The ARKode infrastructure for adaptive one-step methods

Daniel R. Reynolds[1], David J. Gardner[2] and Carol S. Woodward[2]

reynolds@smu.edu, gardner48@llnl.gov, cswoodward@llnl.gov

[1]Department of Mathematics, Southern Methodist University
[2]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

Integrating the Integrators for Nonlinear Evolution Equations
Banff International Research Station
December 2-7, 2018

Outline

Outline

1. Motivation
2. ARKode Background
3. Revised ARKode Infrastructure
4. Timestepper Modules
5. Conclusions, Etc.

Multiphysics Problems

"Multiphysics" problems typically involve a variety of interacting processes:

- System of components coupled in the bulk [cosmology, combustion]
- System of components coupled across interfaces [climate, tokamak fusion]

Multiphysics simulation challenges include:

- Multirate processes, but too close to analytically reformulate.
- Optimal solvers may exist for some pieces, but not for the whole.
- Mixing of stiff/nonstiff processes, a challenge for standard algorithms.

Historical approaches rely on lowest-order time step splittings, may suffer from:

- Low accuracy – typically $\mathcal{O}(h)$-accurate; symmetrization/extrapolation may improve this but at significant cost [Ropp, Shadid & Ober 2005].
- Poor/unknown stability – even when each part utilizes a 'stable' step size, the combined problem may admit unstable modes [Estep et al., 2007].

Need for Flexible Time Integration Libraries

Multiphysics time integration needs:

- Stability/accuracy for each component, as well as inter-physics couplings
- Custom/flexible step sizes for distinct components
- Robust temporal error estimation & adaptivity of step size(s)
- Built-in support for spatial adaptivity
- Ability to apply optimal solver algorithms for individual components
- Support for testing a variety of methods and solution algorithms

Legacy software frameworks enforce overly-rigid standards on applications:

- Fully implicit or fully explicit, without ImEx flexibility.
- Fixed data structures for vectors, matrices, (non)linear solvers.
- Hard-coded parameters – good for most problems, but rarely optimal.

Outline

Additive Runge–Kutta (ARK) Methods [Ascher et al. 1997; Araújo et al. 1997; ...]

ARKode was initially designed to implement adaptive ARK methods, supporting up to two split components: *explicit* and *implicit*,

$$M\dot{y} = f^E(t,y) + f^I(t,y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0,$$

- M is any nonsingular linear operator (mass matrix, typically $M = I$),
- $f^E(t,y)$ contains the explicit terms,
- $f^I(t,y)$ contains the implicit terms.

Combine two s-stage RK methods; denoting $t^*_{n,j} = t_n + c^*_j h_n$, $h_n = t_{n+1} - t_n$:

$$M z_i = M y_n + h_n \sum_{j=1}^{i-1} A^E_{i,j} f^E(t^E_{n,j}, z_j) + h_n \sum_{j=1}^{i} A^I_{i,j} f^I(t^I_{n,j}, z_j), \quad i = 1, \ldots, s,$$

$$M y_{n+1} = M y_n + h_n \sum_{j=1}^{s} \left[b^E_j f^E(t^E_{n,j}, z_j) + b^I_j f^I(t^I_{n,j}, z_j) \right] \quad \text{(solution)}$$

$$M \tilde{y}_{n+1} = M y_n + h_n \sum_{j=1}^{s} \left[\tilde{b}^E_j f^E(t^E_{n,j}, z_j) + \tilde{b}^I_j f^I(t^I_{n,j}, z_j) \right] \quad \text{(embedding)}$$

ARK Coefficients

Two Butcher tables define the method:

- $\left\{c^E, A^E, b^E, \tilde{b}^E\right\}$ define the *explicit Butcher table*
- $\left\{c^I, A^I, b^I, \tilde{b}^I\right\}$ define the *diagonally-implicit Butcher table*

Formulation supports adaptive or fixed-step ERK, DIRK and ARK methods:

- Explicit methods: $A^I = 0$ and all IVP terms are in $f^E(t, y)$.
- Implicit methods: $A^E = 0$ and all IVP terms are in $f^I(t, y)$.
- Tables derived in unison to satisfy inter-component coupling.
- For fixed or user-defined steps h_n: $\tilde{b}^E$ and $\tilde{b}^I$ need not be defined.

Solving each stage z_i, $i = 1, \ldots, s$

Each stage is implicitly defined via a root-finding problem:

$$0 = G_i(z) \equiv Mz - My_n - h_n \left[A^I_{i,i} f^I(t^I_{n,i}, z) + \sum_{j=1}^{i-1} \left(A^E_{i,j} f^E(t^E_{n,j}, z_j) + A^I_{i,j} f^I(t^I_{n,j}, z_j) \right) \right]$$

- if $f^I(t, y)$ is *linear* in y then we must solve a linear system for each z_i,
- otherwise G_i is nonlinear, and requires an iterative nonlinear solver.

Algebraic solver options (see Carol Woodward's talk):

- Nonlinear solver options include: modified Newton, inexact Newton, Anderson-accelerated fixed point, and user-supplied.
- Linear solver options include: dense/band/sparse-direct, preconditioned Krylov iterative, and user-supplied.
- All solvers (except for direct linear) formulated via vector operations, with data structures including: serial, MPI, PETSc, *hypre*, and user-supplied.

ARKode Flexibility Enhancements

Additionally, ARKode includes enhancements for multi-physics codes, including:

- Variety of built-in RK tables; supports user-supplied
- Variety of built-in adaptivity functions; supports user-supplied
- Variety of built-in implicit predictor algorithms
- Ability to specify that problem is linearly implicit
- Ability to resize data structures based on changing IVP size
- All internal solver parameters are user-modifiable

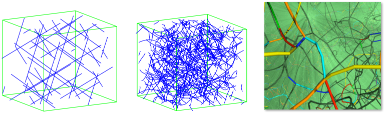
SMU FASTMATH

ARKode Usage

ARKode has been freely-available since 2014. We have specifically worked with applications groups in:

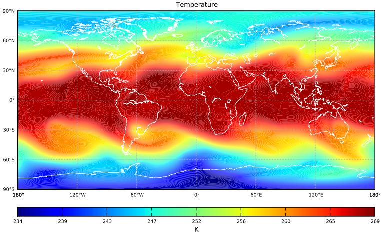
ParaDiS – large-scale simulations of dislocation growth/propagation (material strain hardening) [Gardner et al., *MSMSE*, 2015]

- Examined high-order adaptive DIRK methods.
- Examined nonlinear solvers and options.



Tempest & *HOMME-NH* – non-hydrostatic 3D dynamical cores for atmospheric simulations [Gardner et al., *GMD*, 2018; Vogl et al, *in prep.*]

- Examined ImEx splittings & fixed-step ARK methods for accuracy/stability
- Examined nonlinear/linear solver algorithms for implicit components



Outline

Reconfiguring ARKode into an infrastructure

Over the last year, we have overhauled ARKode to serve as an infrastructure for general, adaptive, one-step time integration methods:

- ARKode provides the outer time integration loop.
- Time-stepping modules handle problem-specific components: how a user defines the IVP itself, and how to take a single time step.
- Time-stepping modules may leverage shared ARKode infrastructure:
  - SUNDIALS' vector, matrix, linear solver and nonlinear solver objects,
  - translation between Jacobians and mass-matrices at the IVP level, to those required within an implicit stage solve,
  - usage modes (interpolation vs "tstop") and dense output interpolation,
  - time adaptivity controllers,
  - implicit predictors,
  - temporal root-finding,
  - etc.

Timestep module requirements

Each timestep module must provide functions to:

- initialize the module once all user-specified options have been set,
- evaluate the full ODE right-hand side function (if partitioned), and
- actually perform a single time step of the method.

To leverage shared algebraic solver infrastructure, it must provide functions to:

- attach implicit linear solver routines and data structures to the module,
- return the linear solver memory structure,
- return function pointer $f_i(t, y)$ corresp. to current implicit stage solve, and
- return γ for the linear system, $\mathcal{A}x = b$ where $\mathcal{A} = M - \gamma \frac{\partial f}{\partial y}(t, y)$.

Timestep modules are also responsible for providing a user interface:

- Define types of supported problems, and allow users to supply problem-defining vectors, functions, parameters, etc.
- Interally create the shared ARKode infrastructure.

Shared infrastructure: ARKLS linear solver interface

Translation layer between SUNDIALS' generic matrix/solver structures ($\mathcal{A}x = b$) and IVP-specific linear systems ($\mathcal{A} \approx M - \gamma \frac{\partial f}{\partial y}(t, y)$).

Provides five routines for timestep modules to use:

- *Initialize*: completes solver initialization based on optional inputs.
- *Setup*: recomputes $\frac{\partial f}{\partial y}(t, y)$ as necessary, including finite-difference approximations for dense/banded matrices.
- *Multiply*: computes the matrix-vector products $\mathcal{A}v$ and $\frac{\partial f}{\partial y}v$.
- *Solve*: solves $\mathcal{A}x = b$ to a desired accuracy (WRMS norm).
- *Free*: frees up memory allocated by the linear solver.

Similar routines are supported for non-identity mass-matrices.

Shared infrastructure: Stepsize adaptivity controllers

A variety of built-in stepsize adaptivity controllers are provided.
Defining q and p as the method and embedding orders, and $\varepsilon_k \approx \|y_k - \tilde{y}_k\|$:

- *PID*: $h_{n+1} = h_n \varepsilon_n^{-k_1/p} \varepsilon_{n-1}^{k_2/p} \varepsilon_{n-2}^{-k_3/p}$
- *PI*: $h_{n+1} = h_n \varepsilon_n^{-k_1/p} \varepsilon_{n-1}^{k_2/p}$
- *I*: $h_{n+1} = h_n \varepsilon_n^{-k_1/p}$
- *explicit Gustafsson*: $h_{n+1} = \begin{cases} h_1 \varepsilon_1^{-1/p} \\ h_n \varepsilon_n^{-k_1/p} \left(\frac{\varepsilon_n}{\varepsilon_{n-1}}\right)^{k_2/p} \end{cases}$
- *implicit Gustafsson*: $h_{n+1} = \begin{cases} h_1 \varepsilon_1^{-1/p} \\ h_n \left(\frac{h_n}{h_{n-1}}\right)^{k_2/p} \varepsilon_n^{-k_1/p} \left(\frac{\varepsilon_n}{\varepsilon_{n01}}\right)^{k_2/p} \end{cases}$
- *ImEx Gustafsson*: h_{n+1} is set to the minimum of the two previous estimates
- *user-supplied*: $h_{n+1} = H\left(y, t, h_n, h_{n-1}, h_{n-2}, \varepsilon_n, \varepsilon_{n-1}, \varepsilon_{n-2}, q, p\right)$

Shared infrastructure: Temporal interpolation module

ARKode provides an integrator-agnostic dense output module based on Hermite polynomial interpolation.

Defining $[t_{n-1}, t_n]$ as the most-recently-computed solution interval, $h_n = t_n - t_{n-1}$ and $\tau = \frac{t-t_n}{h_n}$:

- $\mathcal{O}(h_n)$: $p_0(\tau) = \frac{1}{2}\left(y_{n-1} + y_n\right)$
- $\mathcal{O}(h_n^2)$: $p_1(\tau)$ interpolates $\left\{\, y_{n-1}, y_n \,\right\}$
- $\mathcal{O}(h_n^3)$: $p_2(\tau)$ interpolates $\left\{\, y_{n-1}, y_n, \dot{y}_n \,\right\}$
- $\mathcal{O}(h_n^4)$: $p_3(\tau)$ interpolates $\left\{\, y_{n-1}, y_n, \dot{y}_{n-1}, \dot{y}_n \,\right\}$
- $\mathcal{O}(h_n^5)$: $p_4(\tau)$ interpolates $\left\{\, y_{n-1}, y_n, \dot{y}_{n-1}, \dot{y}_n, \dot{y}\left(t_n - \frac{1}{3}h_n\right) \,\right\}$
- $\mathcal{O}(h_n^6)$: $p_5(\tau)$ interpolates $\left\{\, y_{n-1}, y_n, \dot{y}_{n-1}, \dot{y}_n, \dot{y}\left(t_n - \frac{1}{3}h_n\right), \dot{y}\left(t_n - \frac{2}{3}h_n\right) \,\right\}$

Shared infrastructure: Implicit predictor module

ARKode provides implicit predictors for stages in the step $t_n \to t_{n+1}$, mainly utilizing the interpolation module, $z_i^{(0)} = p\left(c_i \frac{h_{n+1}}{h_n}\right)$:

- *Trivial*: $p(\tau) = y_n$.
- *Maximum order*: $p(\tau) = p_{q_{max}}(\tau)$ for user-specified $q_{max} \in [0, 5]$.
- *Variable order*: $p(\tau) = p_q(\tau)$ where $q = \begin{cases} 3, & \text{if } \tau \le \frac{1}{2}, \\ 2, & \text{if } \frac{1}{2} < \tau \le \frac{3}{4}, \\ 1, & \text{otherwise.} \end{cases}$
- *Cutoff order*: $p(\tau) = p_q(\tau)$ where $q = \begin{cases} q_{max}, & \text{if } \tau \le \frac{1}{2}, \\ 1, & \text{otherwise.} \end{cases}$
- *Bootstrap*: $p(\tau) = y_n$ for $i = 1$; for $i > 1$, $p(\tau)$ interpolates $\{\, y_n, \dot{y}_n, \dot{y}(t_{n,j}) \,\}$ where $t_{n,j} = \max_{j<i}(t_n + c_j h_{n+1})$.

SMU FASTMATH

Shared infrastructure: Butcher table module

ARKode provides a Butcher table data structure, with required fields for $A \in \mathbb{R}^{s\times s}, b \in \mathbb{R}^s, c \in \mathbb{R}^s$ and q; optional fields are $\tilde{b} \in \mathbb{R}^s$ and p.

Users may supply table structures to timestep modules. Alternately, ARKode already includes a variety of tables:

- *Explicit*: 12 embedded tables from $\mathcal{O}(h^2/h)$ through $\mathcal{O}(h^8/h^7)$.
- *Diagonally-implicit*: 12 embedded tables from $\mathcal{O}(h^2/h)$ to $\mathcal{O}(h^5/h^4)$.
- *ImEx*: 3 ARK pairs with orders $\mathcal{O}(h^3/h^2)$, $\mathcal{O}(h^4/h^3)$, and $\mathcal{O}(h^5/h^4)$.

Outline

Continued support for ARK, DIRK and ERK methods

Our existing functionality from previous ARKode versions has been retained:

- *ARKStep* supports ARK, DIRK and ERK methods for problems of the form

$$M\dot{y} = f^E(t, y) + f^I(t, y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0.$$

  - Can fully utilize any Butcher table packaged with ARKode, or any user-supplied tables.
  - Fully retains all functionality of previous ARKode versions.

- *ERKStep* is a leaner module that provides more optimal support for ERK-specific methods applied to the standard IVP problem,

$$\dot{y} = f(t, y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0.$$

Multirate Infinitesimal Step (MIS) stepper [Knoth & Wolke 1998; Schlegel et al. 2009; ...]

David Gardner [LLNL] has implemented a new *MRIStep* module to support $\mathcal{O}(h^2)$ and $\mathcal{O}(h^3)$ MIS methods for explicit-explicit multirate problems:

$$\dot{y} = f^{\{f\}}(t,y) + f^{\{s\}}(t,y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0,$$

- $f^{\{f\}}(t,y)$ contains the "fast" terms; $f^{\{s\}}(t,y)$ contains the "slow" terms;
- $h_s > h_f$; currently both are user-defined, but can be varied between steps;
- slow scale integrated with an ERK method satisfying $c_i < c_{i+1}, i = 1, \ldots, s-1$;
- fast scale is advanced over slow stage $\tau \in [t_n + c_i h_s, t_n + c_{i+1} h_s]$ by solving:

  $$\dot{y} = f^{\{f\}}(\tau, y) + \sum_{k=1}^{j} \alpha_k f^{\{s\}}\left(t_{n,k}^{\{s\}}, z_k^{\{s\}}\right), \quad y(t_n + c_i h_s) = z_i^{\{s\}};$$

  while currently explicit, implicit and ImEx will be added soon;
- only a single traversal of $[t_n, t_{n+1}]$ is required to obtain y_{n+1}.

This module will be included in the next release (this week).

Generalized Additive Runge-Kutta (GARK) stepper [Sandu & Günther, SINUM 2015]

David has also implemented a new *IMEXGARKStep* module to support ImEx GARK methods for problems with two partitions:

$$\dot{y} = f^{\{E\}}(t, y) + f^{\{I\}}(t, y), \quad t \in [t_0, t_f], \quad y(t_0) = y_0.$$

- Users supply Butcher table components $A^{\{E,E\}}$, $A^{\{I,I\}}$, $A^{\{E,I\}}$ and $A^{\{I,E\}}$, corresponding to E-E, I-I, E-I and I-E couplings, respectively; coefficients $b^{\{E\}}$ and $b^{\{I\}}$ define the timestep solution.
- $A^{\{E,E\}}$ and $A^{\{E,I\}}$ must be explicit.
- $A^{\{I,I\}}$ and $A^{\{I,E\}}$ can be diagonally implicit.
- Currently assumes that all tables have the same number of stages.

This module will be included in an upcoming release.

Outline

Conclusions

The ARKode infrastructure flexibly supports extensive studies of optimal algorithms for multiphysics problems:

- Numerous built-in ERK, DIRK, ARK methods, support for user-supplied
- Numerous vector/matrix data structures, support for user-supplied
- Numerous algebraic solver algorithms, support for user-supplied
- Simplifies transition from research-level time integration methods to production software, via reusable infrastructure components:
  - Numerous timestep adaptivity controllers, support for user-supplied
  - Robust temporal interpolation
  - Numerous implicit predictors
  - Extensible Butcher table module simplifies user control over method.

Thanks & Acknowledgements

Collaborators/Students:

- Jean M. Sexton [LBL]
- John Loffeld [LLNL]
- Rujeko Chinomona [SMU, PhD]
- Vu Thai Luan [SMU, postdoc]

Current Grant/Computing Support:

- DOE SciDAC & ECP Programs
- SMU Center for Scientific Computation

Software:

- ARKode – http://faculty.smu.edu/reynolds/arkode
- SUNDIALS – https://computation.llnl.gov/casc/sundials

Support for this work was provided by the Department of Energy, Office of Science project "Frameworks, Algorithms and Scalable Technologies for Mathematics (FASTMath)," under Lawrence Livermore National Laboratory subcontracts B598130, B621355, and B626484.

References

- Ropp & Shadid, *J. Comput. Phys.*, 203, 2005.
- Estep et al., *Comput. Meth. Appl. Mech. Eng.*, 196, 2007.
- Ascher et al., *Applied Numerical Mathematics*, 25, 1997.
- Araújo et al., *SIAM J. Numer. Anal.*, 34, 1997.
- Gardner et al., *Model. Simul. Mater. Sci. Eng.*, 23, 2015.
- Gardner et al., *Geosci. Model Dev.*, 11, 2018.
- Vogl et al., *in preparation*, 2018.
- Knoth & Wolke, *Appl. Numer. Math.*, 1998.
- Schlegel et al., *J. Comput. Appl. Math.*, 2009.
- Sandu & Günther, *SINUM.*, 2015.

SMU FASTMATH