecFn - type for Jacobian times vector function for backward problems.
```

Synopsis

```
This is a script file.
```

Description

```
IDAJacTimesVecFn - type for Jacobian times vector function for backward problems.

   The function JTVFUNB must be defined either as
        FUNCTION [JVB,FLAG] = JTVFUNB(T,YY,YP,YYB,YPB,RRB,VB,CJB)
   or as
        FUNCTION [JVB,FLAG,NEW_DATA] = JTVFUNB(T,YY,YP,YYB,YPB,RRB,VB,CJB,DATA)
   depending on whether a user data structure DATA was specified in
   IDAInit. In either case, it must return the vector JVB, the
   product of the Jacobian (dfB/dyyB + cj * dfB/dypB) and a vector
   vB. The input argument RRB contains the current value of f(t,yy,yp,yyB,ypB).

   The function JTVFUNB must set FLAG=0 if successful, or FLAG~=0 if
   a failure occurred.

   See also IDASetOptions

   NOTE: JTVFUNB is specified through the property JacobianFn to IDASetOptions
   and is used only if the property LinearSolver was set to 'GMRES', 'BiCGStab',
   or 'TFQMR'.
```

IDAPrecSetupFnB

PURPOSE

IDAPrecSetupFnB - type for preconditioner setup function for backward problems.

SYNOPSIS

This is a script file.

DESCRIPTION

IDAPrecSetupFnB - type for preconditioner setup function for backward problems.

```
   The function PSETFUNB must be defined either as
        FUNCTION FLAG = PSETFUNB(T,YY,YP,YYB,YPB,RRB,CJB)
   or as
        FUNCTION [FLAG,NEW_DATA] = PSETFUNB(T,YY,YP,YYB,YPB,RRB,CJB,DATA)
   depending on whether a user data structure DATA was specified in
   IDASetUserData.

   See also IDAPrecSolveFnB, IDAPrecSetupFn, IDASetOptions

   NOTE: PSETFUN and PSETFUNB are specified through the property
   PrecSetupFn to IDASetOptions and are used only if the property
   LinearSolver was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

IDAPrecSolveFnB

PURPOSE

IDAPrecSolveFnB - type for preconditioner solve function.

SYNOPSIS

This is a script file.

DESCRIPTION

IDAPrecSolveFnB - type for preconditioner solve function.

```
   The user-supplied preconditioner solve function PSOLFUNB
   is to solve a linear system P z = r, where P is the
   preconditioner matrix.

   The function PSOLFUNB must be defined either as
        FUNCTION [ZB,FLAG] = PSOLFUNB(T,YY,YP,YYB,YPB,RRB,RB)
   or as
        FUNCTION [ZB,FLAG,NEW_DATA] = PSOLFUNB(T,YY,YP,YYB,YPB,RRB,RB,DATA)
   depending on whether a user data structure DATA was specified in
   IDAInit. In either case, it must return the vector ZB and the
   flag FLAG.

   See also IDAPrecSetupFnB, IDAPrecSolveFn, IDASetOptions

   NOTE: PSOLFUN and PSOLFUNB are specified through the property
   PrecSolveFn to IDASetOptions and are used only if the property
   LinearSolver was set to 'GMRES', 'BiCGStab', or 'TFQMR'.
```

IDAGcommFnB

Purpose

IDAGcommFnB - type for communication function (BBDPre) for backward problems.

Synopsis

This is a script file.

Description

IDAGcommFnB - type for communication function (BBDPre) for backward problems.

```
   The function GCOMFUNB must be defined either as
        FUNCTION FLAG = GCOMFUNB(T, YY, YP, YYB, YPB)
   or as
        FUNCTION [FLAG, NEW_DATA] = GCOMFUNB(T, YY, YP, YYB, YPB, DATA)
   depending on whether a user data structure DATA was specified in
   IDAInit.

   The function GCOMFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also IDAGlocalFnB, IDAGcommFn, IDASetOptions

   NOTES:
     GCOMFUNB is specified through the GcommFn property in IDASetOptions
     and is used only if the property PrecModule is set to 'BBDPre'.

     Each call to GCOMFUNB is preceded by a call to the residual function
     DAEFUN with the same arguments T, YY, YP and YYB and YPB.
     Thus GCOMFUNB can omit any communication done by DAEFUNB if relevant
     to the evaluation of G by GLOCFUNB. If all necessary communication
     was done by DAEFUNB, GCOMFUNB need not be provided.
```

IDAGlocalFnB

Purpose

IDAGlocalFnB - type for RES approximation function (BBDPre) for backward problems.

Synopsis

This is a script file.

Description

IDAGlocalFnB - type for RES approximation function (BBDPre) for backward problems.

```
   The function GLOCFUNB must be defined either as
        FUNCTION [GLOCB, FLAG] = GLOCFUNB(T,YY,YP,YYB,YPB)
   or as
```

```
     FUNCTION [GLOCB, FLAG, NEW_DATA] = GLOCFUNB(T,YY,YP,YYB,YPB,DATA)
  depending on whether a user data structure DATA was specified in
  IDAInit. In either case, it must return the vector GLOCB
  corresponding to an approximation to fB(t,yy,yp,yyB,ypB).

  The function GLOCFUNB must set FLAG=0 if successful, FLAG&lt;0 if an
  unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
  occurred.

  See also IDAGcommFnB, IDAGlocalFn, IDASetOptions

  NOTE: GLOCFUN and GLOCFUNB are specified through the GlocalFn property
  in IDASetOptions and are used only if the property PrecModule
  is set to 'BBDPre'.
```

IDAMonitorFnB

Purpose

IDAMonitorFnB - type of monitoring function for backward problems.

Synopsis

This is a script file.

Description

IDAMonitorFnB - type of monitoring function for backward problems.

```
  The function MONFUNB must be defined as
      FUNCTION [] = MONFUNB(CALL, IDXB, T, Y, YQ)
  It is called after every internal IDASolveB step and can be used to
  monitor the progress of the solver. MONFUNB is called with CALL=0
  from IDAInitB at which time it should initialize itself and it
  is called with CALL=2 from IDAFree. Otherwise, CALL=1.

  It receives as arguments the index of the backward problem (as
  returned by IDAInitB), the current time T, solution vector Y,
  and, if it was computed, the quadrature vector YQ. If quadratures
  were not computed for this backward problem, YQ is empty here.

  If additional data is needed inside MONFUNB, it must be defined
  as
     FUNCTION NEW_MONDATA = MONFUNB(CALL, IDXB, T, Y, YQ, MONDATA)
  If the local modifications to the user data structure need to be
  saved (e.g. for future calls to MONFUNB), then MONFUNB must set
  NEW_MONDATA. Otherwise, it should set NEW_MONDATA=[]
  (do not set NEW_MONDATA = DATA as it would lead to unnecessary copying).

  A sample monitoring function, IDAMonitorB, is provided with CVODES.

  See also IDASetOptions, IDAMonitorB

  NOTES:
```

```
MONFUNB is specified through the MonitorFn property in IDASetOptions.
If this property is not set, or if it is empty, MONFUNB is not used.
MONDATA is specified through the MonitorData property in IDASetOptions.

See IDAMonitorB for an implementation example.
```

5 MATLAB Interface to KINSOL

The MATLAB interface to KINSOL provides access to all functionality of the KINSOL solver.

The interface consists of 5 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions and the types of user-supplied functions are listed in Table 9 and fully documented later in this section. For more in depth details, consult also the KINSOL user guide [1].

To illustrate the use of the KINSOL MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with KINSOL.

Table 9: KINSOL MATLAB interface functions

Functions	KINSetOptions	creates an options structure for KINSOL.
	KINInit	allocates and initializes memory for KINSOL.
	KINSol	solves the nonlinear problem.
	KINGetStats	returns statistics for the KINSOL solver.
	KINFree	deallocates memory for the KINSOL solver.
Function types	KINSysFn	system function
	KINDenseJacFn	dense Jacobian function
	KINBandJacFn	banded Jacobian function
	KINJacTimesVecFn	Jacobian times vector function
	KINPrecSetupFn	preconditioner setup function
	KINPrecSolveFn	preconditioner solve function
	KINGlocalFn	system approximation function (BBDPre)
	KINGcommFn	communication function (BBDPre)

5.1 Interface functions

KINSetOptions

Purpose

KINSetOptions creates an options structure for KINSOL.

Synopsis

```
function options = KINSetOptions(varargin)
```

Description

```
KINSetOptions creates an options structure for KINSOL.

   Usage:

   options = KINSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates a KINSOL
   options structure options in which the named properties have the
   specified values. Any unspecified properties have default values. It is
   sufficient to type only the leading characters that uniquely identify the
   property. Case is ignored for property names.

   options = KINSetOptions(oldoptions,'NAME1',VALUE1,...) alters an existing
   options structure oldoptions.

   options = KINSetOptions(oldoptions,newoptions) combines an existing options
   structure oldoptions with a new options structure newoptions. Any new
   properties overwrite corresponding old properties.

   KINSetOptions with no input arguments displays all property names and their
   possible values.

KINSetOptions properties
(See also the KINSOL User Guide)

UserData - User data passed unmodified to all functions [ empty ]
   If UserData is not empty, all user provided functions will be
   passed the problem data as their last input argument. For example,
   the SYS function must be defined as FY = SYSFUN(Y,DATA).

MaxNumIter - maximum number of nonlinear iterations [ scalar | 200 ]
   Specifies the maximum number of iterations that the nonlinar solver is allowed
   to take.
FuncRelErr - relative residual error [ scalar | eps ]
   Specifies the realative error in computing f(y) when used in difference
   quotient approximation of matrix-vector product J(y)*v.
FuncNormTol - residual stopping criteria [ scalar | eps^(1/3) ]
   Specifies the stopping tolerance on ||fscale*ABS(f(y))||_L-infinity
ScaledStepTol - step size stopping criteria [ scalar | eps^(2/3) ]
   Specifies the stopping tolerance on the maximum scaled step length:
                    ||    y_(k+1) - y_k   ||
```

```
                    || ----------------- ||_L-infinity
                    || |y_(k+1)| + yscale ||
MaxNewtonStep - maximum Newton step size [ scalar | 0.0 ]
   Specifies the maximum allowable value of the scaled length of the Newton step.
InitialSetup - initial call to linear solver setup [ false | true ]
   Specifies whether or not KINSol makes an initial call to the linear solver
   setup function.
MaxNumSetups - [ scalar | 10 ]
   Specifies the maximum number of nonlinear iterations between calls to the
   linear solver setup function (i.e. Jacobian/preconditioner evaluation)
MaxNumSubSetups - [ scalar | 5 ]
   Specifies the maximum number of nonlinear iterations between checks by the
   nonlinear residual monitoring algorithm (specifies length of subintervals).
   NOTE: MaxNumSetups should be a multiple of MaxNumSubSetups.
MaxNumBetaFails - maximum number of beta-condition failures [ scalar | 10 ]
   Specifies the maximum number of beta-condiiton failures in the line search
   algorithm.
EtaForm - Inexact Newton method [ Constant | Type2 | Type1 ]
   Specifies the method for computing the eta coefficient used in the calculation
   of the linear solver convergence tolerance (used only if strategy='InexactNEwton'
   in the call to KINSol):
      lintol = (eta + eps)*||fscale*f(y)||_L2
   which is the used to check if the following inequality is satisfied:
      ||fscale*(f(y)+J(y)*p)||_L2 &lt;= lintol
   Valid choices are:
                         | ||f(y_(k+1))||_L2 - ||f(y_k)+J(y_k)*p_k||_L2 |
   EtaForm='Type1'  eta = ------------------------------------------------
                                       ||f(y_k)||_L2

                                 [ ||f(y_(k+1))||_L2 ]^alpha
   EtaForm='Type2'  eta = gamma * [ ----------------- ]
                                 [  ||f(y_k)||_L2    ]
   EtaForm='Constant'
Eta - constant value for eta [ scalar | 0.1 ]
   Specifies the constant value for eta in the case EtaForm='Constant'.
EtaAlpha - alpha parameter for eta [ scalar | 2.0 ]
   Specifies the parameter alpha in the case EtaForm='Type2'
EtaGamma - gamma parameter for eta [ scalar | 0.9 ]
   Specifies the parameter gamma in the case EtaForm='Type2'
MinBoundEps - lower bound on eps [ false | true ]
   Specifies whether or not the value of eps is bounded below by 0.01*FuncNormtol.
Constraints - solution constraints [ vector ]
   Specifies additional constraints on the solution components.
     Constraints(i) =  0 : no constrain on y(i)
     Constraints(i) =  1 : y(i) &gt;= 0
     Constraints(i) = -1 : y(i) &lt;= 0
     Constraints(i) =  2 : y(i) &gt; 0
     Constraints(i) = -2 : y(i) &lt; 0
   If Constraints is not specified, no constraints are applied to y.

LinearSolver - Type of linear solver [ Dense | Band | GMRES | BiCGStab | TFQMR ]
   Specifies the type of linear solver to be used for the Newton nonlinear solver.
   Valid choices are: Dense (direct, dense Jacobian), GMRES (iterative, scaled
   preconditioned GMRES), BiCGStab (iterative, scaled preconditioned stabilized
```

BiCG), TFQMR (iterative, scaled preconditioned transpose-free QMR).
The GMRES, BiCGStab, and TFQMR are matrix-free linear solvers.

JacobianFn - Jacobian function [function]
This propeerty is overloaded. Set this value to a function that returns Jacobian information consistent with the linear solver used (see Linsolver). If not specified, KINSOL uses difference quotient approximations.
For the Dense linear solver, JacobianFn must be of type KINDenseJacFn and must return a dense Jacobian matrix. For the iterative linear solvers, GMRES, BiCGStab, or TFQMR, JacobianFn must be of type KINJactimesVecFn and must return a Jacobian-vector product.

KrylovMaxDim - Maximum number of Krylov subspace vectors [scalar | 10]
Specifies the maximum number of vectors in the Krylov subspace. This property is used only if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver).

MaxNumRestarts - Maximum number of GMRES restarts [scalar | 0]
Specifies the maximum number of times the GMRES (see LinearSolver) solver can be restarted.

PrecModule - Built-in preconditioner module [BBDPre | UserDefined]
If the PrecModule = 'UserDefined', then the user must provide at least a preconditioner solve function (see PrecSolveFn)
KINSOL provides a built-in preconditioner module, BBDPre which can only be used with parallel vectors. It provide a preconditioner matrix that is block-diagonal with banded blocks. The blocking corresponds to the distribution of the variable vector among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor) of a given function g(t,y) approximating f(y) (see GlocalFn). The blocks are generated by a difference quotient scheme on each processor independently. This scheme utilizes an assumed banded structure with given half-bandwidths, mldq and mudq (specified through LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian block kept by the scheme has half-bandwiths ml and mu (specified through LowerBwidth and UpperBwidth), which may be smaller.

PrecSetupFn - Preconditioner setup function [function]
PrecSetupFn specifies an optional function which, together with PrecSolve, defines a right preconditioner matrix which is an aproximation to the Newton matrix. PrecSetupFn must be of type KINPrecSetupFn.

PrecSolveFn - Preconditioner solve function [function]
PrecSolveFn specifies an optional function which must solve a linear system Pz = r, for given r. If PrecSolveFn is not defined, the no preconditioning will be used. PrecSolveFn must be of type KINPrecSolveFn.

GlocalFn - Local right-hand side approximation funciton for BBDPre [function]
If PrecModule is BBDPre, GlocalFn specifies a required function that evaluates a local approximation to the system function. GlocalFn must be of type KINGlocalFn.

GcommFn - Inter-process communication function for BBDPre [function]
If PrecModule is BBDPre, GcommFn specifies an optional function to perform any inter-process communication required for the evaluation of GlocalFn. GcommFn must be of type KINGcommFn.

LowerBwidth - Jacobian/preconditioner lower bandwidth [scalar | 0]
This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the lower half-bandwidth of the band Jacobian approximation.
If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used (see PrecModule), it specifies the lower half-bandwidth of the retained banded approximation of the local Jacobian block.

```
   LowerBwidth defaults to 0 (no sub-diagonals).
UpperBwidth - Jacobian/preconditioner upper bandwidth [ scalar | 0 ]
   This property is overloaded. If the Band linear solver is used (see LinSolver),
   it specifies the upper half-bandwidth of the band Jacobian approximation.
   If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used
   (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used
   (see PrecModule), it specifies the upper half-bandwidth of the retained
   banded approximation of the local Jacobian block.
   UpperBwidth defaults to 0 (no super-diagonals).
LowerBwidthDQ - BBDPre preconditioner DQ lower bandwidth [ scalar | 0 ]
   Specifies the lower half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).
UpperBwidthDQ - BBDPre preconditioner DQ upper bandwidth [ scalar | 0 ]
   Specifies the upper half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).

Verbose - verbose output [ true | false ]
   Specifies whether or not KINSOL should output additional information
ErrorMessages - Post error/warning messages [ false | true ]
   Note that any errors in KINInit will result in a Matlab error, thus
   stoping execution. Only subsequent calls to KINSOL functions will respect
   the value specified for 'ErrorMessages'.

   See also
        KINDenseJacFn, KINJacTimesVecFn
        KINPrecSetupFn, KINPrecSolveFn
        KINGlocalFn, KINGcommFn
```

KINInit

Purpose

KINInit allocates and initializes memory for KINSOL.

Synopsis

```
function status = KINInit(fct, n, options)
```

Description

KINInit allocates and initializes memory for KINSOL.

```
   Usage:   KINInit ( SYSFUN, N [, OPTIONS ] );

   SYSFUN   is a function defining the nonlinear problem f(y) = 0.
            This function must return a column vector FY containing the
            current value of the residual
   N        is the (local) problem dimension.
   OPTIONS  is an (optional) set of integration options, created with
            the KINSetOptions function.

   See also: KINSetOptions, KINSysFn
```

KINSol

Purpose

KINSol solves the nonlinear problem.

Synopsis

function [status, y] = KINSol(y0, strategy, yscale, fscale)

Description

KINSol solves the nonlinear problem.

```
   Usage:  [STATUS, Y] = KINSol(Y0, STRATEGY, YSCALE, FSCALE)

   KINSol manages the computational process of computing an approximate
   solution of the nonlinear system. If the initial guess (initial value
   assigned to vector Y0) doesn't violate any user-defined constraints,
   then KINSol attempts to solve the system f(y)=0. If an iterative linear
   solver was specified (see KINSetOptions), KINSol uses a nonlinear Krylov
   subspace projection method. The Newton-Krylov iterations are stopped
   if either of the following conditions is satisfied:

       ||f(y)||_L-infinity &lt;= 0.01*fnormtol

       ||y[i+1] - y[i]||_L-infinity &lt;= scsteptol

   However, if the current iterate satisfies the second stopping
   criterion, it doesn't necessarily mean an approximate solution
   has been found since the algorithm may have stalled, or the
   user-specified step tolerance may be too large.

   STRATEGY specifies the global strategy applied to the Newton step if it is
   unsatisfactory. Valid choices are 'None' or 'LineSearch'.
   YSCALE is a vector containing diagonal elements of scaling matrix for vector
   Y chosen so that the components of YSCALE*Y (as a matrix multiplication) all
   have about the same magnitude when Y is close to a root of f(y)
   FSCALE is a vector containing diagonal elements of scaling matrix for f(y)
   chosen so that the components of FSCALE*f(Y) (as a matrix multiplication)
   all have roughly the same magnitude when u is not too near a root of f(y)

   On return, status is one of the following:
     0: KINSol succeeded
     1: The initial y0 already satisfies the stopping criterion given above
     2: Stopping tolerance on scaled step length satisfied
    -1: An error occurred (see printed error message)

   See also KINSetOptions, KINGetstats
```

KINGetStats

Purpose

KINGetStats returns statistics for the main KINSOL solver and the linear

Synopsis

function [si, status] = KINGetStats()

Description

KINGetStats returns statistics for the main KINSOL solver and the linear
solver used.

```
   Usage: STATS = KINGetStats

Fields in the structure STATS

o nfe    - total number evaluations of the nonlinear system function SYSFUN
o nni    - total number of nonlinear iterations
o nbcf   - total number of beta-condition failures
o nbops  - total number of backtrack operations (step length adjustments)
           performed by the line search algorithm
o fnorm  - scaled norm of the nonlinear system function f(y) evaluated at the
           current iterate: ||fscale*f(y)||_L2
o step   - scaled norm (or length) of the step used during the previous
           iteration: ||uscale*p||_L2
o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' or 'BiCGStab' linear solver

o name - 'GMRES' or 'BiCGStab'
o nli  - number of linear solver iterations
o npe  - number of preconditioner setups
o nps  - number of preconditioner solve function calls
o ncfl - number of linear system convergence test failures
```

KINFree

Purpose

KINFree deallocates memory for the KINSOL solver.

Synopsis

function KINFree()

Description

```
KINFree deallocates memory for the KINSOL solver.

   Usage:  KINFree
```

5.2 Function types

KINSysFn

PURPOSE

KINSysFn - type for user provided system function

SYNOPSIS

This is a script file.

DESCRIPTION

KINSysFn - type for user provided system function

```
   The function SYSFUN must be defined as
        FUNCTION [FY, FLAG] = SYSFUN(Y)
   and must return a vector FY corresponding to f(y).
   If a user data structure DATA was specified in KINInit, then
   SYSFUN must be defined as
        FUNCTION [FY, FLAG, NEW_DATA] = SYSFUN(Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector FY,
   the SYSFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function SYSFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also KINInit

   NOTE: SYSFUN is specified through the KINInit function.
```

KINDenseJacFn

PURPOSE

KINDenseJacFn - type for user provided dense Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINDenseJacFn - type for user provided dense Jacobian function.

```
   The function DJACFUN must be defined as
        FUNCTION [J, FLAG] = DJACFUN(Y,FY)
   and must return a matrix J corresponding to the Jacobian of f(y).
```

```
The input argument FY contains the current value of f(y).
If a user data structure DATA was specified in KINInit, then
DJACFUN must be defined as
     FUNCTION [J, FLAG, NEW_DATA] = DJACFUN(Y,FY,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the matrix J and
the flag FLAG, the DJACFUN function must also set NEW_DATA. Otherwise,
it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
to unnecessary copying).

The function DJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also KINSetOptions

NOTE: DJACFUN is specified through the property JacobianFn to KINSetOptions
and is used only if the property LinearSolver was set to 'Dense'.
```

KINBandJacFn

Purpose

KINBandJacFn - type for user provided banded Jacobian function.

Synopsis

This is a script file.

Description

KINBandJacFn - type for user provided banded Jacobian function.

```
The function BJACFUN must be defined as
     FUNCTION [J, FLAG] = BJACFUN(Y, FY)
and must return a matrix J corresponding to the banded Jacobian of f(y).
The input argument FY contains the current value of f(y).
If a user data structure DATA was specified in KINInit, then
BJACFUN must be defined as
     FUNCTION [J, FLAG, NEW_DATA] = BJACFUN(Y, FY, DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the matrix J and
the flag FLAG, the BJACFUN function must also set NEW_DATA. Otherwise,
it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
to unnecessary copying).

The function BJACFUN must set FLAG=0 if successful, FLAG&lt;0 if an
unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
occurred.

See also KINSetOptions

NOTE: BJACFUN is specified through the property JacobianFn to KINSetOptions
and is used only if the property LinearSolver was set to 'Band'.
```

KINJacTimesVecFn

PURPOSE

KINJacTimesVecFn - type for user provided Jacobian times vector function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINJacTimesVecFn - type for user provided Jacobian times vector function.

```
   The function JTVFUN must be defined as
        FUNCTION [JV, NEW_Y, FLAG] = JTVFUN(Y, V, NEW_Y)
   and must return a vector JV corresponding to the product of the
   Jacobian of f(y) with the vector v. On input, NEW_Y indicates if
   the iterate has been updated in the interim. JV must be update
   or reevaluated, if appropriate, unless NEW_Y=false. This flag must
   be reset by the user.
   If a user data structure DATA was specified in KINInit, then
   JTVFUN must be defined as
        FUNCTION [JV, NEW_Y, FLAG, NEW_DATA] = JTVFUN(Y, V, NEW_Y, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector JV, and
   flags NEW_Y and FLAG, the JTVFUN function must also set NEW_DATA. Otherwise,
   it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   If successful, FLAG should be set to 0. If an error occurs, FLAG should
   be set to a nonzero value.

   See also KINSetOptions

   NOTE: JTVFUN is specified through the property JacobianFn to KINSetOptions
   and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINPrecSetupFn

PURPOSE

KINPrecSetupFn - type for user provided preconditioner setup function.

SYNOPSIS

This is a script file.

DESCRIPTION

```
KINPrecSetupFn - type for user provided preconditioner setup function.

   The user-supplied preconditioner setup subroutine should compute
   the right-preconditioner matrix P used to form the scaled preconditioned
   linear system:

   (Df*J(y)*(P^-1)*(Dy^-1)) * (Dy*P*x) = Df*(-F(y))

   where Dy and Df denote the diagonal scaling matrices whose diagonal elements
   are stored in the vectors YSCALE and FSCALE, respectively.

   The preconditioner setup routine (referenced by iterative linear
   solver modules via pset (type KINSpilsPrecSetupFn)) will not be
   called prior to every call made to the psolve function, but will
   instead be called only as often as necessary to achieve convergence
   of the Newton iteration.

   NOTE: If the PRECSOLVE function requires no preparation, then a
   preconditioner setup function need not be given.

   The function PSETFUN must be defined as
        FUNCTION FLAG = PSETFUN(Y, YSCALE, FY, FSCALE)
   The input argument FY contains the current value of f(y), while YSCALE
   and FSCALE are the scaling vectors for solution and system function,
   respectively (as passed to KINSol)

   If a user data structure DATA was specified in KINInit, then
   PSETFUN must be defined as
        FUNCTION [FLAG, NEW_DATA] = PSETFUN(Y, YSCALE, FY, FSCALE, DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the flag FLAG,
   the PSETFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
   to unnecessary copying).

   If successful, PSETFUN must return FLAG=0. For a recoverable error (in
   which case the setup will be retried) it must set FLAG to a positive
   integer value. If an unrecoverable error occurs, it must set FLAG
   to a negative value, in which case the solver will halt.

   See also KINPrecSolveFn, KINSetOptions, KINSol

   NOTE: PSETFUN is specified through the property PrecSetupFn to KINSetOptions
   and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINPrecSolveFn

Purpose

KINPrecSolveFn - type for user provided preconditioner solve function.

Synopsis

This is a script file.

Description

```
KINPrecSolveFn - type for user provided preconditioner solve function.

   The user-supplied preconditioner solve function PSOLFN
   is to solve a linear system P z = r in which the matrix P is
   the preconditioner matrix (possibly set implicitely by PSETFUN)

   The function PSOLFUN must be defined as
        FUNCTION [Z, FLAG] = PSOLFUN(Y, YSCALE, FY, FSCALE, R)
   and must return a vector Z containing the solution of Pz=r.
   The input argument FY contains the current value of f(y), while YSCALE
   and FSCALE are the scaling vectors for solution and system function,
   respectively (as passed to KINSol)

   If a user data structure DATA was specified in KINInit, then
   PSOLFUN must be defined as
        FUNCTION [Z, FLAG, NEW_DATA] = PSOLFUN(Y,YSCALE,FY,FSCALE,R,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector Z and
   the flag FLAG, the PSOLFUN function must also set NEW_DATA. Otherwise,
   it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would
   lead to unnecessary copying).

   If successful, PSOLFUN must return FLAG=0. For a recoverable error it
   must set FLAG to a positive value (in which case the solver will attempt
   to correct). If an unrecoverable error occurs, it must set FLAG
   to a negative value, in which case the solver will halt.

   See also KINPrecSetupFn, KINSetOptions

   NOTE: PSOLFUN is specified through the property PrecSolveFn to KINSetOptions
   and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINGcommFn

Purpose

KINGcommFn - type for user provided communication function (BBDPre).

Synopsis

This is a script file.

Description

```
KINGcommFn - type for user provided communication function (BBDPre).

   The function GCOMFUN must be defined as
        FUNCTION FLAG = GCOMFUN(Y)
   and can be used to perform all interprocess communication necessary
   to evaluate the approximate right-hand side function for the BBDPre
   preconditioner module.
   If a user data structure DATA was specified in KINInit, then
   GCOMFUN must be defined as
        FUNCTION [FLAG, NEW_DATA] = GCOMFUN(Y, DATA)
```

```
   If the local modifications to the user data structure are needed
   in other user-provided functions then the GCOMFUN function must also
   set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set
   NEW_DATA = DATA as it would lead to unnecessary copying).

   The function GCOMFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also KINGlocalFn, KINSetOptions

   NOTES:
     GCOMFUN is specified through the GcommFn property in KINSetOptions
     and is used only if the property PrecModule is set to 'BBDPre'.

     Each call to GCOMFUN is preceded by a call to the system function
     SYSFUN with the same argument Y. Thus GCOMFUN can omit any communication
     done by SYSFUN if relevant to the evaluation of G by GLOCFUN. If all
     necessary communication was done by SYSFUN, GCOMFUN need not be provided.
```

KINGlocalFn

Purpose

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

Synopsis

This is a script file.

Description

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

```
   The function GLOCFUN must be defined as
        FUNCTION [G, FLAG] = GLOCFUN(Y)
   and must return a vector G corresponding to an approximation to f(y)
   which will be used in the BBDPRE preconditioner module. The case where
   G is mathematically identical to F is allowed.
   If a user data structure DATA was specified in KINInit, then
   GLOCFUN must be defined as
        FUNCTION [G, FLAG, NEW_DATA] = GLOCFUN(Y, DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector G,
   the GLOCFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   The function GLOCFUN must set FLAG=0 if successful, FLAG&lt;0 if an
   unrecoverable failure occurred, or FLAG&gt;0 if a recoverable error
   occurred.

   See also KINGcommFn, KINSetOptions

   NOTE: GLOCFUN is specified through the GlocalFn property in KINSetOptions
   and is used only if the property PrecModule is set to 'BBDPre'.
```

6 Supporting modules

This section describes two additional modules in SUNDIALSTB, NVECTOR and PUTILS. The functions in NVECTOR perform various operations on vectors. For serial vectors, all of these operations default to the corresponding MATLAB functions. For parallel vectors, they can be used either on the local portion of the distributed vector or on the global vector (in which case they will trigger an MPI Allreduce operation). The functions in PUTILS are used to run parallel SUNDIALSTB applications. The user should only call the function `mpirun` to launch a parallel MATLAB application. See one of the paralel SUNDIALSTB examples for usage.

The functions in these two additional modules are listed in Table 10 and described in detail in the remainder of this section.

Table 10: The NVECTOR and PUTILS functions

NVECTOR	N_VMax	returns the largest element of x
	N_VMaxNorm	returns the maximum norm of x
	N_VMin	returns the smallest element of x
	N_VDotProd	returns the dot product of two vectors
	N_VWrmsNorm	returns the weighted root mean square norm of x
	N_VWL2Norm	returns the weighted Euclidean L2 norm of x
	N_VL1Norm	returns the L1 norm of x
PUTILS	mpirun	runs parallel examples
	mpiruns	runs the parallel example on a child MATLAB process
	mpistart	lamboot and `MPI_Init` master (if required)

6.1 NVECTOR functions

N_VDotProd

PURPOSE

N_VDotProd returns the dot product of two vectors

SYNOPSIS

function ret = N_VDotProd(x,y,comm)

DESCRIPTION

N_VDotProd returns the dot product of two vectors

```
   Usage:  RET = N_VDotProd ( X, Y [, COMM] )
```

If COMM is not present, N_VDotProd returns the dot product of the local portions of X and Y. Otherwise, it returns the global dot product.

SOURCE CODE

```
function ret = N_VDotProd(x,y,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$


if nargin == 2

  ret = dot(x,y);

else

  ldot = dot(x,y);
  gdot = 0.0;
  MPI_Allreduce(ldot,gdot,'SUM',comm);
  ret = gdot;

end
```

N_VL1Norm

Purpose

`N_VL1Norm returns the L1 norm of x`

Synopsis

`function ret = N_VL1Norm(x,comm)`

Description

```
N_VL1Norm returns the L1 norm of x

   Usage:  RET = N_VL1Norm ( X [, COMM] )

If COMM is not present, N_VL1Norm returns the L1 norm of
the local portion of X. Otherwise, it returns the global
L1 norm..
```

Source Code

```
function ret = N_VL1Norm(x,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$

if nargin == 1

  ret = norm(x,1);

else

  lnrm = norm(x,1);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'MAX',comm);
  ret = gnrm;

end
```

N_VMax

Purpose

`N_VMax returns the largest element of x`

Synopsis

`function ret = N_VMax(x,comm)`

Description

```
N_VMax returns the largest element of x

   Usage:  RET = N_VMax ( X [, COMM] )

If COMM is not present, N_VMax returns the maximum value of
the local portion of X. Otherwise, it returns the global
maximum value.
```

Source Code

```
function ret = N_VMax(x,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$

if nargin == 1

  ret = max(x);

else

  lmax = max(x);
  gmax = 0.0;
  MPI_Allreduce(lmax,gmax,'MAX',comm);
  ret = gmax;

end
```

N_VMaxNorm

Purpose

```
N_VMaxNorm returns the L-infinity norm of x
```

Synopsis

```
function ret = N_VMaxNorm(x, comm)
```

Description

```
N_VMaxNorm returns the L-infinity norm of x

   Usage:  RET = N_VMaxNorm ( X [, COMM] )

If COMM is not present, N_VMaxNorm returns the L-infinity norm
of the local portion of X. Otherwise, it returns the global
L-infinity norm..
```

Source Code

```
function ret = N_VMaxNorm(x, comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$

if nargin == 1

  ret = norm(x,'inf');

else

  lnrm = norm(x,'inf');
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'MAX',comm);
  ret = gnrm;

end
```

N_VMin

Purpose

```
N_VMin returns the smallest element of x
```

Synopsis

```
function ret = N_VMin(x,comm)
```

Description

```
N_VMin returns the smallest element of x
   Usage:  RET = N_VMin ( X [, COMM] )

If COMM is not present, N_VMin returns the minimum value of
the local portion of X. Otherwise, it returns the global
minimum value.
```

Source Code

```
function ret = N_VMin(x,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
```

```
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$

if nargin == 1

  ret = min(x);

else

  lmin = min(x);
  gmin = 0.0;
  MPI_Allreduce(lmin,gmin,'MIN',comm);
  ret = gmin;

end
```

N_VWL2Norm

Purpose

N_VWL2Norm returns the weighted Euclidean L2 norm of x

Synopsis

function ret = N_VWL2Norm(x,w,comm)

Description

```
N_VWL2Norm returns the weighted Euclidean L2 norm of x
   with weight vector w:
   sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)]

   Usage:  RET = N_VWL2Norm ( X, W [, COMM] )

If COMM is not present, N_VWL2Norm returns the weighted L2
norm of the local portion of X. Otherwise, it returns the
global weighted L2 norm..
```

Source Code

```
function ret = N_VWL2Norm(x,w,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
```

```
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date$

if nargin == 2

  ret = dot(x.^2,w.^2);
  ret = sqrt(ret);

else

  lnrm = dot(x.^2,w.^2);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'SUM',comm);

  ret = sqrt(gnrm);

end
```

N_VWrmsNorm

Purpose

N_VWrmsNorm returns the weighted root mean square norm of x

Synopsis

function ret = N_VWrmsNorm(x,w,comm)

Description

```
N_VWrmsNorm returns the weighted root mean square norm of x
with weight vector w:
   sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)/N]

   Usage:  RET = N_VWrmsNorm ( X, W [, COMM] )

If COMM is not present, N_VWrmsNorm returns the WRMS norm
of the local portion of X. Otherwise, it returns the global
WRMS norm..
```

Source Code

```
function ret = N_VWrmsNorm(x,w,comm)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
```

```
% $Revision: 4075 $Date$

if nargin == 2

  ret = dot(x.^2,w.^2);
  ret = sqrt(ret/length(x));

else

  lnrm = dot(x.^2,w.^2);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'SUM',comm);

  ln = length(x);
  gn = 0;
  MPI_Allreduce(ln,gn,'SUM',comm);

  ret = sqrt(gnrm/gn);

end
```

6.2 Parallel utilities

---

mpirun

Purpose

MPIRUN runs parallel examples.

Synopsis

```
function [] = mpirun(fct,npe,dbg)
```

Description

MPIRUN runs parallel examples.

```
  Usage: MPIRUN ( FCT , NPE [, DBG] )

  FCT - function to be executed on all MATLAB processes.
  NPE - number of processes to be used (including the master).
  DBG - flag for debugging [ true | false ]
        If true, spawn MATLAB child processes with a visible xterm.
```

---

mpiruns

Purpose

MPIRUNS runs the parallel example on a child MATLAB process.

Synopsis

```
function [] = mpiruns(fct)
```

Description

MPIRUNS runs the parallel example on a child MATLAB process.

```
  Usage: MPIRUNS ( FCT )

  This function should not be called directly. It is called
  by mpirun on the spawned child processes.
```

---

mpistart

Purpose

MPISTART invokes lamboot (if required) and MPI_Init (if required).

Synopsis

```
function mpistart(nslaves, rpi, hosts)
```

Description

```
MPISTART invokes lamboot (if required) and MPI_Init (if required).

   Usage: MPISTART [ ( NSLAVES [, RPI [, HOSTS] ] ) ]

   MPISTART boots LAM and initializes MPI to match a given number of slave
   hosts (and rpi) from a given list of hosts. All three args optional.

   If they are not defined, HOSTS are taken from a builtin HOSTS list
   (edit HOSTS at the beginning of this file to match your cluster)
   or from the bhost file if defined through LAMBHOST (in this order).

   If not defined, RPI is taken from the builtin variable RPI (edit it
   to suit your needs) or from the LAM_MPI_SSI_rpi environment variable
   (in this order).
```

A Implementation of CVodeMonitor.m

CVodeMonitor

PURPOSE

CVodeMonitor is the default CVODES monitoring function.

SYNOPSIS

function [new_data] = CVodeMonitor(call, T, Y, YQ, YS, data)

DESCRIPTION

```
CVodeMonitor is the default CVODES monitoring function.
   To use it, set the Monitor property in CVodeSetOptions to
   'CVodeMonitor' or to @CVodeMonitor and 'MonitorData' to mondata
   (defined as a structure).

   With default settings, this function plots the evolution of the step
   size, method order, and various counters.

   Various properties can be changed from their default values by passing
   to CVodeSetOptions, through the property 'MonitorData', a structure
   MONDATA with any of the following fields. If a field is not defined,
   the corresponding default value is used.

   Fields in MONDATA structure:
     o stats [ true | false ]
         If true, report the evolution of the step size and method order.
     o cntr [ true | false ]
         If true, report the evolution of the following counters:
         nst, nfe, nni, netf, ncfn (see CVodeGetStats)
     o mode [ 'graphical' | 'text' | 'both' ]
         In graphical mode, plot the evolutions of the above quantities.
         In text mode, print a table.
     o sol  [ true | false ]
         If true, plot solution components.
     o sensi [ true | false ]
         If true and if FSA is enabled, plot sensitivity components.
     o select [ array of integers ]
         To plot only particular solution components, specify their indeces in
         the field select. If not defined, but sol=true, all components are plotted.
     o updt [ integer | 50 ]
         Update frequency. Data is posted in blocks of dimension n.
     o skip [ integer | 0 ]
         Number of integrations steps to skip in collecting data to post.
     o post [ true | false ]
         If false, disable all posting. This option is necessary to disable
         monitoring on some processors when running in parallel.

   See also CVodeSetOptions, CVMonitorFn

   NOTES:
     1. The argument mondata is REQUIRED. Even if only the default options
        are desired, set mondata=struct; and pass it to CVodeSetOptions.
     2. The yQ argument is currently ignored.
```

Source Code

```
function [new_data] = CVodeMonitor(call, T, Y, YQ, YS, data)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date: 2007/05/11 18:51:32 $

if (nargin ~= 6)
  error('Monitor data not defined.');
end

new_data = [];

if call == 0

% Initialize unspecified fields to default values.
  data = initialize_data(data);

% Open figure windows
  if data.post

    if data.grph
      if data.stats | data.cntr
        data.hfg = figure;
      end
%     Number of subplots in figure hfg
      if data.stats
        data.npg = data.npg + 2;
      end
      if data.cntr
        data.npg = data.npg + 1;
      end
    end

    if data.text
      if data.cntr | data.stats
        data.hft = figure;
      end
    end

    if data.sol | data.sensi
      data.hfs = figure;
    end

  end

```

```
% Initialize other private data
  data.i = 0;
  data.n = 1;
  data.t = zeros(1,data.updt);
  if data.stats
    data.h = zeros(1,data.updt);
    data.q = zeros(1,data.updt);
  end
  if data.cntr
    data.nst = zeros(1,data.updt);
    data.nfe = zeros(1,data.updt);
    data.nni = zeros(1,data.updt);
    data.netf = zeros(1,data.updt);
    data.ncfn = zeros(1,data.updt);
  end

  data.first = true;        % the next one will be the first call = 1
  data.initialized = false; % the graphical windows were not initalized

  new_data = data;

  return;

else

% If this is the first call ~= 0,
% use Y and YS for additional initializations

  if data.first

    if isempty(YS)
      data.sensi = false;
    end

    if data.sol | data.sensi

      if isempty(data.select)

        data.N = length(Y);
        data.select = [1:data.N];

      else

        data.N = length(data.select);

      end

      if data.sol
        data.y = zeros(data.N,data.updt);
        data.nps = data.nps + 1;
      end

      if data.sensi
        data.Ns = size(YS,2);
```

```
        data.ys = zeros(data.N, data.Ns, data.updt);
        data.nps = data.nps + data.Ns;
      end

    end

    data.first = false;

  end

% Extract variables from data

  hfg  = data.hfg;
  hft  = data.hft;
  hfs  = data.hfs;
  npg  = data.npg;
  nps  = data.nps;
  i    = data.i;
  n    = data.n;
  t    = data.t;
  N    = data.N;
  Ns   = data.Ns;
  y    = data.y;
  ys   = data.ys;
  h    = data.h;
  q    = data.q;
  nst  = data.nst;
  nfe  = data.nfe;
  nni  = data.nni;
  netf = data.netf;
  ncfn = data.ncfn;

end


% Load current statistics?

if call == 1

  if i ~= 0
    i = i-1;
    data.i = i;
    new_data = data;
    return;
  end

  si = CVodeGetStats;

  t(n) = si.tcur;

  if data.stats
    h(n) = si.hlast;
    q(n) = si.qlast;
  end
```

```
  if data.cntr
    nst(n) = si.nst;
    nfe(n) = si.nfe;
    nni(n) = si.nni;
    netf(n) = si.netf;
    ncfn(n) = si.ncfn;
  end

  if data.sol
    for j = 1:N
      y(j,n) = Y(data.select(j));
    end
  end

  if data.sensi
    for k = 1:Ns
      for j = 1:N
        ys(j,k,n) = YS(data.select(j),k);
      end
    end
  end

end

% Is it time to post?

if data.post & (n == data.updt | call==2)

  if call == 2
    n = n-1;
  end

  if ~data.initialized

    if (data.stats | data.cntr) & data.grph
      graphical_init(n, hfg, npg, data.stats, data.cntr, ...
                     t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if (data.stats | data.cntr) & data.text
      text_init(n, hft, data.stats, data.cntr, ...
                t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if data.sol | data.sensi
      sol_init(n, hfs, nps, data.sol, data.sensi,   ...
               N, Ns, t, y, ys);
    end

    data.initialized = true;

  else

```

```
    if (data.stats | data.cntr) & data.grph
      graphical_update(n, hfg, npg, data.stats, data.cntr, ...
                       t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if (data.stats | data.cntr) & data.text
      text_update(n, hft, data.stats, data.cntr, ...
                  t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if data.sol
      sol_update(n, hfs, nps, data.sol, data.sensi, N, Ns, t, y, ys);
    end

  end

  if call == 2

    if (data.stats | data.cntr) & data.grph
      graphical_final(hfg, npg, data.cntr, data.stats);
    end

    if data.sol | data.sensi
      sol_final(hfs, nps, data.sol, data.sensi, N, Ns);
    end

    return;

  end

  n = 1;

else

  n = n + 1;

end


% Save updated values in data

data.i    = data.skip;
data.n    = n;
data.npg  = npg;
data.t    = t;
data.y    = y;
data.ys   = ys;
data.h    = h;
data.q    = q;
data.nst  = nst;
data.nfe  = nfe;
data.nni  = nni;
data.netf = netf;
data.ncfn = ncfn;
```

```
new_data = data;

return;

%------------------------------------------------------------------------------------------

function data = initialize_data(data)

if ~isfield(data,'mode')
  data.mode = 'graphical';
end
if ~isfield(data,'updt')
  data.updt = 50;
end
if ~isfield(data,'skip')
  data.skip = 0;
end
if ~isfield(data,'stats')
  data.stats = true;
end
if ~isfield(data,'cntr')
  data.cntr = true;
end
if ~isfield(data,'sol')
  data.sol = false;
end
if ~isfield(data,'sensi')
  data.sensi = false;
end
if ~isfield(data,'select')
  data.select = [];
end
if ~isfield(data,'post')
  data.post = true;
end

data.grph = true;
data.text = true;
if strcmp(data.mode,'graphical')
  data.text = false;
end
if strcmp(data.mode,'text')
  data.grph = false;
end

if ~data.sol & ~data.sensi
  data.select = [];
end

% Other initializations
data.npg = 0;
data.nps = 0;
data.hfg = 0;
```

```
data.hft = 0;
data.hfs = 0;
data.h = 0;
data.q = 0;
data.nst = 0;
data.nfe = 0;
data.nni = 0;
data.netf = 0;
data.ncfn = 0;
data.N = 0;
data.Ns = 0;
data.y = 0;
data.ys = 0;

%---------------------------------------------------------------------------

function [] = graphical_init(n, hfg, npg, stats, cntr, ...
                             t, h, q, nst, nfe, nni, netf, ncfn)

fig_name = 'CVODES run statistics';

% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end

figure(hfg);
set(hfg,'Name',fig_name);
set(hfg,'color',[1 1 1]);
pl = 0;

% Time label and figure title

tlab = '\rightarrow   t   \rightarrow';

% Step size and order
if stats
  pl = pl+1;
  subplot(npg,1,pl)
  semilogy(t(1:n),abs(h(1:n)),'-');
  hold on;
  box on;
  grid on;
  xlabel(tlab);
  ylabel('|Step size|');

  pl = pl+1;
  subplot(npg,1,pl)
  plot(t(1:n),q(1:n),'-');
  hold on;
```

```
  box on;
  grid on;
  xlabel(tlab);
  ylabel('Order');
end

% Counters
if cntr
  pl = pl+1;
  subplot(npg,1,pl)
  plot(t(1:n),nst(1:n),'k-');
  hold on;
  plot(t(1:n),nfe(1:n),'b-');
  plot(t(1:n),nni(1:n),'r-');
  plot(t(1:n),netf(1:n),'g-');
  plot(t(1:n),ncfn(1:n),'c-');
  box on;
  grid on;
  xlabel(tlab);
  ylabel('Counters');
end

drawnow;

%------------------------------------------------------------------------

function [] = graphical_update(n, hfg, npg, stats, cntr, ...
                               t, h, q, nst, nfe, nni, netf, ncfn)

figure(hfg);
pl = 0;

% Step size and order
if stats
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n)];
  yd = [get(hc,'YData') abs(h(1:n))];
  set(hc, 'XData', xd, 'YData', yd);

  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n)];
  yd = [get(hc,'YData') q(1:n)];
  set(hc, 'XData', xd, 'YData', yd);
end

% Counters
if cntr
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
```

```
% Attention: Children are loaded in reverse order!
  xd = [get(hc(1),'XData') t(1:n)];
  yd = [get(hc(1),'YData') ncfn(1:n)];
  set(hc(1), 'XData', xd, 'YData', yd);
  yd = [get(hc(2),'YData') netf(1:n)];
  set(hc(2), 'XData', xd, 'YData', yd);
  yd = [get(hc(3),'YData') nni(1:n)];
  set(hc(3), 'XData', xd, 'YData', yd);
  yd = [get(hc(4),'YData') nfe(1:n)];
  set(hc(4), 'XData', xd, 'YData', yd);
  yd = [get(hc(5),'YData') nst(1:n)];
  set(hc(5), 'XData', xd, 'YData', yd);
end

drawnow;

%------------------------------------------------------------------------

function [] = graphical_final(hfg,npg,stats,cntr)

figure(hfg);
pl = 0;

if stats
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = get(hc,'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));

  pl = pl+1;
  subplot(npg,1,pl)
  ylim = get(gca,'YLim');
  ylim(1) = ylim(1) - 1;
  ylim(2) = ylim(2) + 1;
  set(gca,'YLim',ylim);
  set(gca,'XLim',sort([xd(1) xd(end)]));
end

if cntr
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = get(hc(1),'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));
  legend('nst','nfe','nni','netf','ncfn',2);
end

%------------------------------------------------------------------------

function [] = text_init(n,hft,stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

fig_name = 'CVODES run statistics';

```

```
% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end

figure(hft);
set(hft,'Name',fig_name);
set(hft,'color',[1 1 1]);
set(hft,'MenuBar','none');
set(hft,'Resize','off');

% Create text box

margins=[10 10 50 50]; % left, right, top, bottom
pos=get(hft,'position');
tbpos=[margins(1) margins(4) pos(3)-margins(1)-margins(2) ...
        pos(4)-margins(3)-margins(4)];
tbpos(tbpos<1)=1;

htb=uicontrol(hft,'style','listbox','position',tbpos,'tag','textbox');
set(htb,'BackgroundColor',[1 1 1]);
set(htb,'SelectionHighlight','off');
set(htb,'FontName','courier');

% Create table head

tpos = [tbpos(1) tbpos(2)+tbpos(4)+10 tbpos(3) 20];
ht=uicontrol(hft,'style','text','position',tpos,'tag','text');
set(ht,'BackgroundColor',[1 1 1]);
set(ht,'HorizontalAlignment','left');
set(ht,'FontName','courier');
newline = '   time         step      order  |    nst   nfe   nni  netf  ncfn';
set(ht,'String',newline);

% Create OK button

bsize=[60,28];
badjustpos=[0,25];
bpos=[pos(3)/2-bsize(1)/2+badjustpos(1) -bsize(2)/2+badjustpos(2)...
       bsize(1) bsize(2)];
bpos=round(bpos);
bpos(bpos<1)=1;
hb=uicontrol(hft,'style','pushbutton','position',bpos,...
             'string','Close','tag','okaybutton');
set(hb,'callback','close');

% Save handles

handles=guihandles(hft);
guidata(hft,handles);
```

```
for i = 1:n
  newline = '';
  if stats
    newline = sprintf('%10.3e   %10.3e     %1d    |',t(i),h(i),q(i));
  end
  if cntr
    newline = sprintf('%s %5d %5d %5d %5d %5d',...
                      newline,nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  end
  string = get(handles.textbox,'String');
  string{end+1}=newline;
  set(handles.textbox,'String',string);
end

drawnow

%----------------------------------------------------------------------------

function [] = text_update(n,hft,stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

figure(hft);

handles=guidata(hft);

for i = 1:n
   if stats
    newline = sprintf('%10.3e   %10.3e     %1d    |',t(i),h(i),q(i));
  end
  if cntr
    newline = sprintf('%s %5d %5d %5d %5d %5d',...
                      newline,nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  end
  string = get(handles.textbox,'String');
  string{end+1}=newline;
  set(handles.textbox,'String',string);
end

drawnow

%----------------------------------------------------------------------------

function [] = sol_init(n, hfs, nps, sol, sensi, N, Ns, t, y, ys)

fig_name = 'CVODES solution';

% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end
```

```


figure(hfs);
set(hfs,'Name',fig_name);
set(hfs,'color',[1 1 1]);

% Time label

tlab = '\rightarrow   t   \rightarrow';

% Get number of colors in colormap
map = colormap;
ncols = size(map,1);

% Initialize current subplot counter
pl = 0;

if sol

  pl = pl+1;
  subplot(nps,1,pl);
  hold on;

  for i = 1:N
    hp = plot(t(1:n),y(i,1:n),'-');
    ic = 1+(i-1)*floor(ncols/N);
    set(hp,'Color',map(ic,:));
  end
  box on;
  grid on;
  xlabel(tlab);
  ylabel('y');
  title('Solution');

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);
    hold on;

    ys_crt = ys(:,is,1:n);
    for i = 1:N
      hp = plot(t(1:n),ys_crt(i,1:n),'-');
      ic = 1+(i-1)*floor(ncols/N);
      set(hp,'Color',map(ic,:));
    end
    box on;
    grid on;
    xlabel(tlab);
    str = sprintf('s_{%d}',is); ylabel(str);
```

```
    str = sprintf('Sensitivity_%d',is); title(str);

  end

end


drawnow;

%--------------------------------------------------------------------------

function [] = sol_update(n, hfs, nps, sol, sensi, N, Ns, t, y, ys)

figure(hfs);

pl = 0;

if sol

  pl = pl+1;
  subplot(nps,1,pl);

  hc = get(gca,'Children');
  xd = [get(hc(1),'XData') t(1:n)];
% Attention: Children are loaded in reverse order!
  for i = 1:N
    yd = [get(hc(i),'YData') y(N-i+1,1:n)];
    set(hc(i), 'XData', xd, 'YData', yd);
  end

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);

    ys_crt = ys(:,is,:);

    hc = get(gca,'Children');
    xd = [get(hc(1),'XData') t(1:n)];
%   Attention: Children are loaded in reverse order!
    for i = 1:N
      yd = [get(hc(i),'YData') ys_crt(N-i+1,1:n)];
      set(hc(i), 'XData', xd, 'YData', yd);
    end

  end

end

```

```
drawnow;


%------------------------------------------------------------------------

function [] = sol_final(hfs, nps, sol, sensi, N, Ns)

figure(hfs);

pl = 0;

if sol

  pl = pl +1;
  subplot(nps,1,pl);

  hc = get(gca,'Children');
  xd = get(hc(1),'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));

  ylim = get(gca,'YLim');
  addon = 0.1*abs(ylim(2)-ylim(1));
  ylim(1) = ylim(1) + sign(ylim(1))*addon;
  ylim(2) = ylim(2) + sign(ylim(2))*addon;
  set(gca,'YLim',ylim);

  for i = 1:N
    cstring{i} = sprintf('y_{%d}',i);
  end
  legend(cstring);

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);

    hc = get(gca,'Children');
    xd = get(hc(1),'XData');
    set(gca,'XLim',sort([xd(1) xd(end)]));

    ylim = get(gca,'YLim');
    addon = 0.1*abs(ylim(2)-ylim(1));
    ylim(1) = ylim(1) + sign(ylim(1))*addon;
    ylim(2) = ylim(2) + sign(ylim(2))*addon;
    set(gca,'YLim',ylim);

    for i = 1:N
      cstring{i} = sprintf('s%d_{%d}',is,i);
    end
    legend(cstring);
```

```
  end

end

drawnow
```

CVodeMonitorB

Purpose

CVodeMonitorB is the default CVODES monitoring function for backward problems.

Synopsis

```
function [new_data] = CVodeMonitorB(call, idxB, T, Y, YQ, data)
```

Description

```
CVodeMonitorB is the default CVODES monitoring function for backward problems.
   To use it, set the Monitor property in CVodeSetOptions to
   'CVodeMonitorB' or to @CVodeMonitorB and 'MonitorData' to mondata
   (defined as a structure).

   With default settings, this function plots the evolution of the step
   size, method order, and various counters.

   Various properties can be changed from their default values by passing
   to CVodeSetOptions, through the property 'MonitorData', a structure
   MONDATA with any of the following fields. If a field is not defined,
   the corresponding default value is used.

   Fields in MONDATA structure:
     o stats [ true | false ]
         If true, report the evolution of the step size and method order.
     o cntr [ true | false ]
         If true, report the evolution of the following counters:
         nst, nfe, nni, netf, ncfn (see CVodeGetStats)
     o mode [ 'graphical' | 'text' | 'both' ]
         In graphical mode, plot the evolutions of the above quantities.
         In text mode, print a table.
     o sol  [ true | false ]
         If true, plot solution components.
     o select [ array of integers ]
         To plot only particular solution components, specify their indeces in
         the field select. If not defined, but sol=true, all components are plotted.
     o updt [ integer | 50 ]
         Update frequency. Data is posted in blocks of dimension n.
     o skip [ integer | 0 ]
         Number of integrations steps to skip in collecting data to post.
     o post [ true | false ]
         If false, disable all posting. This option is necessary to disable
         monitoring on some processors when running in parallel.
```

```
See also CVodeSetOptions, CVMonitorFnB

NOTES:
  1. The argument mondata is REQUIRED. Even if only the default options
     are desired, set mondata=struct; and pass it to CVodeSetOptions.
  2. The yQ argument is currently ignored.
```

B Implementation of IDAMonitor.m

IDAMonitor

PURPOSE

IDAMonitor is the default IDAS monitoring function.

SYNOPSIS

```
function [new_data] = IDAMonitor(call, T, Y, YQ, YS, data)
```

DESCRIPTION

```
IDAMonitor is the default IDAS monitoring function.
   To use it, set the Monitor property in IDASetOptions to
   'IDAMonitor' or to @IDAMonitor and 'MonitorData' to mondata
   (defined as a structure).

   With default settings, this function plots the evolution of the step
   size, method order, and various counters.

   Various properties can be changed from their default values by passing
   to IDASetOptions, through the property 'MonitorData', a structure
   MONDATA with any of the following fields. If a field is not defined,
   the corresponding default value is used.

   Fields in MONDATA structure:
     o stats [ true | false ]
         If true, report the evolution of the step size and method order.
     o cntr [ true | false ]
         If true, report the evolution of the following counters:
         nst, nfe, nni, netf, ncfn (see IDAGetStats)
     o mode [ 'graphical' | 'text' | 'both' ]
         In graphical mode, plot the evolutions of the above quantities.
         In text mode, print a table.
     o sol  [ true | false ]
         If true, plot solution components.
     o sensi [ true | false ]
         If true and if FSA is enabled, plot sensitivity components.
     o select [ array of integers ]
         To plot only particular solution components, specify their indeces in
         the field select. If not defined, but sol=true, all components are plotted.
     o updt [ integer | 50 ]
         Update frequency. Data is posted in blocks of dimension n.
     o skip [ integer | 0 ]
```

```
        Number of integrations steps to skip in collecting data to post.
    o post [ true | false ]
        If false, disable all posting. This option is necessary to disable
        monitoring on some processors when running in parallel.

  See also IDASetOptions, IDAMonitorFn

  NOTES:
    1. The argument mondata is REQUIRED. Even if only the default options
       are desired, set mondata=struct; and pass it to IDASetOptions.
    2. The yQ argument is currently ignored.
```

Source Code

```
function [new_data] = IDAMonitor(call, T, Y, YQ, YS, data)

% Radu Serban <radu@llnl.gov>
% LLNS Copyright Start
% Copyright (c) 2014, Lawrence Livermore National Security
% This work was performed under the auspices of the U.S. Department
% of Energy by Lawrence Livermore National Laboratory in part under
% Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.
% Produced at the Lawrence Livermore National Laboratory.
% All rights reserved.
% For details, see the LICENSE file.
% LLNS Copyright End
% $Revision: 4075 $Date: 2007/08/21 17:38:42 $

if (nargin ~= 6)
  error('Monitor data not defined.');
end

new_data = [];

if call == 0

% Initialize unspecified fields to default values.
  data = initialize_data(data);

% Open figure windows
  if data.post

    if data.grph
      if data.stats | data.cntr
        data.hfg = figure;
      end
%     Number of subplots in figure hfg
      if data.stats
        data.npg = data.npg + 2;
      end
      if data.cntr
        data.npg = data.npg + 1;
      end
    end

```

```
    if data.text
      if data.cntr | data.stats
        data.hft = figure;
      end
    end

    if data.sol | data.sensi
      data.hfs = figure;
    end

  end

% Initialize other private data
  data.i = 0;
  data.n = 1;
  data.t = zeros(1,data.updt);
  if data.stats
    data.h = zeros(1,data.updt);
    data.q = zeros(1,data.updt);
  end
  if data.cntr
    data.nst = zeros(1,data.updt);
    data.nfe = zeros(1,data.updt);
    data.nni = zeros(1,data.updt);
    data.netf = zeros(1,data.updt);
    data.ncfn = zeros(1,data.updt);
  end

  data.first = true;        % the next one will be the first call = 1
  data.initialized = false; % the graphical windows were not initalized

  new_data = data;

  return;

else

% If this is the first call ~= 0,
% use Y and YS for additional initializations

  if data.first

    if isempty(YS)
      data.sensi = false;
    end

    if data.sol | data.sensi

      if isempty(data.select)

        data.N = length(Y);
        data.select = [1:data.N];

      else
```

```

        data.N = length(data.select);

      end

      if data.sol
        data.y = zeros(data.N,data.updt);
        data.nps = data.nps + 1;
      end

      if data.sensi
        data.Ns = size(YS,2);
        data.ys = zeros(data.N, data.Ns, data.updt);
        data.nps = data.nps + data.Ns;
      end

    end

    data.first = false;

  end

% Extract variables from data

  hfg  = data.hfg;
  hft  = data.hft;
  hfs  = data.hfs;
  npg  = data.npg;
  nps  = data.nps;
  i    = data.i;
  n    = data.n;
  t    = data.t;
  N    = data.N;
  Ns   = data.Ns;
  y    = data.y;
  ys   = data.ys;
  h    = data.h;
  q    = data.q;
  nst  = data.nst;
  nfe  = data.nfe;
  nni  = data.nni;
  netf = data.netf;
  ncfn = data.ncfn;

end


% Load current statistics?

if call == 1

  if i ~= 0
    i = i-1;
    data.i = i;
```

```
    new_data = data;
    return;
  end

  si = IDAGetStats;

  t(n) = si.tcur;

  if data.stats
    h(n) = si.hlast;
    q(n) = si.qlast;
  end

  if data.cntr
    nst(n) = si.nst;
    nfe(n) = si.nfe;
    nni(n) = si.nni;
    netf(n) = si.netf;
    ncfn(n) = si.ncfn;
  end

  if data.sol
    for j = 1:N
      y(j,n) = Y(data.select(j));
    end
  end

  if data.sensi
    for k = 1:Ns
      for j = 1:N
        ys(j,k,n) = YS(data.select(j),k);
      end
    end
  end

end

% Is it time to post?

if data.post & (n == data.updt | call==2)

  if call == 2
    n = n-1;
  end

  if ~data.initialized

    if (data.stats | data.cntr) & data.grph
      graphical_init(n, hfg, npg, data.stats, data.cntr, ...
                     t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if (data.stats | data.cntr) & data.text
      text_init(n, hft, data.stats, data.cntr, ...
```

```
                  t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if data.sol | data.sensi
      sol_init(n, hfs, nps, data.sol, data.sensi,  ...
               N, Ns, t, y, ys);
    end

    data.initialized = true;

  else

    if (data.stats | data.cntr) & data.grph
      graphical_update(n, hfg, npg, data.stats, data.cntr, ...
                       t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if (data.stats | data.cntr) & data.text
      text_update(n, hft, data.stats, data.cntr, ...
                  t, h, q, nst, nfe, nni, netf, ncfn);
    end

    if data.sol
      sol_update(n, hfs, nps, data.sol, data.sensi, N, Ns, t, y, ys);
    end

  end

  if call == 2

    if (data.stats | data.cntr) & data.grph
      graphical_final(hfg, npg, data.cntr, data.stats);
    end

    if data.sol | data.sensi
      sol_final(hfs, nps, data.sol, data.sensi, N, Ns);
    end

    return;

  end

  n = 1;

else

  n = n + 1;

end


% Save updated values in data

data.i    = data.skip;
```

```
data.n     = n;
data.npg   = npg;
data.t     = t;
data.y     = y;
data.ys    = ys;
data.h     = h;
data.q     = q;
data.nst   = nst;
data.nfe   = nfe;
data.nni   = nni;
data.netf  = netf;
data.ncfn  = ncfn;

new_data = data;

return;

%-------------------------------------------------------------------------

function data = initialize_data(data)

if ~isfield(data,'mode')
  data.mode = 'graphical';
end
if ~isfield(data,'updt')
  data.updt = 50;
end
if ~isfield(data,'skip')
  data.skip = 0;
end
if ~isfield(data,'stats')
  data.stats = true;
end
if ~isfield(data,'cntr')
  data.cntr = true;
end
if ~isfield(data,'sol')
  data.sol = false;
end
if ~isfield(data,'sensi')
  data.sensi = false;
end
if ~isfield(data,'select')
  data.select = [];
end
if ~isfield(data,'post')
  data.post = true;
end

data.grph = true;
data.text = true;
if strcmp(data.mode,'graphical')
  data.text = false;
end
```

```
if strcmp(data.mode,'text')
  data.grph = false;
end

if ~data.sol & ~data.sensi
  data.select = [];
end

% Other initializations
data.npg = 0;
data.nps = 0;
data.hfg = 0;
data.hft = 0;
data.hfs = 0;
data.h = 0;
data.q = 0;
data.nst = 0;
data.nfe = 0;
data.nni = 0;
data.netf = 0;
data.ncfn = 0;
data.N = 0;
data.Ns = 0;
data.y = 0;
data.ys = 0;

%------------------------------------------------------------------------------

function [] = graphical_init(n, hfg, npg, stats, cntr, ...
                              t, h, q, nst, nfe, nni, netf, ncfn)

fig_name = 'IDAS run statistics';

% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end

figure(hfg);
set(hfg,'Name',fig_name);
set(hfg,'color',[1 1 1]);
pl = 0;

% Time label and figure title

tlab = '\rightarrow   t   \rightarrow';

% Step size and order
if stats
  pl = pl+1;
```

```
  subplot(npg,1,pl)
  semilogy(t(1:n),abs(h(1:n)),'-');
  hold on;
  box on;
  grid on;
  xlabel(tlab);
  ylabel('|Step size|');

  pl = pl+1;
  subplot(npg,1,pl)
  plot(t(1:n),q(1:n),'-');
  hold on;
  box on;
  grid on;
  xlabel(tlab);
  ylabel('Order');
end

% Counters
if cntr
  pl = pl+1;
  subplot(npg,1,pl)
  plot(t(1:n),nst(1:n),'k-');
  hold on;
  plot(t(1:n),nfe(1:n),'b-');
  plot(t(1:n),nni(1:n),'r-');
  plot(t(1:n),netf(1:n),'g-');
  plot(t(1:n),ncfn(1:n),'c-');
  box on;
  grid on;
  xlabel(tlab);
  ylabel('Counters');
end

drawnow;

%------------------------------------------------------------------------------

function [] = graphical_update(n, hfg, npg, stats, cntr, ...
                               t, h, q, nst, nfe, nni, netf, ncfn)

figure(hfg);
pl = 0;

% Step size and order
if stats
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n)];
  yd = [get(hc,'YData') abs(h(1:n))];
  set(hc, 'XData', xd, 'YData', yd);

  pl = pl+1;
```

```
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n)];
  yd = [get(hc,'YData') q(1:n)];
  set(hc, 'XData', xd, 'YData', yd);
end

% Counters
if cntr
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
% Attention: Children are loaded in reverse order!
  xd = [get(hc(1),'XData') t(1:n)];
  yd = [get(hc(1),'YData') ncfn(1:n)];
  set(hc(1), 'XData', xd, 'YData', yd);
  yd = [get(hc(2),'YData') netf(1:n)];
  set(hc(2), 'XData', xd, 'YData', yd);
  yd = [get(hc(3),'YData') nni(1:n)];
  set(hc(3), 'XData', xd, 'YData', yd);
  yd = [get(hc(4),'YData') nfe(1:n)];
  set(hc(4), 'XData', xd, 'YData', yd);
  yd = [get(hc(5),'YData') nst(1:n)];
  set(hc(5), 'XData', xd, 'YData', yd);
end

drawnow;

%------------------------------------------------------------------------------

function [] = graphical_final(hfg,npg,stats,cntr)

figure(hfg);
pl = 0;

if stats
  pl = pl+1;
  subplot(npg,1,pl)
  hc = get(gca,'Children');
  xd = get(hc,'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));

  pl = pl+1;
  subplot(npg,1,pl)
  ylim = get(gca,'YLim');
  ylim(1) = ylim(1) - 1;
  ylim(2) = ylim(2) + 1;
  set(gca,'YLim',ylim);
  set(gca,'XLim',sort([xd(1) xd(end)]));
end

if cntr
  pl = pl+1;
  subplot(npg,1,pl)
```

```
  hc = get(gca,'Children');
  xd = get(hc(1),'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));
  legend('nst','nfe','nni','netf','ncfn',2);
end

%---------------------------------------------------------------------------

function [] = text_init(n,hft,stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

fig_name = 'IDAS run statistics';

% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end

figure(hft);
set(hft,'Name',fig_name);
set(hft,'color',[1 1 1]);
set(hft,'MenuBar','none');
set(hft,'Resize','off');

% Create text box

margins=[10 10 50 50]; % left, right, top, bottom
pos=get(hft,'position');
tbpos=[margins(1) margins(4) pos(3)-margins(1)-margins(2) ...
       pos(4)-margins(3)-margins(4)];
tbpos(tbpos<1)=1;

htb=uicontrol(hft,'style','listbox','position',tbpos,'tag','textbox');
set(htb,'BackgroundColor',[1 1 1]);
set(htb,'SelectionHighlight','off');
set(htb,'FontName','courier');

% Create table head

tpos = [tbpos(1) tbpos(2)+tbpos(4)+10 tbpos(3) 20];
ht=uicontrol(hft,'style','text','position',tpos,'tag','text');
set(ht,'BackgroundColor',[1 1 1]);
set(ht,'HorizontalAlignment','left');
set(ht,'FontName','courier');
newline = '   time         step      order  |    nst   nfe   nni  netf  ncfn';
set(ht,'String',newline);

% Create OK button

bsize=[60,28];
badjustpos=[0,25];
```

```
bpos=[pos(3)/2-bsize(1)/2+badjustpos(1) -bsize(2)/2+badjustpos(2)...
       bsize(1) bsize(2)];
bpos=round(bpos);
bpos(bpos<1)=1;
hb=uicontrol(hft,'style','pushbutton','position',bpos,...
              'string','Close','tag','okaybutton');
set(hb,'callback','close');

% Save handles

handles=guihandles(hft);
guidata(hft,handles);

for i = 1:n
  newline = '';
  if stats
    newline = sprintf('%10.3e   %10.3e     %1d    |',t(i),h(i),q(i));
  end
  if cntr
    newline = sprintf('%s %5d %5d %5d %5d %5d',...
                      newline,nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  end
  string = get(handles.textbox,'String');
  string{end+1}=newline;
  set(handles.textbox,'String',string);
end

drawnow

%------------------------------------------------------------------------------

function [] = text_update(n,hft,stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

figure(hft);

handles=guidata(hft);

for i = 1:n
   if stats
    newline = sprintf('%10.3e   %10.3e     %1d    |',t(i),h(i),q(i));
  end
  if cntr
    newline = sprintf('%s %5d %5d %5d %5d %5d',...
                      newline,nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  end
  string = get(handles.textbox,'String');
  string{end+1}=newline;
  set(handles.textbox,'String',string);
end

drawnow

%------------------------------------------------------------------------------
```

```
function [] = sol_init(n, hfs, nps, sol, sensi, N, Ns, t, y, ys)

fig_name = 'IDAS solution';

% If this is a parallel job, look for the MPI rank in the global
% workspace and append it to the figure name

global sundials_MPI_rank

if ~isempty(sundials_MPI_rank)
  fig_name = sprintf('%s (PE %d)',fig_name,sundials_MPI_rank);
end


figure(hfs);
set(hfs,'Name',fig_name);
set(hfs,'color',[1 1 1]);

% Time label

tlab = '\rightarrow   t   \rightarrow';

% Get number of colors in colormap
map = colormap;
ncols = size(map,1);

% Initialize current subplot counter
pl = 0;

if sol

  pl = pl+1;
  subplot(nps,1,pl);
  hold on;

  for i = 1:N
    hp = plot(t(1:n),y(i,1:n),'-');
    ic = 1+(i-1)*floor(ncols/N);
    set(hp,'Color',map(ic,:));
  end
  box on;
  grid on;
  xlabel(tlab);
  ylabel('y');
  title('Solution');

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);
```

```
    hold on;

    ys_crt = ys(:,is,1:n);
    for i = 1:N
      hp = plot(t(1:n),ys_crt(i,1:n),'-');
      ic = 1+(i-1)*floor(ncols/N);
      set(hp,'Color',map(ic,:));
    end
    box on;
    grid on;
    xlabel(tlab);
    str = sprintf('s_{%d}',is); ylabel(str);
    str = sprintf('Sensitivity %d',is); title(str);

  end

end


drawnow;

%-----------------------------------------------------------------------------------------

function [] = sol_update(n, hfs, nps, sol, sensi, N, Ns, t, y, ys)

figure(hfs);

pl = 0;

if sol

  pl = pl+1;
  subplot(nps,1,pl);

  hc = get(gca,'Children');
  xd = [get(hc(1),'XData') t(1:n)];
% Attention: Children are loaded in reverse order!
  for i = 1:N
    yd = [get(hc(i),'YData') y(N-i+1,1:n)];
    set(hc(i), 'XData', xd, 'YData', yd);
  end

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);

    ys_crt = ys(:,is,:);

    hc = get(gca,'Children');
```

```
    xd = [get(hc(1),'XData') t(1:n)];
%   Attention: Children are loaded in reverse order!
    for i = 1:N
      yd = [get(hc(i),'YData') ys_crt(N-i+1,1:n)];
      set(hc(i), 'XData', xd, 'YData', yd);
    end

  end

end


drawnow;


%---------------------------------------------------------------------------

function [] = sol_final(hfs, nps, sol, sensi, N, Ns)

figure(hfs);

pl = 0;

if sol

  pl = pl +1;
  subplot(nps,1,pl);

  hc = get(gca,'Children');
  xd = get(hc(1),'XData');
  set(gca,'XLim',sort([xd(1) xd(end)]));

  ylim = get(gca,'YLim');
  addon = 0.1*abs(ylim(2)-ylim(1));
  ylim(1) = ylim(1) + sign(ylim(1))*addon;
  ylim(2) = ylim(2) + sign(ylim(2))*addon;
  set(gca,'YLim',ylim);

  for i = 1:N
    cstring{i} = sprintf('y_{%d}',i);
  end
  legend(cstring);

end

if sensi

  for is = 1:Ns

    pl = pl+1;
    subplot(nps,1,pl);

    hc = get(gca,'Children');
    xd = get(hc(1),'XData');
```

```
    set(gca,'XLim',sort([xd(1) xd(end)]));

    ylim = get(gca,'YLim');
    addon = 0.1*abs(ylim(2)-ylim(1));
    ylim(1) = ylim(1) + sign(ylim(1))*addon;
    ylim(2) = ylim(2) + sign(ylim(2))*addon;
    set(gca,'YLim',ylim);

    for i = 1:N
      cstring{i} = sprintf('s%d_{%d}',is,i);
    end
    legend(cstring);

  end

end

drawnow
```

IDAMonitorB

Purpose

```
IDAMonitorB is the default IDAS monitoring function for backward problems.
```

Synopsis

```
function [new_data] = IDAMonitorB(call, idxB, T, Y, YQ, data)
```

Description

```
IDAMonitorB is the default IDAS monitoring function for backward problems.
   To use it, set the Monitor property in IDASetOptions to
   'IDAMonitorB' or to @IDAMonitorB and 'MonitorData' to mondata
   (defined as a structure).

   With default settings, this function plots the evolution of the step
   size, method order, and various counters.

   Various properties can be changed from their default values by passing
   to IDASetOptions, through the property 'MonitorData', a structure
   MONDATA with any of the following fields. If a field is not defined,
   the corresponding default value is used.

   Fields in MONDATA structure:
     o stats [ true | false ]
         If true, report the evolution of the step size and method order.
     o cntr [ true | false ]
         If true, report the evolution of the following counters:
         nst, nfe, nni, netf, ncfn (see IDAGetStats)
     o mode [ 'graphical' | 'text' | 'both' ]
         In graphical mode, plot the evolutions of the above quantities.
         In text mode, print a table.
     o sol  [ true | false ]
```

```
      If true, plot solution components.
  o select [ array of integers ]
      To plot only particular solution components, specify their indeces in
      the field select. If not defined, but sol=true, all components are plotted.
  o updt [ integer | 50 ]
      Update frequency. Data is posted in blocks of dimension n.
  o skip [ integer | 0 ]
      Number of integrations steps to skip in collecting data to post.
  o post [ true | false ]
      If false, disable all posting. This option is necessary to disable
      monitoring on some processors when running in parallel.

See also IDASetOptions, IDAMonitorFnB

NOTES:
  1. The argument mondata is REQUIRED. Even if only the default options
     are desired, set mondata=struct; and pass it to IDASetOptions.
  2. The yQ argument is currently ignored.
```

References

[1] A. M. Collier, A. C. Hindmarsh, R. Serban, and C.S. Woodward. User Documentation for KINSOL v2.6.0. Technical Report UCRL-SM-208116, LLNL, 2009.

[2] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS, suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Soft.*, (31):363–396, 2005.

[3] A. C. Hindmarsh and R. Serban. User Documentation for CVODES v2.6.0. Technical report, LLNL, 2009. UCRL-SM-208111.

[4] R. Serban, C. Petra, and A.C. Hindmarsh. User Documentation for IDAS v1.0.0. Technical report, LLNL, 2009. UCRL-SM-234051.

Index