Lawrence Livermore National Laboratory

SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers

Carol S. Woodward

Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

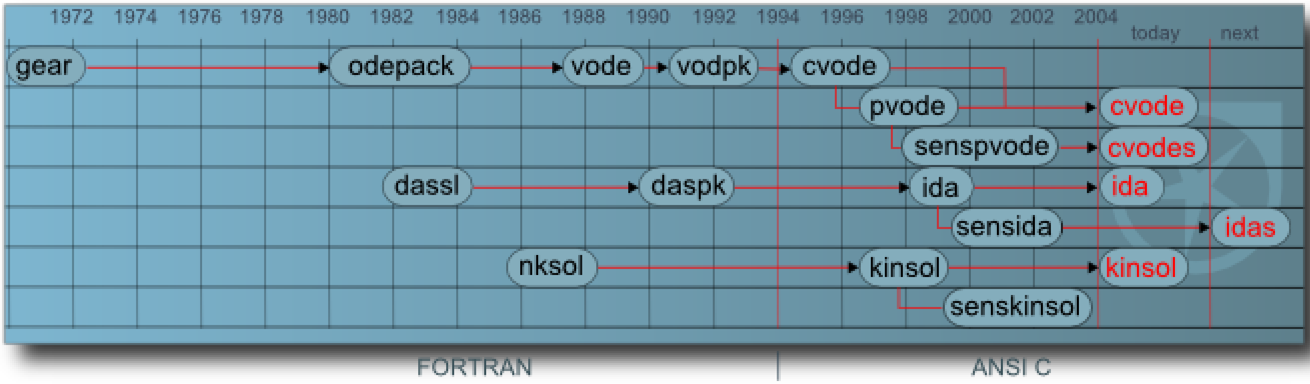
UCRL-PRES-213978

Outline

- SUNDIALS Overview
- ODE and DAE integration
  - Initial value problems
  - Implicit integration methods
- Nonlinear Systems
  - Newton’s method and inexact Newton’s method
  - Preconditioning
- SUNDIALS: usage, applications, and availability
- Upcoming additions

LLNL has a long history of R&D in ODE/DAE methods and software

- Fortran solvers written at LLNL:
  - VODE: stiff/nonstiff ODE systems, with direct linear solvers
  - VODPK: with Krylov linear solver (GMRES)
  - NKSOL: Newton-Krylov solver - nonlinear algebraic systems
  - DASPK: DAE system solver (from DASSL)
- Recent focus has been on sensitivity analysis
- Organized into a single suite, SUNDIALS, written in C and including CVODE and CVODES, IDA, IDAS, and KINSOL



May 2009

Push to solve large, parallel systems motivated rewrites in C

- CVODE: rewrite of VODE/VODPK [Cohen, Hindmarsh, 94]
- PVODE: parallel CVODE [Byrne and Hindmarsh, 98]
- KINSOL: rewrite of NKSOL [Taylor and Hindmarsh, 98]
- IDA: rewrite of DASPK [Hindmarsh and Taylor, 99]
- Sensitivity variants: SensPVODE, SensIDA, SensKINSOL [Brown, Grant, Hindmarsh, Lee, 00-01]
- New sensitivity-capable solvers:
  - CVODES [Hindmarsh and Serban, 02]
  - IDAS [Serban, Petra, and Hindmarsh, 09]

- Organized into a single suite, SUNDIALS, including CVODE and CVODES, IDA, IDAS, and KINSOL

The SUNDIALS package offers Newton solvers, time integration, and sensitivity solvers

- **CVODE: implicit ODE solver, y' = f(y, t)**
  - **Variable-order, variable step BDF (stiff) or implicit Adams (nonstiff)**
  - **Nonlinear systems solved by Newton or functional iteration**
  - **Linear systems by direct (dense or band) or iterative solvers**
- **IDA: implicit DAE solver, F(t, y, y') = 0**
  - **Variable-order, variable step BDF**
  - **Nonlinear system solved by Newton iteration**
  - **Linear systems by direct (dense or band) or iterative solvers**
- **KINSOL: Newton solver, F(u) = 0**
  - **Inexact and Modified (with dense solve) Newton**
  - **Linear systems by iterative or dense direct solvers**
- **CVODES: sensitivity-capable (forward & adjoint) CVODE**
- **IDAS: sensitivity-capable (forward & adjoint) IDA**
- **Iterative linear Krylov solvers: GMRES, BiCGStab, TFQMR**

SUNDIALS was designed to easily interface with legacy codes

- **Philosophy: *Keep codes simple to use***
- **Written in C**
  - **Fortran interfaces: FCVODE, FIDA, and FKINSOL**
  - **Matlab interfaces: sundialsTB (CVODES, IDA, & KINSOL)**
- **Written in a data structure neutral manner**
  - **No specific assumptions about data**
  - **Application-specific data representations can be used**
- **Modular implementation**
  - **Vector modules**
  - **Linear solver modules**
- **Require minimal problem information, but offer user control over most parameters**

Initial value problems (IVPs) come in the form of ODEs and DAEs

- The general form of an IVP is given by

$$F(t, \dot{x}, x) = 0$$
$$x(t_0) = x_0$$

- **If $\partial F / \partial \dot{x}$ is invertible, we solve for $\dot{x}$ to obtain an ordinary differential equation (ODE), but this is not always the best approach**
- **Else, the IVP is a differential algebraic equation (DAE)**

- **A DAE has differentiation index *i* if *i* is the minimal number of analytical differentiations needed to extract an explicit ODE**

Stiffness of an equation can significantly impact whether implicit methods are needed

- (Ascher and Petzold, 1998): If the system has widely varying time scales, and the phenomena that change on fast scales are *stable*, then the problem is stiff
- Stiffness depends on
  - Jacobian eigenvalues, λ_j
  - System dimension
  - Accuracy requirements
  - Length of simulation
- In general a problem is stiff on $[t_0, t_1]$ if

$$\boxed{(t_1 - t_0)\min_j \Re(\lambda_j) << -1}$$

Dalquist test problem shows impact of stability on step sizes for explicit and implicit methods

Dalquist test equation: $\dot{y} = \lambda y, \; y(0) = y_0$

Exact solution: $y(t_n) = y_0 e^{\lambda t_n}$

Absolute stability requirement

$$|y_n| \le |y_{n-1}|, \quad n = 1,2,\ldots$$

If Re(λ)<0, then $|y(t_n)|$ decays exponentially; we cannot tolerate growth in the approximate solution y_n

Region of absolute stability of an integrator written as:

$y_n = R(z)y_{n-1}$, with time step $z = h\lambda$

$$S = \{z \in C; |R(z)| \le 1\}$$

Forward and backward Euler show different stability restrictions

- Forward Euler: $\boldsymbol{y_n = y_{n-1} + h(\lambda y_{n-1}) \Rightarrow R(z) = |1 + h\lambda|}$

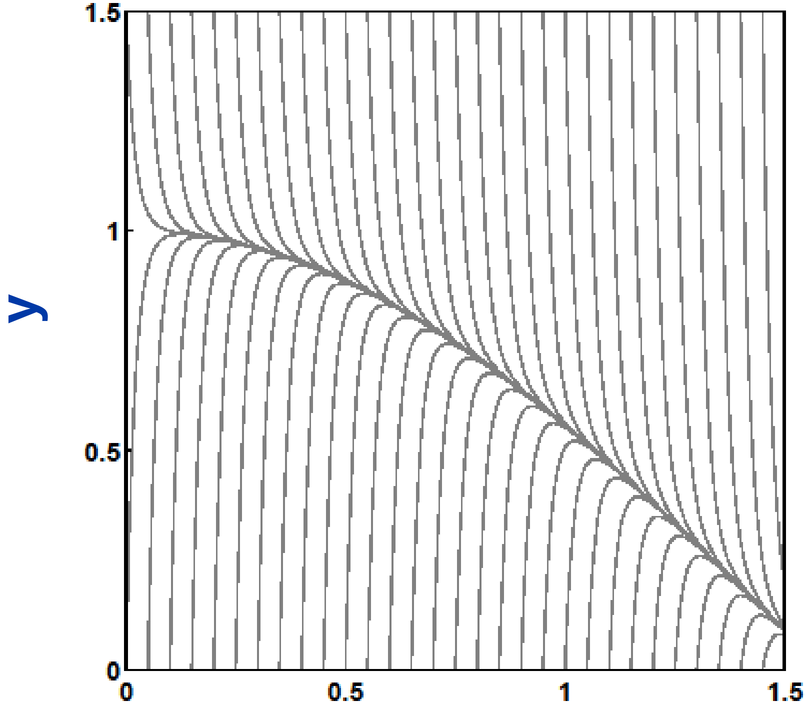
So, if λ < 0, FE has the step size restriction: $\boldsymbol{h \le \frac{2}{|\lambda|}}$

- Backward Euler: $\boldsymbol{y_n = y_{n-1} + h(\lambda y_n) \Rightarrow R(z) = \left|\frac{1}{1 - h\lambda}\right|}$

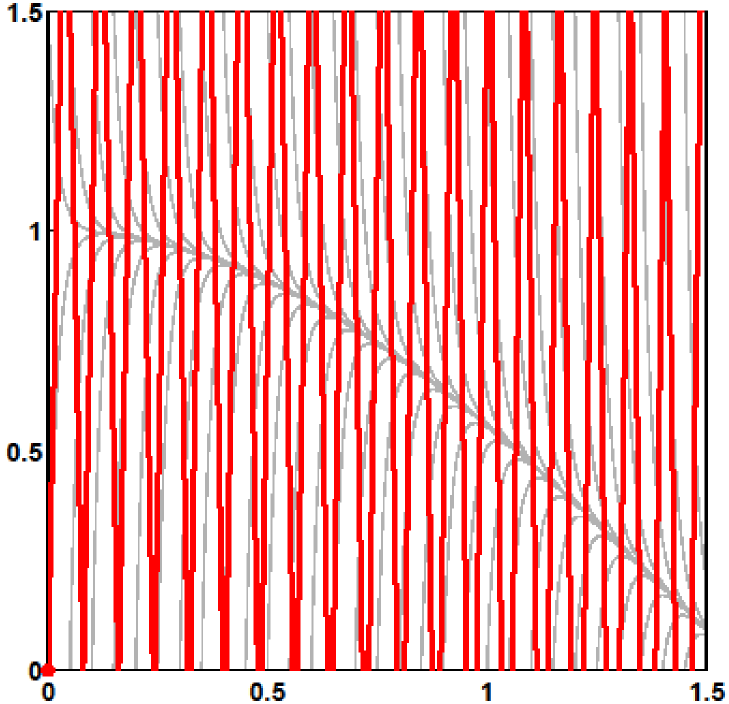
So, if λ < 0, BE has the step size restriction: $\boldsymbol{h > 0}$

Curtiss and Hirchfelder example

$$\dot{y} = -50\left(y - \cos(t)\right) \quad \lambda = -50$$



Solution curves

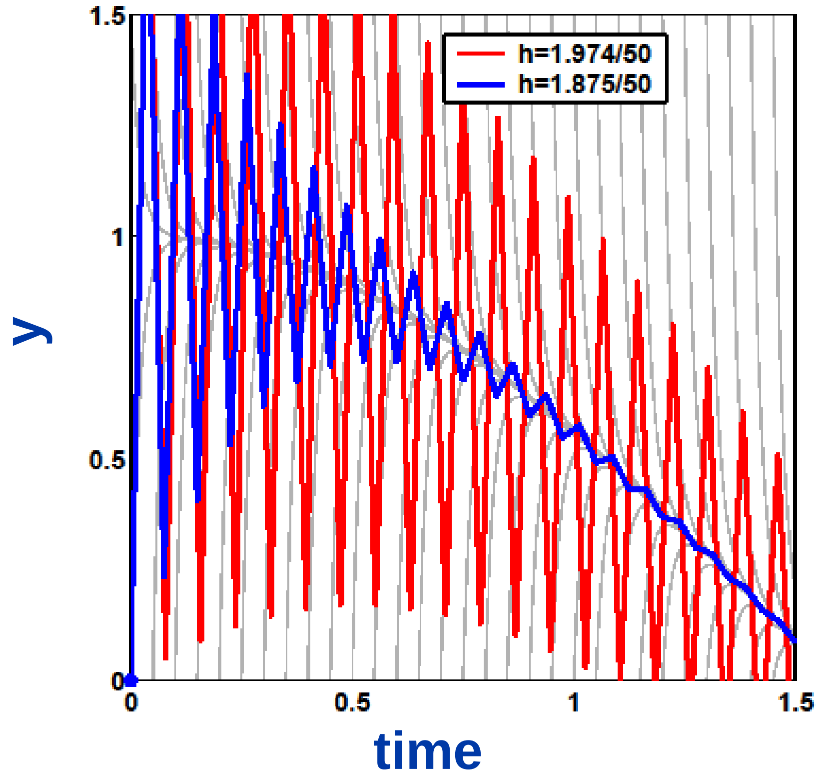


Forward Euler

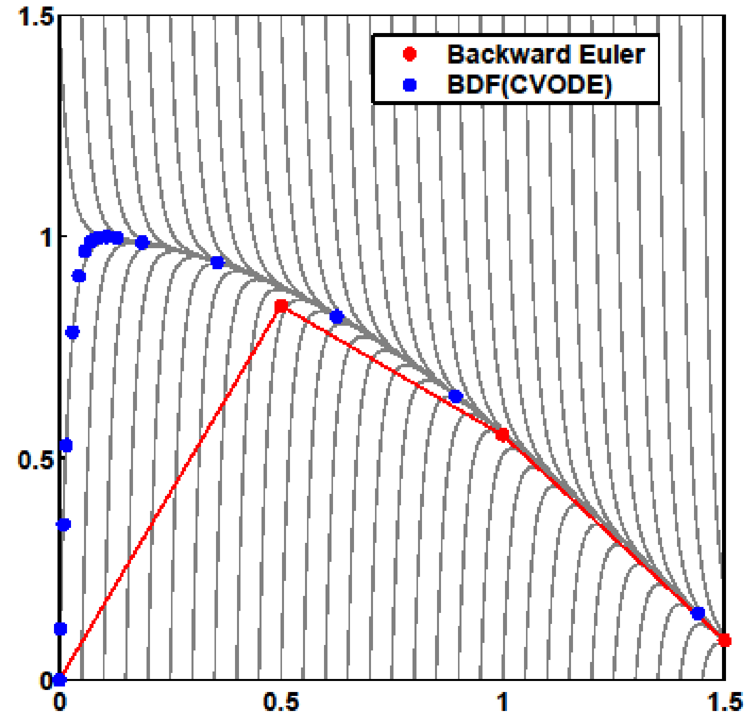
h=2.01/50

Curtiss and Hirchfelder example

$$\dot{y} = -50(y - \cos(t)) \quad \lambda = -50$$



Forward Euler



Implicit schemes

h=0.5 for BE

SUNDIALS has implementations of Linear Multistep Methods (LMM)

General form of LMM: $$\sum_{i=0}^{K_1} \alpha_{n,i} y_{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{y}_{n-i} = 0$$

- Two methods:
  - Adams-Moulton (nonstiff); $K_1 = 1$, $K_2 = k$, $k = 1,\ldots,12$
  - BDF (stiff); $K_1 = k$, $K_2 = 0$, $k = 1,\ldots,5$

- Nonlinear systems (BDF)
  - ODE:

$$\dot{y} = f(y) \qquad G(y_n) \equiv y_n - \beta_0 h_n f(t, y_n) - \sum_{i=1}^{k} \alpha_{n,i} y_{n-i} = 0$$

  - DAE:

$$F(\dot{y}, y) = 0 \qquad G(y_n) \equiv F\left(t, (\beta_0 h_n)^{-1} \sum_{i=1}^{k} \alpha_{n,i} y_{n-i}, y_n \right) = 0$$

Stability is very restricted for higher orders of BDF methods

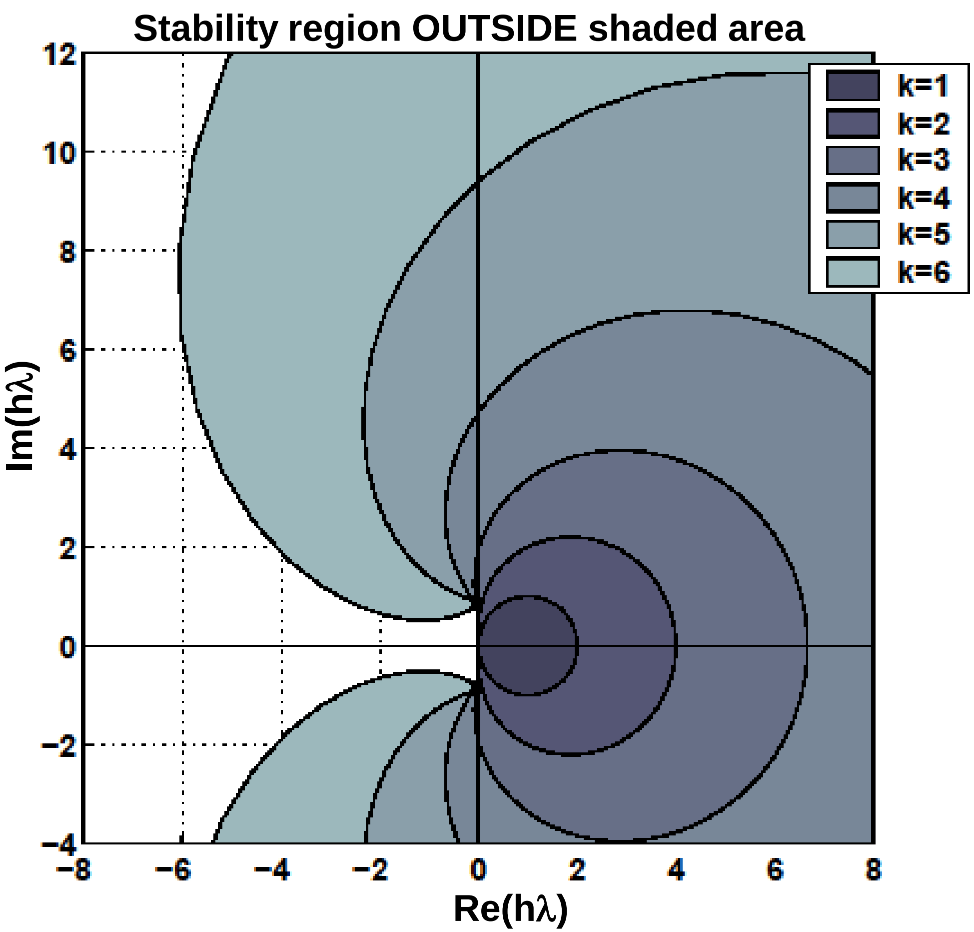
$$y_n - \beta_0 h_n \dot{y}_n = \sum_{i=1}^{k} \alpha_{n,i} y_{n-i}$$

Regions of instability grow with the order

CVODE and IDA allow up to order 5

CVODE includes an optional stability limit detection algorithm:

- **Based on linear analysis**
- **Limits step if it detects a potential stability problem**



CVODE solves $\dot{y} = f(t, y)$

- Variable order and variable step size methods:
  - BDF (backward differentiation formulas) for stiff systems
  - Implicit Adams for nonstiff systems
- (Stiff case) Solves time step for the system $\dot{y} = f(t, y)$
  - applies an explicit predictor to give $y_{n(0)}$

$$y_{n(0)} = \sum_{j=1}^{q} \alpha_j^p y_{n-j} + \Delta t \beta_1^p \dot{y}_{n-1}$$

  - applies an implicit corrector with $y_{n(0)}$ as the initial guess

$$y_n = \sum_{j=1}^{q} \alpha_j y_{n-j} + \Delta t \beta_0 f_n(y_n)$$

Time steps and order are chosen to minimize the local truncation error

- Time steps are chosen by:
  - Estimate the error: $E(\Delta t) = C(y_n - y_{n(0)})$
    - Accept step if $||E(\Delta t)||_{WRMS} < 1$
    - Reject step otherwise
  - Estimate error at the next step, $\Delta t'$, as

$$E(\Delta t') \approx (\Delta t' / \Delta t)^{q+1} E(\Delta t)$$

  - Choose next step so that $||E(\Delta t')||_{WRMS} < 1$
- Choose method order by:
  - Estimate error for next higher and lower orders
  - Choose the order that gives the largest time step meeting the error condition

Computations weighted so no component disproportionally impacts convergence

- An absolute tolerance is specified for each solution component, ATOL^i
- A relative tolerance is specified for all solution components, RTOL
- Norm calculations are weighted by:

$$\mathbf{ewt}^{i} = \frac{1}{\mathbf{RTOL}\cdot\left|\mathbf{y}^{i}\right| + \mathbf{ATOL}^{i}} \qquad \|\mathbf{y}\|_{\mathbf{WRMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{ewt}^{i}\cdot\mathbf{y}^{i}\right)^{2}}$$

- Bound time integration error with:

$$\left\|\mathbf{y}_{n} - y_{n(0)}\right\| < \tfrac{1}{6}$$

The 1/6 factor tries to account for estimation errors

Nonlinear system will require nonlinear solves

- Use predicted value as the initial iterate for the nonlinear solver
- Nonstiff systems: Functional iteration

$$y_{n(m+1)} = \beta_0 h_n f\left(y_{n(m)}\right) + \sum_{i=1}^{q} \alpha_{n,i} y_{n-i}$$

- Stiff systems: Newton iteration

$$M\left(y_{n(m+1)} - y_{n(m)}\right) = -G\left(y_{n(m)}\right)$$

  - ODE: $M \approx I - \gamma\,\partial f/\partial y, \quad \gamma = \beta_0 h_n$
  - DAE: $M \approx \partial F/\partial y + \gamma\,\partial F/\partial \dot{y}, \quad \gamma = 1/\left(\beta_0 h_n\right)$

SUNDIALS provides many options for linear solvers

- Iterative linear solvers
  - Result in inexact Newton solver
  - Scaled preconditioned solvers: GMRES, Bi-CGStab, TFQMR
  - Only require matrix-vector products
  - Require preconditioner for the Newton matrix, *M*
- Jacobian information (matrix or matrix-vector product) can be supplied by the user or estimated with finite difference quotients
- Two options require serial environments and some pre-defined structure to the data
  - Direct dense
  - Direct band

An inexact Newton-Krylov method can be used to solve the implicit systems

- Krylov iterative methods find the linear system solution in a Krylov subspace: $K(J,r) = \{r, \quad Jr, \quad J^2 r, \quad \ldots\}$
- Only require matrix-vector products
- Difference approximations to the matrix-vector product are used,

$$J(x)v \approx \frac{F(x + \theta v) - F(x)}{\theta}$$

- Matrix entries need never be formed, and memory savings can be used for a better preconditioner

IDA solves $F(t, y, y') = 0$

- C rewrite of DASPK [Brown, Hindmarsh, Petzold]
- Variable order / variable coefficient form of BDF
- Targets: implicit ODEs, index-1 DAEs, and Hessenberg index-2 DAEs
- Optional routine solves for consistent values of y_0 and y_0'
  - Semi-explicit index-1 DAEs, differential components known, algebraic unknown OR all of y_0' specified, y_0 unknown
- Nonlinear systems solved by Newton-Krylov method

- Optional constraints: $y^i > 0$, $y^i < 0$, $y^i \geq 0$, $y^i \leq 0$

KINSOL solves $F(u) = 0$

- C rewrite of Fortran NKSOL (Brown and Saad)
- Inexact Newton solver: solves $J\,\Delta u^n = -F(u^n)$ approximately
- Modified Newton option (with direct solves) – this freezes the Newton matrix over a number of iterations
- Krylov solver: scaled preconditioned GMRES, TFQMR, Bi-CGStab
  - Optional restarts for GMRES
  - Preconditioning on the right: $(J\,P^{-1})(Ps) = -F$
- Direct solvers: dense and band (serial & special structure)
- Optional constraints: $u_i > 0$, $u_i < 0$, $u_i \geq 0$ or $u_i \leq 0$
- Can scale equations and/or unknowns
- Dynamic linear tolerance selection

An inexact Newton's method is used to solve the nonlinear problem

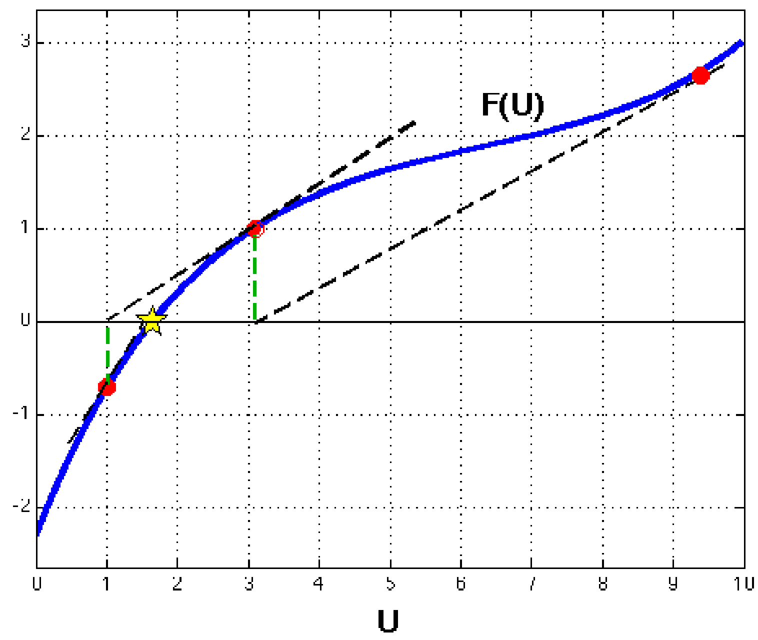
1. Starting with x^0, want x^* such that $F(x^*) = 0$

2. Repeat for each k until $\left\| \mathbf{F}(\mathbf{x}^{k+1}) \right\| \leq \mathbf{tol}$

   a. Solve (approximately)

   $$\mathbf{J}(\mathbf{x}^k)\mathbf{s}^k = -\mathbf{F}(\mathbf{x}^k)$$

   b. Update, $x^{k+1} = x^k + \lambda s^k$



courtesy of D. Reynolds (SMU)

- tol may be chosen adaptively based on accuracy requirements
- λ is a search parameter
- $\|.\|$ is a weighted L-2 norm

Linear stopping tolerances must be chosen to prevent "oversolves"

The linear system is solved to a given tolerance:

$$\left\| \mathbf{F}(\mathbf{x}^{k}) + \mathbf{J}(\mathbf{x}^{k})\mathbf{s}^{k+1} \right\| \leq \eta^{k} \left\| \mathbf{F}(\mathbf{x}^{k}) \right\|$$

- Newton method assumes a linear model
  - Bad approximation far from solution, loose tol.
  - Good approximation close to solution, tight tol.
- Eisenstat and Walker (SISC 96)
  - Choice 1 $\eta^{k} = \left| \left\| F^{k} \right\| - \left\| F^{k-1} - J^{k-1} s^{k-1} \right\| \right| \Big/ \left\| F^{k-1} \right\|$
  - Choice 2 $\eta^{k} = 0.9 \left(\left\| \mathbf{F}^{(k)} \right\| \Big/ \left\| \mathbf{F}^{(k-1)} \right\| \right)^{2}$
- ODE literature $\eta^{k} = 0.05$

Inexact methods maintain the fast rate of convergence of Newton's method

- Convergence of Newton's method is *q-quadratic* locally, for some constant C

$$\left\| \mathbf{x}^{k+1} - \mathbf{x}^{*} \right\| \le \mathbf{C} \left\| \mathbf{x}^{k} - \mathbf{x}^{*} \right\|^{2}$$

- Convergence of an inexact Newton method is
  - *q-linear* if η^{k} is constant in *k*
  - *q-super-linear* if $\lim_{k \to \infty} \eta^{k} = 0$
  - *q-quadratic* if for some constant C

$$\left\| \mathbf{F}(\mathbf{x}^{k}) + \mathbf{J}(\mathbf{x}^{k}) \mathbf{s}^{k+1} \right\| \le \mathbf{C} \left\| \mathbf{F}(\mathbf{x}^{k}) \right\|^{2}$$

- Eisenstat and Walker methods are *q-quadratic*

Line-search globalization for Newton's method can enhance robustness

- User can select:
  - Inexact Newton
  - Inexact Newton with line search
- Line searches can provide more flexibility in the initial guess (larger time steps)
- Take, $x^{k+1} = x^k + \lambda s^{k+1}$, for λ chosen appropriately (to satisfy the Goldstein-Armijo conditions):
  - sufficient decrease in F relative to the step length
  - minimum step length relative to the initial rate of decrease
  - full Newton step when close to the solution

Preconditioning is essential for large problems as Krylov methods can stagnate

- Preconditioner P must approximate Newton matrix, yet be reasonably efficient to evaluate and solve.
- Typical P (for time-dep. ODE problem) is $I - \gamma \tilde{J}, \quad \tilde{J} \approx J$
- The user must supply two routines for treatment of P:
  - Setup: evaluate and preprocess P (infrequently)
  - Solve: solve systems Px=b (frequently)
- User can save and reuse approximation to J, as directed by the solver
- SUNDIALS offers hooks for user-supplied preconditioning
- Band and block-banded preconditioners are supplied for use with the supplied vector structure

Sensitivity Analysis

- Sensitivity Analysis (SA) is the study of how the variation in the output of a model (numerical or otherwise) can be apportioned, qualitatively or quantitatively, to different sources of variation in inputs.
- Applications:
  - Model evaluation (most and/or least influential parameters), Model reduction, Data assimilation, Uncertainty quantification, Optimization (parameter estimation, design optimization, optimal control, …)
- Approaches:
  - Forward sensitivity analysis
  - Adjoint sensitivity analysis

The SUNDIALS vector module is generic

- Data vector structures can be user-supplied
- The generic NVECTOR module defines:
  - A `content` structure (void *)
  - An `ops` structure – pointers to actual vector operations supplied by a vector definition
- Each implementation of NVECTOR defines:
  - Content structure specifying the actual vector data and any information needed to make new vectors (problem or grid data)
  - Implemented vector operations
  - Routines to clone vectors
- Note that all parallel communication resides in reduction operations: dot products, norms, mins, etc.

SUNDIALS provides serial and parallel NVECTOR implementations

- *Use is optional*

- Vectors are laid out as an array of doubles (or floats)
- Appropriate lengths (local, global) are specified
- Operations are fast since stride is always 1
- All vector operations are provided for both serial and parallel cases
- For the parallel vector, MPI is used for global reductions

- These serve as good templates for creating a user-supplied vector structure around a user's own existing structures

SUNDIALS provides Fortran interfaces

- CVODE, IDA, and KINSOL
- Cross-language calls go in both directions:
- Fortran user code ←→ interfaces ←→ CVODE/KINSOL/IDA

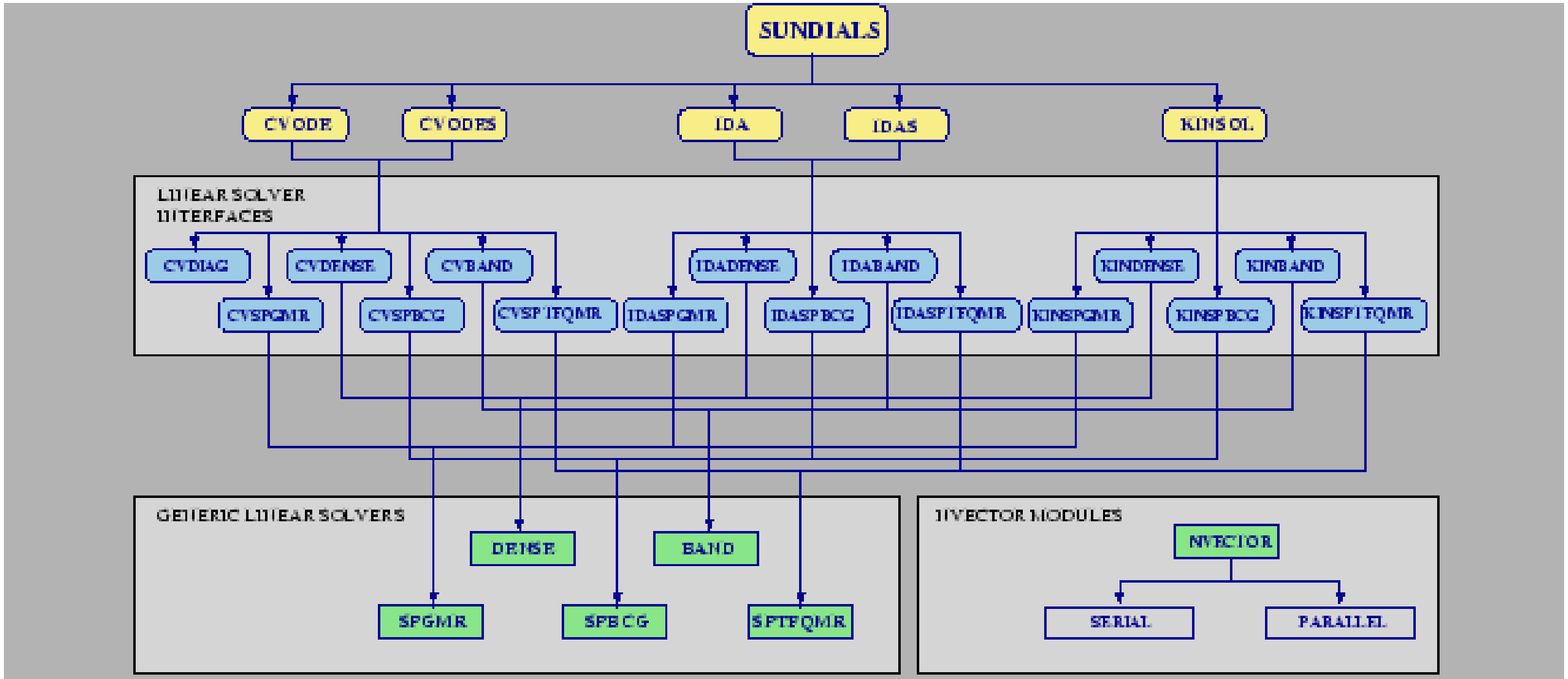
- Fortran main → interfaces to solver routines
- Solver routines → interface to user's problem-defining routine and preconditioning routines

- For portability, all user routines have fixed names
- Examples are provided

SUNDIALS provides Matlab interfaces

- CVODES, KINSOL, and IDAS
- The core of each interface is a single MEX file which interfaces to solver-specific user-callable functions
- Guiding design philosophy: make interfaces equally familiar to both SUNDIALS and Matlab users
  - all user-provided functions are Matlab m-files
  - all user-callable functions have the same names as the corresponding C functions
  - unlike the Matlab ODE solvers, we provide the more flexible SUNDIALS approach in which the 'Solve' function only returns the solution at the next requested output time.
- Includes complete documentation (including through the Matlab help system) and several examples

Structure of SUNDIALS



High-level diagram (note that none of the Lapack-based linear solver modules are represented.)

SUNDIALS code usage is similar across the suite

- Have a series of Set/Get routines to set options
- For CVODE with parallel vector implementation:

```
#include “cvode.h”
#include “cvode_spgmr.h”
#include “nvector_*.h”

y = N_VNew_*(n,…);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
flag = CVodeSet*(…);
flag = CVodeInit(cvmem,rhs,t0,y,…);
flag = CVSpgmr(cvmem,…);
for(tout = …) {
   flag = CVode(cvmem, …,y,…);  }

NV_Destroy(y);
CVodeFree(&cvmem);
```

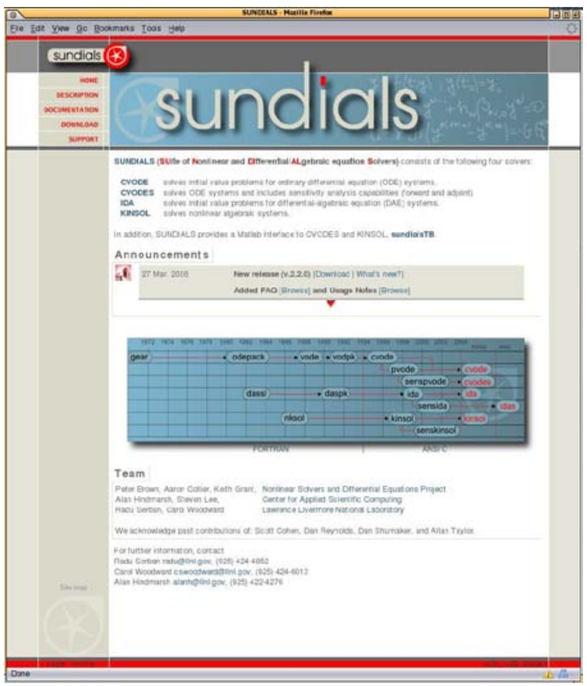
Availability

Open source BSD license

https://computation.llnl.gov/casc/sundials

Publications

https://computation.llnl.gov/casc/sundials/documentation/documentation.html



Web site:

Individual codes download

SUNDIALS suite download

User manuals

User group email list

The SUNDIALS Team:

Alan Hindmarsh, Radu Serban, and Carol Woodward