Example Programs for ARKode

Release 2.1.1

Daniel R. Reynolds
Department of Mathematics
Southern Methodist University

May 7, 2018



CONTENTS

1 Serial C example problems **3**
- 1.1 ark_analytic . . . 3
- 1.2 ark_analytic_nonlin . . . 4
- 1.3 ark_brusselator . . . 5
- 1.4 ark_brusselator_fp . . . 7
- 1.5 ark_robertson . . . 7
- 1.6 ark_robertson_root . . . 8
- 1.7 ark_brusselator1D . . . 9
- 1.8 ark_brusselator1D_klu . . . 11
- 1.9 ark_brusselator1D_FEM_slu . . . 11
- 1.10 ark_heat1D . . . 13
- 1.11 ark_heat1D_adapt . . . 14
- 1.12 ark_KrylovDemo_prec . . . 15

2 OpenMP C example problems **17**
- 2.1 ark_brusselator1D_omp . . . 17

3 Parallel C example problems **19**
- 3.1 ark_diurnal_kry_bbd_p . . . 19
- 3.2 ark_diurnal_kry_p . . . 20

4 Parallel Hypre example problems **21**
- 4.1 ark_diurnal_kry_ph . . . 21

5 Serial C++ example problems **23**
- 5.1 ark_analytic_sys . . . 23

6 Parallel C++ example problems **25**
- 6.1 ark_heat2D . . . 25

7 Serial Fortran 77 example problems **29**
- 7.1 fark_diurnal_kry_bp . . . 29
- 7.2 fark_roberts_dnsL . . . 30

8 Parallel Fortran 77 example problems **31**
- 8.1 fark_diag_kry_bbd_p . . . 31
- 8.2 fark_diag_non_p . . . 31

9 Serial Fortran 90 example problems **33**
- 9.1 ark_bruss . . . 33

10 Parallel Fortran 90 example problems **35**
10.1 fark_heat2D 35

Bibliography **37**

This is the documentation for the ARKode examples. ARKode is an adaptive step time integration package for stiff, nonstiff and multi-rate systems of ordinary differential equations (ODEs). The ARKode solver is a component of the SUNDIALS suite of nonlinear and differential/algebraic equation solvers. It is designed to have a similar user experience to the CVODE solver, with user modes to allow adaptive integration to specified output times, return after each internal step and root-finding capabilities, for calculations both in serial and parallel (via MPI). The default integration and solver options should apply to most users, though complete control over all internal parameters and time adaptivity algorithms is enabled through optional interface routines.

ARKode is developed by Southern Methodist University, with support by the US Department of Energy through the FASTMath SciDAC Institute, under subcontract B598130 from Lawrence Livermore National Laboratory.

Along with the ARKode solver, we have created a suite of example problems demonstrating its usage on applications written in C, C++ and Fortran 77 and Fortran 90. These examples demonstrate a large variety of ARKode solver options, including explicit, implicit and ImEx solvers, root-finding, Newton and fixed-point nonlinear solvers, direct and iterative linear solvers, adaptive resize capabilities, and the Fortran solver interface. While these examples are not an exhaustive set of all possible usage scenarios, they are designed to show a variety of exemplars, and can be used as templates for new problems using ARKode's solvers. Further information on the ARKode package itself may be found in the accompanying ARKode user guide *[R2017]*.

The following tables summarize the salient features of each of the example problems in this document. Each example is designed to be relatively self-contained, so that you need only study and/or emulate the problem that is most closely related to your own. We group these examples according to programming language (C, C++, Fortran 77, Fortran 90).

ARKode example problems written in C are summarized in the table below, and are further described in the chapters *Serial C example problems*, *OpenMP C example problems*, *Parallel C example problems* and *Parallel Hypre example problems*.

Problem	Integrator	Nonlinear	Linear	Size	Extras
ark_analytic	DIRK	Newton	Dense	1	
ark_analytic_nonlin	ERK	N.A.	N.A.	1	
ark_brusselator	DIRK	Newton	Dense	3	
ark_brusselator_fp	ARK	Fixed-point	N.A.	3	
ark_robertson	DIRK	Newton	Dense	3	
ark_robertson_root	DIRK	Newton	Dense	3	rootfinding
ark_brusselator1D	DIRK	Newton	Band	3N	
ark_brusselator1D_omp	DIRK	Newton	Band	3N	OpenMP-enabled
ark_brusselator1D_klu	DIRK	Newton	KLU	3N	sparse matrices
ark_brusselator1D_FEM_sl	DIRK	Newton	SuperLU_MT	3N	finite-element, $M \neq I$, sparse matrices
ark_heat1D	DIRK	Newton	PCG	N	
ark_heat1D_adapt	DIRK	Newton	PCG	(dynamic)	adaptive vector resizing
ark_KrylovDemo_prec	DIRK	Newton	SPGMR	216	multiple preconditioners
ark_diurnal_kry_bbd_p	DIRK	Newton	SPGMR	200	parallel, BBD preconditioner
ark_diurnal_kry_p	DIRK	Newton	SPGMR	200	parallel, block-diagonal precond.
ark_diurnal_kry_ph	DIRK	Newton	SPGMR	200	HYPRE parallel vector

ARKode example problems written in C++ are summarized in the table below, and are further described in the chapters *Serial C++ example problems* and *Parallel C++ example problems*.

Problem	Integrator	Nonlinear	Linear	Size	Extras
ark_analytic_sys	DIRK	Newton	Dense	3	
ark_heat2D	DIRK	Newton	PCG	$nx * ny$	parallel

ARKode example problems written in Fortran 77 are summarized in the table below, and are further described in the

chapters *Serial Fortran 77 example problems* and *Parallel Fortran 77 example problems*.

Problem	Integrator	Nonlinear	Linear	Size	Extras
fark_diurnal_kry_bp	DIRK	Newton	SPGMR	10	banded preconditioner
fark_roberts_dnsL	DIRK	Newton	Dense	3	LAPACK dense solver, rootfinding
fark_diag_kry_bbd_p	DIRK	Newton	SPGMR	10*NProcs	parallel BBD preconditioner
fark_diag_non_p	ERK	N.A.	N.A.	10*NProcs	parallel

ARKode example problems written in Fortran 90 are summarized in the table below, and are further described in the chapters *Serial Fortran 90 example problems* and *Parallel Fortran 90 example problems*.

Problem	Integrator	Nonlinear	Linear	Size	Extras
ark_bruss	ARK	Newton	Dense	3	
fark_heat2D	DIRK	Newton	PCG	$nx * ny$	parallel

CHAPTER

ONE

SERIAL C EXAMPLE PROBLEMS

1.1 ark_analytic

This is a very simple C example showing how to use the ARKode solver interface.

The problem is that of a scalar-valued initial value problem (IVP) that is linear in the dependent variable y, but nonlinear in the independent variable t:

$$\frac{dy}{dt} = \lambda y + \frac{1}{1+t^2} - \lambda \arctan(t),$$

where $0 \leq t \leq 10$ and $y(0) = 0$. The stiffness of the problem may be tuned via the parameter λ. The value of λ must be negative to result in a well-posed problem; for values with magnitude larger than 100 or so the problem becomes quite stiff. Here, we choose $\lambda = -100$. After each unit time interval, the solution is output to the screen.

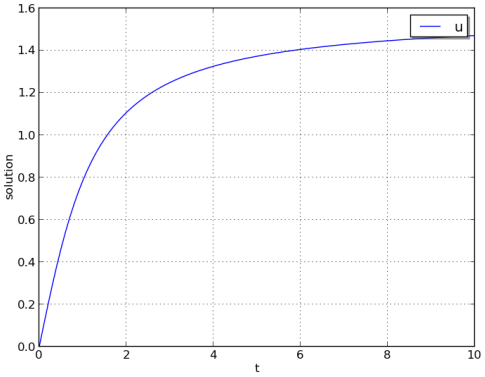
1.1.1 Numerical method

The example routine solves this problem using a diagonally-implicit Runge-Kutta method. Each stage is solved using the built-in modified Newton iteration, but since the ODE is linear in y these should only require a single iteration per stage. Internally, Newton will use the SUNLINSOL_DENSE linear solver via the ARKDLS interface, which in the case of this scalar-valued problem is just division. The example file contains functions to evaluate both $f(t, y)$ and $J(t, y) = \lambda$.

We specify the relative and absolute tolerances, $rtol = 10^{-6}$ and $atol = 10^{-10}$, respectively. Aside from these choices, this problem uses only the default ARKode solver parameters.

1.1.2 Solutions

This problem is included both as a simple example, but also because it has an analytical solution, $y(t) = \arctan(t)$. As seen in the plots below, the computed solution tracks the analytical solution quite well (left), and results in errors below those specified by the input error tolerances (right).



1.2 ark_analytic_nonlin

This example problem is only marginally more difficult than the preceding problem, in that the ODE right-hand side function is nonlinear in the solution y. While the implicit solver from the preceding problem would also work on this example, because it is not stiff we use this to demonstrate how to use ARKode's explicit solver interface.

The ODE problem is

$$\frac{dy}{dt} = (t+1)e^{-y},$$

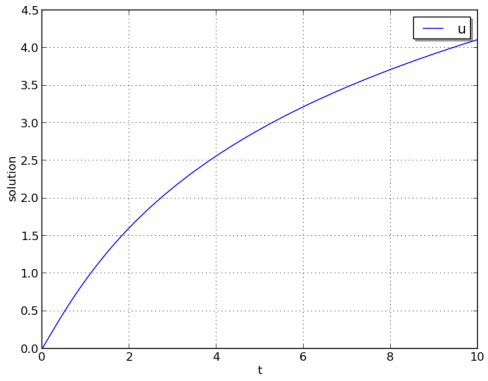
for the interval $t \in [0.0, 10.0]$, with initial condition $y(0) = 0$. This has analytical solution $y(t) = \log\left(\frac{t^2}{2} + t + 1\right)$.

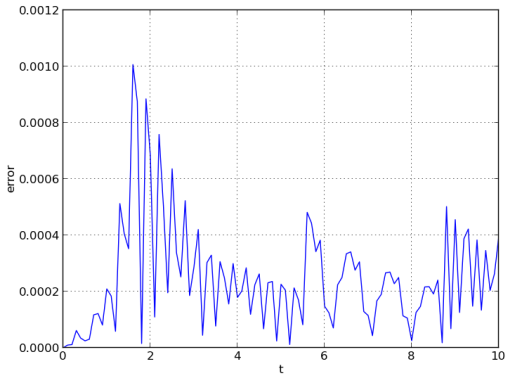
1.2.1 Numerical method

This program solves the problem with the ERK method. Output is printed every 1.0 units of time (10 total). Run statistics (optional outputs) are printed at the end.

1.2.2 Solutions

This problem is included both as a simple example to test the nonlinear solvers within ARKode, but also because it has an analytical solution, $y(t) = \log\left(\frac{t^2}{2} + t + 1\right)$. As seen in the plots below, the computed solution tracks the analytical solution quite well (left), and results in errors comparable with those specified by the requested error tolerances (right).





1.3 ark_brusselator

We now wish to exercise the ARKode solvers on more challenging nonlinear ODE systems. The following test simulates a brusselator problem from chemical kinetics, and is widely used as a standard benchmark problem for new solvers. The ODE system has 3 components, $Y = [u, v, w]$, satisfying the equations,

$$\begin{aligned}
\frac{du}{dt} &= a - (w+1)u + vu^2, \\
\frac{dv}{dt} &= wu - vu^2, \\
\frac{dw}{dt} &= \frac{b-w}{\varepsilon} - wu.
\end{aligned}$$

We integrate over the interval $0 \le t \le 10$, with the initial conditions $u(0) = u_0$, $v(0) = v_0$, $w(0) = w_0$. After each unit time interval, the solution is output to the screen.

The problem implements 3 different testing scenarios:

Test 1: $u_0 = 3.9$, $v_0 = 1.1$, $w_0 = 2.8$, $a = 1.2$, $b = 2.5$, and $\varepsilon = 10^{-5}$

Test 2: $u_0 = 1.2$, $v_0 = 3.1$, $w_0 = 3$, $a = 1$, $b = 3.5$, and $\varepsilon = 5 \cdot 10^{-6}$

Test 3: $u_0 = 3$, $v_0 = 3$, $w_0 = 3.5$, $a = 0.5$, $b = 3$, and $\varepsilon = 5 \cdot 10^{-4}$

The example problem currently selects test 2, though that value may be easily adjusted to explore different testing scenarios.

1.3.1 Numerical method

This program solves the problem with the DIRK method, using a Newton iteration with the SUNLINSOL_DENSE linear solver module via the ARKDLS interface. Additionally, this example provides a routine to ARKDLS to compute the dense Jacobian.

The problem is run using scalar relative and absolute tolerances of $rtol = 10^{-6}$ and $atol = 10^{-10}$, respectively.

10 outputs are printed at equal intervals, and run statistics are printed at the end.

1.3.2 Solutions

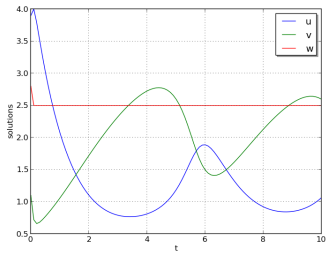
The computed solutions will of course depend on which test is performed:

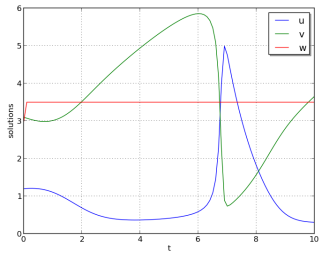
Test 1: Here, all three components exhibit a rapid transient change during the first 0.2 time units, followed by a slow and smooth evolution.

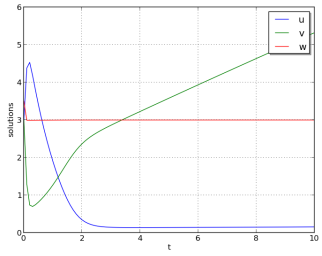
Test 2: Here, w experiences a fast initial transient, jumping 0.5 within a few steps. All values proceed smoothly until around $t = 6.5$, when both u and v undergo a sharp transition, with u increaseing from around 0.5 to 5 and v decreasing from around 6 to 1 in less than 0.5 time units. After this transition, both u and v continue to evolve somewhat rapidly for another 1.4 time units, and finish off smoothly.

Test 3: Here, all components undergo very rapid initial transients during the first 0.3 time units, and all then proceed very smoothly for the remainder of the simulation.

Unfortunately, there are no known analytical solutions to the Brusselator problem, but the following results have been verified in code comparisons against both CVODE and the built-in ODE solver `ode15s` from Matlab:







Brusselator solution plots: left is test 1, center is test 2, right is test 3.

1.4 ark_brusselator_fp

This test problem is a duplicate the `ark_brusselator` problem above, but with a few key changes in the methods used for time integration and nonlinear solver. As with the previous test, this problem has 3 dependent variables u, v and w, that depend on the independent variable t via the IVP system

$$\begin{aligned}
\frac{du}{dt} &= a - (w+1)u + vu^2, \\
\frac{dv}{dt} &= wu - vu^2, \\
\frac{dw}{dt} &= \frac{b-w}{\varepsilon} - wu.
\end{aligned}$$

We integrate over the interval $0 \le t \le 10$, with the initial conditions $u(0) = u_0$, $v(0) = v_0$, $w(0) = w_0$. After each unit time interval, the solution is output to the screen.

Again, we have 3 different testing scenarios,

Test 1: $u_0 = 3.9$, $v_0 = 1.1$, $w_0 = 2.8$, $a = 1.2$, $b = 2.5$, and $\varepsilon = 10^{-5}$

Test 2: $u_0 = 1.2$, $v_0 = 3.1$, $w_0 = 3$, $a = 1$, $b = 3.5$, and $\varepsilon = 5 \cdot 10^{-6}$

Test 3: $u_0 = 3$, $v_0 = 3$, $w_0 = 3.5$, $a = 0.5$, $b = 3$, and $\varepsilon = 5 \cdot 10^{-4}$

with test 2 selected within in the example file.

1.4.1 Numerical method

This program solves the problem with the ARK method, in which we have split the right-hand side into stiff ($f_i(t,y)$) and non-stiff ($f_e(t,y)$) components,

$$f_i(t,y) = \begin{bmatrix} 0 \\ 0 \\ \frac{b-w}{\varepsilon} \end{bmatrix} \qquad f_e(t,y) = \begin{bmatrix} a - (w+1)u + vu^2 \\ wu - vu^2 \\ -wu \end{bmatrix}.$$

Also unlike the previous test problem, we solve the resulting implicit stages using the available accelerated fixed-point solver, enabled through a call to `ARKodeSetFixedPoint`, with an acceleration subspace of dimension 3.

10 outputs are printed at equal intervals, and run statistics are printed at the end.

1.5 ark_robertson

Our next two tests simulate the Robertson problem, corresponding to the kinetics of an autocatalytic reaction, corresponding to the CVODE example of the same name. This is an ODE system with 3 components, $Y = [u,\ v,\ w]^T$, satisfying the equations,

$$\begin{aligned}
\frac{du}{dt} &= -0.04u + 10^4 vw, \\
\frac{dv}{dt} &= 0.04u - 10^4 vw - 3 \cdot 10^7 v^2, \\
\frac{dw}{dt} &= 3 \cdot 10^7 v^2.
\end{aligned}$$

We integrate over the interval $0 \le t \le 10^{11}$, with initial conditions $Y(0) = [1,\ 0,\ 0]^T$.

1.5.1 Numerical method

This program is constructed to solve the problem with the DIRK solver. Implicit subsystems are solved using a Newton iteration with the SUNLINSOL_DENSE dense linear solver module via the ARKDLS interface; a routine is provided to ARKDLS to supply the Jacobian matrix.

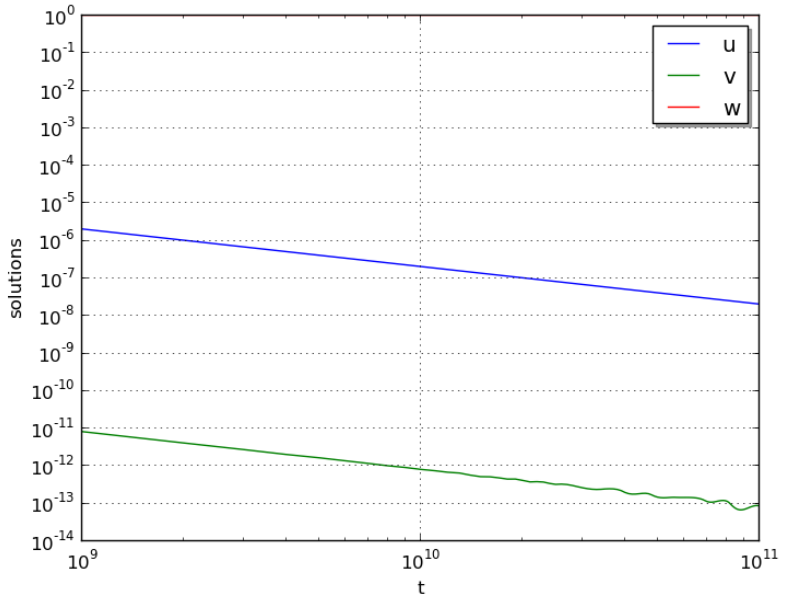
The problem is run using scalar relative and absolute tolerances of $rtol = 10^{-4}$ and $atol = 10^{-11}$, respectively.

100 outputs are printed at equal intervals, and run statistics are printed at the end.

1.5.2 Solutions

Due to the linearly-spaced requested output times in this example, and since we plot in a log-log scale, by the first output at $t = 10^9$, the solutions have already undergone a sharp transition from their initial values of $(u, v, w) = (1, 0, 0)$. For additional detail on the early evolution of this problem, see the following example, that requests logarithmically-spaced output times.

From the plot here, it is somewhat difficult to see the solution values for w, which here all have a value of 1 ± 10^{-5}. Additionally, we see that near the end of the evolution, the values for v begin to exhibit oscillations; this is due to the fact that by this point those values have fallen below their specified absolute tolerance. A smoother behavior (with an increase in time steps) may be obtained by reducing the absolute tolerance for that variable.



1.6 ark_robertson_root

We again test the Robertson problem, but in this example we will utilize both a logarithmically-spaced set of output times (to properly show the solution behavior), as well as ARKode's root-finding capabilities. Again, the Robertson

problem consists of an ODE system with 3 components, $Y = [u,\ v,\ w]^T$, satisfying the equations,

$$\begin{aligned}\frac{du}{dt} &= -0.04u + 10^4 vw,\\ \frac{dv}{dt} &= 0.04u - 10^4 vw - 3\cdot 10^7 v^2,\\ \frac{dw}{dt} &= 3\cdot 10^7 v^2.\end{aligned}$$

We integrate over the interval $0 \le t \le 10^{11}$, with initial conditions $Y(0) = [1,\ 0,\ 0]^T$.

Additionally, we supply the following two root-finding equations:

$$\begin{aligned}g_1(u) &= u - 10^{-4},\\ g_2(w) &= w - 10^{-2}.\end{aligned}$$

While these are not inherently difficult nonlinear equations, they easily serve the purpose of determining the times at which our solutions attain desired target values.

1.6.1 Numerical method

This program solves the problem with the DIRK solver. Implicit subsystems are solved using a Newton iteration with the SUNLINSOL_DENSE linear solver module via the ARKDLS interface; a routine is supplied to provide the dense Jacobian matrix.

The problem is run using scalar relative and vector absolute tolerances. Here, we choose relative tolerance $rtol = 10^{-4}$, and set absolute tolerances on u, v and w of 10^{-8}, 10^{-11} and 10^{-8}, respectively.

100 outputs are printed at equal intervals, and run statistics are printed at the end.

However, unlike in the previous problem, while integrating the system, we use the rootfinding feature of ARKode to find the times at which either $u = 10^{-4}$ or $w = 10^{-2}$.

1.6.2 Solutions

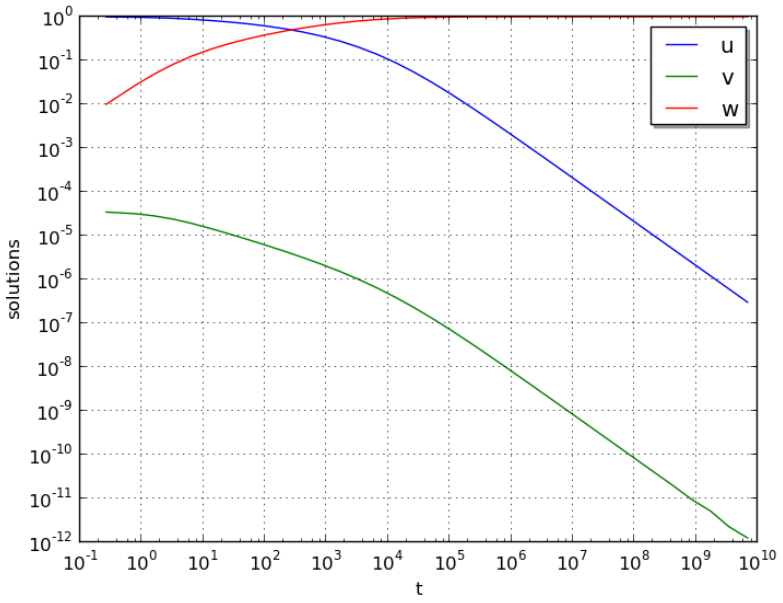
In the solutions below, we now see the early-time evolution of the solution components for the Robertson ODE system.

We note that when running this example, the root-finding capabilities of ARKode report outside of the typical logarithmically-spaced output times to declare that at time $t = 0.264019$ the variable w attains the value 10^{-2}, and that at time $t = 2.07951\cdot 10^7$ the variable u attains the value 10^{-4}; both of our thresholds specified by the root-finding function `g()`.

1.7 ark_brusselator1D

We now investigate a time-dependent system of partial differential equations. We adapt the previously-described brusselator test problem by adding diffusion into the chemical reaction network. We again have a system with 3 components, $Y = [u,\ v,\ w]^T$ that satisfy the equations,

$$\begin{aligned}\frac{\partial u}{\partial t} &= d_u \frac{\partial^2 u}{\partial x^2} + a - (w+1)u + vu^2,\\ \frac{\partial v}{\partial t} &= d_v \frac{\partial^2 v}{\partial x^2} + wu - vu^2,\\ \frac{\partial w}{\partial t} &= d_w \frac{\partial^2 w}{\partial x^2} + \frac{b-w}{\varepsilon} - wu.\end{aligned}$$



However, now these solutions are also spatially dependent. We integrate for $t \in [0, 10]$, and $x \in [0, 1]$, with initial conditions

$$u(0, x) = a + \frac{1}{10}\sin(\pi x),$$
$$v(0, x) = \frac{b}{a} + \frac{1}{10}\sin(\pi x),$$
$$w(0, x) = b + \frac{1}{10}\sin(\pi x),$$

and with stationary boundary conditions, i.e.

$$\frac{\partial u}{\partial t}(t, 0) = \frac{\partial u}{\partial t}(t, 1) = 0,$$
$$\frac{\partial v}{\partial t}(t, 0) = \frac{\partial v}{\partial t}(t, 1) = 0,$$
$$\frac{\partial w}{\partial t}(t, 0) = \frac{\partial w}{\partial t}(t, 1) = 0.$$

We note that these can also be implemented as Dirichlet boundary conditions with values identical to the initial conditions.

1.7.1 Numerical method

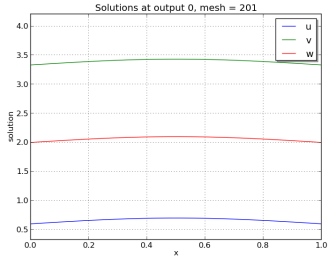
We employ a *method of lines* approach, wherein we first semi-discretize in space to convert the system of 3 PDEs into a larger system of ODEs. To this end, the spatial derivatives are computed using second-order centered differences, with the data distributed over N points on a uniform spatial grid. As a result, ARKode approaches the problem as one involving $3N$ coupled ODEs.

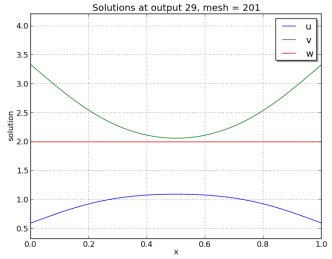
The problem is run using $N = 201$ spatial points, with parameters $a = 0.6$, $b = 2.0$, $d_u = 0.025$, $d_v = 0.025$, $d_w = 0.025$ and $\varepsilon = 10^{-5}$. We specify scalar relative and absolute solver tolerances of $rtol = 10^{-6}$ and $atol = 10^{-10}$, respectively.

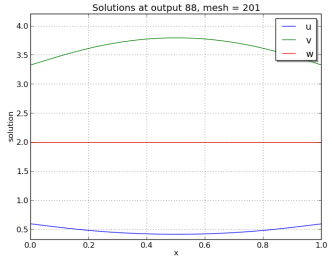
This program solves the problem with a DIRK method, using a Newton iteration with the SUNLINSOL_BAND linear solver module via the ARKDLS interface; a routine is supplied to fill the banded Jacobian matrix.

100 outputs are printed at equal intervals, and run statistics are printed at the end.

1.7.2 Solutions







Brusselator PDE solution snapshots: left is at time $t = 0$, center is at time $t = 2.9$, right is at time $t = 8.8$.

1.8 ark_brusselator1D_klu

This problem is mathematically identical to the preceding problem, *ark_brusselator1D*, but instead of using the SUNMATRIX_BAND banded matrix module and SUNLINSOL_BAND linear solver module, it uses the SUNMATRIX_SPARSE sparse matrix module with the SUNLINSOL_KLU linear solver module. These are still provided to ARKode using the ARKDLS direct linear solver interface, and again a routine is provided to supply a compressed-sparse-column version of the Jacobian matrix. Additionally, the solution is only output 10 times instead of 100.

1.9 ark_brusselator1D_FEM_slu

This problem is mathematically identical to the preceding problems, *ark_brusselator1D* and *ark_brusselator1D_klu*, but utilizes a different set of numerical methods.

1.9.1 Numerical method

As with the preceding problems, we employ a method of lines approach, wherein we first semi-discretize in space to convert the system of 3 PDEs into a larger system of ODEs. However, in this example we discretize in space using a standard piecewise linear, Galerkin finite element method, over a non-uniform discretization of the interval $[0, 1]$ into 100 subintervals. To this end, we must integrate each term in each equation, multiplied by test functions, over each subinterval, e.g.

$$\int_{x_i}^{x_{i+1}} \left(a - (w+1)u + vu^2\right) \varphi \, \mathrm{d}x.$$

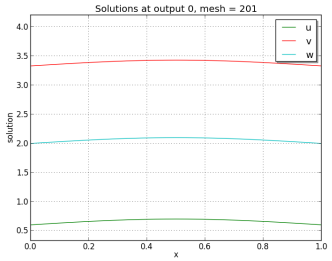
Since we employ piecewise linear basis and trial functions, the highest nonlinearity in the model is a quartic polynomial. We therefore approximate these integrals using a three-node Gaussian quadrature, exact for polynomials up to degree six.

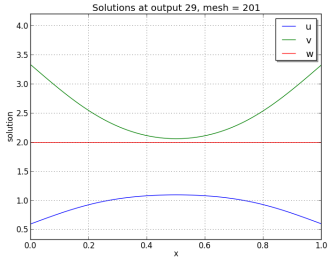
After this spatial semi-discretization, the system of three PDEs is passed to ARKode as a system of $3N$ coupled ODEs, as with the preceding problem.

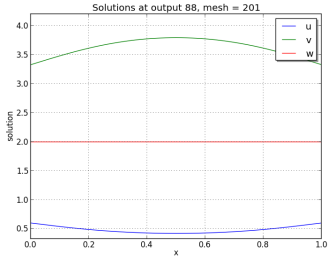
As with the preceding problem *ark_brusselator1D_klu*, this example solves the problem with a DIRK method, using a Newton iteration, and the SUNMATRIX_SPARSE module. However, this example uses the SUNLINSOL_SUPERLUMT linear solver module, both for the Newton systems having Jacobian $A = M - \gamma J$, as well as for the mass-matrix-only linear systems with matrix M. Functions implementing both J and M in compressed-sparse-column format are supplied.

100 outputs are printed at equal intervals, and run statistics are printed at the end.

1.9.2 Solutions







Finite-element Brusselator PDE solution snapshots (created using the supplied Python script, `plot_brusselator1D_FEM.py`): left is at time $t = 0$, center is at time $t = 2.9$, right is at time $t = 8.8$.

1.10 ark_heat1D

As with the previous brusselator problem, this example simulates a simple one-dimensional partial differential equation; in this case we consider the heat equation,

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + f,$$

for $t \in [0, 10]$, and $x \in [0, 1]$, with initial condition $u(0, x) = 0$, stationary boundary conditions,

$$\frac{\partial u}{\partial t}(t, 0) = \frac{\partial u}{\partial t}(t, 1) = 0,$$

and a point-source heating term,

$$f(t, x) = \begin{cases} 1 & \text{if } x = 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

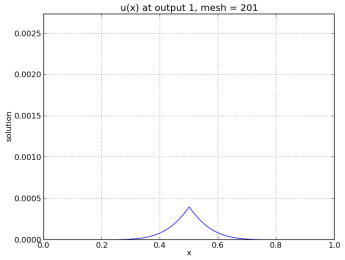
1.10.1 Numerical method

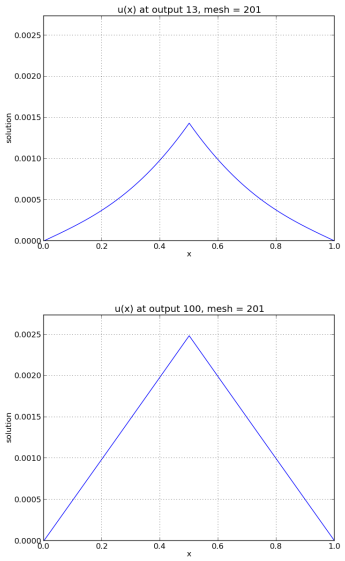
As with the *ark_brusselator1D* test problem, this test computes spatial derivatives using second-order centered differences, with the data distributed over N points on a uniform spatial grid.

In this example, we use $N = 201$ spatial points, with heat conductivity parameter $k = 0.5$, and discretize the equation using second-order centered finite-differences. The problem is run using scalar relative and absolute solver tolerances of $rtol = 10^{-6}$ and $atol = 10^{-10}$, respectively.

This program solves the problem with a DIRK method, utilizing a Newton iteration. The primary utility in including this example is that since the Newton linear systems are now symmetric, we solve these using the SUNLINSOL_PCG iterative linear solver, through the ARKSPILS linear solver interface. A routine to perform the Jacobian-vector product routine is supplied, in order to provide an example of its use.

1.10.2 Solutions





One-dimensional heat PDE solution snapshots: left is at time $t = 0.01$, center is at time $t = 0.13$, right is at time $t = 1.0$.

1.11 ark_heat1D_adapt

This problem is mathematically identical to the *ark_heat1D* test problem. However, instead of using a uniform spatial grid, this test problem utilizes a dynamically-evolving spatial mesh. The PDE under consideration is a simple one-dimensional heat equation,

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + f,$$

for $t \in [0, 10]$, and $x \in [0, 1]$, with initial condition $u(0, x) = 0$, stationary boundary conditions,

$$\frac{\partial u}{\partial t}(t, 0) = \frac{\partial u}{\partial t}(t, 1) = 0,$$

and a point-source heating term,

$$f(t, x) = \begin{cases} 1 & \text{if } x = 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

1.11.1 Numerical method

We again employ a method-of-lines discretization approach. The spatial derivatives are computed using a three-point centered stencil, that is accurate to $O(\Delta x_i^2)$ if the neighboring points are equidistant from the central point, i.e. $x_{i+1} - x_i = x_i - x_{i-1}$; however, if these neighbor distances are unequal the approximation reduces to first-order accuracy. The spatial mesh is initially distributed uniformly over 21 points in $[0, 1]$, but as the simulation proceeds the mesh is [crudely] adapted to add points to the center of subintervals bordering any node where $\left|\frac{\partial^2 u}{\partial x^2}\right| > 0.003$. We note that the spatial adaptivity approach employed in this example is *ad-hoc*, designed only to exemplify ARKode usage on a problem with varying size (not to show optimally-adaptive spatial refinement methods).

This program solves the problem with a DIRK method, utilizing a Newton iteration and the SUNLINSOL_PCG iterative linear solver. Additionally, the test problem utilizes ARKode's spatial adaptivity support (via `ARKodeResize`), allowing retention of the major ARKode data structures across vector length changes.

1.12 ark_KrylovDemo_prec

This problem is an ARKode clone of the CVODE problem, `cv_KrylovDemo_prec`. This is a demonstration program using the SUNLINSOL_SPGMR linear solver module. As explained more thoroughly in *[HSR2017]*, the problem is a stiff ODE system that arises from a system of PDEs modeling a six-species food web population model, with predator-prey interaction and diffusion on the unit square in two dimensions. We have a system with 6 components, $C = [c^1, c^2, \ldots, c^6]^T$ that satisfy the equations,

$$\frac{\partial c^i}{\partial t} = d_i \left(\frac{\partial^2 c^i}{\partial x^2} + \frac{\partial^2 c^i}{\partial y^2} \right) + f_i(x, y, c), \quad i = 1, \ldots, 6.$$

where

$$f_i(x, y, c) = c^i \left(b_i + \sum_{j=1}^{ns} a_{i,j} c^j \right).$$

Here, the first three species are prey and the last three are predators. The coefficients $a_{i,j}, b_i, d_i$ are:

$$a_{i,j} = \begin{cases} -1, & i = j, \\ -0.5 \times 10^{-6}, & i \le 3, j > 3, \\ 10^4, & i > 3, j \le 3 \end{cases} \quad b_i = \begin{cases} (1 + xy), & i \le 3, \\ -(1 + xy), & i > 3 \end{cases} \quad d_i = \begin{cases} 1, & i \le 3, \\ \frac{1}{2}, & i > 3 \end{cases}$$

The spatial domain is $(x, y) \in [0, 1]^2$; the time domain is $t \in [0, 10]$, with initial conditions

$$c^i(x, y) = 10 + i\sqrt{4x(1 - x)}\sqrt{4y(1 - y)}$$

and with homogeneous Neumann boundary conditions, $\nabla c^i \cdot \vec{n} = 0$.

1.12.1 Numerical method

We employ a method of lines approach, wherein we first semi-discretize in space to convert the system of 6 PDEs into a larger system of ODEs. To this end, the spatial derivatives are computed using second-order centered differences, with the data distributed over $Mx * My$ points on a uniform spatial grid. As a result, ARKode approaches the problem as one involving $6 * Mx * My$ coupled ODEs.

This program solves the problem with a DIRK method, using a Newton iteration with the preconditioned SUNLINSOL_SPGMR iterative linear solver module, and ARKSPILS interface. 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.
2. A block-diagonal matrix based on the partial derivatives of the interaction terms f only, using block-grouping (computing only a subset of the 3×3 blocks).

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 orthogonalization options are tested. In the series of runs, `ARKodeInit`, `SUNSPGMR`, `ARKSpilsSetLinearSolver`, `SUNSPGMRSetGSType`, `ARKSpilsSetEpsLin` and `ARKSpilsSetPreconditioner` are called only for the first run, whereas `ARKodeReInit`, `SUNSPGMRSetPrecType` and `SUNSPGMRSetGSType` are called to re-initialize the integrator and update linear solver parameters for each of the remaining three runs.

A problem description, performance statistics at selected output times, and final statistics are written to standard output. On the first run, solution values are also printed at output times. Error and warning messages are written to standard error, but there should be no such messages.

CHAPTER

TWO

OPENMP C EXAMPLE PROBLEMS

2.1 ark_brusselator1D_omp

This problem is mathematically identical to the one-dimensional reaction-diffusion brusselator model, *ark_brusselator1D*. As before, we investigate a time-dependent system of partial differential equations with 3 components, $Y = [u,\ v,\ w]^T$ that satisfy the equations,

$$\begin{aligned}
\frac{\partial u}{\partial t} &= d_u \frac{\partial^2 u}{\partial x^2} + a - (w+1)u + vu^2, \\
\frac{\partial v}{\partial t} &= d_v \frac{\partial^2 v}{\partial x^2} + wu - vu^2, \\
\frac{\partial w}{\partial t} &= d_w \frac{\partial^2 w}{\partial x^2} + \frac{b-w}{\varepsilon} - wu.
\end{aligned}$$

We integrate for $t \in [0, 10]$, and $x \in [0, 1]$, with initial conditions

$$\begin{aligned}
u(0,x) &= a + \frac{1}{10}\sin(\pi x), \\
v(0,x) &= \frac{b}{a} + \frac{1}{10}\sin(\pi x), \\
w(0,x) &= b + \frac{1}{10}\sin(\pi x),
\end{aligned}$$

and with stationary boundary conditions, i.e.

$$\begin{aligned}
\frac{\partial u}{\partial t}(t,0) &= \frac{\partial u}{\partial t}(t,1) = 0, \\
\frac{\partial v}{\partial t}(t,0) &= \frac{\partial v}{\partial t}(t,1) = 0, \\
\frac{\partial w}{\partial t}(t,0) &= \frac{\partial w}{\partial t}(t,1) = 0.
\end{aligned}$$

2.1.1 Numerical method

The numerical method is identical to the previous implementation, except that we now use SUNDIALS' OpenMP-enabled vector kernel module, NVECTOR_OPENMP, and have similarly threaded the supplied right-hand side and banded Jacobian construction functions.

CHAPTER

THREE

PARALLEL C EXAMPLE PROBLEMS

3.1 ark_diurnal_kry_bbd_p

This problem is an ARKode clone of the CVODE problem, `cv_diurnal_kry_bbd_p`. As described in *[HSR2017]*, this problem models a two-species diurnal kinetics advection-diffusion PDE system in two spatial dimensions,

$$\frac{\partial c_i}{\partial t} = K_h \frac{\partial^2 c_i}{\partial x^2} + V\frac{\partial c_i}{\partial x} + \frac{\partial}{\partial y}\left(K_v(y)\frac{\partial c_i}{\partial y}\right) + R_i(c_1, c_2, t), \quad i = 1, 2$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 * c_1 * c_3 - q_2 * c_1 * c_2 + 2 * q_3(t) * c_3 + q_4(t) * c_2, \\ R_2(c_1, c_2, t) &= q_1 * c_1 * c_3 - q_2 * c_1 * c_2 - q_4(t) * c_2, \\ K_v(y) &= K_{v0} e^{y/5}. \end{aligned}$$

Here K_h, V, K_{v0}, 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 spatial domain $(x, y) \in [0, 20] \times [30, 50]$, with homogeneous Neumann boundary conditions, and for time interval $t \in [0, 86400]$ sec (1 day).

We enforce the initial conditions

$$\begin{aligned} c_1(x, y) &= 10^6 \chi(x)\eta(y) \\ c_2(x, y) &= 10^{12} \chi(x)\eta(y) \\ \chi(x) &= 1 - \sqrt{\frac{x - 10}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}} \\ \eta(y) &= 1 - \sqrt{\frac{y - 40}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}}. \end{aligned}$$

3.1.1 Numerical method

We employ a method of lines approach, wherein we first semi-discretize in space to convert the system of 2 PDEs into a larger system of ODEs. To this end, the spatial derivatives are computed using second-order centered differences, with the data distributed over $Mx * My$ points on a uniform spatial grid. As a result, ARKode approaches the problem as one involving $2 * Mx * My$ coupled ODEs.

The problem is decomposed in parallel into uniformly-sized subdomains, with two subdomains in each direction (four in total), and where each subdomain has five points in each direction (i.e. $Mx = My = 10$).

This program solves the problem with a DIRK method, using a Newton iteration with the preconditioned SUNLINSOL_SPGMR iterative linear solver through the ARKSPILS interface.

The preconditioner matrix used is block-diagonal, with banded blocks, constructed using the ARKBBDPRE module. Each block is generated using difference quotients, with half-bandwidths `mudq = mldq = 10`, but the retained banded blocks have half-bandwidths `mukeep = mlkeep = 2`. A copy of the approximate Jacobian is saved and conditionally reused within the preconditioner routine.

Two runs are made for this problem, first with left and then with right preconditioning.

Performance data and sampled solution values are printed at selected output times, and all performance counters are printed on completion.

3.2 ark_diurnal_kry_p

This problem is an ARKode clone of the CVODE problem, `cv_diurnal_kry_p`. As described in *[HSR2017]*, this test problem models a two-species diurnal kinetics advection-diffusion PDE system in two spatial dimensions,

$$\frac{\partial c_i}{\partial t} = K_h \frac{\partial^2 c_i}{\partial x^2} + V\frac{\partial c_i}{\partial x} + \frac{\partial}{\partial y}\left(K_v(y)\frac{\partial c_i}{\partial y}\right) + R_i(c_1, c_2, t), \quad i = 1, 2$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 * c_1 * c_3 - q_2 * c_1 * c_2 + 2 * q_3(t) * c_3 + q_4(t) * c_2, \\ R_2(c_1, c_2, t) &= q_1 * c_1 * c_3 - q_2 * c_1 * c_2 - q_4(t) * c_2, \\ K_v(y) &= K_{v0} e^{y/5}. \end{aligned}$$

Here K_h, V, K_{v0}, 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 spatial domain $(x, y) \in [0, 20] \times [30, 50]$, with homogeneous Neumann boundary conditions, and for time interval $t \in [0, 86400]$ sec (1 day).

We enforce the initial conditions

$$\begin{aligned} c^1(x, y) &= 10^6 \chi(x)\eta(y) \\ c^2(x, y) &= 10^{12} \chi(x)\eta(y) \\ \chi(x) &= 1 - \sqrt{\frac{x - 10}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}} \\ \eta(y) &= 1 - \sqrt{\frac{y - 40}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}}. \end{aligned}$$

3.2.1 Numerical method

We employ a method of lines approach, wherein we first semi-discretize in space to convert the system of 2 PDEs into a larger system of ODEs. To this end, the spatial derivatives are computed using second-order centered differences, with the data distributed over $Mx * My$ points on a uniform spatial grid. As a result, ARKode approaches the problem as one involving $2 * Mx * My$ coupled ODEs.

The problem is decomposed in parallel into uniformly-sized subdomains, with two subdomains in each direction (four in total), and where each subdomain has five points in each direction (i.e. $Mx = My = 10$).

This program solves the problem with a DIRK method, using a Newton iteration with the preconditioned SUNLINSOL_SPGMR iterative linear solver, through the ARKSPILS interface.

The preconditioner matrix used is block-diagonal, with block-diagonal portion of the Newton matrix used as a left preconditioner. A copy of the block-diagonal portion of the Jacobian is saved and conditionally reused within the preconditioner routine.

Performance data and sampled solution values are printed at selected output times, and all performance counters are printed on completion.

CHAPTER

FOUR

PARALLEL HYPRE EXAMPLE PROBLEMS

4.1 ark_diurnal_kry_ph

This problem is mathematically identical to the parallel C example problem *ark_diurnal_kry_p*. As before, this test problem models a two-species diurnal kinetics advection-diffusion PDE system in two spatial dimensions,

$$\frac{\partial c_i}{\partial t} = K_h \frac{\partial^2 c_i}{\partial x^2} + V\frac{\partial c_i}{\partial x} + \frac{\partial}{\partial y}\left(K_v(y)\frac{\partial c_i}{\partial y}\right) + R_i(c_1, c_2, t), \quad i = 1, 2$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 * c_1 * c_3 - q_2 * c_1 * c_2 + 2 * q_3(t) * c_3 + q_4(t) * c_2, \\ R_2(c_1, c_2, t) &= q_1 * c_1 * c_3 - q_2 * c_1 * c_2 - q_4(t) * c_2, \\ K_v(y) &= K_{v0} e^{y/5}. \end{aligned}$$

Here K_h, V, K_{v0}, 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 spatial domain $(x, y) \in [0, 20] \times [30, 50]$, with homogeneous Neumann boundary conditions, and for time interval $t \in [0, 86400]$ sec (1 day).

We enforce the initial conditions

$$\begin{aligned} c^1(x, y) &= 10^6 \chi(x)\eta(y) \\ c^2(x, y) &= 10^{12} \chi(x)\eta(y) \\ \chi(x) &= 1 - \sqrt{\frac{x - 10}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}} \\ \eta(y) &= 1 - \sqrt{\frac{y - 40}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}}. \end{aligned}$$

4.1.1 Numerical method

The numerical method is identical to the previous implementation, except that we now use the HYPRE parallel vector module, NVECTOR_PARHYP. The output of these two examples is identical. Below, we discuss only the main differences between the two implementations; familiarity with the HYPRE library is helpful.

We use the HYPRE IJ vector interface to allocate the template vector and create the parallel partitioning:

```
HYPRE_IJVectorCreate(comm, my_pe*local_N, (my_pe + 1)*local_N - 1, &Uij);
HYPRE_IJVectorSetObjectType(Uij, HYPRE_PARCSR);
HYPRE_IJVectorInitialize(Uij);
```

The *initialize* call means that vector elements are ready to be set using the IJ interface. We choose the initial condition vector $x_0 = x(t_0)$ as the template vector, and we set its values in the `SetInitialProfiles(...)` function. We complete assembly of the HYPRE vector with the calls:

```
HYPRE_IJVectorAssemble(Uij);
HYPRE_IJVectorGetObject(Uij, (void**) &Upar);
```

The *assemble* call is collective and makes the HYPRE vector ready to use. The sets the handle `Upar` to the actual HYPRE vector. The handle is then passed to the `N_VMake` function, which creates the template `N_Vector`, `u`, as a wrapper around the HYPRE vector. All of the other vectors used in the computation are created by cloning this template vector.

Furthermore, since the template vector does not own the underlying HYPRE vector (it was created using the `HYPRE_IJVectorCreate` call above), so it is the user's responsibility to destroy it by calling `HYPRE_IJVectorDestroy(Uij)` after the template vector `u` has been destroyed. This function will destroy both the HYPRE vector and its IJ interface.

To access individual elements of the solution and derivative vectors `u` and `udot` in the IVP right-hand side function, `f`, the user needs to first extract the HYPRE vector by calling `N_VGetVector_ParHyp`, and then use HYPRE-specific methods to access the data from that point on.

Note: Currently, interfaces to HYPRE solvers and preconditioners are not available directly through the SUNDIALS interfaces, however these could be utilized directly as preconditioners for SUNDIALS' Krylov solvers. Direct interfaces to the HYPRE solvers will be provided in subsequent SUNDIALS releases. The current HYPRE vector interface is included in this release mainly for testing purposes and as a preview of functionality to come.

CHAPTER

FIVE

SERIAL C++ EXAMPLE PROBLEMS

5.1 ark_analytic_sys

This example demonstrates the use of ARKode's fully implicit solver on a stiff ODE system that has a simple analytical solution. The problem is that of a linear ODE system,

$$\frac{dy}{dt} = Ay$$

where $A = VDV^{-1}$. In this example, we use

$$V = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 2 & 1 \\ 0 & -1 & 2 \end{bmatrix}, \qquad V^{-1} = \frac{1}{4}\begin{bmatrix} 5 & 1 & -3 \\ 2 & 2 & -2 \\ 1 & 1 & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} -1/2 & 0 & 0 \\ 0 & -1/10 & 0 \\ 0 & 0 & \lambda \end{bmatrix}.$$

where λ is a large negative number. The analytical solution to this problem may be computed using the matrix exponential,

$$Y(t) = Ve^{Dt}V^{-1}Y(0).$$

We evolve the problem for t in the interval $\left[0,\ \frac{1}{20}\right]$, with initial condition $Y(0) = [1,\ 1,\ 1]^T$.

5.1.1 Numerical method

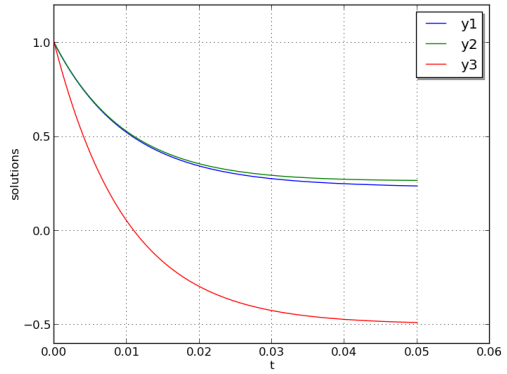
The stiffness of the problem is directly proportional to the value of λ. The value of λ should be negative to result in a well-posed ODE; for values with magnitude larger than 100 the problem becomes quite stiff.

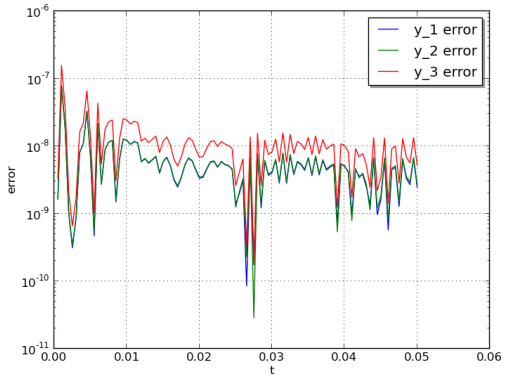
Here, we choose $\lambda = -100$, along with scalar relative and absolute tolerances of $rtol = 10^{-6}$ and $atol = 10^{-10}$, respectively.

This program solves the problem with the DIRK method, Newton iteration with the SUNMATRIX_DENSE matrix module and accompanying SUNLINSOL_DENSE linear solver module, ARKDLS direct linear solver interface, and a user-supplied dense Jacobian routine. Output is printed every 0.005 units of time (10 total). Run statistics (optional outputs) are printed at the end.

5.1.2 Solutions

This problem is included both as a simple example to test systems of ODE within ARKode on a problem having an analytical solution, $Y(t) = Ve^{Dt}V^{-1}Y(0)$. As seen in the plots below, the computed solution tracks the analytical solution quite well (left), and results in errors with exactly the magnitude as specified by the requested error tolerances (right).


y1
y2
y3
solutions
t
1.0
0.5
0.0
−0.5
0.00
0.01
0.02
0.03
0.04
0.05
0.06


y_1 error
y_2 error
y_3 error
error
t
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
0.00
0.01
0.02
0.03
0.04
0.05
0.06

CHAPTER

SIX

PARALLEL C++ EXAMPLE PROBLEMS

6.1 ark_heat2D

ARKode provides one parallel C++ example problem, that extends our previous *ark_heat1D* test to now simulate a two-dimensional heat equation,

$$\frac{\partial u}{\partial t} = k_x \frac{\partial^2 u}{\partial x^2} + k_y \frac{\partial^2 u}{\partial y^2} + h,$$

for $t \in [0, 0.3]$, and $(x, y) \in [0, 1]^2$, with initial condition $u(0, x, y) = 0$, stationary boundary conditions,

$$\frac{\partial u}{\partial t}(t, 0, y) = \frac{\partial u}{\partial t}(t, 1, y) = \frac{\partial u}{\partial t}(t, x, 0) = \frac{\partial u}{\partial t}(t, x, 1) = 0,$$

and a periodic heat source,

$$h(x, y) = \sin(\pi x) \sin(2\pi y).$$

Under these conditions, the problem has an analytical solution of the form

$$u(t, x, y) = \frac{1 - e^{-(k_x + 4k_y)\pi^2 t}}{(k_x + 4k_y)\pi^2} \sin(\pi x) sin(2\pi y).$$

6.1.1 Numerical method

The spatial derivatives are computed using second-order centered differences, with the data distributed over $nx \times ny$ points on a uniform spatial grid.

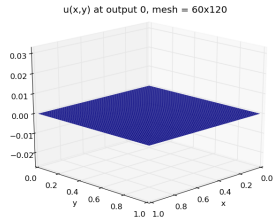
The problem is set up to use spatial grid parameters $nx = 60$ and $ny = 120$, with heat conductivity parameters $k_x = 0.5$ and $k_y = 0.75$. The problem is run using scalar relative and absolute solver tolerances of $rtol = 10^{-5}$ and $atol = 10^{-10}$.

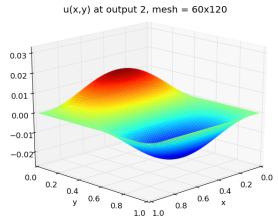
As with the 1D version, this program solves the problem with a DIRK method, that itself uses a Newton iteration and SUNLINSOL_PCG iterative linear solver through the ARKSPILS interface. However, unlike the previous example, here the PCG solver is preconditioned using a single Jacobi iteration, and uses ARKSPILS' built-in finite-difference Jacobian-vector product routine. Additionally, this problem uses MPI and the NVECTOR_PARALLEL module for parallelization.

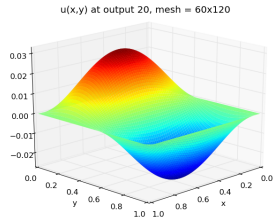
6.1.2 Solutions

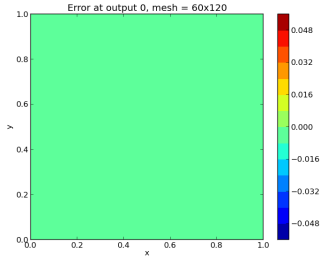
Top row: 2D heat PDE solution snapshots, the left is at time $t = 0$, center is at time $t = 0.03$, right is at time $t = 0.3$. Bottom row, absolute error in these solutions. Note that the relative error in these results is on the order 10^{-4},

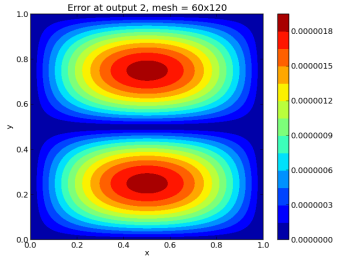
corresponding to the spatial accuracy of the relatively coarse finite-difference mesh. All plots are created using the supplied Python script, `plot_heat2D.py`.

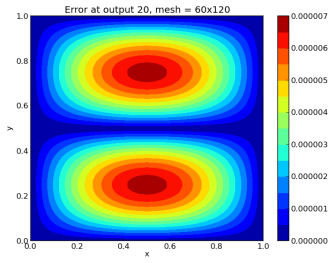











Error at output 20, mesh = 60x120
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0.000000

CHAPTER

SEVEN

SERIAL FORTRAN 77 EXAMPLE PROBLEMS

7.1 fark_diurnal_kry_bp

This problem is an ARKode clone of the CVODE problem, `fcv_diurnal_kry_bp`. As described in *[HSR2017]*, this problem models a two-species diurnal kinetics advection-diffusion PDE system in two spatial dimensions,

$$\frac{\partial c_i}{\partial t} = K_h \frac{\partial^2 c_i}{\partial x^2} + V\frac{\partial c_i}{\partial x} + \frac{\partial}{\partial y}\left(K_v(y)\frac{\partial c_i}{\partial y}\right) + R_i(c_1, c_2, t), \quad i = 1, 2$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 * c_1 * c_3 - q_2 * c_1 * c_2 + 2 * q_3(t) * c_3 + q_4(t) * c_2, \\ R_2(c_1, c_2, t) &= q_1 * c_1 * c_3 - q_2 * c_1 * c_2 - q_4(t) * c_2, \\ K_v(y) &= K_{v0} e^{y/5}. \end{aligned}$$

Here K_h, V, K_{v0}, 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 spatial domain $(x, y) \in [0, 20] \times [30, 50]$, with homogeneous Neumann boundary conditions, and for time interval $t \in [0, 86400]$ sec (1 day).

We enforce the initial conditions

$$\begin{aligned} c^1(x, y) &= 10^6 \chi(x)\eta(y) \\ c^2(x, y) &= 10^{12} \chi(x)\eta(y) \\ \chi(x) &= 1 - \sqrt{\frac{x - 10}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}} \\ \eta(y) &= 1 - \sqrt{\frac{y - 40}{10}} + \frac{1}{2}\sqrt[4]{\frac{x - 10}{10}}. \end{aligned}$$

7.1.1 Numerical method

We employ a method of lines approach, wherein we first semi-discretize in space to convert the system of 2 PDEs into a larger system of ODEs. To this end, the spatial derivatives are computed using second-order centered differences, with the data distributed over $Mx * My$ points on a uniform spatial grid. As a result, ARKode approaches the problem as one involving $2 * Mx * My$ coupled ODEs. In this problem, we use a relatively coarse uniform mesh with $Mx = My = 10$.

This program solves the problem with a DIRK method, using a Newton iteration with the preconditioned SUNLINSOL_SPGMR iterative linear solver module, and the ARKSPILS interface.

The left preconditioner used is a banded matrix, constructed using the ARKBP module. The banded preconditioner matrix is generated using difference quotients, with half-bandwidths `mu = ml = 2`.

Performance data and sampled solution values are printed at selected output times, and all performance counters are printed on completion.

7.2 fark_roberts_dnsL

This problem is an ARKode clone of the CVODE problem, `fcv_roberts_dnsL`. As described in *[HSR2017]*, this problem models the kinetics of a three-species autocatalytic reaction. This is an ODE system with 3 components, $Y = [y_1,\ y_2,\ y_3]^T$, satisfying the equations,

$$\begin{aligned}
\frac{dy_1}{dt} &= -0.04y_1 + 10^4 y_2 y_3, \\
\frac{dy_2}{dt} &= 0.04y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2, \\
\frac{dy_3}{dt} &= 3 \cdot 10^7 y_2^2.
\end{aligned}$$

We integrate over the interval $0 \le t \le 4 \cdot 10^{10}$, with initial conditions $Y(0) = [1,\ 0,\ 0]^T$.

Additionally, we supply the following two root-finding equations:

$$\begin{aligned}
g_1(u) &= u - 10^{-4}, \\
g_2(w) &= w - 10^{-2}.
\end{aligned}$$

While these are not inherently difficult nonlinear equations, they easily serve the purpose of determining the times at which our solutions attain desired target values.

7.2.1 Numerical method

This program solves the problem with a DIRK method, using a Newton iteration with the SUNLINSOL_LAPACKDENSE linear solver module and ARKDLS interface.

As with the *ark_robertson_root* problem, we enable ARKode's rootfinding module to find the times at which either $u = 10^{-4}$ or $w = 10^{-2}$.

Performance data and solution values are printed at selected output times, along with additional output at rootfinding events. All performance counters are printed on completion.

CHAPTER

EIGHT

PARALLEL FORTRAN 77 EXAMPLE PROBLEMS

8.1 fark_diag_kry_bbd_p

This problem is an ARKode clone of the CVODE problem, `fcv_diag_kry_bbd_p`. As described in *[HSR2017]*, this problem models a stiff, linear, diagonal ODE system,

$$\frac{\partial y_i}{\partial t} = -\alpha i y_i, \quad i = 1, \dots N.$$

Here $\alpha = 10$ and $N = 10N_P$, where N_P is the number of MPI tasks used for the problem. The problem has initial conditions $y_i = 1$ and evolves for the time interval $t \in [0, 1]$.

8.1.1 Numerical method

This program solves the problem with a DIRK method, using a Newton iteration with the preconditioned SUNLINSOL_SPGMR iterative linear solver module and ARKSPILS interface.

A diagonal preconditioner matrix is used, formed automatically through difference quotients within the ARKBBDPRE module. Since ARKBBDPRE is developed for use of a block-banded preconditioner, in this solver each block is set to have half-bandwidths `mudq = mldq = 0` to retain only the diagonal portion.

Two runs are made for this problem, first with left and then with right preconditioning (`IPRE` is first set to 1 and then to 2).

Performance data is printed at selected output times, and maximum errors and final performance counters are printed on completion.

8.2 fark_diag_non_p

This problem is an ARKode clone of the CVODE problem, `fcv_diag_non_p`. As described in *[HSR2017]*, this problem models a nonstiff, linear, diagonal ODE system,

$$\frac{\partial y_i}{\partial t} = -\alpha i y_i, \quad i = 1, \dots N.$$

Here $\alpha = \frac{10}{N}$ and $N = 10N_P$, where N_P is the number of MPI tasks used for the problem. The problem has initial conditions $y_i = 1$ and evolves for the time interval $t \in [0, 1]$.

8.2.1 Numerical method

This program solves the problem with an ERK method, and hence does not require either a nonlinear or linear solver for integration.

Performance data is printed at selected output times, and maximum errors and final performance counters are printed on completion.

CHAPTER

NINE

SERIAL FORTRAN 90 EXAMPLE PROBLEMS

9.1 ark_bruss

This test problem is a Fortran-90 version of the same brusselator problem as before, *ark_brusselator*, in which the "test 1" parameters are hard-coded into the solver. As with the previous test, this problem has 3 dependent variables u, v and w, that depend on the independent variable t via the IVP system

$$\begin{aligned}
\frac{du}{dt} &= a - (w+1)u + vu^2, \\
\frac{dv}{dt} &= wu - vu^2, \\
\frac{dw}{dt} &= \frac{b-w}{\varepsilon} - wu.
\end{aligned}$$

We integrate over the interval $0 \le t \le 10$, with the initial conditions $u(0) = 3.9$, $v(0) = 1.1$, $w(0) = 2.8$, and parameters $a = 1.2$, $b = 2.5$ and $\varepsilon = 10^{-5}$. After each unit time interval, the solution is output to the screen.

9.1.1 Numerical method

Since this driver and utility functions are written in Fortran-90, this example demonstrates the use of the FARKODE interface for the ARKode solver. For time integration, this example uses the fourth-order additive Runge-Kutta IMEX method, where the right-hand sides are broken up as

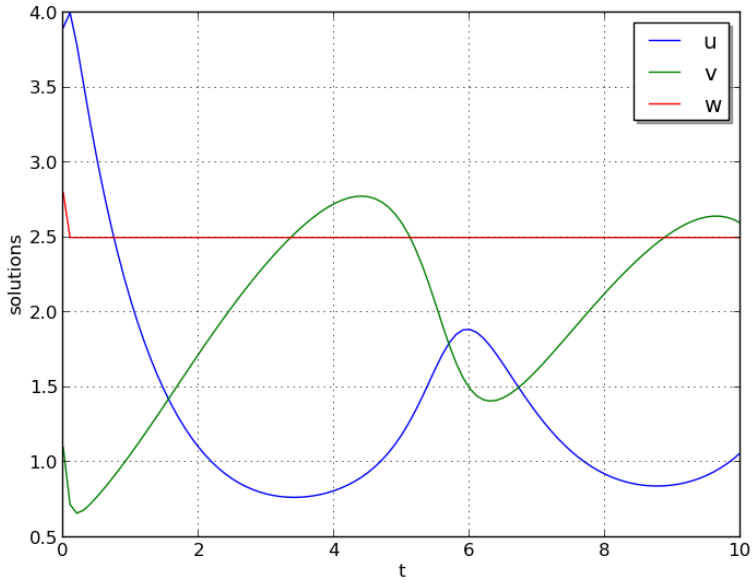
$$f_E(t,u,v,w) = \begin{pmatrix} a - (w+1)u + vu^2 \\ wu - vu^2 \\ -wu \end{pmatrix}, \quad \text{and} \quad f_I(t,u,v,w) = \begin{pmatrix} 0 \\ 0 \\ \frac{b-w}{\varepsilon} \end{pmatrix}.$$

The implicit systems are solved using the built-in modified Newton iteration, with the SUNMATRIX_DENSE matrix module and accompanying SUNLINSOL_DENSE linear solver module, through the ARKDLS interface. Both the Jacobian routine and right-hand side functions are supplied by functions provided in the example file.

The only non-default solver options are the tolerances $atol = 10^{-10}$ and $rtol = 10^{-6}$, adaptivity method 2 (I controller), a maximum of 8 Newton iterations per step, a nonlinear solver convergence coefficient $nlscoef = 10^{-8}$, and a maximum of 1000 internal time steps.

9.1.2 Solutions

With this setup, all three solution components exhibit a rapid transient change during the first 0.2 time units, followed by a slow and smooth evolution, as seen in the figure below. Note that these results identically match those from the previous C example with the same equations.


u
v
w
solutions
t
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
10

CHAPTER

TEN

PARALLEL FORTRAN 90 EXAMPLE PROBLEMS

10.1 fark_heat2D

This test problem is a Fortran-90 version of the same two-dimensional heat equation problem as in C++, *ark_heat2D*. This models a simple two-dimenaional heat equation,

$$\frac{\partial u}{\partial t} = k_x \frac{\partial^2 u}{\partial x^2} + k_y \frac{\partial^2 u}{\partial y^2} + h,$$

for $t \in [0, 0.3]$, and $(x, y) \in [0, 1]^2$, with initial condition $u(0, x, y) = 0$, stationary boundary conditions,

$$\frac{\partial u}{\partial t}(t, 0, y) = \frac{\partial u}{\partial t}(t, 1, y) = \frac{\partial u}{\partial t}(t, x, 0) = \frac{\partial u}{\partial t}(t, x, 1) = 0,$$

and a periodic heat source,

$$h(x, y) = \sin(\pi x) \sin(2\pi y).$$

Under these conditions, the problem has an analytical solution of the form

$$u(t, x, y) = \frac{1 - e^{-(k_x + 4k_y)\pi^2 t}}{(k_x + 4k_y)\pi^2} \sin(\pi x) sin(2\pi y).$$

10.1.1 Numerical method

The spatial derivatives are computed using second-order centered differences, with the data distributed over $nx \times ny$ points on a uniform spatial grid.

The spatial grid is set to $nx = 60$ and $ny = 120$. The heat conductivity parameters are $k_x = 0.5$ and $k_y = 0.75$.

As with the C++ version, this program solves the problem with a DIRK method, that itself uses a Newton iteration and SUNLINSOL_PCG iterative linear solver module through the ARKSPILS interface. Also like the C++ version, the PCG solver is preconditioned using a single Jacobi iteration, and uses ARKSPILS' finite-difference Jacobian-vector product approximation routine for the PCG polver. Additionally, this problem uses MPI and the Fortran interface to the NVECTOR_PARALLEL module for parallelization.

[HSR2017] A.C. Hindmarsh, R. Serban and D.R. Reynolds. Example Programs for CVODE v3.1.0. Technical Report UCRL-SM-208110, LLNL, 2017.

[R2017] D.R. Reynolds. User Documentation for ARKode v2.1.1. Technical Report LLNL-CODE-667205, LLNL, 2017.