The PVODE Trio: PVODE, KINSOL, and IDA

Alan C. Hindmarsh
Lawrence Livermore National Laboratory

Preprint

UCRL-JC-145579

October 2001

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

The PVODE Trio:
PVODE, KINSOL, and IDA

Alan C. Hindmarsh

Collaborators:
Peter Brown, Keith Grant, Steven Lee,
Radu Serban, Dan Shumaker, Allan Taylor,
Carol Woodward

Nonlinear Solvers & Differential Eqns. Project
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

ACTS Toolkit Workshop
11 October 2001

Background

LLNL has a long history of R & D in ODE methods and software, and closely related areas, with emphasis on applications to PDEs.

Popular Fortran solvers written at LLNL:

- VODE: ODE initial value problems for stiff/nonstiff systems, with direct solution of linear systems [Brown, Byrne, Hindmarsh]
- VODPK: Variant of VODE with preconditioned Krylov solution of linear systems (GMRES iteration) [Brown, Byrne, Hindmarsh]
- NKSOL: Newton-Krylov (GMRES) solver for nonlinear algebraic systems [Brown & Saad]
- DASPK: Differential-algebraic system solver (from DASSL) with direct and preconditioned Krylov solution of linear systems [Brown, Hindmarsh, Petzold]

Areas of special interest in recent years:

- parallel solution of large problems
- sensitivity of solution w.r.t. model parameters

Background (cont.)

Starting in 1993, the push to solve large systems in parallel motivated work to write or rewrite solvers in C.

The first result:

CVODE = C rewrite of VODE + VODPK
[Cohen & Hindmarsh, 1994]

CVODE

IVP: $\dot{y} = f(t, y), \quad y(t_0) = y_0, \quad y \in \mathbf{R}^N$

Methods: variable-order, variable-step

- BDF = Backward Differentiation Formulas (stiff)
  (Fixed-Leading Coeff. form)
- Implicit Adams (nonstiff)

Nonlinear systems solved by:

- Newton (stiff)
- Functional Iteration (nonstiff)

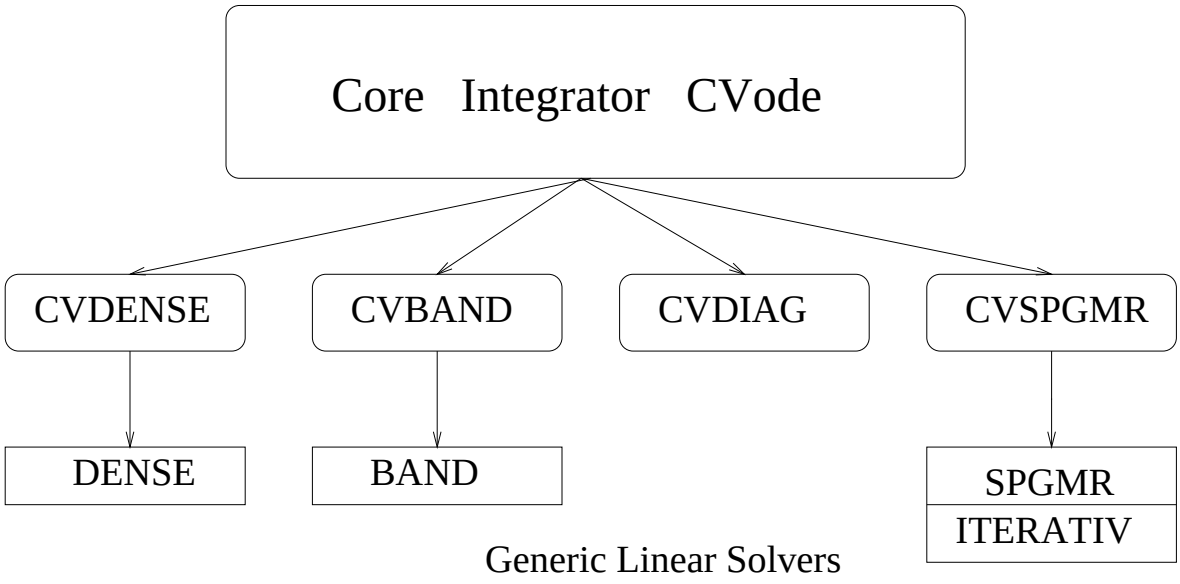
Linear systems solved by:

- Dense or band direct solver – user or internal Jacobian
- SPGMR = Scaled Preconditioned GMRES:
  unrestarted, matrix-free, left/right preconditioning,
  user routines for preconditioning setup/solve

Code organization completely redone:

- Memory allocation
- Linear solver modules separate from core integrator
- Each linear solver has interface + generic solver
- Separate module of vector kernels (linear sums, dot products, norms, etc.) on vectors of type `N_Vector`

CVODE Organization



NVECTOR

LLNLTYPS

LLNLMATH

PVODE

PVODE (1998) = Parallel extension of CVODE

Methods: as in CVODE, but no direct linear solvers

- nonstiff method
- stiff method with SPGMR linear solver

Parallelism assumes:

- SPMD model
- all N-vectors identically distributed across processors

Parallelism achieved by doing a parallel rewrite of the NVECTOR module. Versions:

- Cray SHMEM version (not released)
- MPI version
- user can supply own parallel NVECTOR module

Released package is MPI_PVODE.

PVODE Usage

Unlike the user of the Fortran solvers, the PVODE user calls several routines for various parts of solution process.

- Set local vector length
- `machEnv = PVecInitMPI(...)`: init. NVECTOR
- Set initial values of y (type `N_Vector`)
- `mem = CVODEMalloc(...)`: initialize PVODE
- `CVSpgmr(...)`: if Newton, specify SPGMR and preconditioner setup and solve routines
- `for (tout=...)  ier = CVode(...)`: integrate
- `CVodeFree`: free PVODE memory

Errors are controlled via user input tolerances:

- `rtol` = scalar relative tolerance
- `atol` = absolute tolerance = scalar or vector

Resulting error weights $\texttt{rtol}|y^i|+\texttt{atol}^i$ are also used to scale GMRES.

PVODE - Preconditioning

Preconditioner P must approximate Newton matrix, yet be reasonably efficient to evaluate and solve.

From linear multistep method,

$y_n = h\beta_0 \dot{y}_n +$ sum of known past values,

where $\dot{y}_n = f(t_n, y_n)$ (h = stepsize, β_0 = BDF coeff.), the Newton matrix is $I - \gamma J, \quad J = \partial f / \partial y, \ \gamma \equiv h\beta_0$.

Typical P is $I - \gamma \tilde{J}$ with $\tilde{J} \sim J$, possibly a crude approximation.

Treatment of P is in two phases:

- evaluate and preprocess P (infrequently)
- solve systems $Px = b$ (frequently)

User can save $\tilde{J}$ and reuse it when γ changes (trading computation for storage), as directed by PVODE.

The user must supply routines for setup and solve of P, but the package offers help:

- Example illustrates operator-split preconditioner for reaction-diffusion systems
- BBD module supplied: Band-Block-Diagonal preconditioner

Other linear solvers useful, given choice of $\tilde{J}$.

PVODE - BBD Preconditioner

Directed at PDE-based problems, using Domain Decomposition

Time-dependent PDE system, with spatial discretization, gives ODE system $\dot{y} = f(t, y)$.

Decompose domain into M non-overlapping subdomains.

DD induces block form $y = (y_1, \cdots, y_M)$, same for f.

Use this distribution for PVODE on M processors.

But $f_m(t, y)$ depends on both y_m and ghost cell data from other $y_{m'}$, typically in a local manner.

Build preconditioner P by:

- computing $\partial f_m / \partial y_m$ (ignore coupling)
- replacing f by $g \approx f$ ($g = f$ allowed)

E.g., g may have smaller set of ghost cell data.

On processor m, use J_m = banded difference quotient approximation to $\partial g_m / \partial y_m$, then

$$P = diag[P_1, \cdots, P_M] \ , \quad P_m = I_m - \gamma J_m$$

Solve $Px = b$ by band LU and backsolve ops. on each processor (setup = evaluation + LU, solve = backsolve).

PVODE - BBD (cont.)

User supplies g as two routines:

- `gcomm`: inter-processor communication of data needed to evaluate g_m
- `glocal`: evaluate g_m on processor m

User also supplies:

- half-bandwidths `ml,mu` of band matrix J_m
- half-bandwidths `mldq,mudq` for use in D.Q. algorithm (cost of J_m is `mldq+mudq+2` evaluations of g_m)

(1) `ml,mu` may be smaller than `mldq,mudq` – trading lower matrix costs for slower convergence.

(2) Both pairs of half-bandwidths may be less than the true values for $\partial g_m/\partial y_m$, for efficiency.

(3) Both pairs may depend on m.

PVODE - Fortran/C Interfaces

Fortran applications are accommodated, via a set of interface routines.

(Fortran user) $\longleftrightarrow$ (interfaces) $\longleftrightarrow$ PVODE

Cross-language calls go in both directions:

Fortran Main $\longrightarrow$ interfaces to solver routines

Solver routines $\longrightarrow$ interfaces to user's f etc.

For portability, all user routines have fixed names.

Small examples are provided.

KINSOL

Solves $F(u) = 0, \quad F : R^N \to R^N$, given a guess u_0.

C rewrite of Fortran NKSOL [Brown & Saad]

Method is Inexact Newton:

Newton correction equation $J\Delta u_n = -F(u_n)$ is solved only approximately, with a preconditioned Krylov method.

Krylov solver: SPGMR = Scaled Precond. GMRES

- restarts allowed
- preconditioning on the right: $(JP^{-1})(P\Delta) = -F$

Krylov iteration requires matrix-vector products $J(u)v$, done by:

- user-supplied routine, or
- difference quotient $[F(u + \sigma v) - F(u)]/\sigma$

Choice of Newton strategies:

- Inexact Newton
- Inexact Newton with Linesearch/Backtrack

(NKSOL also had a Dogleg Method; KINSOL does not.)

KINSOL (cont.)

Optional inequality constraints: $u^i > 0$ or $u^i < 0$

Error controls:

1. Newton stopping test: $\|D_F F(u_n)\| < ftol$ with input scaling D_F for F and input tolerance $ftol$.

2. Krylov stopping test: $\|J\Delta_k + F\| < \eta_k \|F\|$ with three choices:

- $\eta_k =$ constant
- two 'forcing term' choices of Eisenstat/Walker [1996]

3. For step control and choice of σ, user must also supply $D_u =$ scaling for u.

KINSOL – BBD Preconditioner

Package includes band-block-diagonal preconditioner module analogous to PVODE's BBD.

Defined via $g \approx F$:

$$P = diag[P_1, \cdots, P_M] \ , \quad P_m = J_m \approx \partial g_m / \partial y_m$$

J_m is banded, via difference quotients, with user-supplied half-bandwidths for D.Q. alg. and retained matrix.

KINSOL Code Organization

Same basic organization as CVODE/PVODE
(only one linear solver choice at present)

Shared modules:

- generic SPGMR solver
- NVECTOR modules
  - serial
  - parallel (MPI version only)

User supplies routines for:

- F
- P setup and solve (optional)
- Jv product (optional)
- `gcomm, glocal` (for BBD preconditioner)

Examples provided with user preconditioner and BBD

Package of Fortran/C interfaces provided

IDA

Solves Initial Value Problem for DAE system

$$F(t, y, y') = 0,$$
$$F : R \times R^N \times R^N \to R^N,$$
given y_0, y'_0 at $t = t_0$

C rewrite of Fortran DASPK [Brown/Hindmarsh/Petzold]

Method: Variable-order BDF, variable-coefficient
(Fixed-Leading-Coefficient form)

Newton corrections involve Newton matrix

$$J = \partial F/\partial y + \alpha \partial F/\partial y'$$

$\alpha = \alpha_0/h$ ($h =$ stepsize, $\alpha_0 =$BDF coeff.)

Linear systems solved by:

- direct solve (dense or banded, user or internal J) (serial version only)
- SPGMR = Scaled Precond. GMRES
  - restarts allowed
  - preconditioning on left: $(P^{-1}J)(\Delta y) = -P^{-1}F$
  - user routines for P setup & solve

Optional inequality constraints:

$y^i > 0$ or $y^i < 0$ or $y^i \geq 0$ or $y^i \leq 0$

IDA - Initial Condition Calculation

User input y_0, y_0' may or may not be consistent ($F = 0$), but must be for integration to succeed.

Optional user-callable routine solves for consistent values, for two classes of problems:

- Semi-explicit index-1 systems, differential components of y_0 known, algebraic components unknown
- All of y_0' specified, y_0 unknown

IDA solves $F(t_0, y_0, y_0') = 0$ for unknown components of y_0 and y_0', using

- Newton iteration with Linesearch
- existing linear system solver machinery (+ tricks)

IDA - BBD Preconditioner

Package includes band-block-diagonal preconditioner module analogous to PVODE's BBD.

Defined via $G \approx F$:

$P = diag[P_1, \cdots, P_M]$

$P_m = J_m \approx \partial G_m / \partial y_m + \alpha \partial G_m / \partial y'_m$

J_m is banded, via difference quotients, with user-supplied half-bandwidths for D.Q. alg. and retained matrix.

IDA - Code Organization

Same basic organization as CVODE/PVODE

Shared modules:

- generic dense, band, SPGMR solvers
- NVECTOR modules
  - serial
  - parallel (MPI version only)

User supplies routines for:

- F
- J for direct solve (optional)
- P setup and solve for SPGMR (optional)
- `gcomm, glocal` (for BBD preconditioner)

Examples provided with user preconditioner and BBD

Sensitivity Analysis

In addition to the solution y or u, we want its sensitivity (first-order) with respect to parameters in the problem (or initial conditions).

(1) ODEs.

If $p = (p_1, \cdots, p_m)$ and $\dot{y} = f(t, y, p)$, $y(t_0) = y_0(p)$, we want $s = \partial y / \partial p$ $(N \times m)$.

Each column $s_i = \partial y / \partial p_i$ satisfies another ODE

$$\dot{s}_i = J s_i + \partial f / \partial p_i \quad (J = \partial f / \partial y)$$

with initial values $s_i(t_0) = \partial y_0 / \partial p_i$.

SensPVODE [Lee/Hindmarsh/Brown] integrates the extended ODE system for $Y = (y, w_1, \cdots, w_m)$, where $w_i = \bar{p}_i s_i$ and $\bar{p}_i =$ scale factor $\sim p_i$.

Evaluation of $\dot{w}_i = \bar{p}_i \dot{s}_i$ done by difference quotients (range of choices) or by Automatic Differentiation.

Jacobian of extended system, of size $N(m+1)$, is approximated by $diag[J, \cdots, J]$. Appropriate preconditioner is $diag[P, \cdots, P]$. Linear systems involve added solve operations but no added matrix setup operations.

Sensitivity Analysis (cont.)

(2) Nonlinear Systems.

$F(u,p) = 0,\ s = \partial u/\partial p \Rightarrow$

$$Js_i = -\partial F/\partial p_i \quad (J = \partial F/\partial u)\ .$$

SensKINSOL [Grant/Hindmarsh/Taylor] solves for u (if not done already by KINSOL), then solves linear systems for $w_i = \bar{p}_i s_i$.

(3) DAEs.

$F(t,y,y',p) = 0,\ s = \partial y/\partial p \Rightarrow$

$$\frac{\partial F}{\partial y}s_i + \frac{\partial F}{\partial y'}s_i' + \frac{\partial F}{\partial p_i} = 0$$

SensIDA [Lee/Hindmarsh] integrates the extended DAE system for $Y = (y, w_1, \cdots, w_m)$, where $w_i = \bar{p}_i s_i$.

Newton matrix of extended system is approximated by $diag[J, \cdots, J]$ (J = Newton matrix of original system).

Sensitivity Analysis (cont.)

(4) Alternative approach in ODE and DAE cases:

Staggered corrector iteration: Solve for $y_n \approx y(t_n)$, then for the $s_i(t_n)$. Saves time if y_n problem has trouble.

Experimental extension to CVODE/PVODE in progress [Serban].

Applications

* PVODE is being used in a parallel 3D tokamak turbulence model in LLNL's Magnetic Fusion Energy Division. A typical run has 7 unknowns on a $64 \times 64 \times 40$ mesh, with up to 60 processors.

* KINSOL with a HYPRE Multigrid preconditioner is being applied within LLNL/CASC to solve a nonlinear Richards equation for pressures in porous media flows. Fully scalable solution performance obtained on up to 225 processors of ASCI Blue.

* PVODE, KINSOL, IDA, with MG preconditioner, are being used to solve 3D neutral particle transport problems within LLNL/CASC. Scalable performance obtained on up to 5800 processors on ASCI Red.

* SensPVODE, SensKINSOL, and SensIDA have been used to determine solution sensitivities of neutral particle transport applications at LLNL w.r.t. various material properties, for solution uncertainty quantification.

* IDA and SensIDA are being used in a cloud and aerosol microphysics model at LLNL to study cloud formation processes, in study of model parameter sensitivity.

Sources and References

Publications listed are available from the ACTS Toolkit page and/or the CASC/NSDE Project website,
www.llnl.gov/CASC/nsde/

Sources for PVODE, KINSOL, IDA are available at the LLNL/CASC Software Download Site,
www.llnl.gov/CASC/download/download_home.html

[1] S. D. Cohen and A. C. Hindmarsh, "CVODE User Guide," LLNL Report UCRL-MA-118618, Sept. 1994.

[2] Scott D. Cohen and Alan C. Hindmarsh, "CVODE, a Stiff/Nonstiff ODE Solver in C," Computers in Physics, vol. 10, no. 2 (March/April 1996), pp. 138-143.

[3] Michael R. Wittman, "Testing of PVODE, a Parallel ODE Solver," LLNL Report UCRL-ID-125562, August 1996. Compares SHMEM_PVODE and MPI_PVODE.

[4] George D. Byrne and Alan C. Hindmarsh, "User Documentation for PVODE, An ODE Solver for Parallel Computers," LLNL Report UCRL-ID-130884, May 1998.

[5] Allan G. Taylor and Alan C. Hindmarsh, "User Documentation for KINSOL, A Nonlinear Solver for Sequential and Parallel Computers," LLNL Report UCRL-ID-131185, July 1998.

[6] George D. Byrne and Alan C. Hindmarsh, "PVODE, An ODE Solver for Parallel Computers," in Int. J. High Perf. Comput. Applic., vol. 13, no. 4 (1999), pp. 354-365.

[7] Alan C. Hindmarsh and Allan G. Taylor, "User Documentation for IDA, a Differential-Algebraic Equation Solver for Sequential and Parallel Computers," LLNL Report UCRL-MA-136910, December 1999.

[8] Alan C. Hindmarsh, "The PVODE and IDA Algorithms", LLNL Report UCRL-ID-141558, December 2000. Detailed algorithm descriptions.