Example Programs for IDAS v3.0.1

Radu Serban and Alan C. Hindmarsh
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

December 18, 2018

LLNL-TR-437091

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

This report is intended to serve as a companion document to the User Documentation of IDAS [4]. It provides details, with listings, on the example programs supplied with the IDAS distribution package.

The IDAS distribution contains examples of the following types: serial and parallel examples of Initial Value Problem (IVP) integration, serial and parallel examples of forward sensitivity analysis (FSA), and serial and parallel examples of adjoint sensitivity analysis (ASA). The names of all the examples are given in the following table.

	Serial examples	Parallel examples

IVP	`idasRoberts_dns`
	`idasRoberts_klu` `idasRoberts_sps`
	`idasAkzoNob_dns` `idasSlCrank_dns`
	`idasHeat2D_bnd` `idasHeat2D_kry`
	`idasFoodWeb_bnd` `idasFoodWeb_bnd_omp`
	`idasFoodWeb_kry_omp` `idasKrylovDemo_ls`
		`idasHeat2D_kry_p`
		`idasHeat2D_kry_bbd_p`
		`idasFoodWeb_kry_p`
		`idasFoodWeb_kry_bbd_p`
		`idasBruss_kry_bbd_p`

FSA	`idasRoberts_FSA_dns`
	`idasRoberts_FSA_klu` `idasRoberts_FSA_sps`
	`idasSlCrank_FSA_dns`
		`idasBruss_FSA_kry_bbd_p`
		`idasHeat2D_FSA_kry_bbd_p`

ASA	`idasRoberts_ASAi_dns`
	`idasRoberts_ASAi_klu` `idasRoberts_ASAi_sps`
	`idasAkzoNob_ASAi_dns` `idasHessian_ASA_FSA`
		`idasBruss_ASAp_kry_bbd_p`

With the exception of "demo"-type example files, the names of all the examples distributed with SUNDIALS are of the form `[slv][PbName]_[SA]_[ls]_[prec]_[p]`, where

[slv] identifies the solver (for IDAS examples this is `idas`);

[PbName] identifies the problem;

[SA] identifies sensitivity analysis examples. This field can be one of: `FSA` for forward sensitivity examples, `ASAi` for adjoint sensitivity examples using an integral-form model output, or `ASAp` for adjoint sensitivity examples using an pointwise model output;

[ls] identifies the linear solver module used;

[prec] indicates the IDAS preconditioner module used (if applicable — for examples using a Krylov linear solver and the IDABBDPRE module, this will be `bbd`);

[p] indicates an example using the parallel vector module NVECTOR_PARALLEL.

The examples are briefly described next. Note that the IDAS distribution includes all of the IDA C examples (denoted here as examples for IVP integration). More details on these can be found in the IDA Example Program document [2].

Supplied in the *srcdir*`/examples/idas/serial` directory are the following serial examples (using the NVECTOR_SERIAL module):

- `idasRoberts_dns` solves the Robertson chemical kinetics problem [3], which consists of two differential equations and one algebraic constraint. It also uses the rootfinding feature of IDAS.

  The problem is solved with the SUNLINSOL_DENSE linear solver using a user-supplied Jacobian.

- `idasRoberts_klu` is the same as `idasRoberts_dns` but uses the KLU sparse direct linear solver.

- `idasRoberts_sps` is the same as `idasRoberts_dns` but uses the SuperLUMT sparse direct linear solver (with one thread).

- `idasAkzoNob_dns` solves the Akzo-Nobel chemical kinetics problem, which consists of six nonlinear DAEs of index 1. The problem originates from Akzo Nobel Central research in Arnhern, The Netherlands, and describes a chemical process in which two species are mixed, while carbon dioxide is continuously added.

  The problem is solved with the SUNLINSOL_DENSE linear solver using the default difference quotient dense Jacobian approximation.

- `idasHeat2D_bnd` solves a 2-D heat equation, semidiscretized to a DAE on the unit square.

  This program solves the problem with the SUNLINSOL_BAND linear solver and the default difference-quotient Jacobian approximation. For purposes of illustration, `IDACalcIC` is called to compute correct values at the boundary, given incorrect values as input initial guesses. The constraint $u > 0.0$ is imposed for all components.

- `idasHeat2D_kry` solves the same 2-D heat equation problem as `idasHeat2D_bnd`, with the Krylov linear solver SUNLINSOL_SPGMR. The preconditioner uses only the diagonal elements of the Jacobian.

- `idasFoodWeb_bnd` solves a system of PDEs modeling a food web problem, with predator-prey interaction and diffusion, on the unit square in 2-D.

  The PDEs are discretized in space to a system of DAEs which are solved using the SUNLINSOL_BAND linear solver with the default difference-quotient Jacobian approximation.

- `idasSlCrank_dns` solves a system of index-2 DAEs, modeling a planar slider-crank mechanism.

  The problem is obtained through a stabilized index reduction (Gear-Gupta-Leimkuhler) starting from the index-3 DAE equations of motion derived using three generalized coordinates and two algebraic position constraints. The program also computes the time-averaged kinetic energy as a quadrature.

- `idasKrylovDemo_ls` solves the same problem as `idasHeat2D_kry`, with three Krylov linear solvers SUNLINSOL_SPGMR, SUNLINSOL_SPBCGS, and SUNLINSOL_SPTFQMR. The preconditioner uses only the diagonal elements of the Jacobian.

- `idasRoberts_FSA_dns` solves the same kinetics problem as in `idasRoberts_dns`. IDAS also computes both its solution and solution sensitivities with respect to the three

reaction rate constants appearing in the model. This program solves the problem with the SUNLINSOL_DENSE linear solver, and a user-supplied Jacobian routine.

- `idasRoberts_FSA_klu` solves the same problem as in `idasRoberts_FSA_dns` but uses the sparse direct solver KLU.

- `idasRoberts_FSA_sps` solves the same problem as in `idasRoberts_FSA_dns` but uses the sparse solver SuperLUMT.

- `idasSlCrank_FSA_dns` solves a system of index-2 DAEs, modeling a planar slider-crank mechanism.

  This example computes both its solution and solution sensitivities with respect to the problem parameters k (spring constant) and c (damper constant), and then uses them to evaluate the gradient of the cumulative kinetic energy of the system.

- `idasRoberts_ASAi_dns` solves the same kinetics problem as in `idasRoberts_dns`.
  Here the adjoint capability of IDAS is used to compute gradients of a functional of the solution with respect to the three reaction rate constants appearing in the model. This program solves both the forward and backward problems with the SUNLINSOL_DENSE linear solver, and user-supplied Jacobian routines.

- `idasRoberts_ASAi_klu` solves the same problem as in `idasRoberts_ASAi_dns`, but uses the sparse direct solver KLU.

- `idasRoberts_ASAi_sps` solves the same problem as in `idasRoberts_ASAi_dns`, but uses the sparse solver SuperLUMT.

- `idasAkzoNob_ASAi_dns` solves the Akzo-Nobel chemical kinetics problem.
  The adjoint capability of IDAS is used to compute gradients with respect to the initial conditions of the integral over time of the concentration of the first species.

- `idasHessian_ASA_FSA` is an example of using the *forward-over-adjoint* method for computing 2nd-order derivative information, in the form of Hessian-times-vector products.

Supplied in the *srcdir*`/examples/idas/parallel` directory are the following parallel examples (using the NVECTOR_PARALLEL module):

- `idasHeat2D_kry_p` solves the same 2-D heat equation problem as `idasHeat2D_kry`, with SUNLINSOL_SPGMR in parallel, and with a user-supplied diagonal preconditioner,

- `idasHeat2D_kry_bbd_p` solves the same problem as `idasHeat2D_kry_p`.

  This program uses the SUNLINSOL_SPGMR linear solver in parallel, and the band-block-diagonal preconditioner IDABBDPRE with half-bandwidths equal to 1.

- `idasFoodWeb_kry_p` solves the same food web problem as `idasFoodWeb_bnd`, but with SUNLINSOL_SPGMR and a user-supplied preconditioner.

  The preconditioner supplied to SUNLINSOL_SPGMR is the block-diagonal part of the Jacobian with $n_s \times n_s$ blocks arising from the reaction terms only (n_s = number of species).

- `idasFoodWeb_kry_bbd_p` solves the same food web problem as `idasFoodWeb_kry_p`.

  This program solves the problem using SUNLINSOL_SPGMR in parallel and the IDABBDPRE preconditioner.

- `idasBruss_kry_bbd_p` solves the two-species time-dependent PDE known as the Brusselator problem, using the SUNLINSOL_SPGMR linear solver and the IDABBDPRE preconditioner.

  The PDEs are discretized by central differencing on a 2D spatial mesh. The system is actually implemented on submeshes, processor by processor.

- `idasBruss_FSA_kry_bbd_p` solves the Brusselator problem with the forward sensitivity capability in IDAS used to compute solution sensitivities with respect to two of the problem parameters, and then the gradient of a model output functional, written as the final time value of the spatial integral of the first PDE component.

- `idasHeat2D_FSA_kry_bbd_p` solves the same problem as `idaHeat2D_kry_p`, but using the IDABBDPRE preconditioner, and with forward sensitivity enabled to compute the solution sensitivity with respect to two coefficients of the original PDE.

- `idasBruss_ASAp_kry_bbd_p` solves the same problem as `idasBruss_FSA_kry_bbd_p` but using an adjoint sensitivity approach for computing the gradient of the model output functional.

Supplied in the *srcdir*`/examples/idas/C_openmp` directory are the following examples, using the OpenMP NVECTOR module:

- `idasFoodWeb_bnd_omp` solves the same problem as in `idasFoodWeb_bnd` but uses the OpenMP module.

- `idasFoodWeb_kry_omp` solves the same problem as in `idasFoodWeb_kry` but uses the OpenMP module.

In the following sections, we give detailed descriptions of some (but not all) of the sensitivity analysis examples. We do not discuss the examples for IVP integration; for those, the interested reader should consult the IDA Examples document [2]. Any IDA problem will work with IDAS with only two modifications: (1) the main program should include the header file `idas.h` instead of `ida.h`, and (2) the loader command must reference *builddir*`/lib/libsundials_idas.`*lib* instead of *builddir*`/lib/libsundials_ida.`*lib*.

We also give our output files for each of thes examples described below, but users should be cautioned that their results may differ slightly from these. Differences in solution values may differ within the tolerances, and differences in cumulative counters, such as numbers of steps or Newton iterations, may differ from one machine environment to another by as much as 10% to 20%.

In the descriptions below, we make frequent references to the IDAS User Guide [4]. All citations to specific sections (e.g. §4.2) are references to parts of that user guide, unless explicitly stated otherwise.

Note The examples in the IDAS distribution were written in such a way as to compile and run for any combination of configuration options during the installation of SUNDIALS (see Appendix A in the User Guide). As a consequence, they contain portions of code that will not typically be present in a user program. For example, all example programs make use of the variables `SUNDIALS_EXTENDED_PRECISION` and `SUNDIALS_DOUBLE_PRECISION` to test if the solver libraries were built in extended or double precision, and use the appropriate conversion specifiers in `printf` functions. Similarly, all forward sensitivity examples can be run with or without sensitivity computations enabled and, in the former case, with various combinations of methods and error control strategies. This is achieved in these example through the program arguments.

2 Forward sensitivity analysis example problems

For all the IDAS examples, either of the two sensitivity method options, `IDA_SIMULTANEOUS` or `IDA_STAGGERED`, can be used, and sensitivities may be included in the error test or not (`errconS` set to `SUNTRUE` or `SUNFALSE`, respectively, in the call to `IDASetSensErrCon`).

Descriptions of one serial example (`idasSlCrank_FSA_dns`) and one parallel example (`idasBruss_FSA_kry_bbd_p`) are provided in the following two subsections. For details on the other examples, the reader is directed to the comments in their source files.

2.1 A serial dense example: idasSlCrank_FSA_dns

To illustrate the use of IDAS in a forward sensitivity analysis (FSA) problem, using the serial vector representation, we present in this section a problem from multibody system dynamics. Besides introducing the FSA capabilities of IDAS, this example also illustrates the proper treatment of such problems within IDA and IDAS (a stabilized index reduction is required).

The multibody system considered here consists of two bodies (crank and connecting rod) with a translational-spring-damper (TSD) and a constant force acting on the connecting rod. The system has a single degree of freedom. It is modeled with the three generalized coordinates indicated in Fig. 1 (crank angle, horizontal position of the translational joint, and angle of the connecting rod) and therefore has two constraints. The local reference frame on the crank is positioned at the revolute joint on the ground. The crank has length a, mass m_1, and moment of inertia J_1 (with respect to the local frame). The local reference frame on the connecting rod is positioned at the translational joint. The connecting rod has length 2, mass m_2, and moment of inertia J_2. The TSD has spring constant k, damping constant c, and free length l_0. A constant horizontal force F acts on the connecting rod.

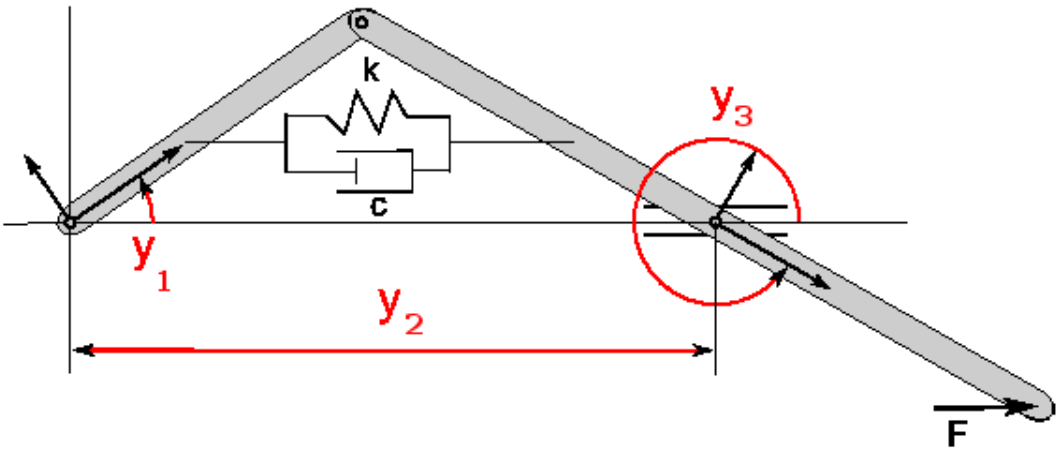

Figure 1: Slider-crank mechanism modeled with three generalized coordinates.

The equations of motion can be written as

$$M(y)\ddot{y} = Q(y,\dot{y}) - \Phi_y^T(y)\lambda$$
$$\Phi(y) = 0 \ ,$$

where $y \in R^3$ is the vector of generalized coordinates, $M(y)$ is the generalized mass matrix, and Q is a vector of generalized applied forces. $\Phi(y) \in R^2$ represents the (algebraic) position-level constraints and Φ_y is its Jacobian with respect to y. $\lambda \in R^2$ are Lagrange multipliers corresponding to the constraint forces. For its solution with IDAS, the above index-3 DAE is reformulated as a stabilized index-2 DAE (Gear-Gupta-Leimkuhler formulation, [1]) by introducing two additional Lagrange multipliers μ and appending the velocity constraints. Converting to first order differential equations, we obtain:

$$\begin{aligned} \dot{y} &= v - \Phi_y^T(y)\mu \\ M(y)\dot{v} &= Q(y,v) - \Phi_y^T(y)\lambda \\ \Phi(y) &= 0 \\ \Phi_y(y)v &= 0 \ , \end{aligned} \tag{1}$$

where $v = \dot{y}$ are the generalized velocities.

For the mechanical system under consideration, the position constraints can be written as

$$\Phi(y) = \begin{bmatrix} y_2 - a\cos(y_1) - a\cos(y_3) \\ a\sin(y_1) + \sin(y_3) \end{bmatrix}$$

while the generalized force takes the form

$$Q(y,v) = \begin{bmatrix} -(f/\ell)a[\sin(y_3 - y_1)/2 + y_2\sin(y_1)]/2 \\ (f/\ell)[\cos(y_3)/2 - y_2 + a\cos(y_1)/2] + F \\ -(f/\ell)[y_2\sin(y_3) - a\sin(y_3 - y_1)/2]/2 - F\sin(y_3) \end{bmatrix} \ ,$$

where

$$\begin{aligned} f =& k(\ell - \ell_0) + c\ell' \\ \ell^2 =& y_2^2 - y_2[\cos(y_3) + a\cos(y_1)] + (1 + a^2)/4 + a\cos(y_3 - y_1)/2 \\ 2\ell\ell' =& 2y_2v_2 - v_2[\cos(y_3) + a\cos(y_1)] + y_2[\sin(y_3)v_3 + a\sin(y_1)v_1] \\ & - a\sin(y_3 - y_1)(v_3 - v_1)/2 \ . \end{aligned}$$

The generalized mass matrix is diagonal: $M = diag\{J_1, m_2, J_2\}$.

In the case treated here, $a = .5$, $J_1 = 1$, $J_2 = 2$, $m_1 = m_2 = 1$, $F = 1$, $k = 1$, $c = 1$, and $\ell_0 = 1$. The final time is $t_f = 10$.

The system (1) is solved with IDAS using a state vector $Y = [y, v, \lambda, \mu] \in R^{10}$. The initial conditions (at $t = 0$) are set to consistent values, given as follows:

$$\begin{aligned} & y_1 = \pi/2 \\ & y_3 = \arcsin(-a) \\ & y_2 = \cos(y_3) \\ & v_1 = v_2 = v_3 = 0 \\ & \lambda_1 = \lambda_2 = \mu_1 = \mu_2 = 0 \\ & dy_1/dt = dy_2/dt = dy_3/dt = 0 \\ & dv_1/dt = [Q_1]_{t=0}/J_1 \\ & dv_2/dt = [Q_2]_{t=0}/m_2 \\ & dv_3/dt = [Q_3]_{t=0}/J_2 \\ & d\lambda_1/dt = d\lambda_2/dt = d\mu_1/dt = d\mu_2/dt = 0 \ . \end{aligned}$$

The problem is solved with a relative tolerance of 10^{-6} and a (scalar) absolute tolerance of 10^{-7}. Note that the algebraic variables λ and μ are excluded from the error test (by specifying them through `IDASetId` and invoking `IDASetSuppressAlg`).

The two parameters of the TSD, k and c, are considered in a forward sensitivity analysis of this model. Solution sensitivities with respect to those parameters are computed and then used to estimate the gradient of the integrated kinetic energy of the system,

$$G = \int_{t_0}^{t_f} \left(\frac{1}{2} J_1 \dot{y}_1^2 + \frac{1}{2} m_2 \dot{y}_2^2 + \frac{1}{2} J_2 \dot{y}_3^2 \right) dt \,. \tag{2}$$

This is then compared against gradient approximations based on (backward, forward, and central) finite differences. The sensitivity residuals are evaluated using the IDAS internal finite-difference approximation. Computation of the gradient of the integral in G takes advantage of the IDAS feature for computing sensitivities of pure quadrature equations.

Figure 2 shows the sensitivities of the horizontal position of the translational joint ($x = y_2$) with respect to the TSD parameters k and c, superimposed over the solution itself.

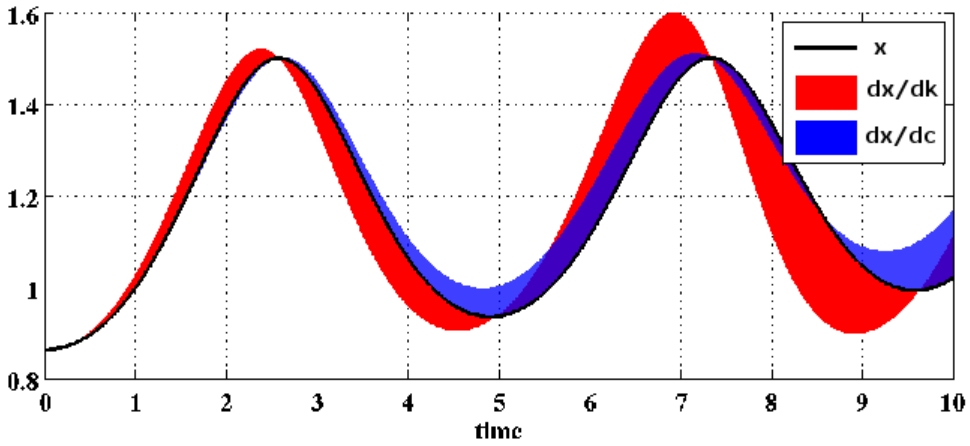

Figure 2: Sensitivities of the solution component y_2 with respect to the TSD parameters.

The following output is generated by `idasSlCrank_FSA_dns` when computing sensitivities with the `IDA_SIMULTANEOUS` method and full error control:

idasSlCrank_FSA_dns sample output

```
Slider-Crank example for IDAS:

Forward integration ... done!

Final Run Statistics:

Number of steps                    = 231
Number of residual evaluations     = 1131
Number of Jacobian evaluations     = 42
Number of nonlinear iterations     = 711
Number of error test failures      = 0
Number of nonlinear conv. failures = 0
-----------------------------------------
  G =       3.3366157997761721
```

```
-----------------------------------------

-------------F O R W A R D------------------
   dG/dp:    3.3346e-01  -3.6375e-01
-----------------------------------------



   Checking using Finite Differences

---------------BACKWARD-------------------
   dG/dp:    3.3344e-01  -3.6375e-01
-----------------------------------------

---------------FORWARD--------------------
   dG/dp:    3.3345e-01  -3.6375e-01
-----------------------------------------

--------------CENTERED--------------------
   dG/dp:    3.3345e-01  -3.6375e-01
-----------------------------------------
```

2.2 A parallel example using IDABBDPRE: idasBruss_FSA_kry_bbd_p

The `idasBruss_FSA_kry_bbd_p` program solves the two-species time-dependent PDE known as the Brusselator problem, using the SUNLINSOL_SPGMR linear solver and the IDABBDPRE preconditioner.

With subscripts on u and v denoting partial derivatives, the PDEs are as follows:

$$\partial u/\partial t = \epsilon_1(u_{xx} + u_{yy}) + u^2 v - (B+1)u + A$$
$$\partial v/\partial t = \epsilon_2(v_{xx} + v_{yy}) - u^2 v + Bu$$

on the unit square in (x, y), and for $0 \le t \le t_f = 1$. The constants involved are $\epsilon_1 = \epsilon_2 = 0.002$, $A = 1$, and $B = 3.4$. The boundary conditions are Neumann (zero derivatives). The initial conditions are given by:

$$u = 1 - 0.5\cos(\pi y)$$
$$v = 3.5 - 2.5\cos(\pi x)$$

The PDEs are discretized by central differencing on a uniform 2D spatial mesh. The boundary conditions are handled by copying values from the first interior mesh line to a line of ghost values on each side of the square. The system is actually implemented on submeshes, processor by processor.

Here the forward sensitivity capability in IDAS is used to compute solution sensitivities with respect the two parameters ϵ_i. From those, we compute the corresponding sensitivities of the final spatial average of u,

$$g = \int\int u(x, y, t_f)\ dx\ dy$$

by means of a spatial integration of the sensitivities:

$$dg/d\epsilon_i = \int\int \partial u(x, y, t_f)/\partial\epsilon_i\ dx\ dy\ .$$

The following output is generated by `idasBruss_FSA_kry_bbd_p` when computing sensitivities with the `IDA_SIMULTANEOUS` method and full error control:

```
mpirun -np 4 idasBruss_FSA_kry_bbd_p -sensi sim t
```

idasBruss_FSA_kry_bbd_p sample output

```
 Brusselator PDE -  DAE parallel example problem for IDA

Number of species ns: 2     Mesh dimensions: 82 x 82
Total system size: 13448
Subgrid dimensions: 41 x 41     Processor array: 2 x 2
Tolerance parameters:  rtol = 1e-05   atol = 1e-05
Linear solver: SUNLinSol_SPGMR     Max. Krylov dimension maxl: 16
Preconditioner: band-block-diagonal (IDABBDPRE), with parameters
     mudq = 82,  mldq = 82,  mukeep = 2,  mlkeep = 2
CalcIC called to correct initial concentrations

------------------------------------------------------------
  t        bottom-left  top-right    | nst  k      h
------------------------------------------------------------

0.00e+00   5.0038e-01   1.4996e+00   |   0  0   1.0000e-06
           1.0019e+00   5.9981e+00   |

1.00e-03   4.9944e-01   1.5076e+00   |  11  1   5.1200e-04
           1.0034e+00   5.9896e+00   |

1.00e-02   4.9119e-01   1.5832e+00   |  17  2   2.0480e-03
           1.0170e+00   5.9082e+00   |

1.00e-01   4.2223e-01   3.0684e+00   |  31  4   7.4353e-03
           1.1419e+00   4.3097e+00   |

4.00e-01   3.0652e-01   5.4104e+00   |  72  5   7.4353e-03
           1.4714e+00   6.1133e-01   |

7.00e-01   2.7048e-01   4.1053e+00   | 112  5   7.4353e-03
           1.7403e+00   7.8907e-01   |

1.00e+00   2.6100e-01   3.1024e+00   | 152  5   7.4353e-03
           1.9881e+00   1.0113e+00   |

------------------------------------------------------------

Final statistics:

Number of steps                    = 152
Number of residual evaluations     = 972
Number of nonlinear iterations     = 164
Number of error test failures      = 1
Number of nonlinear conv. failures = 0

Number of linear iterations        = 806
Number of linear conv. failures    = 0

Number of preconditioner setups    = 15
Number of preconditioner solves    = 1304
Number of local residual evals.    = 2490
```

```
The average of u on the domain:
g = 1.62453

Sensitivities of g:
w.r.t. eps0 =  16.1571227152
w.r.t. eps1 =  -3.7188364999
```

3 Adjoint sensitivity analysis example problems

The next two subsections describe a serial example (`idasAkzoNob_ASAi_dns`) and a parallel one (`idasBruss_ASAp_kry_bbd_p`). For details on the other examples, the reader is directed to the comments in their source files.

3.1 A serial dense example: idasAkzoNob_ASAi_dns

The `idasAkzoNob_ASAi_dns` program solves the Akzo-Nobel chemical kinetics problem, which consists of six nonlinear DAEs. The system has index 1. The problem originates from Akzo Nobel Central research in Arnhern, The Netherlands, and describes a chemical process in which two species are mixed, while carbon dioxide is continuously added.

The problem is of the form

$$\begin{aligned} y' &= f(y,z) \\ 0 &= g(y,z) \end{aligned} \tag{3}$$

with $y \in R^5$ and $z \in R$. The function f is defined by

$$f(y,z) = \begin{bmatrix} -2r_1 & +r_2 & -r_3 & -r_4 & & \\ -\frac{1}{2}r_1 & & & -r_4 & -\frac{1}{2}r_5 & +F_{in} \\ r_1 & -r_2 & +r_3 & & & \\ & -r_2 & +r_3 & -2r_4 & & \\ & r_2 & -r_3 & & +r_5 & \end{bmatrix}$$

where the r_i and F_{in} are auxiliary variables, given by

$$\begin{aligned} r_1 &= k_1 y_1^4 y_2^{1/2} \\ r_2 &= k_2 y_3 y_4 \\ r_3 &= \frac{k_2}{K} y_1 y_5 \\ r_4 &= k_3 y_1 y_4^{1/2} \\ r_5 &= k_4 z^2 y_2^{1/2} \\ F_{in} &= klA\left(\frac{p(CO_2)}{H} - y_2\right) \ . \end{aligned}$$

The function g in the algebraic equation is defined by

$$g(y,z) = K_s y_1 y_4 - z \ .$$

It is clear from the fact that the Jacobian $\partial g/\partial z$ is non-singular that the DAE (3) has (differentiation) index 1. See http://pitagora.dm.uniba.it/∼testset/report/chemakzo.pdf for details.

The problem is solved with the SUNLINSOL_DENSE linear solver using the default difference quotient dense Jacobian approximation. The adjoint capability of IDAS is used to compute gradients with respect to the initial conditions of the integral

$$G = \int_0^{t_f} y_1 dt \ ,$$

where y_1 is the concentration of the first species. The sensitivity of G is the solution of the adjoint system, evaluated at $t = 0$.

The output generated by `idasAkzoNob_ASAi_dns` is shown below.

idasAkzoNob_ASAi_dns sample output

```
Adjoint Sensitivity Example for Akzo-Nobel Chemical Kinetics
------------------------------------------------------------
Sensitivity of G = int_t0^tf (y1) dt with respect to IC.
------------------------------------------------------------

Forward integration ... done ( nst = 457 )
G:                31.2642162580310945
------------------------------------------------------------

Backward integration ... done ( nst = 277 )
dG/dy0:            2.2207e+01
                  -6.2695e+01
                  -2.5114e+00
                   9.1837e+01
                   3.5176e+00
                   3.6976e-01
------------------------------------------------------------
```

3.2 A parallel example using IDABBDPRE: idasBruss_ASAp_kry_bbd_p

The `idasBruss_ASAp_kry_bbd_p` program solves the same problem as `idasBruss_kry_bbd_p` and `idasBruss_FSA_kry_bbd_p`, namely the Brusselator PDE system. (See §2.2 above.) In addition, it uses an adjoint sensitivity approach to compute the gradients of the model output functional

$$g(t) = \int \int u(t, x, y)\ dx\ dy\ .$$

For perturbations δu_0 and δv_0 in the initial profiles u and v, the perturbation of g at the final time is

$$\delta g(t_f) = \int \int [\lambda(0, x, y)\delta u_0 + \mu(0, x, y)\delta v_0]\ dx\ dy\ ,$$

where $\lambda(t, x, y)$ and $\mu(t, x, y)$ are the solutions of the adjoint PDEs,

$$\begin{aligned} \partial\lambda/\partial t &= -\epsilon_1(\lambda_{xx} + \lambda_{yy}) - (2uv - B - 1)\lambda + (2uv - B)\mu \\ \partial\mu/\partial t &= -\epsilon_2(\mu_{xx} + \mu_{yy}) - u^2\lambda + u^2\mu\ , \end{aligned}$$

with Neumann boundary conditions, and initial (final time) conditions

$$\lambda(t_f, x, y) = 1\ , \quad \mu(t_f, x, y) = 0\ .$$

The adjoint PDEs are discretized and solved in the same way as the Brusselator PDEs.

A sample output generated by `idasBruss_ASAp_kry_bbd_p` is shown below.

idasBruss_ASAp_kry_bbd_p sample output

```
Starting integration of the FORWARD problem
```

```
 BRUSSELATOR: chemically reacting system

Number of species ns: 2     Mesh dimensions: 42 x 42
Total system size: 3528
Subgrid dimensions: 21 x 21     Processor array: 2 x 2
Tolerance parameters:  rtol = 1e-05   atol = 1e-05
Linear solver: SUNLinSol_SPGMR     Max. Krylov dimension maxl: 16
Preconditioner: band-block-diagonal (IDABBDPRE), with parameters
     mudq = 42,  mldq = 42,  mukeep = 2,  mlkeep = 2
------------------------------------------------------------
  t        bottom-left  top-right    | nst  k      h
------------------------------------------------------------

1.00e+00   2.6132e-01   3.0982e+00   | 154  5   1.1112e-02
           1.9993e+00   1.0125e+00   |

------------------------------------------------------------

Final statistics:

Number of steps                    = 154
Number of residual evaluations     = 392
Number of nonlinear iterations     = 174
Number of error test failures      = 2
Number of nonlinear conv. failures = 0

Number of linear iterations        = 216
Number of linear conv. failures    = 0

Number of preconditioner setups    = 17
Number of preconditioner solves    = 392
Number of local residual evals.    = 1462

                 BACKWARD problem
------------------------------------------------------------

Final statistics:

Number of steps                    = 106
Number of residual evaluations     = 276
Number of nonlinear iterations     = 125
Number of error test failures      = 0
Number of nonlinear conv. failures = 0

Number of linear iterations        = 151
Number of linear conv. failures    = 0

Number of preconditioner setups    = 18
Number of preconditioner solves    = 276
Number of local residual evals.    = 1548
```

References

[1] C.W. Gear, B. Leimkuhler, and G.K. Gupta. Automatic Integration of Euler-Lagrange Equations with Constraints. *J. Comput. Appl. Math.*, 12/13:77–90, 1985.

[2] A. C. Hindmarsh, R. Serban, and A. Collier. Example Programs for IDA v4.0.1. Technical Report UCRL-SM-208113, LLNL, 2018.

[3] H. H. Robertson. The solution of a set of reaction rate equations. In J. Walsh, editor, *Numerical analysis: an introduction*, pages 178–182. Academ. Press, 1966.

[4] R. Serban, C. Petra, and A. C. Hindmarsh. User Documentation for IDAS v3.0.1. Technical Report UCRL-SM-234051, LLNL, 2018.