Example Programs for CVODES v4.0.1

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

December 18, 2018

UCRL-SM-208115

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 CVODES [1]. It provides details, with listings, on the example programs supplied with the CVODES distribution package.

The CVODES 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 these examples are given in the following table. In addition, there is an example using OpenMP.

	Serial examples	Parallel examples

IVP	`cvsRoberts_dns` `cvsRoberts_dnsL`
	`cvsRoberts_dns_uw` `cvsRoberts_dns_constraints`
	`cvsRoberts_klu` `cvsRoberts_sps`
	`cvsAdvDiff_bnd` `cvsAdvDiff_bndL`
	`cvsDirunal_kry` `cvsDiurnal_kry_bp`
	`cvsDirectDemo_ls` `cvsKrylovDemo_ls`
	`cvsKrylovDemo_prec`
		`cvsAdvDiff_non_p`
		`cvsDiurnal_kry_p`
		`cvsDirunal_kry_bbd_p`

FSA	`cvsRoberts_FSA_dns` `cvsRoberts_FSA_dns_constraints`
	`cvsRoberts_FSA_klu` `cvsRoberts_FSA_sps`
	`cvsAdvDiff_FSA_non` `cvsDiurnal_FSA_kry`
		`cvsAdvDiff_FSA_non_p`
		`cvsDiurnal_FSA_kry_p`

ASA	`cvsRoberts_ASAi_dns` `cvsRoberts_ASAi_dns_constraints`
	`cvsRoberts_ASAi_klu` `cvsRoberts_ASAi_sps`
	`cvsAdvDiff_ASAi_bnd` `cvsFoodWeb_ASAi_kry`
	`cvsFoodWeb_ASAp_kry` `cvsHessian_ASA_FSA`
		`cvsAdvDiff_ASAp_non_p`
		`cvsAtmDisp_ASAi_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 CVODES examples this is `cvs`);

[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 (for examples using fixed-point iteration for the nonlinear system solver, `non` specifies that no linear solver was used);

[prec] indicates the CVODES preconditioner module used, `bp` for CVBANDPRE or `bbd` for CVBBDPRE (only if applicable, for examples using a Krylov linear solver);

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

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

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

`cvsRoberts_dns` solves a chemical kinetics problem consisting of three rate equations. This program solves the problem with the BDF method and Newton iteration, with the SUNLINSOL_DENSE linear solver module and a user-supplied Jacobian routine. It also uses the rootfinding feature of CVODES.

`cvsRoberts_dns_constraints` is the same as `cvsRoberts_dns` but imposes the constraint $u \geq 0.0$ for all components.

`cvsRoberts_dnsL` is the same as `cvsRoberts_dns` but uses the SUNLINSOL_LAPACKDENSE linear solver module.

`cvsRoberts_dns_uw` is the same as `cvsRoberts_dns` but demonstrates the user-supplied error weight function feature of CVODES.

`cvsRoberts_klu` is the same as `cvsRoberts_dns` but uses the SUNLINSOL_KLU sparse direct linear solver module.

`cvsRoberts_sps` is the same as `cvsRoberts_dns` but uses the SUNLINSOL_SUPERLUMT sparse direct linear solver module (with one thread).

`cvsAdvDiff_bnd` solves the semi-discrete form of an advection-diffusion equation in 2-D.
This program solves the problem with the BDF method and Newton iteration, with the SUNLINSOL_BAND linear solver module and a user-supplied Jacobian routine.

`cvsAdvDiff_bndL` is the same as `cvsAdvDiff_bnd` but uses the SUNLINSOL_LAPACKBAND linear solver module.

`cvsDiurnal_kry` solves the semi-discrete form of a two-species diurnal kinetics advection-diffusion PDE system in 2-D.
The problem is solved with the BDF/GMRES method (i.e. using the SUNLINSOL_SPGMR linear solver) and the block-diagonal part of the Newton matrix as a left preconditioner. A copy of the block-diagonal part of the Jacobian is saved and conditionally reused within the preconditioner setup routine.

`cvsDiurnal_kry_bp` solves the same problem as `cvsDiurnal_kry`, with the BDF/GMRES method and a banded preconditioner, generated by difference quotients, using the module CVBANDPRE.
The problem is solved twice: with preconditioning on the left, then on the right.

`cvsDirectDemo_ls` is a demonstration program for CVODES with direct linear solvers. Two separate problems are solved using both the Adams and BDF linear multistep methods in combination with fixed-point and Newton iterations.
The first problem is the Van der Pol oscillator for which the Newton iteration cases use the following types of Jacobian approximations: (1) dense, user-supplied, (2) dense, difference-quotient approximation, (3) diagonal approximation. The second problem is a linear ODE with a banded lower triangular matrix derived from a 2-D advection PDE. In this case, the Newton iteration cases use the following types of Jacobian approximation: (1) banded, user-supplied, (2) banded, difference-quotient approximation, (3) diagonal approximation.

cvsKrylovDemo_ls solves the same problem as cvsDiurnal_kry, with the BDF method, but with three Krylov linear solver modules: SUNLINSOL_SPGMR, SUNLINSOL_SPBCGS, and SUNLINSOL_SPTFQMR.

cvsKrylovDemo_prec is a demonstration program with the GMRES linear solver.
This program solves a stiff ODE system that arises from a system of partial differential equations. The PDE system is a six-species food web population model, with predator-prey interaction and diffusion on the unit square in two dimensions.
The ODE system is solved using Newton iteration and the SUNLINSOL_SPGMR linear solver module (scaled preconditioned GMRES).
The preconditioner matrix used is the product of two matrices: (1) a matrix, only defined implicitly, based on a fixed number of Gauss-Seidel iterations using the diffusion terms only; and (2) a block-diagonal matrix based on the partial derivatives of the interaction terms only, using block-grouping.
Four different runs are made for this problem. The product preconditoner is applied on the left and on the right. In each case, both the modified and classical Gram-Schmidt options are tested.

cvsRoberts_FSA_dns solves a 3-species kinetics problem (from cvsRoberts_dns).
CVODES 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 BDF method, Newton iteration with the SUNLINSOL_DENSE linear solver module, and a user-supplied Jacobian routine. It also uses the user-supplied error weight function feature of CVODES.

cvsRoberts_FSA_dns_constraints is the same as cvsRoberts_FSA_dns but imposes the constraint u 0.0 for all components.

cvsRoberts_FSA_klu is the same as cvsRoberts_FSA_dns but uses the SUNLINSOL_KLU sparse direct linear solver module.

cvsRoberts_FSA_sps is the same as cvsRoberts_FSA_dns but uses the SUNLINSOL_SUPERLUMT sparse direct linear solver module.

cvsAdvDiff_FSA_non solves the semi-discrete form of an advection-diffusion equation in 1-D.
CVODES computes both its solution and solution sensitivities with respect to the advection and diffusion coefficients. This program solves the problem with the option for nonstiff systems, i.e. Adams method and fixed-point iteration.

cvsDiurnal_FSA_kry solves the semi-discrete form of a two-species diurnal kinetics advection-diffusion PDE system in 2-D space (from cvsDiurnal_kry).
CVODES computes both its solution and solution sensitivities with respect to two parameters affecting the kinetic rate terms. The problem is solved with the BDF/GMRES method (i.e. using the SUNLINSOL_SPGMR linear solver) and the block-diagonal part of the Newton matrix as a left preconditioner.

cvsRoberts_ASAi_dns solves a 3-species kinetics problem (from cvsRoberts_dns).
The adjoint capability of CVODES 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 BDF method, Newton iteration with the SUNLINSOL_DENSE linear solver, and user-supplied Jacobian routines.

`cvsRoberts_ASAi_dns_constraints` is the same as `cvsRoberts_ASAi_dns` but imposes the constraint u 0.0 for all components.

`cvsRoberts_ASAi_klu` is the same as `cvsRoberts_ASAi_dns` but uses the SUNLINSOL_KLU sparse direct linear solver module.

`cvsRoberts_ASAi_sps` is the same as `cvsRoberts_ASAi_dns` but uses the SUNLINSOL_SUPERLUMT sparse direct linear solver module.

`cvsAdvDiff_ASAi_bnd` solves a semi-discrete 2-D advection-diffusion equation (from `cvsAdvDiff_bnd`).
The adjoint capability of CVODES is used to compute gradients of the average (over both time and space) of the solution with respect to the initial conditions. This program solves both the forward and backward problems with the BDF method, Newton iteration with the SUNLINSOL_BAND linear solver, and user-supplied Jacobian routines.

`cvsFoodWeb_ASAi_kry` solves a stiff ODE system that arises from a system of partial differential equations (from `cvsKrylovDemo_prec`). The PDE system is a six-species food web population model, with predator-prey interaction and diffusion on the unit square in two dimensions.
The adjoint capability of CVODES is used to compute gradients of the average (over both time and space) of the concentration of a selected species with respect to the initial conditions of all six species. Both the forward and backward problems are solved with the BDF/GMRES method (i.e. using the SUNLINSOL_SPGMR linear solver module) and the block-diagonal part of the Newton matrix as a left preconditioner.

`cvsFoodWeb_ASAp_kry` solves the same problem as `cvsFoodWeb_ASAi_kry`, but computes gradients of the average over space at the *final time* of the concentration of a selected species with respect to the initial conditions of all six species.

`cvsHessian_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/cvodes/parallel` directory are the following seven parallel examples (using the NVECTOR_PARALLEL module):

`cvsAdvDiff_non_p` solves the semi-discrete form of a 1-D advection-diffusion equation. This program solves the problem with the option for nonstiff systems, i.e. Adams method and fixed-point iteration.

`cvsDiurnal_kry_p` is a parallel implementation of `cvsDiurnal_kry`.

`cvsDiurnal_kry_bbd_p` solves the same problem as `cvsDiurnal_kry_p`, with BDF and the GMRES linear solver, using a block-diagonal matrix with banded blocks as a preconditioner, generated by difference quotients, using the module CVBBDPRE.

`cvsAdvDiff_FSA_non_p` is a parallel version of `cvsAdvDiff_FSA_non`.

- `cvsDiurnal_FSA_kry_p` is a parallel version of `cvsDiurnal_FSA_kry`.

- `cvsAdvDiff_ASAp_non_p` solves a semi-discrete 1-D advection-diffusion equation (from `cvsAdvDiff_non_p`).
  The adjoint capability of CVODES is used to compute gradients of the average over space of the solution at the *final time* with respect to both the initial conditions and the advection and diffusion coefficients in the model. This program solves both the forward and backward problems with the option for nonstiff systems, i.e. Adams method and fixed-point iteration.

- `cvsAtmDisp_ASAi_kry_bbd_p` solves an adjoint sensitivity problem for an advection-diffusion PDE in 2-D or 3-D using the BDF/GMRES method and the CVBBDPRE preconditioner module on both the forward and backward phases.
  The adjoint capability of CVODES is used to compute the gradient of the space-time average of the squared solution norm with respect to problem parameters which parametrize a distributed volume source.

Supplied in *srcdir*/`examples/cvodes/C_openmp` is an example, `cvsAdvDiff_bnd_omp`, which solves the same problem as `cvsAdvDiff_bnd` but using the OpenMP NVECTOR 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 CVODE Examples document [2]. Any CVODE program will work with CVODES with only two modifications: (1) the main program should include the header file `cvodes.h` instead of `cvode.h`, and (2) the loader command must reference *builddir*/`lib/libsundials_cvodes.`*lib* instead of *builddir*/`lib/libsundials_cvode.`*lib*.

We also give our output files for each of the 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%.

The final section of this report describes a set of tests done with CVODES in a parallel environment (using NVECTOR_PARALLEL) on a modification of the `cvsDiurnal_kry_p` example.

In the descriptions below, we make frequent references to the CVODES User Guide [1]. 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 CVODES 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 CVODES examples, any of three sensitivity method options (CV_SIMULTANEOUS, CV_STAGGERED, or CV_STAGGERED1) can be used, and sensitivities may be included in the error test or not (error control set on SUNTRUE or SUNFALSE, respectively).

The next three sections describe in detail two serial examples (cvsAdvDiff_FSA_non and cvsRoberts_FSA_dns), and a parallel one (cvsDiurnal_FSA_kry_p). For details on the other examples, the reader is directed to the comments in their source files.

2.1 A serial nonstiff example: cvsAdvDiff_FSA_non

As a first example of using CVODES for forward sensitivity analysis, we treat the simple advection-diffusion equation for $u = u(t, x)$

$$\frac{\partial u}{\partial t} = q_1 \frac{\partial^2 u}{\partial x^2} + q_2 \frac{\partial u}{\partial x} \tag{1}$$

for $0 \le t \le 5$, $0 \le x \le 2$, and subject to homogeneous Dirichlet boundary conditions and initial values given by

$$\begin{aligned} u(t,0) &= 0\,, \quad u(t,2) = 0 \\ u(0,x) &= x(2-x)e^{2x}\,. \end{aligned} \tag{2}$$

The nominal values of the problem parameters are $q_1 = 1.0$ and $q_2 = 0.5$. A system of MX ODEs is obtained by discretizing the x-axis with MX+2 grid points and replacing the first and second order spatial derivatives with their central difference approximations. Since the value of u is constant at the two endpoints, the semi-discrete equations for those points can be eliminated. With u_i as the approximation to $u(t, x_i)$, $x_i = i(\Delta x)$, and $\Delta x = 2/(\mathtt{MX} + 1)$, the resulting system of ODEs, $\dot{u} = f(t, u)$, can now be written:

$$\dot{u}_i = q_1 \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + q_2 \frac{u_{i+1} - u_{i-1}}{2(\Delta x)}\,. \tag{3}$$

This equation holds for $i = 1, 2, \ldots,$ MX, with the understanding that $u_0 = u_{MX+1} = 0$.

The sensitivity systems for $s^1 = \partial u/\partial q_1$ and $s^2 = \partial u/\partial q_2$ are simply

$$\begin{aligned} \frac{ds_i^1}{dt} &= q_1 \frac{s_{i+1}^1 - 2s_i^1 + s_{i-1}^1}{(\Delta x)^2} + q_2 \frac{s_{i+1}^1 - s_{i-1}^1}{2(\Delta x)} + \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} \\ s_i^1(0) &= 0.0 \end{aligned} \tag{4}$$

and

$$\begin{aligned} \frac{ds_i^2}{dt} &= q_1 \frac{s_{i+1}^2 - 2s_i^2 + s_{i-1}^2}{(\Delta x)^2} + q_2 \frac{s_{i+1}^2 - s_{i-1}^2}{2(\Delta x)} + \frac{u_{i+1} - u_{i-1}}{2(\Delta x)} \\ s_i^1(0) &= 0.0\,. \end{aligned} \tag{5}$$

This problem uses the Adams (non-stiff) integration formula and fixed-point iteration. It is unrealistically simple*, but serves to illustrate use of the forward sensitivity capabilities in CVODES.

*Increasing the number of grid points to better resolve the PDE spatially will lead to a stiffer ODE for which the Adams integration formula will not be suitable.

The cvsAdvDiff_FSA_non.c file begins by including several header files, including the main CVODES header file, the sundials_types.h header file for the definition of the realtype type, and the NVECTOR_SERIAL header file for the definitions of the serial N_Vector type and operations on such vectors. Following that are definitions of problem constants and a data block for communication with the f routine. That block includes the problem parameters and the mesh dimension.

The main program begins by processing and verifying the program arguments, followed by allocation and initialization of the user-defined data structure. Next, the vector of initial conditions is created (by calling N_VNew_Serial) and initialized (in the function SetIC). The next code block creates and allocates memory for the CVODES object.

If sensitivity calculations were turned on through the command line arguments, the main program continues with setting the scaling parameters pbar and the array of flags plist. In this example, the scaling factors pbar are used both for the finite difference approximation to the right-hand sides of the sensitivity systems (4) and (5) and in calculating the absolute tolerances for the sensitivity variables. The flags in plist are set to indicate that sensitivities with respect to both problem parameters are desired. The array of NS = 2 vectors uS for the sensitivity variables is created by calling N_VCloneVectorArray_Serial and set to contain the initial values ($s_i^1(0) = 0.0$, $s_i^2(0) = 0.0$).

The next three calls set optional inputs for sensitivity calculations: the sensitivity variables are included or excluded from the error test (the boolean variable err_con is passed as a command line argument), the control variable rho is set to a value ZERO = 0 to indicate the use of second-order centered directional derivative formulas for the approximations to the sensitivity right-hand sides, and the array of scaling factors pbar is passed to CVODES. Memory for sensitivity calculations is allocated by calling CVodeSensInit1 which also specifies the sensitivity solution method (sensi_meth is passed as a command line argument), and the initial conditions for the sensitivity variables. The problem parameters p and the arrays pbar and plist are passed to CVodeSetSensParam.

Next, in a loop over the NOUT output times, the program calls the integration routine CVode. On a successful return, the program prints the maximum norm of the solution u at the current time and, if sensitivities were also computed, extracts and prints the maximum norms of $s^1(t)$ and $s^2(t)$. The program ends by printing some final integration statistics and freeing all allocated memory.

The f function is a straightforward implementation of Eqn. (3). The rest of the source file contains definitions of private functions. The last two, PrintFinalStats and check_flag, can be used with minor modifications by any CVODES user code to print final CVODES statistics and to check return flags from CVODES interface functions, respectively.

Results generated by cvsAdvDiff_FSA_non are shown in Fig. 1. The output generated by cvsAdvDiff_FSA_non when computing sensitivities with the CV_SIMULTANEOUS method and full error control (cvsAdvDiff_FSA_non -sensi sim t) is as follows:

cvsAdvDiff_FSA_non sample output

```

1-D advection-diffusion equation, mesh size = 10
Sensitivity: YES ( SIMULTANEOUS + FULL ERROR CONTROL )

============================================================
     T     Q       H      NST                    Max norm
============================================================
5.000e-01  4  7.577e-03   115
```

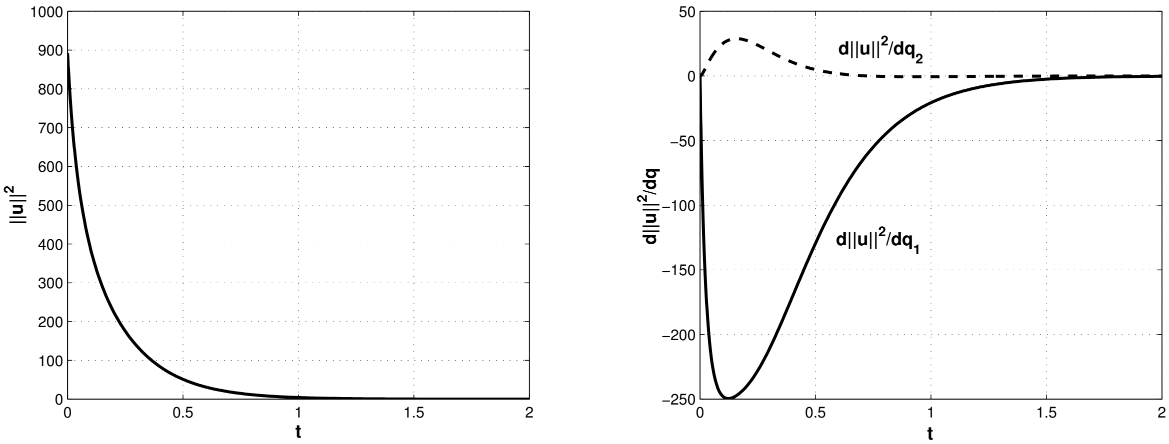

Figure 1: Results for the `cvsAdvDiff_FSA_non` example problem. The time evolution of the squared solution norm, $jjujj^2$, is shown on the left. The gure on the right shows the evolution of the sensitivities of $jjujj^2$ with respect to the two problem parameters.

```
                                    Solution          3.0529e+00
                                    Sensitivity 1     3.8668e+00
                                    Sensitivity 2     6.2020e-01
------------------------------------------------------------
1.000e+00  3  4.126e-03   187
                                    Solution          8.7533e-01
                                    Sensitivity 1     2.1743e+00
                                    Sensitivity 2     1.8909e-01
------------------------------------------------------------
1.500e+00  2  1.181e-02   265
                                    Solution          2.4948e-01
                                    Sensitivity 1     9.1825e-01
                                    Sensitivity 2     7.3921e-02
------------------------------------------------------------
2.000e+00  2  9.433e-03   328
                                    Solution          7.1095e-02
                                    Sensitivity 1     3.4666e-01
                                    Sensitivity 2     2.8228e-02
------------------------------------------------------------
2.500e+00  2  3.946e-03   398
                                    Solution          2.0259e-02
                                    Sensitivity 1     1.2300e-01
                                    Sensitivity 2     1.0085e-02
------------------------------------------------------------
3.000e+00  2  9.370e-03   470
                                    Solution          5.7731e-03
                                    Sensitivity 1     4.1958e-02
                                    Sensitivity 2     3.4556e-03
------------------------------------------------------------
3.500e+00  2  1.010e-02   540
                                    Solution          1.6451e-03
                                    Sensitivity 1     1.3922e-02
                                    Sensitivity 2     1.1669e-03
------------------------------------------------------------
4.000e+00  2  4.255e-03   638
                                    Solution          4.6881e-04
                                    Sensitivity 1     4.5275e-03
```

```
                                  Sensitivity 2    3.8633e-04
---------------------------------------------------------------
4.500e+00  1  5.757e-03   716
                                  Solution         1.3404e-04
                                  Sensitivity 1    1.4539e-03
                                  Sensitivity 2    1.2576e-04
---------------------------------------------------------------
5.000e+00  1  6.420e-03   798
                                  Solution         3.8640e-05
                                  Sensitivity 1    4.6496e-04
                                  Sensitivity 2    4.0583e-05
---------------------------------------------------------------

Final Statistics

nst     =   798

nfe     =  1408
netf    =     1    nsetups  =     0
nni     =  1405    ncfn     =   125

nfSe    =  2816    nfeS     =  5632
netfs   =     0    nsetupsS =     0
nniS    =     0    ncfnS    =     0
```

2.2 A serial dense example: cvsRoberts_FSA_dns

This example is a modi cation of the chemical kinetics example `cvRoberts_dns` described in [2]. It computes, in addition to the solution of the IVP, sensitivities of the solution with respect to the three reaction rates involved in the model. The ODEs are written as:

$$
\begin{aligned}
y_1 &= \quad p_1 y_1 + p_2 y_2 y_3 \\
y_2 &= p_1 y_1 \quad p_2 y_2 y_3 \quad p_3 y_2^2 \\
y_3 &= p_3 y_2^2 \, ,
\end{aligned}
\tag{6}
$$

with initial conditions at $t_0 = 0$, $y_1 = 1$ and $y_2 = y_3 = 0$. The nominal values of the reaction rate constants are $p_1 = 0.04$, $p_2 = 10^4$ and $p_3 = 3 \;\; 10^7$. The sensitivity systems that are solved together with (6) are

$$
\begin{aligned}
\underset{\sim}{s}_i &= \begin{bmatrix} p_1 & p_2 y_3 & p_2 y_2 \\ p_1 & p_2 y_3 \quad 2p_3 y_2 & p_2 y_2 \\ 0 & 2p_3 y_2 & 0 \end{bmatrix} s_i + \frac{\partial f}{\partial p_i} \, , \quad s_i(t_0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \, , \quad i = 1, 2, 3 \\
\frac{\partial f}{\partial p_1} &= \begin{bmatrix} y_1 \\ y_1 \\ 0 \end{bmatrix} \, , \quad \frac{\partial f}{\partial p_2} = \begin{bmatrix} y_2 y_3 \\ y_2 y_3 \\ 0 \end{bmatrix} \, , \quad \frac{\partial f}{\partial p_3} = \begin{bmatrix} 0 \\ y_2^2 \\ y_2^2 \end{bmatrix} \, .
\end{aligned}
\tag{7}
$$

The main program is described below with emphasis on the sensitivity related components. These explanations, together with those given for the code `cvRoberts_dns` in [2], will also provide the user with a template for instrumenting an existing simulation code to perform forward sensitivity analysis. As will be seen from this example, an existing simulation code can be modi ed to compute sensitivity variables (in addition to state variables) by only inserting a few CVODES calls into the main program.

First note that no new header files need be included. In addition to the constants already defined in cvRoberts_dns, we define the number of model parameters, NP (= 3), the number of sensitivity parameters, NS (= 3), and a constant ZERO = 0.0.

As mentioned in §5.1, the user data structure data must provide access to the array of model parameters as the only way for CVODES to communicate parameter values to the right-hand side function f. In the cvsRoberts_FSA_dns example this is done by defining data to be of type UserData, i.e. a pointer to a structure which contains an array of NP realtype values.

Four user-supplied functions are defined. The function f, passed to CVodeInit, computes the right-hand side of the ODE (6), while Jac computes the dense Jacobian of the problem and is attached to the dense linear solver module SUNLINSOL_DENSE through a call to CVodeSetJacFn. The function fS computes the right-hand side of each sensitivity system (7) for one parameter at a time and is therefore of type SensRhs1. Finally, the function ewt computes the error weights for the WRMS norm estimations within CVODES.

The program prologue ends by defining six private helper functions. The first two, ProcessArgs and WrongArgs (which would not be present in a typical user code), parse and verify the command line arguments to cvsRoberts_FSA_dns, respectively. After each successful return from the main CVODES integrator, the functions PrintOutput and PrintOutputS print the state and sensitivity variables, respectively. The function PrintFinalStats is called after completion of the integration to print solver statistics. The function check_flag is used to check the return flag from any of the CVODES interface functions called by main.

The main program begins with definitions and type declarations. Among these, it defines the vector pbar of NS scaling factors for the model parameters p, and the array yS of vectors (of type N_Vector) which will contain the initial conditions and solutions for the sensitivity variables. It also declares the variable data of type UserData which will contain the user-defined data structure to be passed to CVODES and used in the evaluation of the ODE right-hand sides.

The first code block in main deals with reading and interpreting the command line arguments. cvsRoberts_FSA_dns can be run with or without sensitivity computations turned on and with different selections for the sensitivity method and error control strategy.

The user's data structure is then allocated and its field p is set to contain the values of the three problem parameters. The next block of code is identical to that in cvRoberts_dns.c (see [2]) and involves allocation and initialization of the state variables, and creation and initialization of cvode_mem, the CVODES solver memory. It specifies that a user-provided function (ewt) is to be used for computing the error weights. It also attaches SUNLINSOL_DENSE, with a non-NULL Jacobian function, as the linear solver to be used in the Newton nonlinear iteration.

If sensitivity analysis is enabled (through the command line arguments), the main program will then set the scaling parameters pbar ($\mathtt{pbar}_i = \mathtt{p}_i$, which can typically be used for nonzero model parameters). Next, the program allocates memory for yS, by calling the NVECTOR_SERIAL function N_VCloneVectorArray_Serial, and initializaes all sensitivity variables to 0.0.

The call to CVodeSensInit1 specifies the sensitivity solution method through the argument sensi_meth (read from the command line arguments) as one of CV_SIMULTANEOUS, CV_STAGGERED, or CV_STAGGERED1. It also specifies the user-defined routine, fS, for evaluation of the right-hand sides of sensitivity equations.

The next three calls specify optional inputs for forward sensitivity analysis: specifying that sensitivity tolerances are to be based on pbar, the error control strategy (read from the command line arguments), and the information on the model parameters. In this example,

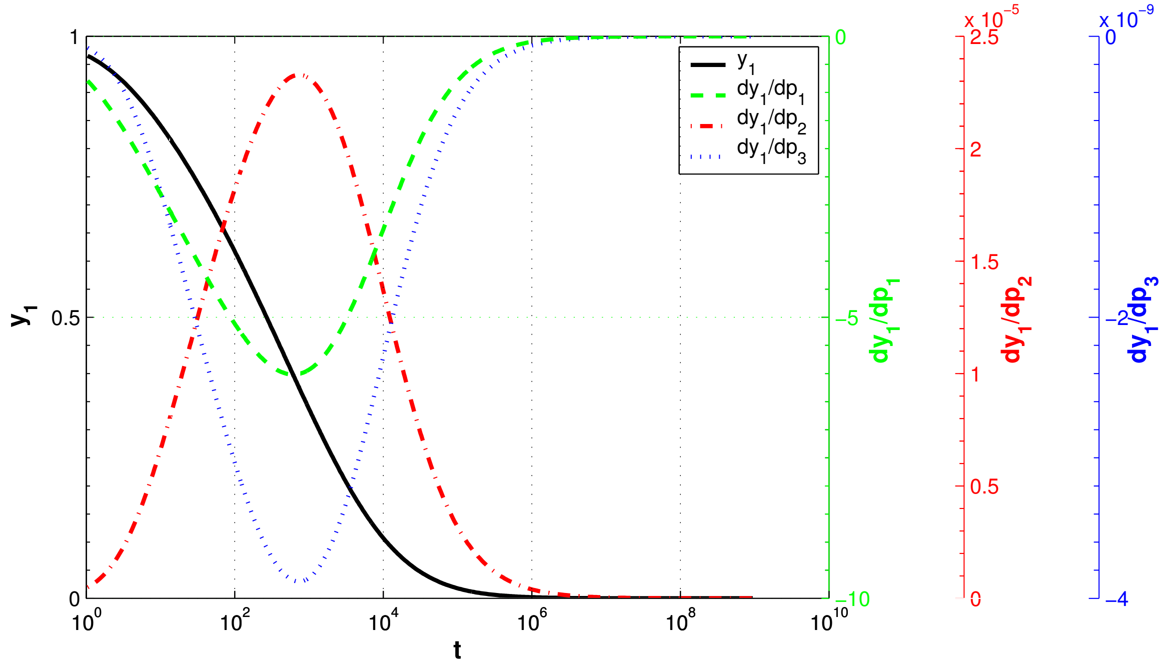

Figure 2: Results for the `cvsRoberts_FSA_dns` example problem: time evolution of y_1 and its sensitivities with respect to the three problem parameters. (Note the four different vertical scales.)

only `pbar` is needed for the estimation of absolute sensitivity variable tolerances; neither `p` nor `plist` is required since the sensitivity right-hand sides are computed in the user function `fS`. As a consequence, we pass `NULL` for the corresponding arguments in `CVodeSetSensParams`.

Note that this example uses the default estimates for the relative and absolute tolerances `rtolS` and `atolS` for sensitivity variables, based on the tolerances for state variables and the scaling parameters `pbar` (see §2.6 for details).

Next, in a loop over the `NOUT` output times, the program calls the integration routine `CVode` which, if sensitivity analysis was initialized through the call to `CVodeSensInit1`, computes both state and sensitivity variables. However, `CVode` returns only the state solution at `tout` in the vector `y`. The program tests the return from `CVode` for a value other than `CV_SUCCESS` and prints the state variables. Sensitivity variables at `tout` are loaded into `yS` by calling `CVodeGetSens`. The program tests the return from `CVodeGetSens` for a value other than `CV_SUCCESS` and then prints the sensitivity variables.

Finally, the program prints some statistics (function `PrintFinalStats`) and deallocates memory through calls to `N_VDestroy_Serial`, `N_VDestroyVectorArray_Serial`, `CVodeFree`, and `free` for the user data structure.

The user-supplied functions `f` (for the right-hand side of the original ODEs) and `Jac` (for the system Jacobian) are identical to those in `cvRoberts_dns.c`, with the notable exeption that model parameters are extracted from the user-defined data structure `data`, which must first be cast to the `UserData` type. Similarly, the user-supplied function `ewt` is identical to that in `cvRoberts_dns_uw.c`. The user-supplied function `fS` computes the sensitivity right-hand side for the `iS`-th sensitivity equation.

Results generated by `cvsRoberts_FSA_dns` are shown in Fig. 2. The following output is generated by `cvsRoberts_FSA_dns` when computing sensitivities with the `CV_SIMULTANEOUS` method and full error control (`cvsRoberts`

```
4.000e+07  4  1.776e+06   753
                  Solution         5.2039e-05   2.0817e-10   9.9995e-01
                  Sensitivity 1   -2.5991e-03  -5.1931e-09   2.5991e-03
                  Sensitivity 2    1.0396e-08   2.0772e-14  -1.0397e-08
                  Sensitivity 3   -1.7330e-12  -6.9328e-18   1.7330e-12
-----------------------------------------------------------------------
4.000e+08  4  2.766e+07   802
                  Solution         5.2106e-06   2.0842e-11   9.9999e-01
                  Sensitivity 1   -2.6063e-04  -5.2149e-10   2.6063e-04
                  Sensitivity 2    1.0425e-09   2.0859e-15  -1.0425e-09
                  Sensitivity 3   -1.7366e-13  -6.9467e-19   1.7367e-13
-----------------------------------------------------------------------
4.000e+09  2  4.183e+08   836
                  Solution         5.1881e-07   2.0752e-12   1.0000e-00
                  Sensitivity 1   -2.5907e-05  -5.1717e-11   2.5907e-05
                  Sensitivity 2    1.0363e-10   2.0687e-16  -1.0363e-10
                  Sensitivity 3   -1.7293e-14  -6.9174e-20   1.7293e-14
-----------------------------------------------------------------------
4.000e+10  2  3.799e+09   859
                  Solution         6.5181e-08   2.6072e-13   1.0000e-00
                  Sensitivity 1   -2.4884e-06  -3.3032e-12   2.4884e-06
                  Sensitivity 2    9.9534e-12   1.3213e-17  -9.9534e-12
                  Sensitivity 3   -2.1727e-15  -8.6908e-21   2.1727e-15
-----------------------------------------------------------------------

Final Statistics

nst     =   859

nfe     =  1222
netf    =    29    nsetups  =   142
nni     =  1218    ncfn     =     4

nfSe    =  3666    nfeS     =     0
netfs   =     0    nsetupsS =     0
nniS    =     0    ncfnS    =     0

nje    =    24    nfeLS     =     0
```

2.3 A parallel example with user preconditioner: cvsDiurnal_FSA_kry_p

As an example of using the forward sensitivity capabilities in CVODES with the Krylov linear solver SUNLINSOL_SPGMR and the NVECTOR_PARALLEL module, we describe a test problem (derived from `cvDiurnal_kry_p`) that solves the semi-discrete form of a two-species diurnal kinetics advection-di usion PDE system in 2-D space, for which we also compute solution sensitivities with respect to problem parameters (q_1 and q_2) that appear in the kinetic rate terms.

The PDE system is

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + V \frac{\partial c^i}{\partial x} + \frac{\partial}{\partial y} K_v(y) \frac{\partial c^i}{\partial y} + R^i(c^1, c^2, t) \quad (i = 1, 2)\,, \tag{8}$$

where the superscripts i are used to distinguish the two chemical species, and where the reaction terms are given by

$$
\begin{aligned}
R^1(c^1,c^2,t) &= -q_1c^1c^3 - q_2c^1c^2 + 2q_3(t)c^3 + q_4(t)c^2 \ , \\
R^2(c^1,c^2,t) &= q_1c^1c^3 - q_2c^1c^2 - q_4(t)c^2 \ .
\end{aligned}
\tag{9}
$$

The spatial domain is $0 \le x \le 20$, $30 \le y \le 50$ (in km). The various constants and parameters are: $K_h = 4.0 \cdot 10^{-6}$, $V = 10^{-3}$, $K_v = 10^{-8}\exp(y/5)$, $q_1 = 1.63 \cdot 10^{-16}$, $q_2 = 4.66 \cdot 10^{-16}$, $c^3 = 3.7 \cdot 10^{16}$, and the diurnal rate constants are defined as:

$$
q_i(t) = \left\{ \begin{array}{ll} \exp[-a_i/\sin\omega t], & \text{for } \sin\omega t > 0 \\ 0, & \text{for } \sin\omega t \le 0 \end{array} \right\} \quad (i = 3,4) \, ,
$$

where $\omega = \pi/43200$, $a_3 = 22.62$, $a_4 = 7.601$. The time interval of integration is $[0, 86400]$, representing 24 hours measured in seconds.

Homogeneous Neumann boundary conditions are imposed on each boundary, and the initial conditions are

$$
\begin{aligned}
c^1(x,y,0) = 10^6\alpha(x)\beta(y) \ , \quad & c^2(x,y,0) = 10^{12}\alpha(x)\beta(y) \ , \\
\alpha(x) &= 1 - (0.1x - 1)^2 + (0.1x - 1)^4/2 \ , \\
\beta(y) &= 1 - (0.1y - 4)^2 + (0.1y - 4)^4/2 \ .
\end{aligned}
\tag{10}
$$

We discretize the PDE system with central differencing, to obtain an ODE system $\dot{u} = f(t,u)$ representing (8). In this case, the discrete solution vector is distributed across many processes. Specifically, we may think of the processes as being laid out in a rectangle, and each process being assigned a subgrid of size `MXSUB`$\times$`MYSUB` of the $x - y$ grid. If there are `NPEX` processes in the x direction and `NPEY` processes in the y direction, then the overall grid size is `MX`$\times$`MY` with `MX`=`NPEX`$\times$`MXSUB` and `MY`=`NPEY`$\times$`MYSUB`, and the size of the ODE system is $2 \cdot$`MX`$\cdot$`MY`.

To compute f in this setting, the processes pass and receive information as follows. The solution components for the bottom row of grid points assigned to the current process are passed to the process below it, and the solution for the top row of grid points is received from the process below the current process. The solution for the top row of grid points for the current process is sent to the process above the current process, while the solution for the bottom row of grid points is received from that process by the current process. Similarly, the solution for the first column of grid points is sent from the current process to the process to its left, and the last column of grid points is received from that process by the current process. The communication for the solution at the right edge of the process is similar. If this is the last process in a particular direction, then message passing and receiving are bypassed for that direction.

The overall structure of `main` is very similar to that of the code `cvsRoberts_FSA_dns` described above, with differences arising from the use of the parallel NVECTOR module, NVECTOR_PARALLEL. On the other hand, the user-supplied routines in `cvsDiurnal_FSA_kry_p`, `f` for the right-hand side of the original system, `Precond` for the preconditioner setup, and `PSolve` for the preconditioner solve, are identical to those defined in the example program `cvDiurnal_kry_p` described in [2]. The only difference is in the routine `fcalc`, which operates on local data only and contains the actual calculation of $f(t,u)$, where the problem parameters are first extracted from the user data structure `data`. The program `cvsDiurnal_FSA_kry_p` defines no additional user-supplied routines, as it uses the CVODES internal difference quotient routines to compute the sensitivity equation right-hand sides.

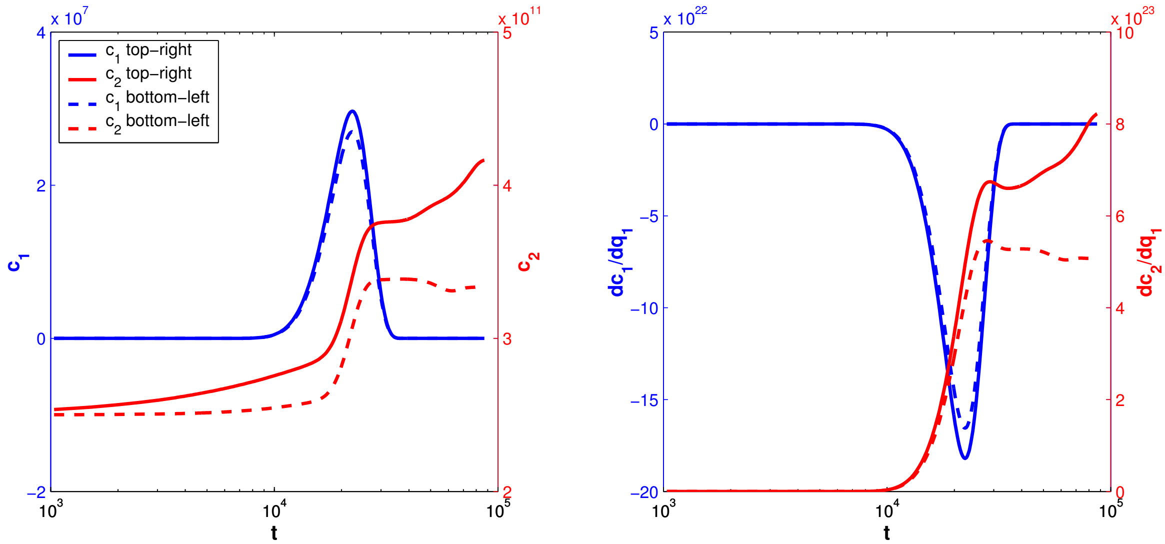

Figure 3: Results for the `cvsDiurnal_FSA_kry_p` example problem: time evolution of c_1 and c_2 at the bottom-left and top-right corners (left) and of their sensitivities with respect to q_1.

Sample results generated by `cvsDiurnal_FSA_kry_p` are shown in Fig. 3. These results were generated on a (2   40)   (2   40) spatial grid. The following output is generated by `cvsDiurnal_FSA_kry_p` when computing sensitivities with the `CV_SIMULTANEOUS` method and full error control (`mpirun -np 4 cvsDiurnal_FSA_kry_p -sensi sim t`):

`cvsDiurnal_FSA_kry_p` sample output

```
2-species diurnal advection-diffusion problem
Sensitivity: YES ( SIMULTANEOUS + FULL ERROR CONTROL )

========================================================================
     T      Q       H       NST                   Bottom left   Top right
========================================================================
7.200e+03   3   3.475e+01   345
                                 Solution          1.0468e+04   1.1185e+04
                                                   2.5267e+11   2.6998e+11
                                 ----------------------------------------
                                 Sensitivity 1    -6.4201e+19  -6.8598e+19
                                                   7.1177e+19   7.6556e+19
                                 ----------------------------------------
                                 Sensitivity 2    -4.3853e+14  -5.0065e+14
                                                  -2.4407e+18  -2.7843e+18
------------------------------------------------------------------------
1.440e+04   3   5.071e+01   862
                                 Solution          6.6590e+06   7.3008e+06
                                                   2.5819e+11   2.8329e+11
                                 ----------------------------------------
                                 Sensitivity 1    -4.0848e+22  -4.4785e+22
                                                   5.9549e+22   6.7173e+22
                                 ----------------------------------------
                                 Sensitivity 2    -4.5235e+17  -5.4318e+17
                                                  -6.5418e+21  -7.8315e+21
------------------------------------------------------------------------
2.160e+04   3   5.422e+01  1115
```

```
                                    Solution          2.6650e+07   2.9308e+07
                                                      2.9928e+11   3.3134e+11
                                    ---------------------------------------
                                    Sensitivity 1    -1.6346e+23  -1.7976e+23
                                                      3.8203e+23   4.4991e+23
                                    ---------------------------------------
                                    Sensitivity 2    -7.6601e+18  -9.4433e+18
                                                     -7.6459e+22  -9.4501e+22
-----------------------------------------------------------------------------
2.880e+04  3  4.027e+01  1446
                                    Solution          8.7021e+06   9.6501e+06
                                                      3.3804e+11   3.7510e+11
                                    ---------------------------------------
                                    Sensitivity 1    -5.3375e+22  -5.9187e+22
                                                      5.4487e+23   6.7430e+23
                                    ---------------------------------------
                                    Sensitivity 2    -4.8855e+18  -6.1040e+18
                                                     -1.7194e+23  -2.1518e+23
-----------------------------------------------------------------------------
3.600e+04  4  6.446e+01  1550
                                    Solution          1.4040e+04   1.5609e+04
                                                      3.3868e+11   3.7652e+11
                                    ---------------------------------------
                                    Sensitivity 1    -8.6141e+19  -9.5762e+19
                                                      5.2718e+23   6.6030e+23
                                    ---------------------------------------
                                    Sensitivity 2    -8.4328e+15  -1.0549e+16
                                                     -1.8439e+23  -2.3096e+23
-----------------------------------------------------------------------------
4.320e+04  4  1.552e+02  1802
                                    Solution         -6.7943e-09  -1.7531e-08
                                                      3.3823e+11   3.8035e+11
                                    ---------------------------------------
                                    Sensitivity 1     1.5377e+08  -1.8226e+09
                                                      5.2753e+23   6.7448e+23
                                    ---------------------------------------
                                    Sensitivity 2     4.9296e+03  -1.7707e+04
                                                     -1.8454e+23  -2.3595e+23
-----------------------------------------------------------------------------
5.040e+04  4  1.552e+02  1848
                                    Solution         -3.3333e-09  -1.0074e-08
                                                      3.3582e+11   3.8645e+11
                                    ---------------------------------------
                                    Sensitivity 1     7.6593e+08   2.3212e+09
                                                      5.2067e+23   6.9664e+23
                                    ---------------------------------------
                                    Sensitivity 2     3.2953e+07   1.2254e+08
                                                     -1.8214e+23  -2.4370e+23
-----------------------------------------------------------------------------
5.760e+04  5  2.333e+02  1871
                                    Solution         -8.0165e-13  -2.6806e-12
                                                      3.3203e+11   3.9090e+11
                                    ---------------------------------------
                                    Sensitivity 1    -1.3115e+05  -4.2823e+05
                                                      5.0825e+23   7.1205e+23
                                    ---------------------------------------
                                    Sensitivity 2     6.8742e+01   1.6059e+02
                                                     -1.7780e+23  -2.4910e+23
-----------------------------------------------------------------------------
```

```
6.480e+04  5  2.801e+02  1893
                                 Solution         -2.8173e-08  -9.8429e-08
                                                   3.3130e+11   3.9634e+11
                                 ---------------------------------------
                                 Sensitivity 1     2.2918e+09   7.9585e+09
                                                   5.0442e+23   7.3274e+23
                                 ---------------------------------------
                                 Sensitivity 2     7.1238e+05   2.7790e+06
                                                  -1.7646e+23  -2.5633e+23
------------------------------------------------------------------------
7.200e+04  4  1.003e+02  2580
                                 Solution         -1.1403e-08  -6.9110e-08
                                                   3.3297e+11   4.0389e+11
                                 ---------------------------------------
                                 Sensitivity 1     6.8126e+08   4.1340e+09
                                                   5.0783e+23   7.6382e+23
                                 ---------------------------------------
                                 Sensitivity 2    -3.8340e+07  -2.6839e+08
                                                  -1.7765e+23  -2.6721e+23
------------------------------------------------------------------------
7.920e+04  4  4.453e+02  2608
                                 Solution          4.8775e-18   2.7563e-17
                                                   3.3344e+11   4.1203e+11
                                 ---------------------------------------
                                 Sensitivity 1     1.2984e+02   7.7701e+02
                                                   5.0730e+23   7.9960e+23
                                 ---------------------------------------
                                 Sensitivity 2    -4.4037e-01  -3.1248e+00
                                                  -1.7747e+23  -2.7972e+23
------------------------------------------------------------------------
8.640e+04  5  7.396e+02  2619
                                 Solution         -2.5590e-20  -1.5317e-19
                                                   3.3518e+11   4.1625e+11
                                 ---------------------------------------
                                 Sensitivity 1     1.6342e+00   9.8016e+00
                                                   5.1171e+23   8.2142e+23
                                 ---------------------------------------
                                 Sensitivity 2    -5.6895e-03  -4.0306e-02
                                                  -1.7901e+23  -2.8736e+23
------------------------------------------------------------------------

Final Statistics

nst     =  2619

nfe     =  3582
netf    =   150    nsetups  =   436
nni     =  3580    ncfn     =     6

nfSe    =  7164    nfeS     = 14328
netfs   =     0    nsetupsS =     0
nniS    =     0    ncfnS    =     0
```

3 Adjoint sensitivity analysis example problems

The next three sections describe in detail a serial example (cvsRoberts_ASAi_dns) and two parallel examples (cvsAdvDiff_ASAp_non_p and cvsAtmDisp_ASAi_kry_bbd_p) that perform adjoint sensitivity analysis. For details on the other examples, the reader is directed to the comments in their source les.

3.1 A serial dense example: cvsRoberts_ASAi_dns

As a rst example of using CVODES for adjoint sensitivity analysis, we examine the chemical kinetics problem (from cvsRoberts_FSA_dns)

$$\begin{aligned} y_1 &= \quad p_1 y_1 + p_2 y_2 y_3 \\ y_2 &= p_1 y_1 \quad p_2 y_2 y_3 \quad p_3 y_2^2 \\ y_3 &= p_3 y_2^2 \\ y(t_0) &= y_0\,, \end{aligned} \tag{11}$$

for which we want to compute the gradient with respect to p of

$$G(p) = \int_{t_0}^{T} y_3 dt, \tag{12}$$

without having to compute the solution sensitivities dy/dp. Following the derivation in x2.7, and taking into account the fact that the initial values of (11) do not depend on the parameters p, by (2.21) this gradient is simply

$$\frac{dG}{dp} = \int_{t_0}^{T} \left(g_p + \lambda^T f_p \right) dt\,, \tag{13}$$

where $g(t, y, p) = y_3$, f is the vector-valued function de ning the right-hand side of (11), and λ is the solution of the adjoint problem (2.20),

$$\begin{aligned} \lambda &= \quad (f_y)^T \lambda \quad (g_y)^T \\ \lambda(T) &= 0\,. \end{aligned} \tag{14}$$

In order to avoid saving intermediate λ values just for the evaluation of the integral in (13), we extend the backward problem with the following N_p quadrature equations

$$\begin{aligned} \xi &= g_p^T + f_p^T \lambda \\ \xi(T) &= 0\,, \end{aligned} \tag{15}$$

which yield $\xi(t_0) = \quad \int_{t_0}^{T}$

The calling program includes the CVODES header files cvodes.h for CVODES definitions and interface function prototypes, the header file nvector_serial.h for the definition of the serial implementation of the NVECTOR module, NVECTOR_SERIAL, the header files sunmatrix_dense.h and sunlinsol_dense.h for the dense SUNMATRIX and SUNLINSOL modules, the header file sundials_types.h for the definition of realtype and sunindextype, and the file sundials_math.h for the definition of the SUNRabs macro. This program also includes two user-defined accessor macros, Ith and IJth, that are useful in writing the problem functions in a form closely matching their mathematical description, i.e. with components numbered from 1 instead of from 0. Following that, the program defines problem-specific constants and a user-defined data structure, which will be used to pass the values of the parameters p to various user routines. The constant STEPS defines the number of integration steps between two consecutive checkpoints. The program prologue ends with the prototypes of four user-supplied functions that are called by CVODES. The first two provide the right-hand side and dense Jacobian for the forward problem, and the last two provide the right-hand side and dense Jacobian for the backward problem.

The main function begins with type declarations and continues with the allocation and initialization of the user data structure, which contains the values of the parameters p. Next, it allocates and initializes y with the initial conditions for the forward problem, allocates and initializes q for the quadrature used in computing the value G, and finally sets the scalar relative tolerance reltolQ and vector absolute tolerance abstolQ for the quadrature variables. No tolerances for the state variables are defined since cvsRoberts_ASAi_dns uses its own function to compute the error weights for WRMS norm estimates of state solution vectors.

The call to CVodeCreate creates the main integrator memory block for the forward integration and specifies the CV_BDF integration method. The call to CVodeInit initializes the forward integration by specifying the initial conditions. The call to CVodeWFtolerances specifies a function that computes error weights. The next call specifies the optional user data pointer data. The linear solver is selected to be SUNLINSOL_DENSE through calls to create the template Jacobian matrix and dense linear solver objects (SUNDenseMatrix and SUNLinSol_Dense), and to attach these to the CVODES integrator via the call to CVodeSetLinearSolver. The user-provided Jacobian routine Jac is specified through a call to CVodeSetJacFn.

The next code block initializes quadrature computations in the forward phase, by allocating CVODES memory for quadrature integration (the call to CVodeQuadInit specifies the right-hand side fQ of the quadrature equation and the initial values of the quadrature variable), setting the integration tolerances for the quadrature variables, and finally including the quadrature variable in the error test.

Allocation for the memory block of the combined forward-backward problem is accomplished through the call to CVadjInit which specifies STEPS = 150, the number of steps between two checkpoints, and specifies cubic Hermite interpolation.

The call to CVodeF requests the solution of the forward problem to TOUT. If successful, at the end of the integration, CVodeF will return the number of saved checkpoints in the argument ncheck (optionally, a list of the checkpoints can be obtained by calling CVodeGetAdjCheckPointsInfo and the checkpoint information printed).

The next segment of code deals with the setup of the backward problem. First, a serial vector yB of length NEQ is allocated and initalized with the value of $\lambda(= 0.0)$ at the final time (TB1 = 4.0E7). A second serial vector qB of dimension NP is created and initialized to 0.0. This vector corresponds to the quadrature variables ξ whose values at t_0 will be the components of the desired gradient of $\partial G/\partial p$ (after a sign change). Following that, the program sets the relative and absolute tolerances for the backward integration.

The CVODES memory for the backward integration is created and allocated by the calls to the interface routines CVodeCreateB and CVodeInitB which specify the CV_BDF integration method, among other things. The dense linear solver is created and initialized by calling the SUNDenseMatrix, SUNLinSol_Dense and CVodeSetLinearSolverB routines, and specifying a non-NULL Jacobian routine JacB and user data data.

The tolerances for the integration of quadrature variables, reltolB and abstolQB, are speci ed through CVodeQuadSStolerancesB. The call to CVodeSetQuadErrConB indicates that ξ should be included in the error test. Quadrature computation is initialized by calling CVodeQuadInitB which speci es the right-hand side of the quadrature equations as fQB.

The actual solution of the backward problem is accomplished through two calls to CVodeB | one for intermediate output at t = 40, and one for the nal time T0 = 0. At each point, the backward solution yB (= λ) is obtained with a call to CVodeGetB and the forward solution with a call to CVodeGetAdjY. The values of the quadrature variables ξ at time T0 are loaded in qB by calling the extraction routine CVodeGetQuadB. The negative of qB gives the gradient $\partial G/\partial p$.

The main program then carries out a second backward problem. It calls to CVodeReInitB and CVodeQuadReInitB to re-initialize the backward memory block for a new adjoint computation with a di erent nal time (TB2 = 50). This is followed by two calls to CVodeB, one for intermediate output at t = 40 and one for the nal values at t = 0. Finally, the gradient $\partial G/\partial p$ of the second function G is printed.

The main program ends by freeing previously allocated memory by calling CVodeFree (for the CVODES memory for the forward problem), CVadjFree (for the memory allocated for the combined problem), and N_VFree_Serial (for the various vectors).

The user-supplied functions f and Jac for the right-hand side and Jacobian of the forward problem are straightforward expressions of its mathematical formulation (11). The function ewt is the same as the one for cvRoberts_dns_uw.c. The function fQ implements (16), while fB, JacB, and fQB are mere translations of the backward problem (14) and (15).

The output generated by cvsRoberts_ASAi_dns is shown below.

cvsRoberts_ASAi_dns sample output

```

Adjoint Sensitivity Example for Chemical Kinetics
-------------------------------------------------

ODE: dy1/dt = -p1*y1 + p2*y2*y3
     dy2/dt =  p1*y1 - p2*y2*y3 - p3*(y2)^2
     dy3/dt =  p3*(y2)^2

Find dG/dp for
     G = int_t0^tB0 g(t,p,y) dt
     g(t,p,y) = y3


Create and allocate CVODES memory for forward runs
Forward integration ... done ( nst = 766 )

ncheck = 5

--------------------------------------------------------
G:            3.9983e+07
--------------------------------------------------------
```

```
Create and allocate CVODES memory for backward run
Backward integration from tB0 =   4.0000e+07

---------------------------------------------------------
returned t:   4.0000e+01
tout:         4.0000e+01
lambda(t):    3.9967e+07   3.9967e+07   3.9967e+07
y(t):         7.1583e-01   9.1855e-06   2.8416e-01
---------------------------------------------------------

Done ( nst = 212 )
---------------------------------------------------------
returned t:   0.0000e+00
lambda(t0):   3.9967e+07   3.9967e+07   3.9967e+07
y(t0):        1.0000e+00   0.0000e+00   0.0000e+00
dG/dp:        7.6842e+05  -3.0691e+00   5.1144e-04
---------------------------------------------------------

Re-initialize CVODES memory for backward run
Backward integration from tB0 =   5.0000e+01

---------------------------------------------------------
returned t:   4.0000e+01
tout:         4.0000e+01
lambda(t):    2.8959e-01   1.7624e+00   9.3567e+00
y(t):         7.1583e-01   9.1855e-06   2.8416e-01
---------------------------------------------------------

Done ( nst = 186 )
---------------------------------------------------------
returned t:   0.0000e+00
lambda(t0):   8.4190e+00   1.6097e+01   1.6097e+01
y(t0):        1.0000e+00   0.0000e+00   0.0000e+00
dG/dp:        1.7341e+02  -5.0590e-04   8.4321e-08
---------------------------------------------------------

Free memory
```

3.2 A parallel nonstiff example: cvsAdvDiff_ASAp_non_p

As an example of using the CVODES adjoint sensitivity module with the parallel vector module NVECTOR_PARALLEL, we describe a sample program that solves the following problem: Consider the 1-D advection-diffusion equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= p_1 \frac{\partial^2 u}{\partial x^2} + p_2 \frac{\partial u}{\partial x} \\ 0 = x_0 &\le x \le x_1 = 2 \\ 0 = t_0 &\le t \le t_f = 2.5 \,, \end{aligned} \tag{17}$$

with boundary conditions $u(t, x_0) = u(t, x_1) = 0$, $\forall t$, and initial condition $u(t_0, x) = u_0(x) = x(2 - x)e^{2x}$. Also consider the function

$$g(t) = \int_{x_0}^{x_1} u(t, x) dx \,.$$

We wish to find, through adjoint sensitivity analysis, the gradient of $g(t_f)$ with respect to $p = [p_1; p_2]$ and the perturbation in $g(t_f)$ due to a perturbation δu_0 in u_0.

The approach we take in the program `cvsAdvDiff_ASAp_non_p` is to first derive an adjoint PDE which is then discretized in space and integrated backwards in time to yield the desired sensitivities. A straightforward extension to PDEs of the derivation given in §2.7 gives

$$\frac{dg}{dp}(t_f) = \int_{t_0}^{t_f} dt \int_{x_0}^{x_1} dx \mu \cdot \left[\frac{\partial^2 u}{\partial x^2}; \frac{\partial u}{\partial x} \right] \tag{18}$$

and

$$\delta g|_{t_f} = \int_{x_0}^{x_1} \mu(t_0, x) \delta u_0(x) dx \,, \tag{19}$$

where μ is the solution of the adjoint PDE

$$\begin{aligned} &\frac{\partial \mu}{\partial t} + p_1 \frac{\partial^2 \mu}{\partial x^2} - p_2 \frac{\partial \mu}{\partial x} = 0 \\ &\mu(t_f, x) = 1 \\ &\mu(t, x_0) = \mu(t, x_1) = 0 \,. \end{aligned} \tag{20}$$

Both the forward problem (17) and the backward problem (20) are discretized on a uniform spatial grid of size $M_x + 2$ with central differencing and with boundary values eliminated, leaving ODE systems of size $N = M_x$ each. As always, we deal with the time quadratures in (18) by introducing the additional equations

$$\begin{aligned} \dot{\xi}_1 &= \int_{x_0}^{x_1} dx \mu \frac{\partial^2 u}{\partial x^2} \,, \quad \xi_1(t_f) = 0 \,, \\ \dot{\xi}_2 &= \int_{x_0}^{x_1} dx \mu \frac{\partial u}{\partial x} \,, \quad \xi_2(t_f) = 0 \,, \end{aligned} \tag{21}$$

yielding

$$\frac{dg}{dp}(t_f) = - [\xi_1(t_0); \xi_2(t_0)]$$

The space integrals in (19) and (21) are evaluated numerically, on the given spatial mesh, using the trapezoidal rule.

Note that $\mu(t_0, x^*)$ is nothing but the perturbation in $g(t_f)$ due to a δ-function perturbation $\delta u_0(x) = \delta(x - x^*)$ in the initial conditions. Therefore, $\mu(t_0, x)$ completely describes $\delta g(t_f)$ for any perturbation δu_0.

Both the forward and the backward problems are solved with the option for nonstiff systems, i.e. using the Adams method with fixed-point iteration for the solution of the nonlinear systems. The overall structure of the `main` function is very similar to that of the code `cvsRoberts_ASAi_dns` discussed previously with differences arising from the use of the parallel NVECTOR module. Unlike `cvsRoberts_ASAi_dns`, the example `cvsAdvDiff_ASAp_non_p` illustrates computation of the additional quadrature variables by appending `NP` equations to the adjoint system. This approach can be a better alternative to using special treatment of the quadrature equations when their number is too small for parallel treatment.

Besides the parallelism implemented by CVODES at the NVECTOR level, this example uses MPI calls to parallelize the calculations of the right-hand side routines `f` and `fB` and of the spatial integrals involved. The forward problem has size `NEQ = MX`, while the backward problem has size `NB = NEQ + NP`, where `NP = 2` is the number of quadrature equations in

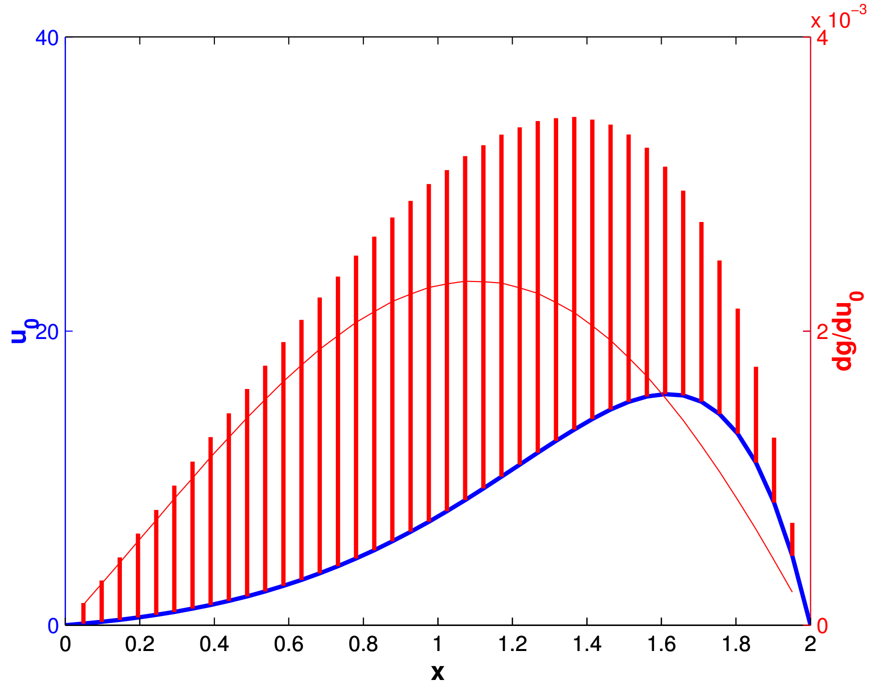

Figure 4: Results for the `cvsAdvDiff_ASAp_non_p` example problem. The gradient of $g(t_f)$ with respect to the initial conditions u_0 is shown superimposed over the values u_0.

(21). The use of the total number of available processes on two problems of di erent sizes deserves some comments, as this is typical in adjoint sensitivity analysis. Out of the total number of available processes, namely `nprocs`, the rst `npes = nprocs - 1` processes are dedicated to the integration of the ODEs arising from the semi-discretization of the PDEs (17) and (20), and receive the same load on both the forward and backward integration phases. The last process is reserved for the integration of the quadrature equations (21), and is therefore inactive during the forward phase. Of course, for problems involving a much larger number of quadrature equations, more than one process could be reserved for their integration. An alternative would be to redistribute the `NB` backward problem variables over all available processes, without any relationship to the load distribution of the forward phase. However, the approach taken in `cvsAdvDiff_ASAp_non_p` has the advantage that the communication strategy adopted for the forward problem can be directly transferred to communication among the rst `npes` processes during the backward integration phase.

We must also emphasize that, although inactive during the forward integration phase, the last process *must* participate in that phase with a *zero local array length*. This is because, during the backward integration phase, this process must have its own local copy of variables (such as `cvadj_mem`) that were set only during the forward phase.

Using `MX = 40` on 4 processes, the gradient of $g(t_f)$ with respect to the two problem parameters is obtained as $dg/dp(t_f)$ = [1.13856; 1.01023]. The gradient of $g(t_f)$ with respect to the initial conditions is shown in Fig. 4. The gradient is plotted superimposed over the initial conditions. Sample output generated by `cvsAdvDiff_ASAp_non_p`, for `MX = 20`, is shown below.

`cvsAdvDiff_ASAp_non_p sample output`

```
g(tf) = 2.444739e-02

dgdp(tf)
  [ 1]: -1.502107e-01
  [ 2]: -1.097739e-02

mu(t0)
  [ 1]: 2.776607e-04
  [ 2]: 5.619775e-04
  [ 3]: 8.477404e-04
  [ 4]: 1.126412e-03
  [ 5]: 1.393777e-03
  [ 6]: 1.639607e-03
  [ 7]: 1.861184e-03
  [ 8]: 2.047397e-03
  [ 9]: 2.197434e-03
  [10]: 2.300275e-03
  [11]: 2.357283e-03
  [12]: 2.358593e-03
  [13]: 2.307827e-03
  [14]: 2.197332e-03
  [15]: 2.032873e-03
  [16]: 1.809960e-03
  [17]: 1.536162e-03
  [18]: 1.210898e-03
  [19]: 8.430003e-04
  [20]: 4.362428e-04
```

3.3 A parallel example using CVBBDPRE: cvsAtmDisp_ASAi_kry_bbd_p

As a more elaborate example of a parallel adjoint sensitivity calculation, we describe next the program `cvsAtmDisp_ASAi_kry_bbd_p` provided with CVODES. This example models an atmospheric release with an advection-di usion PDE in 2-D or 3-D and computes the gradient with respect to source parameters of the space-time average of the squared norm of the concentration. Given a known velocity eld $v(t,x)$ and source function S, the transport equation for the concentration $c(t,x)$ in a domain is given by

$$
\begin{aligned}
\frac{\partial c}{\partial t} \quad k\, r^2 c + v \quad r\, c + S = 0\,, \text{ in } (0,T) \\
\frac{\partial c}{\partial n} = g\,, \text{ on } (0,T) \quad \partial \\
c = c_0(x)\,, \text{ in } \quad \text{at } t = 0\,,
\end{aligned}
\tag{22}
$$

where is a box in $\mathbb{R}^2$ or $\mathbb{R}^3$ and n is the normal to the boundary of . We assume homogeneous boundary conditions ($g = 0$) and a zero initial concentration everywhere in ($c_0(x$

where x_i is the location of the source of intensity $S(x_i) = p_i$, and λ is solution of the adjoint PDE

$$
\begin{aligned}
-\frac{\partial \lambda}{\partial t} - k \nabla^2 \lambda - v \cdot \nabla \lambda &= c(t,x)\,, \text{ in } (T,0) \times \Omega \\
(k \nabla \lambda + v\lambda) \cdot n &= 0\,, \text{ on } (0,T) \times \partial\Omega \\
\lambda &= 0\,, \text{ in } \Omega \text{ at } t = T\,.
\end{aligned}
\tag{25}
$$

The PDE (22) is semi-discretized in space with central finite differences, with the boundary conditions explicitly taken into account by using layers of ghost cells in every direction. If the direction x^i of Ω is discretized into m_i intervals, this leads to a system of ODEs of dimension $N = \prod_1^d (m_i + 1)$, with $d = 2$, or $d = 3$. The source term S is parameterized as a piecewise constant function and yielding N parameters in the problem. The nominal values of the source parameters correspond to two Gaussian sources.

The source code as supplied runs the 2-D problem. To obtain the 3-D version, add a line `#define USE3D` at the top of `main`.

The adjoint PDE (25) is discretized to a system of ODEs in a similar fashion. The space integrals in (23) and (24) are simply approximated by their Riemann sums, while the time integrals are resolved by appending pure quadrature equations to the systems of ODEs.

We use BDF with the SUNLINSOL_SPGMR linear solver module and the CVBBDPRE preconditioner for both the forward and the backward integration phases. The value of G is computed on the forward phase as a quadrature, while the components of the gradient dG/dp are computed as quadratures during the backward integration phase. All quadrature variables are included in the corresponding error tests.

Communication between processes for the evaluation of the ODE right-hand sides involves passing the solution on the local boundaries (lines in 2-D, surfaces in 3-D) to the 4 (6 in 3-D) neighboring processes. This is implemented in the function `f_comm`, called in `f` and `fB` before evaluation of the local residual components. Since there is no additional communication required for the CVBBDPRE preconditioner, a `NULL` pointer is passed for `gloc` and `glocB` in the calls to `CVBBDPrecInit` and `CVBBDPrecInitB`, respectivley.

For the sake of clarity, the `cvsAtmDisp_ASAi_kry_bbd_p` example does not use the most memory-efficient implementation possible, as the local segment of the solution vectors (`y` on the forward phase and `yB` on the backward phase) and the data received from neighboring processes is loaded into a temporary array `y_ext` which is then used exclusively in computing the local components of the right-hand sides.

Note that if `cvsAtmDisp_ASAi_kry_bbd_p` is given any command line argument, it will generate a series of MATLAB files which can be used to visualize the solution. The results of a 2-D simulation and adjoint sensitivity analysis with `cvsAtmDisp_ASAi_kry_bbd_p` on a 80×80 grid and $2 \times 4 = 8$ processes are shown in Fig. 5. Results in 3-D[†], on a $80 \times 80 \times 40$ grid and $2 \times 4 \times 2 = 16$ processes are shown in Figs. 6 and 7. A sample output generated by `cvsAtmDisp_ASAi_kry_bbd_p` for a 2D calculation is shown below.

cvsAtmDisp_ASAi_kry_bbd_p sample output

```
Parallel Krylov adjoint sensitivity analysis example
2D Advection diffusion PDE with homogeneous Neumann B.C.
Computes gradient of G = int_t_Omega ( c_i^2 ) dt dOmega
with respect to the source values at each grid point.
```

[†]The name of the executable for the 3-D version is `cvsAtmDisp_ASAi_kry_bbd_p3D`.

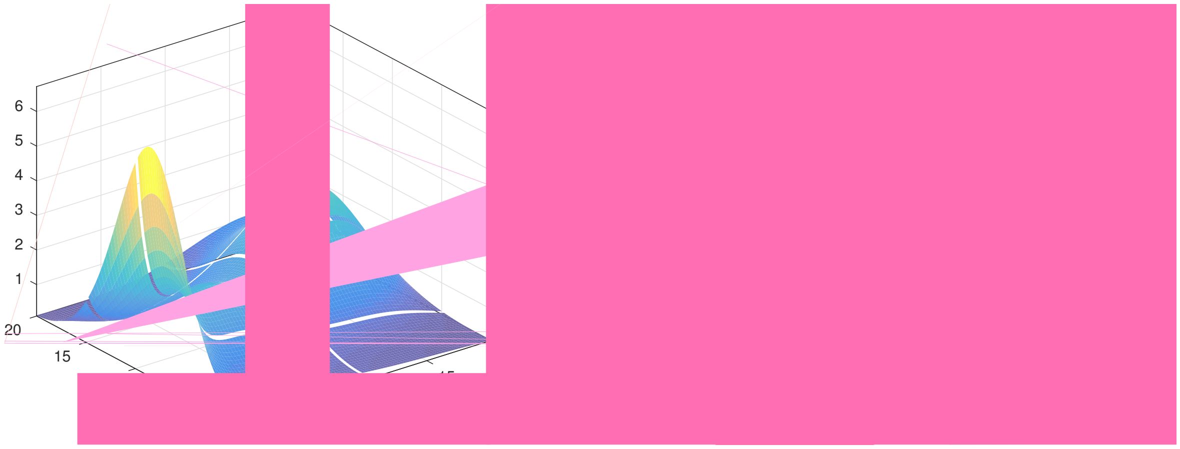

Figure 5: Results for the `cvsAtmDisp_ASAi_kry_bbd_p` example problem in 2D. The gradient with respect to the source parameters is pictured on the left. On the right, the gradient was color-coded and superimposed over the nominal value of the source parameters.

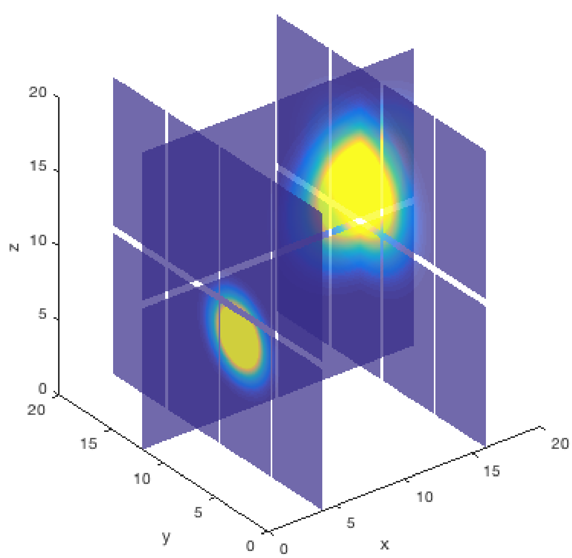

Figure 6: Results for the `cvsAtmDisp_ASAi_kry_bbd_p` example problem in 3D. Nominal values of the source parameters.

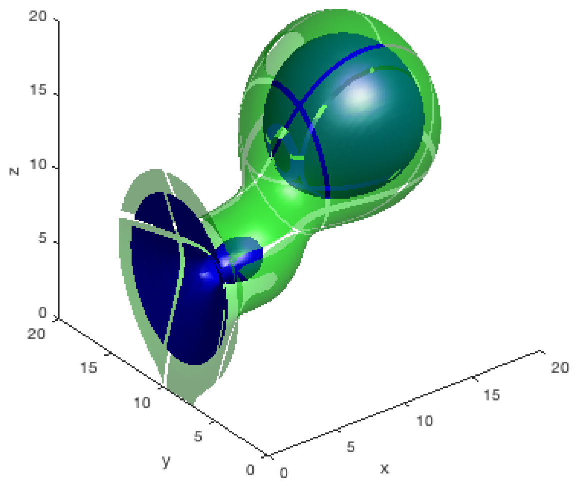

Figure 7: Results for the `cvsAtmDisp_ASAi_kry_bbd_p` example problem in 3D. Two isosurfaces of the gradient with respect to the source parameters. They correspond to values of 0.25 (green) and 0.4 (blue).

```
Domain:
   0.000000 < x < 20.000000    mx = 80   npe_x = 2
   0.000000 < y < 20.000000    my = 80   npe_y = 4

Begin forward integration... done.   G = 4.791513e+03

Final Statistics..

lenrw    =  85469     leniw =    420
llrw     =  78788     lliw  =    202
nst      =    174
nfe      =    178     nfel  =    310
nni      =    175     nli   =    310
nsetups  =     18     netf  =      0
npe      =      4     nps   =    482
ncfn     =      0     ncfl  =      0

Begin backward integration... done.

Final Statistics..

lenrw    = 150999     leniw =    420
llrw     =  78788     lliw  =    202
nst      =    118
nfe      =    134     nfel  =    277
nni      =    131     nli   =    277
nsetups  =     16     netf  =      0
npe      =      3     nps   =    399
```

```
ncfn     =       0     ncfl  =       0
```

4 Parallel tests

The most preeminent advantage of CVODES over existing sensitivity solvers is the possibility of solving very large-scale problems on massively parallel computers. To illustrate this point we present speedup results for the integration and forward sensitivity analysis for an ODE system generated from the following 2-species diurnal kinetics advection-diffusion PDE system in 2 space dimensions. This work was reported in [3]. The PDE is a modification of that described in [4], and takes the form:

$$\frac{dc_i}{dt} = K_h \frac{d^2 c_i}{dx^2} + v \frac{dc_i}{dx} + K_v \frac{d^2 c_i}{dz^2} + R_i(c_1, c_2, t)\,, \quad \text{for } i = 1, 2\,,$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 c_1 c_3 - q_2 c_1 c_2 + 2 q_3(t) c_3 + q_4(t) c_2, \\ R_2(c_1, c_2, t) &= q_1 c_1 c_3 - q_2 c_1 c_2 - q_4(t) c_2\,, \end{aligned}$$

K_h, K_v, v, q_1, q_2, and c_3 are constants, and $q_3(t)$ and $q_4(t)$ vary diurnally. The problem is posed on the square $0 \le x \le 20$, $30 \le z \le 50$ (all in km), with homogeneous Neumann boundary conditions, and for time t in $0 \le t \le 86400$ (1 day). The PDE system is treated by central differences on a uniform mesh, except for the advection term, which is treated with a biased 3-point difference formula. The initial profiles are proportional to a simple polynomial in x and a hyperbolic tangent function in z.

The solution with CVODES is done with the BDF/GMRES method (i.e. using the SUNLINSOL_SPGMR linear solver module) and the block-diagonal part of the Newton matrix as a left preconditioner. A copy of the block-diagonal part of the Jacobian is saved and conditionally reused within the preconditioner setup function.

The problem is solved by CVODES using P processes, treated as a rectangular process grid of size $p_x \times p_z$. Each process is assigned a subgrid of size $n = n_x \times n_z$ of the (x, z) mesh. Thus the actual mesh size is $N_x \times N_z = (p_x n_x) \times (p_z n_z)$, and the ODE system size is $N = 2 N_x N_z$. Parallel performance tests were performed on ASCI Frost, a 68-node, 16-way SMP system with POWER3 375 MHz processors and 16 GB of memory per node. We present timing results for the integration of only the state equations (column STATES), as well as for the computation of forward sensitivities with respect to the diffusion coefficients K_h and K_v using the staggered corrector method without and with error control on the sensitivity variables (columns STG and STG_FULL, respectively). Run times for a global problem size of $N = 2 N_x N_y = 2 \cdot 1600 \cdot 400 = 1,280,000$ are shown in Fig. 8 and listed below.

P	STATES	STG	STG_FULL
4	460.31	1414.53	2208.14
8	211.20	646.59	1064.94
16	97.16	320.78	417.95
32	42.78	137.51	210.84
64	19.50	63.34	83.24
128	13.78	42.71	55.17
256	9.87	31.33	47.95

We note that there was not enough memory to solve the problem (even without carrying sensitivities) using fewer processes.

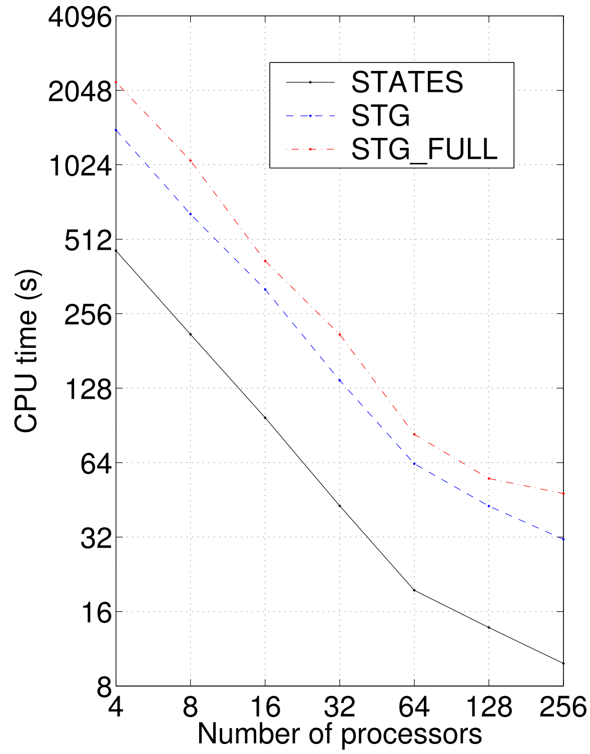

Figure 8: Speedup results for the integration of the state equations only (solid line), staggered sensitivity analysis without error control on the sensitivity variables (dashed line), and staggered sensitivity analysis with full error control (dotted line)

The departure from the ideal line of slope -1 is explained by the interplay of several conflicting processes. On one hand, when increasing the number of processes, the preconditioner quality decreases, as it incorporates a smaller and smaller fraction of the Jacobian, and the cost of interprocess communication increases. On the other hand, decreasing the number of processes leads to an increase in the cost of the preconditioner setup phase and to a larger local problem size which can lead to a point where a node starts memory-paging to disk.

References

[1] A. C. Hindmarsh and R. Serban. User Documentation for CVODES v4.0.1. Technical report, LLNL, 2018. UCRL-SM-208111.

[2] A. C. Hindmarsh, R. Serban, and D. R. Reynolds. Example Programs for CVODE v4.0.1. Technical report, LLNL, 2018. UCRL-SM-208110.

[3] R. Serban and A. C. Hindmarsh. CVODES, the sensitivity-enabled ODE solver in SUNDIALS. In *Proceedings of the 5th International Conference on Multibody Systems, Nonlinear Dynamics and Control*, Long Beach, CA, 2005. ASME.

[4] M. R. Wittman. Testing of PVODE, a Parallel ODE Solver. Technical Report UCRL-ID-125562, LLNL, August 1996.