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 was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

LLNL-PRES-641695

Outline

- SUNDIALS Overview
- ODE integration
  - CVODE
  - ARKode
- DAE integration
  - IDA
- Sensitivity Analysis
- Nonlinear Systems
  - KINSOL
  - Fixed point solver
- SUNDIALS: usage, applications, and availability

SUite of Nonlinear and DIfferential-ALgebraic Solvers

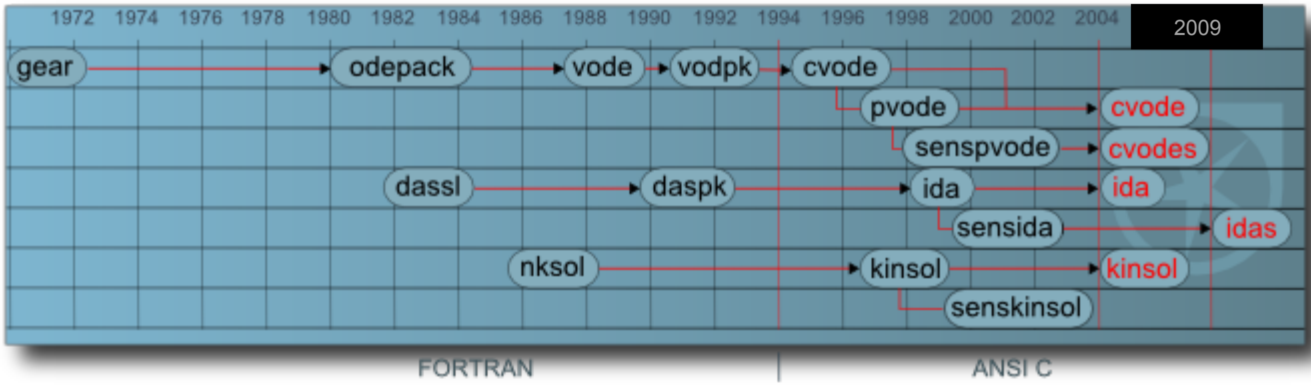
- Suite of time integrators and nonlinear solvers
  - ODE and DAE time integrators with forward and adjoint sensitivity capabilities, Newton-Krylov nonlinear solver
  - Written in C with interfaces to Fortran and Matlab
  - Designed to be incorporated into existing codes
  - Modular implementation: users can supply own data structures
    - Linear solvers / preconditioners
    - Vector structures – core data structure for all the codes
    - Supplied with serial and MPI parallel structures
- Freely available, released under BSD license

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

LLNL has a strong history of nonlinear solver and time integration research

SUNDIALS package evolved from innovation in methods and software

- Newton solvers evolved from the first Newton-Krylov method and code for PDEs
- ODE codes from odepack (> 200K downloads)
- DAE codes from DASSL



SUNDIALS 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
- CVODES and IDAS: sensitivity-capable (forward & adjoint)
- Adaptive time step and order selection minimize local truncation error
- KINSOL: Newton solver, F(u) = 0
  - Inexact and Modified (with dense solve) Newton
  - Linear systems by iterative or dense direct solvers
- Iterative linear Krylov solvers: GMRES, BiCGStab, TFQMR

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

- Variable order and variable step size Linear Multistep Methods

$$\sum_{j=0}^{K_1} \alpha_{n,j} y_{n-j} + \Delta t_n \sum_{j=0}^{K_2} \beta_{n,j} \dot{y}_{n-j} = 0$$

- Adams-Moulton (nonstiff); K_1 = 1, K_2 = k, k = 1,…,12
- Backward Differentiation Formulas [BDF] (stiff); K_1 = k, K_2 = 0, k = 1,…,5
- Rootfinding capability - finds roots of user-defined functions, $g_i(t,y)$
- The stiff solvers execute a predictor-corrector scheme:

Explicit predictor to give $y_{n(0)}$

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

Implicit corrector with $y_{n(0)}$ as initial iterate

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

Convergence and errors are measured against user-specified tolerances

- 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:

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

- Bound time integration error with:

$$\|y_n - y_{n(0)}\| < \frac{1}{6}$$

The 1/6 factor tries to account for estimation errors

Time steps 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, Δt’, 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

Nonlinear systems at each time step will require nonlinear solves

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

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

- Stiff systems: Newton iteration

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

ODE $\dot{y} = f(y)$

$M \approx I - \gamma \partial f / \partial y \qquad \gamma = \beta_0 \Delta t_n$

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

DAE $F(\dot{y}, y) = 0$

$M \approx \partial F / \partial y + \gamma \partial F / \partial \dot{y} \qquad \gamma = 1/\left(\beta_0 \Delta t_n\right)$

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

CVODE offers Newton, Newton-Krylov and function iteration as nonlinear solvers

- Non-stiff systems can use function iteration, or a fixed point solver
- Stiff systems generally require a Newton nonlinear solver
  - SUNDIALS provides dense solvers or hooks to LAPACK
    - Can reuse Jacobian over multiple steps -> modified Newton
  - Newton-Krylov solvers only require matrix-vector products
    - Approximations to the matrix-vector product are used,

$$J(y)v \approx \frac{G(y+\epsilon v) - G(y)}{\epsilon}$$

    - Matrix entries need never be formed

We are adding Runge-Kutta (RK) ODE time integrators to SUNDIALS via ARKode

- RK methods are multistage: allow high order accuracy without long step history (enabling spatial adaptivity)
- Implicit RK methods require multiple nonlinear solves per time step
- Additive RK methods apply a pair of explicit (ERK) and implicit (DIRK) methods to a split system, allowing accurate and stable approximations for multi-rate problems.
- Can decompose the system into “fast” and “slow” components to be treated with DIRK and ERK solvers
- ARKode provides 3rd to 5th order ARK, 2nd to 5th order DIRK and 2nd to 6th order ERK methods; also supports user-supplied methods.
- Applies advanced error estimators, adaptive time stepping, Newton and fixed-point iterative solvers
- ARKode will be released with SUNDIALS later this year

http://faculty.smu.edu/reynolds/arkode

ARKode solves $M\dot{y} = f_E(t,y) + f_I(t,y)$

- Variable step size additive Runge-Kutta Methods:

$$Mz_i = My_{n-1} + h_n \sum_{j=0}^{i-1} A^E_{i,j} f_E(t_{n-1} + c_j h_n, z_j) + h_n \sum_{j=0}^{i} A^I_{i,j} f_I(t_{n-1} + c_j h_n, \boxed{z_j}),$$

$$My_n = My_{n-1} + h_n \sum_{i=0}^{s} b_i \left(f_E(t_{n-1} + c_i h_n, z_i) + f_I(t_{n-1} + c_i h_n, z_i)\right),$$

$$M\tilde{y}_n = My_{n-1} + h_n \sum_{i=0}^{s} \tilde{b}_i \left(f_E(t_{n-1} + c_i h_n, z_i) + f_I(t_{n-1} + c_i h_n, z_i)\right).$$

- ERK methods use $A^I=0$; DIRK methods use $A^E=0$,
- z_i, $i = 1,\ldots,s$ are the inner stage solutions,
- y_n is the time-evolved solution, and
- $\tilde{y}_n$ is the embedded solution (used for error estimation),
- M may be the identity (ODEs) or a non-singular mass matrix (FEM).

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

- The general form of an IVP is given by

$$\boxed{\begin{aligned} F(t,\dot{x},x) &= 0 \\ x(t_0) &= x_0 \end{aligned}}$$

- 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

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
- Rootfinding capability - finds roots of user-defined functions, $g_i(t,y,y')$
- Nonlinear systems solved by Newton-Krylov method

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

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

Sensitivity Analysis Approaches

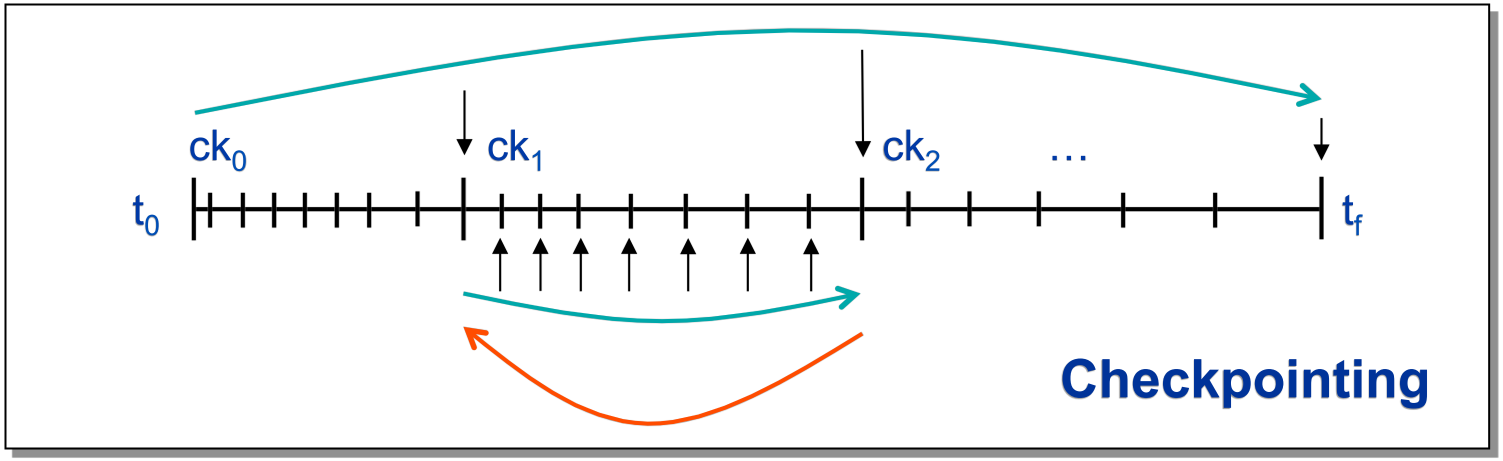
Parameter dependent system

$$\begin{cases} F(x,\dot{x},t,p) = 0 \\ x(0) = x_0(p) \end{cases}$$

FSA	**ASA**
$\begin{cases} F_{\dot{x}}\dot{s}_i + F_x s_i + F_{p_i} = 0 \\ s_i(0) = dx_0 / dp_i \end{cases}, \quad i = 1,\ldots,N_p$	$\begin{cases} (\lambda^* F_{\dot{x}})' - \lambda^* F_x = -g_x \\ \lambda^* F_{\dot{x}} x_p = \ldots \quad \text{at } t = T \end{cases}$
$g(t,x,p)$ $\dfrac{dg}{dp} = g_x s + g_p$	$G(x,p) = \int_0^T g(t,x,p)dt$ $\dfrac{dG}{dp} = \int_0^T (g_p - \lambda^* F_p)dt - \left(\lambda^* F_{\dot{x}} x_p\right)\Big\|_0^T$
Computational cost: $(1+N_p)N_x$ increases with N_p	Computational cost: $(1+N_g)N_x$ increases with N_g

Adjoint Sensitivity Analysis Implementation

- Solution of the forward problem is required for the adjoint problem → need predictable and compact storage of solution values for the solution of the adjoint system



- Cubic Hermite or variable-degree polynomial interpolation
- Simulations are reproducible from each checkpoint
- Force Jacobian evaluation at checkpoints to avoid storing it
- Store solution and first derivative
- Computational cost: 2 forward and 1 backward integrations

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
- Backtracking and line search options for robustness
- Dynamic linear tolerance selection

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

Fixed point and Picard iteration will be added to KINSOL in the next release

- Define an iterative scheme to solve F(h) = h - G(h) = 0 as,

```
Initialize $h^0$.
For $k = 0,1,\ldots,$ until $\|F(h^k)\| < \tau$
    Set $h^{k+1} = G(h^k)$.
end
```

- Picard iteration is a fixed point method formed from writing F as the difference of a linear, Lu, and a nonlinear, N(u), operator

$$F(u) = Lu - N(u); \quad L^{-1}N(u) = u - L^{-1}F(u) \equiv G(u)$$

$$u^{k+1} \approx u^k - L^{-1}F(u^k) = G(u^k)$$

Like Newton with L approximating J

- Fixed point iteration has a global but linear convergence theory
- Requires G to be a contraction $\|G(x) - G(y)\| \le \gamma \|x - y\|, \quad \gamma < 1$

KINSOL will have both Picard and fixed point iterations *with acceleration*

SUNDIALS provides many options for linear solvers

- Iterative Krylov 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*
- Two options require serial environments and some pre-defined structure to the data
  - Direct dense
  - Direct band
- Jacobian information (matrix or matrix-vector product) can be supplied by the user or estimated with finite difference quotients

We are developing a SUNDIALS interface to sparse direct solvers

- Requires serial vector kernel now – only for transfer of RHS information for Jacobian systems
- Will generalize to more generic vector interface in the future
- Matrix information is passed via new SUNDIALS sparse_matrix structure which utilizes a compressed sparse column format
- First instantiation is an interface to SuperLU_MT (multi-threaded version of SuperLU)
- Will also develop interfaces to KLU (serial) and possibly PARDISO (threaded)

Preconditioning is essential for large problems as Krylov methods can stagnate

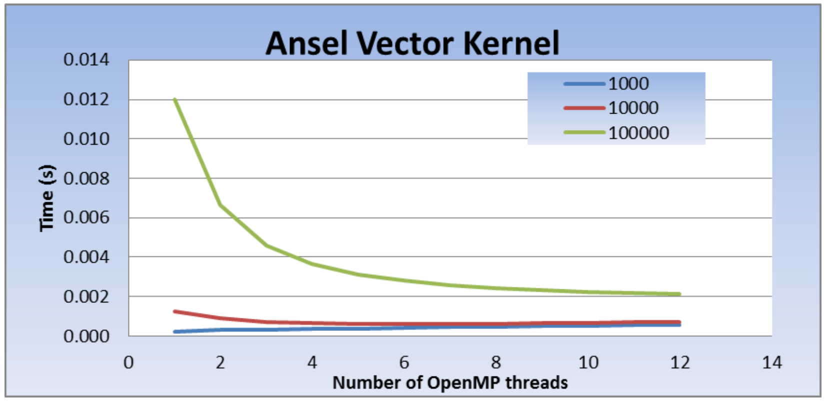
- Preconditioner P must approximate Newton matrix, yet be reasonably efficient to evaluate and solve.
- Typical P (for time-dep. ODE problem) is $I - \gamma\tilde{J}, \quad \tilde{J} \approx J$
- The user must supply two routines for treatment of P:
  - Setup: evaluate and preprocess P (infrequently)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

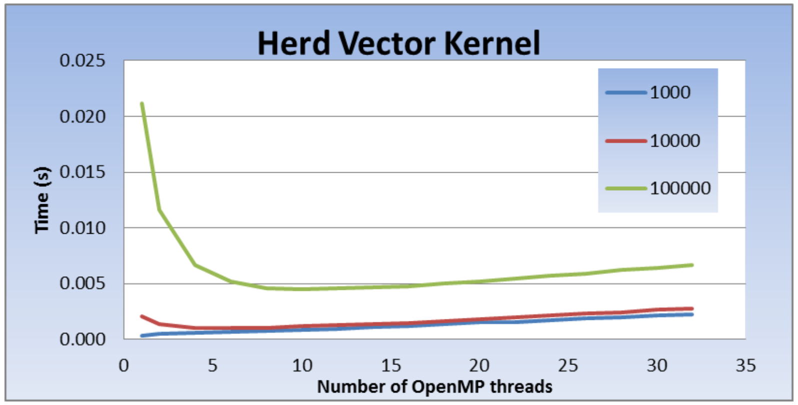
The SUNDIALS vector module is generic

- 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 operations provided for both serial and MPI parallel cases
- Can serve as templates for creating a user-supplied vector
- OpenMP and pThreads vector kernels coming soon. Preliminary performance tests indicate that 10K length required to see benefit





FASTMATH

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

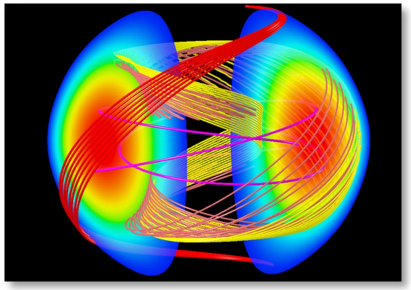
SUNDIALS code usage is similar across the suite

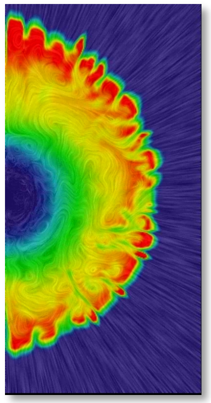
- Have a series of Set/Get routines to set options
- For CVODE with parallel vector implementation:

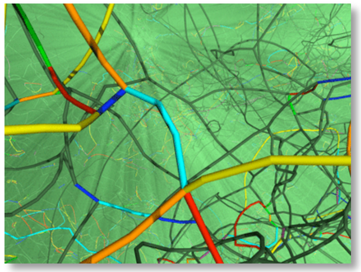
```
#include “cvode.h”
#include “cvode_spgmr.h”
#include “nvector_*.h”

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);
```

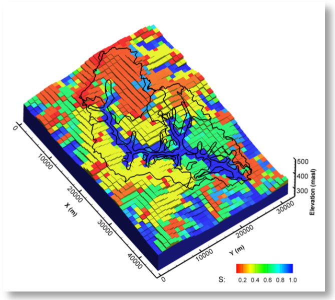
SUNDIALS has been used worldwide in applications from research and industry

- Power grid modeling (RTE France, ISU)
- Simulation of clutches and power train parts (LuK GmbH & Co.)
- Electrical and heat generation within battery cells (CD-adapco)
- 3D parallel fusion (SMU, U. York, LLNL)
- Implicit hydrodynamics in core collapse supernova (Stony Brook)
- Dislocation dynamics (LLNL)
- Sensitivity analysis of chemically reacting flows (Sandia)
- Large-scale subsurface flows (CO Mines, LLNL)
- Optimization in simulation of energy-producing algae (NREL)
- Micromagnetic simulations (U. Southampton)


Magnetic reconnection


Core collapse supernova


Dislocation dynamics



Subsurface flow

More than 3,000 downloads each year

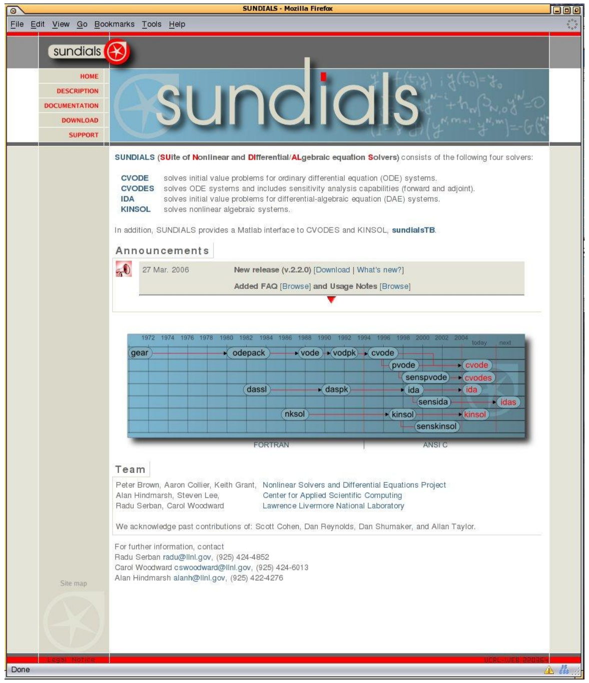
Availability

Open source BSD license

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

Publications

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



SUNDIALS - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
sundials
HOME
DESCRIPTION
DOCUMENTATION
DOWNLOAD
SUPPORT
sundials

SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) consists of the following four solvers:

CVODE	solves initial value problems for ordinary differential equation (ODE) systems.
CVODES	solves ODE systems and includes sensitivity analysis capabilities (forward and adjoint).
IDA	solves initial value problems for differential-algebraic equation (DAE) systems.
KINSOL	solves nonlinear algebraic systems.

In addition, SUNDIALS provides a Matlab interface to CVODES and KINSOL, **sundialsTB**.

Announcements

27 Mar. 2006 — **New release (v.2.2.0)** [Download | What's new?]

Added FAQ [Browse] **and Usage Notes** [Browse]

Team

Peter Brown, Aaron Collier, Keith Grant, Alan Hindmarsh, Steven Lee, Radu Serban, Carol Woodward — Nonlinear Solvers and Differential Equations Project, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

We acknowledge past contributions of: Scott Cohen, Dan Reynolds, Dan Shumaker, and Allan Taylor.

For further information, contact
Radu Serban radu@llnl.gov, (925) 424-4852
Carol Woodward cswoodward@llnl.gov, (925) 424-6013
Alan Hindmarsh alanh@llnl.gov, (925) 422-4276

Web site:

- Individual codes download
- SUNDIALS suite download
- User manuals
- User group email list

The SUNDIALS Team:

Alan Hindmarsh, Radu Serban, Dan Reynolds, Carol Woodward, and Eddy Banks