

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

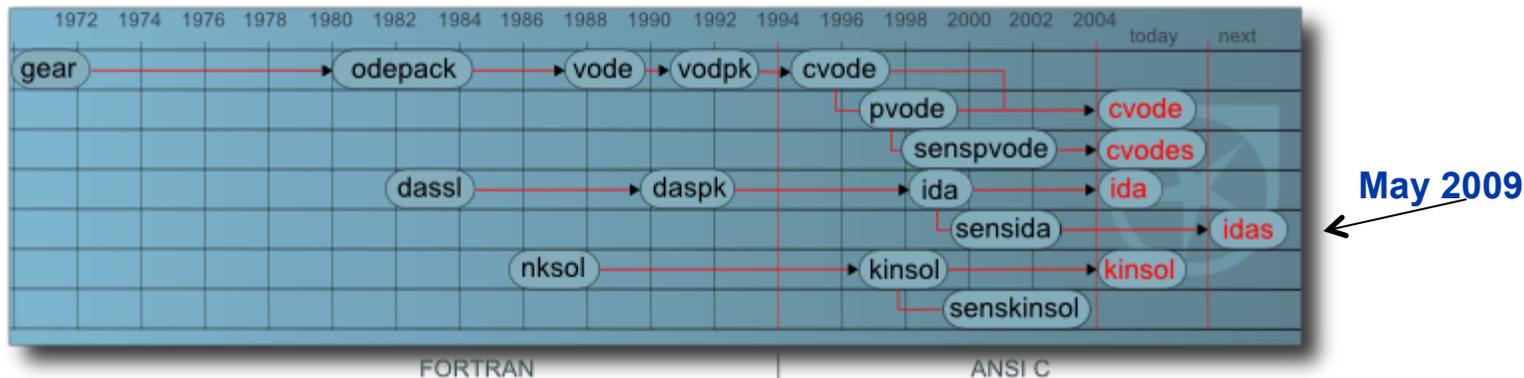
Outline

- SUNDIALS Overview
- ODE and DAE integration
 - Initial value problems
 - Implicit integration methods
- Nonlinear Systems
 - Newton's method and inexact Newton's method
 - Preconditioning
- Sensitivity analysis
 - Definitions, applications, methods
 - Forward sensitivity analysis
 - Adjoint sensitivity analysis
- 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



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



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

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

- 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



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

$$(t_1 - t_0) \min_j \Re(\lambda_j) \ll -1$$

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

Dalquist test equation: $\dot{\mathbf{y}} = \lambda \mathbf{y}, \mathbf{y}(0) = \mathbf{y}_0$

Exact solution: $\mathbf{y}(t_n) = \mathbf{y}_0 e^{\lambda t_n}$

Absolute stability requirement

$$|\mathbf{y}_n| \leq |\mathbf{y}_{n-1}|, \quad n = 1, 2, \dots$$

If $\text{Re}(\lambda) < 0$, then $|\mathbf{y}(t_n)|$ decays exponentially, and we cannot tolerate growth in \mathbf{y}_n

Region of absolute stability of an integrator written as:

$\mathbf{y}_n = R(z)\mathbf{y}_{n-1}$, with time step $z = h\lambda$

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

Forward and backward Euler show different stability restrictions

- Forward Euler: $\mathbf{y}_n = \mathbf{y}_{n-1} + h(\lambda \mathbf{y}_{n-1}) \Rightarrow R(z) = |1 + h\lambda|$

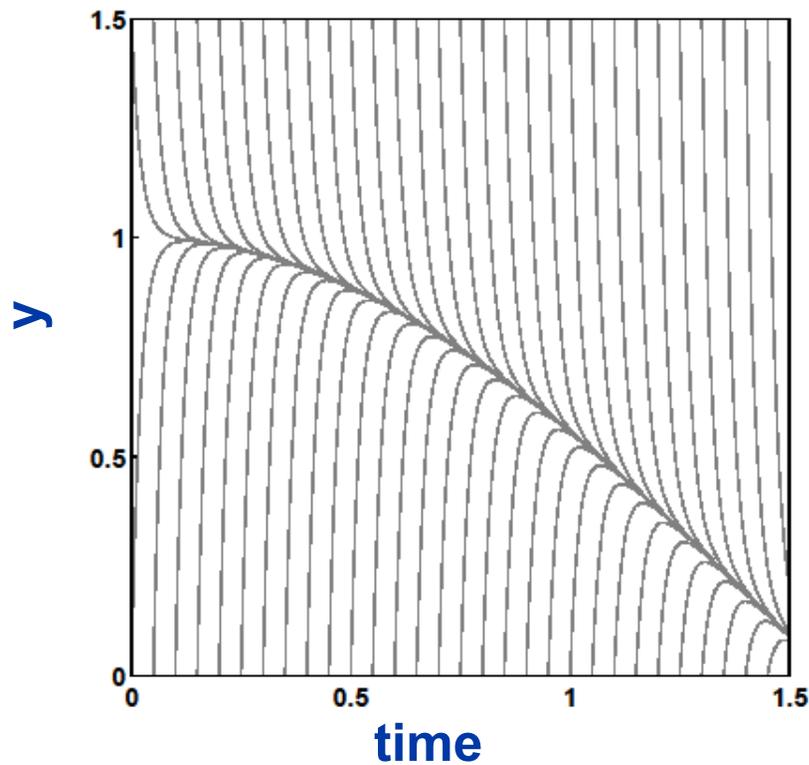
So, if $\lambda < 0$, FE has the step size restriction: $h \leq \frac{2}{|\lambda|}$

- Backward Euler: $\mathbf{y}_n = \mathbf{y}_{n-1} + h(\lambda \mathbf{y}_n) \Rightarrow R(z) = \left| \frac{1}{1 - h\lambda} \right|$

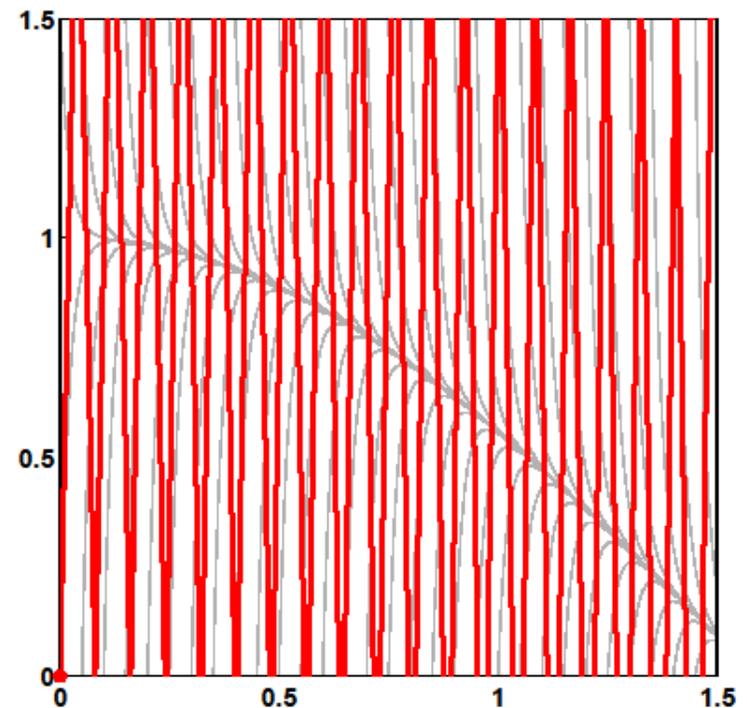
So, if $\lambda < 0$, BE has the step size restriction: $h > 0$

Curtiss and Hirschfelder example

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



Solution curves

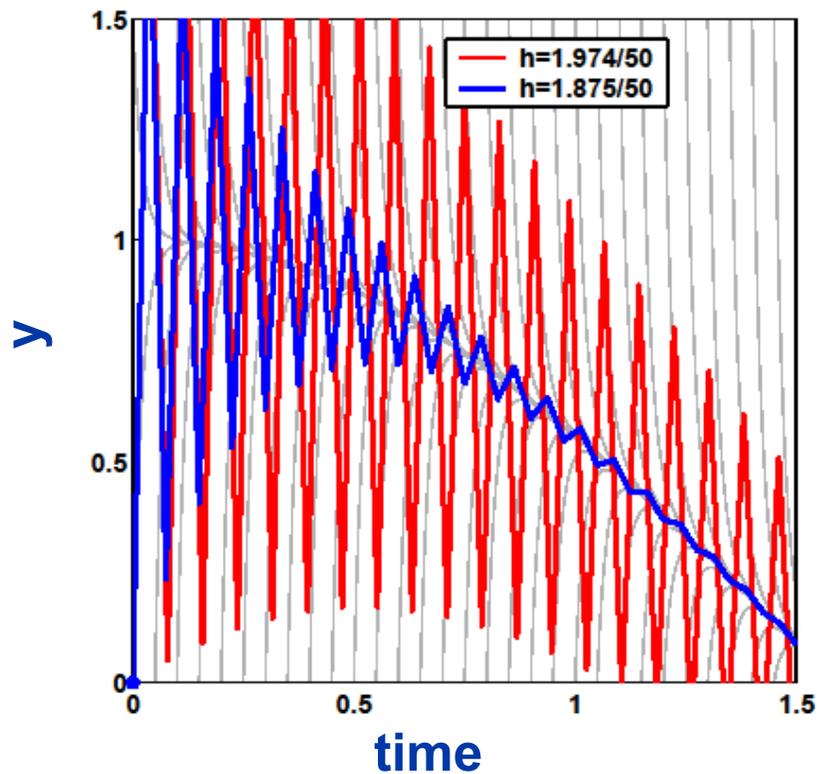


Forward Euler

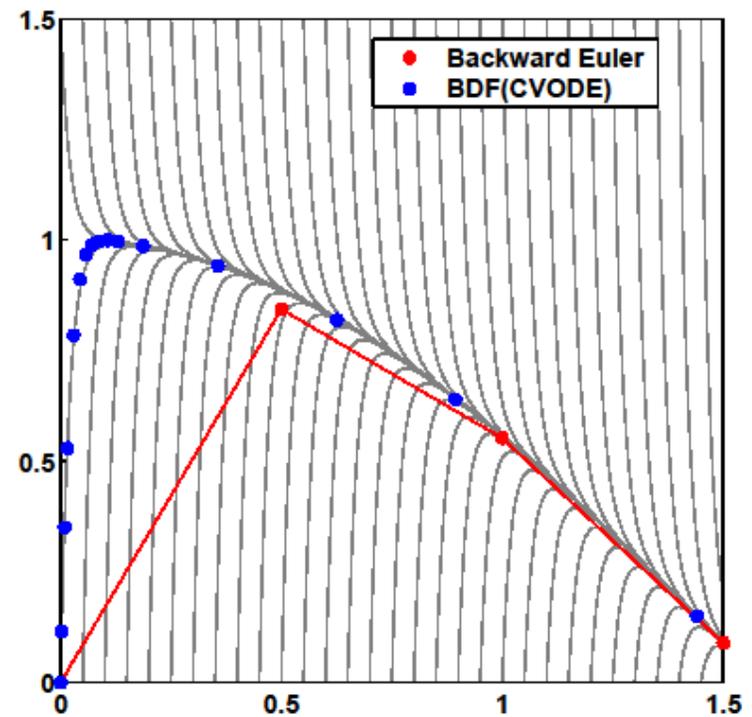
$h=2.01/50$

Curtiss and Hirschfelder 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:
$$\sum_{i=0}^{K_1} \alpha_{n,i} \mathbf{y}_{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{\mathbf{y}}_{n-i} = \mathbf{0}$$

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

- Nonlinear systems (BDF)

- ODE:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \mathbf{G}(\mathbf{y}_n) \equiv \mathbf{y}_n - \beta_0 h_n \mathbf{f}(t, \mathbf{y}_n) - \sum_{i=1}^k \alpha_{n,i} \mathbf{y}_{n-i} = \mathbf{0}$$

- DAE:

$$\mathbf{F}(\dot{\mathbf{y}}, \mathbf{y}) = \mathbf{0} \quad \mathbf{G}(\mathbf{y}_n) \equiv \mathbf{F}\left(t, (\beta_0 h_n)^{-1} \sum_{i=1}^k \alpha_{n,i} \mathbf{y}_{n-i}, \mathbf{y}_n\right) = \mathbf{0}$$



Stability is very restricted for higher orders of BDF methods

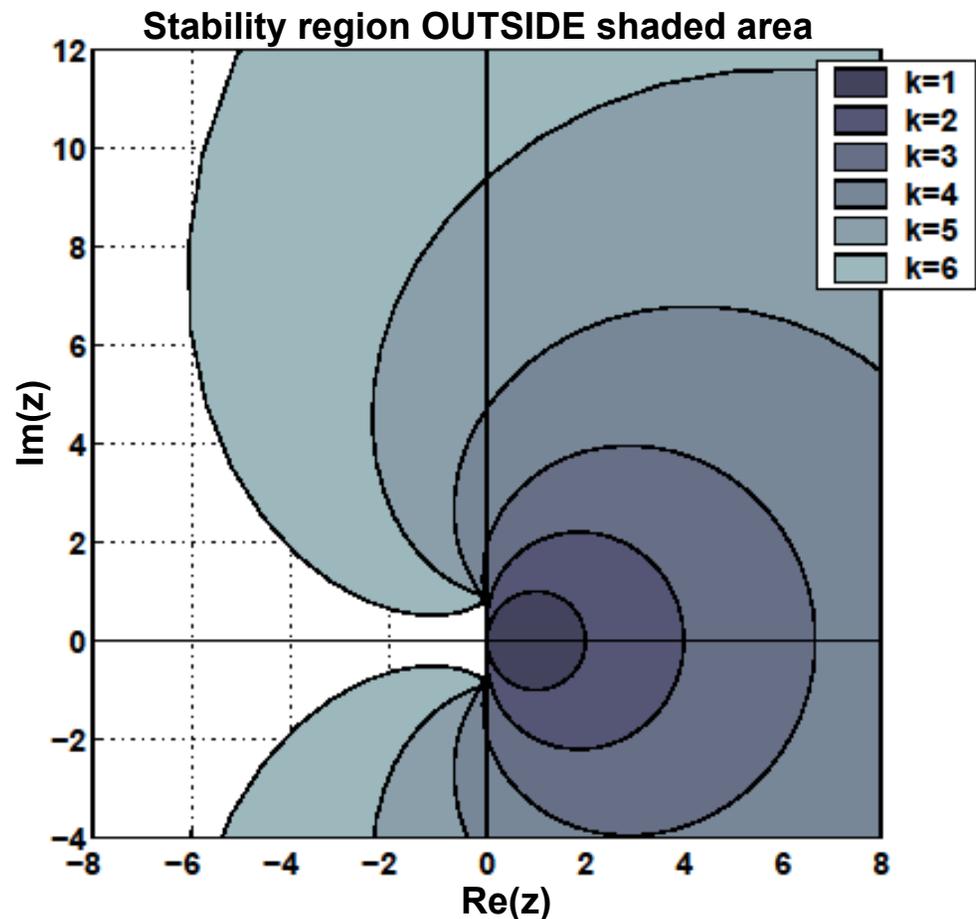
$$\mathbf{y}_n - \beta_0 \mathbf{h}_n \dot{\mathbf{y}}_n = \sum_{i=1}^k \alpha_{n,i} \mathbf{y}_{n-i}$$

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

- 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 $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$
 - applies an explicit predictor to give $\mathbf{y}_{n(0)}$

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

- applies an implicit corrector with $\mathbf{y}_{n(0)}$ as the initial guess

$$\mathbf{y}_n = \sum_{j=1}^q \alpha_j \mathbf{y}_{n-j} + \Delta t \beta_0 \mathbf{f}_n(\mathbf{y}_n)$$

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, $\Delta t'$, 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 disproportionately 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:

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

- Bound time integration error with:

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

The $1/6$ factor tries to account for estimation errors

Nonlinear system will require nonlinear solves

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

$$\mathbf{y}_{n(m+1)} = \beta_0 \mathbf{h}_n \mathbf{f}(\mathbf{y}_{n(m)}) + \sum_{i=1}^q \alpha_{n,i} \mathbf{y}_{n-i}$$

- Stiff systems: Newton iteration

$$\mathbf{M}(\mathbf{y}_{n(m+1)} - \mathbf{y}_{n(m)}) = -\mathbf{G}(\mathbf{y}_{n(m)})$$

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

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: $K(\mathbf{J}, \mathbf{r}) = \{\mathbf{r}, \mathbf{J}\mathbf{r}, \mathbf{J}^2\mathbf{r}, \dots\}$
- Only require matrix-vector products
- Difference approximations to the matrix-vector product are used,

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

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



CVODE and IDA are equipped with a rootfinding capability

- Finds roots of user-defined functions, $g_i(t, y)$ or $g_i(t, y, y')$
- Roots are found by looking at sign changes, so only roots of odd multiplicity are found
- Checks each time interval for sign change
 - When sign changes are found, apply a modified secant method
 - Tight tolerance: $\tau = 100 * U * (|t_n| + |\Delta t|)$; $U =$ unit roundoff
- Checks for $g_i(t, y) = 0$ every time g_i is evaluated; if $g_i(t, y) = 0$, then root is reported
- If $g_i(t^*, y) = 0$ for some t^*
 - $g_i(t^* + \delta, y)$ is computed for some small δ in direction of integration
 - Integration stops if any $g_i(t + \delta, y) = 0$
 - Ensures values of g_i are nonzero at some past value of t , beyond which a search for roots is done

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

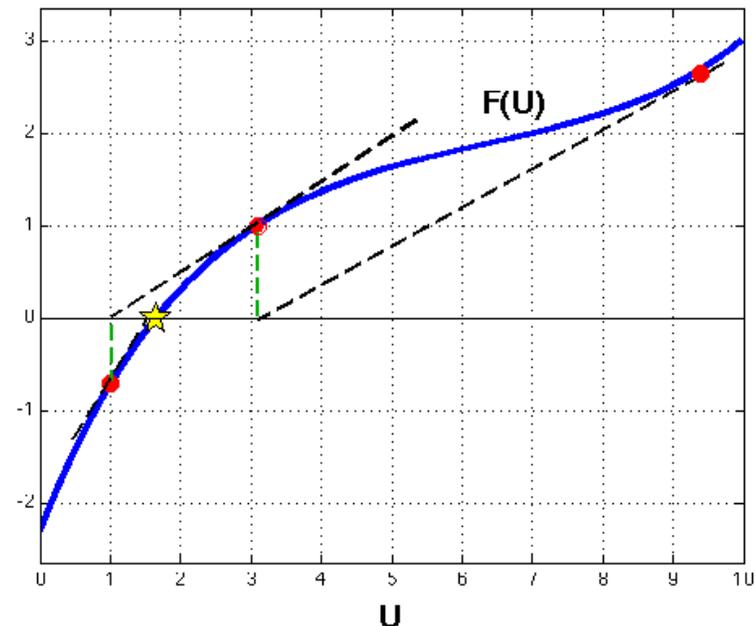
1. Starting with x^0 , want x^* such that $F(x^*) = 0$
2. Repeat for each k until $\|F(x^{k+1})\| \leq \text{tol}$

a. Solve (approximately)

$$J(x^k)s^k = -F(x^k)$$

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

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



courtesy of D. Reynolds (SMU)

Linear stopping tolerances must be chosen to prevent “oversolves”

The linear system is solved to a given tolerance:

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

- 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 $\eta^k = \frac{\|\mathbf{F}^k\| - \|\mathbf{F}^{k-1} - \mathbf{J}^{k-1}\mathbf{s}^{k-1}\|}{\|\mathbf{F}^{k-1}\|}$
 - Choice 2 $\eta^k = 0.9 \left(\frac{\|\mathbf{F}^{(k)}\|}{\|\mathbf{F}^{(k-1)}\|} \right)^2$
- 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

$$\|\mathbf{x}^{k+1} - \mathbf{x}^*\| \leq C \|\mathbf{x}^k - \mathbf{x}^*\|^2$$

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

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

- 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 $I - \gamma \tilde{J}$, $\tilde{J} \approx J$
- 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
 - Can use *hypre* or PetSc or ...
- 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



Sensitivity Analysis Approaches

Parameter dependent system

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

FSA

ASA

$$\begin{cases} F_{\dot{x}} \dot{s}_i + F_x s_i + F_{p_i} = 0 \\ s_i(0) = dx_0/dp_i \end{cases}, \quad i = 1, \dots, N_p$$

$$\begin{cases} (\lambda^* F_{\dot{x}})' - \lambda^* F_x = -g_x \\ \lambda^* F_{\dot{x}} x_p = \dots \quad \text{at } t = T \end{cases}$$

$$g(t, x, p)$$

$$G(x, p) = \int_0^T g(t, x, p) dt$$

$$\frac{dg}{dp} = g_x s + g_p$$

$$\frac{dG}{dp} = \int_0^T (g_p - \lambda^* F_p) dt - (\lambda^* F_{\dot{x}} x_p)_0^T$$

Computational cost:

$(1+N_p)N_x$ increases with N_p

Computational cost:

$(1+N_g)N_x$ increases with N_g



FSA - Methods

- **Staggered Direct Method:** On each time step, converge Newton iteration for state variables, then solve linear sensitivity system
 - Requires formation and storage of Jacobian matrices, Not matrix-free, Errors in finite-difference Jacobians lead to errors in sensitivities
- ✓ **Simultaneous Corrector Method:** On each time step, solve the nonlinear system simultaneously for solution and sensitivity variables
 - Block-diagonal approximation of the combined system Jacobian, Requires formation of sensitivity R.H.S. at every iteration
- ✓ **Staggered Corrector Method:** On each time step, converge Newton for state variables, then iterate to solve sensitivity system
 - With Krylov



FSA – Generation of the sensitivity system

- Analytical
- Automatic differentiation
 - ADIFOR, ADIC, ADOLC
 - complex-step derivatives
- Directional derivative approximation

CVODES case

$$\dot{x} = f(t, x, p)$$

$$\dot{s}_i = \frac{\partial f}{\partial x} s_i + \frac{\partial f}{\partial p_i}$$

$$\frac{\partial f}{\partial x} s_i \approx \frac{f(t, x + \sigma_x s_i, p) - f(t, x - \sigma_x s_i, p)}{2\sigma_x}$$

$$\frac{\partial f}{\partial p_i} \approx \frac{f(t, x, p + \sigma_i e_i) - f(t, x, p - \sigma_i e_i)}{2\sigma_i}$$

$$\begin{cases} \sigma_i = |\bar{p}_i| \sqrt{\max(\text{rtol}, \varepsilon)} \\ \sigma_x = \frac{1}{\max(1/\sigma_i, \|s_i\|_{WRMS}/|\bar{p}_i|)} \end{cases}$$

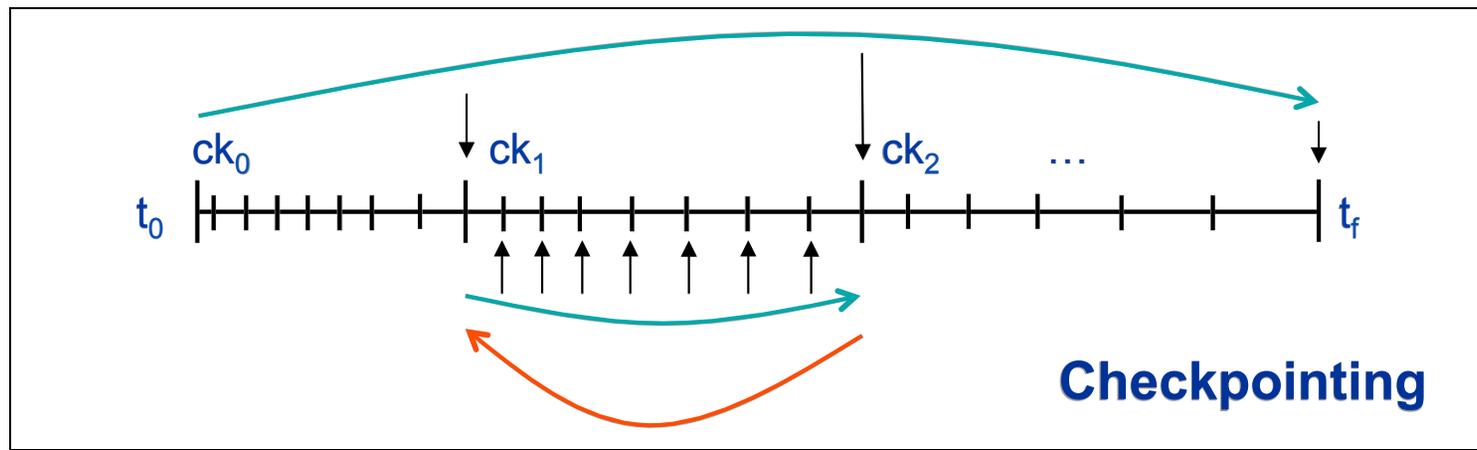
or

$$\frac{\partial f}{\partial x} s_i + \frac{\partial f}{\partial p_i} \approx \frac{f(t, x + \sigma s_i, p + \sigma e_i) - f(t, x - \sigma s_i, p - \sigma e_i)}{2\sigma} \quad \sigma = \min(\sigma_i, \sigma_x)$$



ASA – 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

ASA – Generation of the sensitivity system

- Analytical
 - Tedious
 - For PDEs: in general, adjoint and discretization operators do NOT commute
- Automatic differentiation
 - Certainly the most attractive alternative
 - Reverse AD tools not as mature as forward AD tools
- Finite difference approximation
 - NOT an option (computational cost equivalent to FSA!)



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 \leftrightarrow interfaces \leftrightarrow CVODE/KINSOL/IDA

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

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

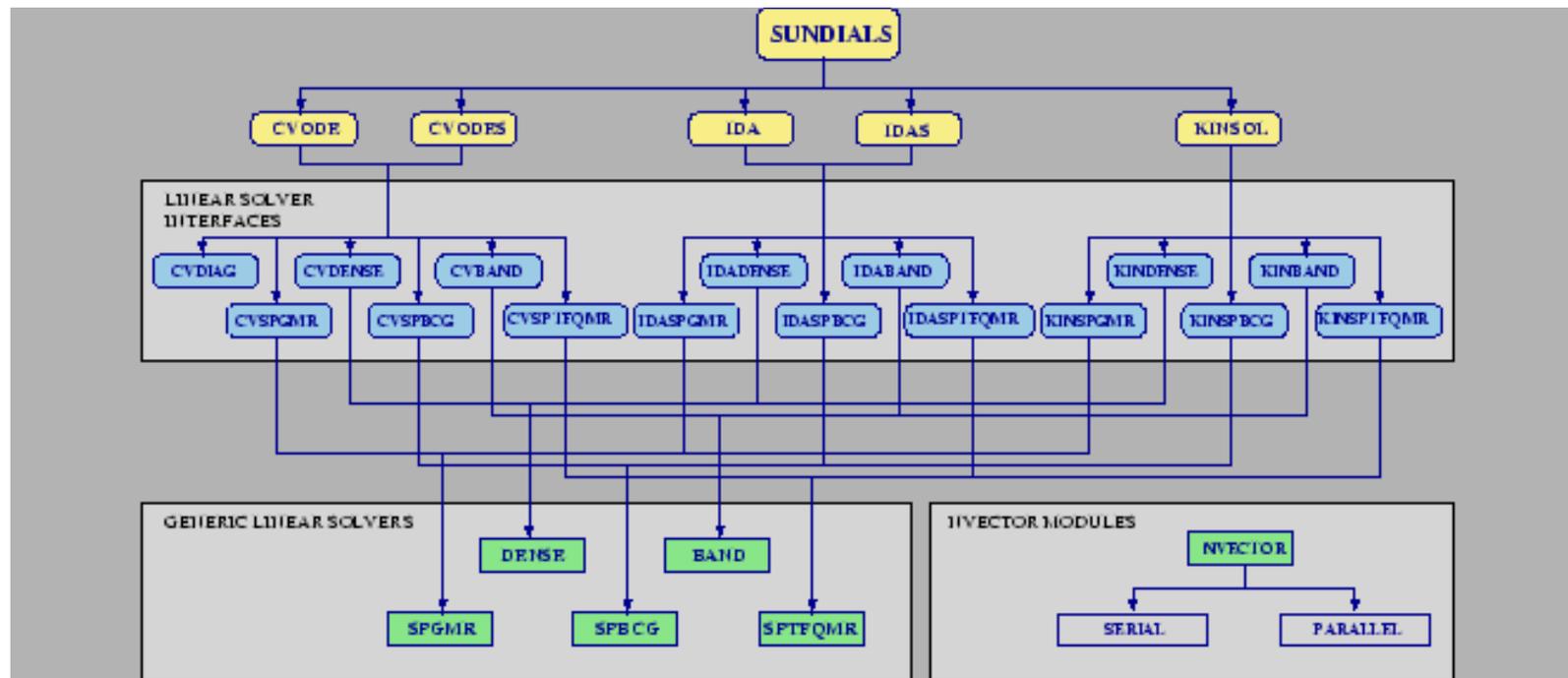


SUNDIALS provides Matlab interfaces

- 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);
```



Forward Sensitivity Analysis in SUNDIALS

User main routine

Specification of problem parameters

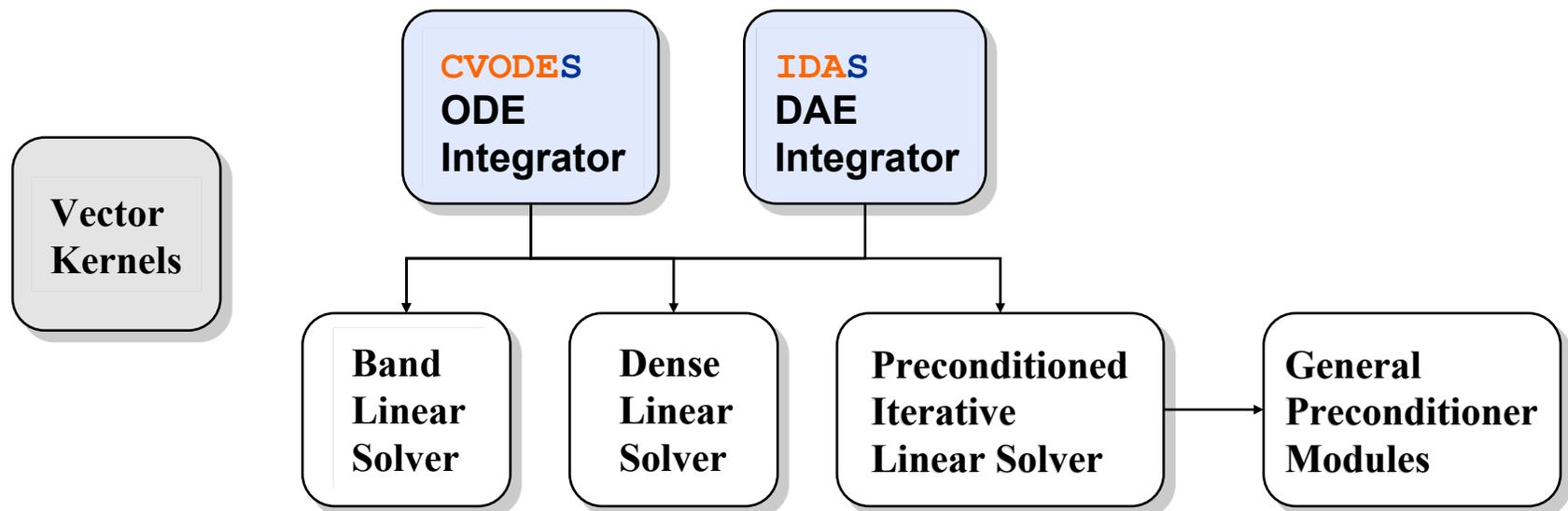
Activation of sensitivity computation

User problem-defining function

User preconditioner function

Options

- sensitivity approach (simultaneous or staggered)
- sensitivity residuals: analytical, FD(DQ), AD, CS
- error control on sensitivity variables
- user-defined tolerances for sensitivity variables



Forward Sensitivity Analysis in SUNDIALS

```
#include "cvodes.h"
#include "cvodes_spgmr.h"
#include "nvector_*.h"

y = NV_New*(n,...);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
flag = CVodeSet*(...);
flag = CVodeMalloc(cvmem,rhs,t0,y,...);
flag = CVSpgmr(cvmem,...);
yS = NV_NewVectorArray_*(Ns,...);
flag = CVodeSetSens*(...);
flag = CVodeSensMalloc(cvmem,...,yS);
for(tout = ...) {
    flag = CVode(cvmem, ...,y,...);
    flag = CVodeGetSens(cvmem,t,yS);
}
NV_Destroy(y);
NV_DestroyVectorArray(yS,Ns);
CVodeFree(&cvmem);
```

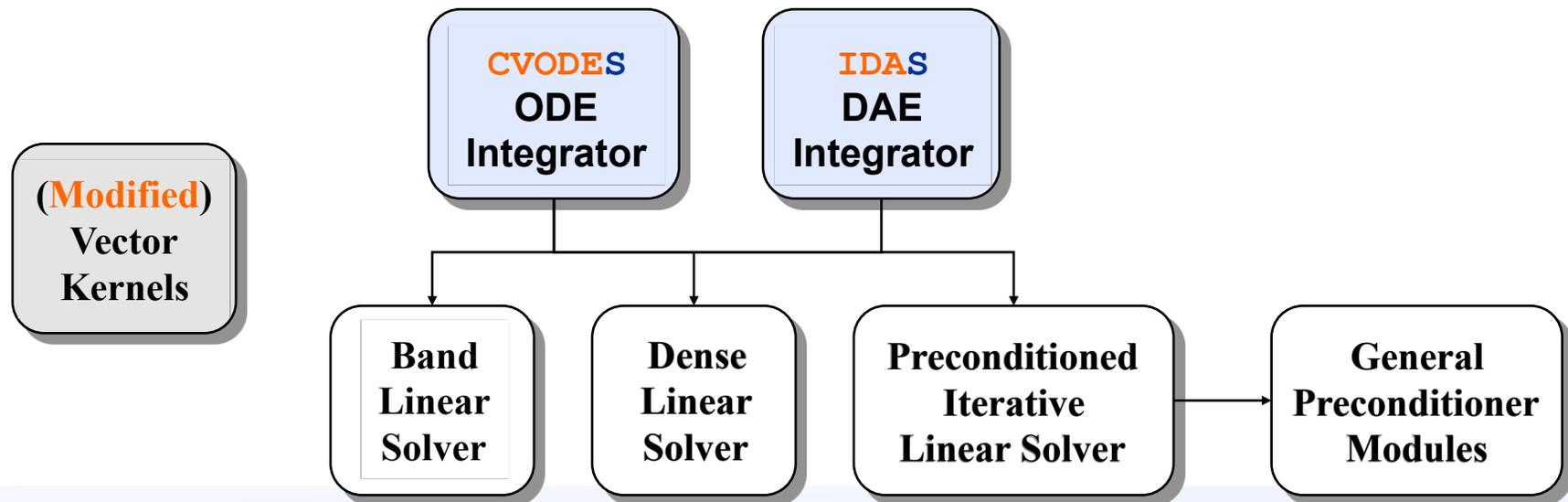


Adjoint Sensitivity Analysis in SUNDIALS

User main routine
Activation of sensitivity computation
User problem-defining function
User reverse function
User preconditioner function
User reverse preconditioner function

Implementation

- check point approach; total cost is 2 forward solutions + 1 backward solution
- integrate any system backwards in time
- may require modifications to some user-defined vector kernels



Adjoint Sensitivity Analysis in SUNDIALS

```
#include "cvodes.h"
#include "cvodea.h"
#include "cvodes_spgmr.h"
#include "nvector_*.h"

y = N_VNew_(n,...);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
CVodeSet*(...); CVodeMalloc(...); CVSpgmr(...);

cvadj = CVadjMalloc(cvmem,STEPS);
flag = CVodeF(cvadj,...,&nchk);
yB = N_VNew_(nB,...);
CVodeSet*B(...); CVodeMallocB(...); CVSpgmrB(...);
for(tout = ...) {
    flag = CVodeB(cvadj, ...,yB,...);
}
NV_Destroy(y);
NV_Destroy(yB);
CVodeFree(&cvmem);
CVadjFree(&cvadj);
```



Upcoming additions to SUNDIALS

- **Accelerated fixed point nonlinear solver**
 - **Anderson acceleration: preliminary version in place, now refining**
- **Vector kernels supporting multi/many core architectures**
- **Time integrators for multi-rate problems: ARKODE from Dan Reynolds (SMU)**



The ARKODE Solver

We are currently working on a new solver that will extend SUNDIALS to support multi-rate systems of ordinary differential equations.

Like CVODE, this solver applies advanced error estimators and adaptive time stepping to efficiently evolve systems of ODEs

Unlike CVODE, this solver allows a user to decompose the ODE system into “fast” and “slow” components, applying implicit and explicit solvers to these components, respectively



The ARKODE Design

ARKODE solvers are based on additive Runge-Kutta (ARK) methods

- **Comprised of a pair of explicit and diagonally-implicit Runge-Kutta methods**
- **ERK and DIRK methods derived in coordination, to guarantee accuracy of each method as well as their coupling**

Built-in coefficients providing from 3rd to 5th order accurate methods

May also be run in purely explicit or purely implicit mode

One step (multi stage) solvers that work naturally with spatially-adaptive PDE simulations

The ARKODE Algorithms

Data structures and iterative solvers used within ARKODE match the rest of SUNDIALS

General vector-based implementation, for serial, parallel, or user-defined data structures

Inexact Newton methods, with preconditioned Krylov linear solvers for parallel and serial problems

Modified Newton methods with LAPACK linear solvers for serial problems

For more information: <http://faculty.smu.edu/reynolds/arkode>



Applications of SUNDIALS

- CVODE and KINSOL are being used in parallel fusion simulations at SMU
- KINSOL is being used to solve for implicit hydrodynamics in core collapse supernova simulations at SUNY-Stony Brook
- Parallel CVODE is being used in a 3D tokamak turbulence model (BOUT++) in LLNL's Magnetic Fusion Energy Division.
- KINSOL with a HYPRE multigrid preconditioner is being applied to solve a nonlinear Richards' equation for pressure in porous media flows.
- CVODE, KINSOL, IDA, with MG preconditioner, are being used to solve 3D neutral particle transport problems in CASC.
- ...



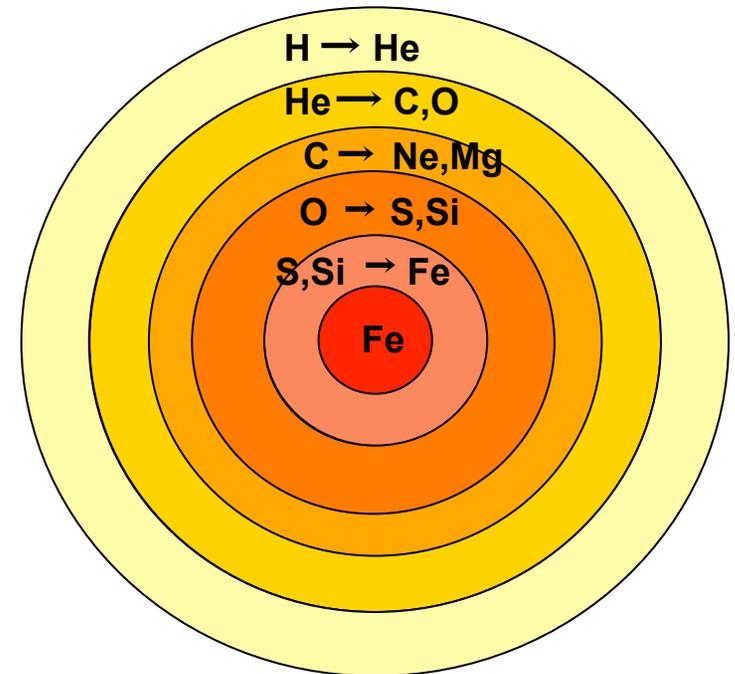
Applications with sensitivity analysis

- CVODES used for sensitivity analysis of chemically reacting flows (SciDAC collaboration with Sandia Livermore).
- CVODES used for sensitivity analysis of radiation transport (diffusion approximation).
- KINSOL+CVODES used for inversion of large-scale time-dependent PDEs (atmospheric releases).
- ...



Stellar collapse simulations must deliver high accuracy over long time periods

- Evolve by fusion burning, iron core collapse, blow up, and neutrino radiation cooling
- Stellar collapse, supernova explosion, and neutron star formation is one of the most energetic, *but poorly understood*, phenomena in astrophysics
- Models: of radiation transport, hydrodynamics & self-gravity
- Neutrinos carry off ~99% of energy; 1% is observed (have to model this)



Onionskin-like structure

$$\Delta t_{CFL,core}(10^{-7} s) < \text{desired step} < \Delta t_{CFL,shock}(\sim 10^{-4} s) < t_{cooling}(10s)$$

We model the hydrodynamics with a Lagrangian formulation and pose the discrete system as a DAE

Lagrangian Eqns
(fluid flow)

+

Self-gravity

+

Conservation form

=>

Stellar collapse hydrodynamics model

$$\frac{V(U_n) - V(U_{n-1})}{\Delta t} = \theta F(U_n) + (1 - \theta) F(U_{n-1}),$$

$$G(U_n) = 0, \quad \text{self - gravity}$$

Implicit approach
works for Eulerian
forms as well

We apply Newton-Krylov to the space-
discretized form of this DAE system
using the SUNDIALS KINSOL package

w/. D. Reynolds (SMU) & D. Swesty (SUNY SB)



We have customized the Newton-Krylov method for the stellar collapse simulation

- Non-differentiabilities in discretization schemes result in convergence problems with finite differences
 - Limiters and artificial viscosity increase stability for shocks but result in discontinuities in the Jacobian
 - Freeze parameters in Newton iteration to smooth Jacobian
 - Approximate Jacobian entries at the start of each iteration

$$[J_H(U^k)v]_{ij} \approx \frac{H_i(U^k + \sigma e_j) - H_i(U^k)}{\sigma}$$

- Form the preconditioner by extracting only the spatially local entries that couple the variables within a given cell
 - Place them in a block diagonal matrix
 - Solve each block exactly

We had to take care in scaling and constraints

- Unknowns and equations at differing scales can cause difficulties with stopping criteria

- Use a weighted RMS norm for convergence:

$$\|H(U)\| = \left(\|DH(U)\|_2^2 / N \right), \quad D = \text{diag}(d_1^{-1}, \dots, d_N^{-1})$$

- D gives the typical equation magnitude

- Certain variables have positivity constraints which can be violated in Newton updates or differencing

- Apply a log transform for density, temperature, and radius

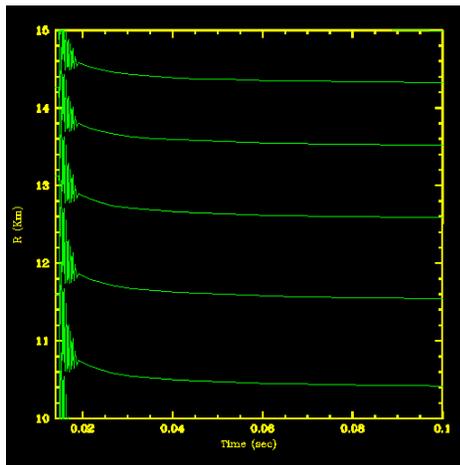
$$\tilde{\rho} = \log(\rho)$$

- Increases nonlinearity but, in practice, this adds only up to 1 Newton iteration

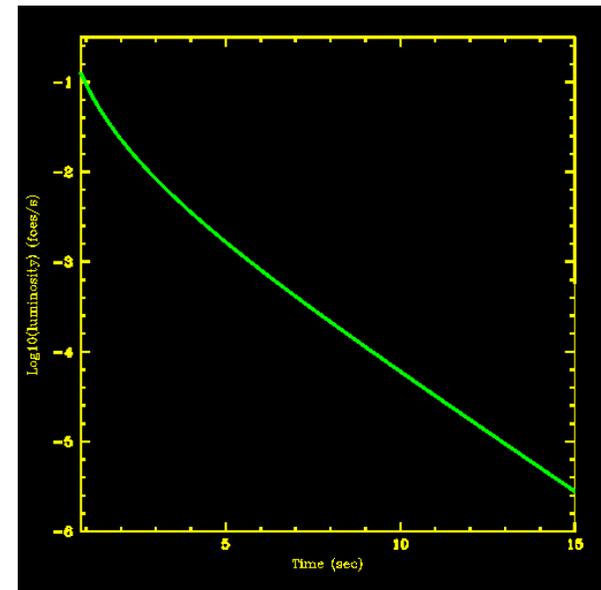
Implicit approach enabled the first radiation-hydrodynamic modeling of the entire proto-neutron star cooling

- Initial central density: $5 \times 10^{14} \text{ g/cm}^3$
- Initial radiation distribution contributes $\frac{1}{2}$ of pressure support in the star center and diminishes radially
- Star contracts as neutrinos diffuse out
- *NK for neutrino MGFLD as well as hydrodynamics*

Mass contours show contraction



Neutrino cooling signal decays over 15s timescale



Explicit CFL restriction:

$\Delta t \sim 2.5 \times 10^{-8} \text{ s} \Rightarrow 10^9 \text{ steps}$

Implicit used $\Delta t \sim 2.5 \times 10^{-5} \text{ s}$

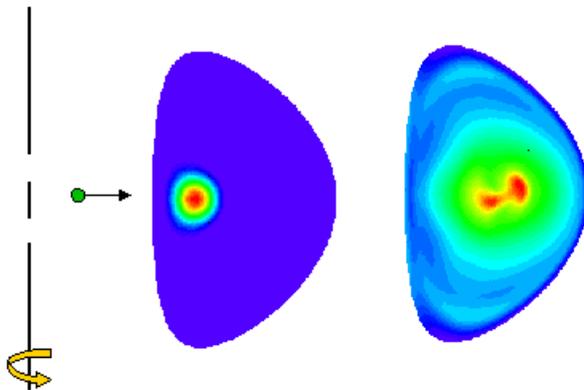
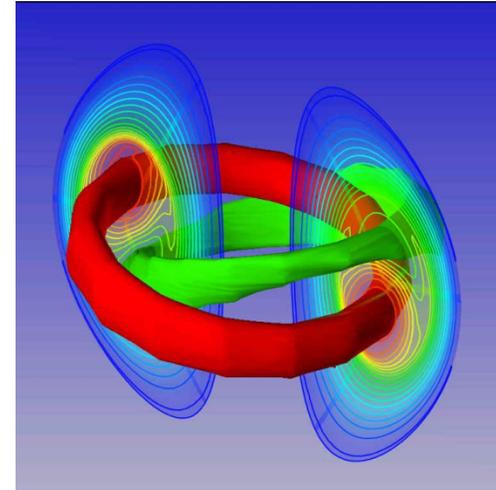
0.1% of the number of explicit steps!



Newton-Krylov methods have been applied to resistive magnetohydrodynamics simulations

Magnetic Reconnection

- Breaking & reconnecting oppositely-directed field lines in a plasma
- Instabilities replace hot plasma core with cool plasma, halting fusion
- Sweet-Parker reconnection is $\sim 10^5$ times slower than fastest waves



Pellet Injection Fueling

- Shoot frozen hydrogen pellets into the plasma at high velocity ~ 500 m/s
- Want location of mass deposition
- Pellet motion is $\sim 10^4$ times slower than fastest waves

Both applications require *large-scale, long-time* simulations

We use a conservative form of the single-fluid resistive magnetohydrodynamics equations

Euler eqns (fluid flow) + Low-freq. Maxwell eqns (electromagnetic fields) + Conservation form

=>

Resistive magnetohydrodynamics

$$\partial_t U + \nabla \cdot F(U) - \nabla \cdot F_v(U) = 0$$

$$U = (\rho, \rho v, B, e)^T$$

$F(U)$ = hyperbolic fluxes

$$f(U) \equiv \nabla \cdot F_v(U) - \nabla \cdot F(U)$$

$F_v(U)$ = diffusive fluxes

We view this model as a system of ODEs: $\partial_t U = f(U)$

High order BDF time integrator, CVODE, with Newton-Krylov from SUNDIALS Package

w/ *D. Reynolds (SMU) & R. Samtaney (KAUST)*

Lawrence Livermore National Laboratory



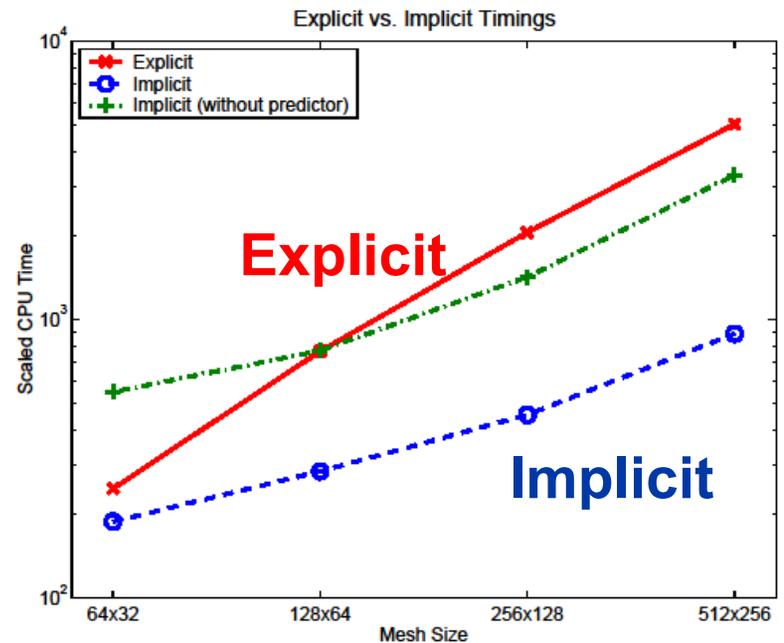
The implicit approach gave faster solutions than the original explicit approach on our reconnection problem

- GEM magnetic reconnection challenge (Brin et al., 2001)
- 2D, characteristic velocity is the Alfvén speed
- Small magnetic resistivity and fluid viscosity
- No preconditioning
- $O(\Delta t^5)$ implicit method
- $O(\Delta t^4)$ explicit Runge-Kutta

Implicit is almost 6x faster than explicit

Preconditioning further improved these results

Reynolds, Samtaney, & W., JCP, 2006



Problem Size	Avg Step Size	
	Explicit	Implicit
64x32	0.029	0.067
128x64	0.015	0.057
256x128	0.008	0.056
512x256	0.004	0.059

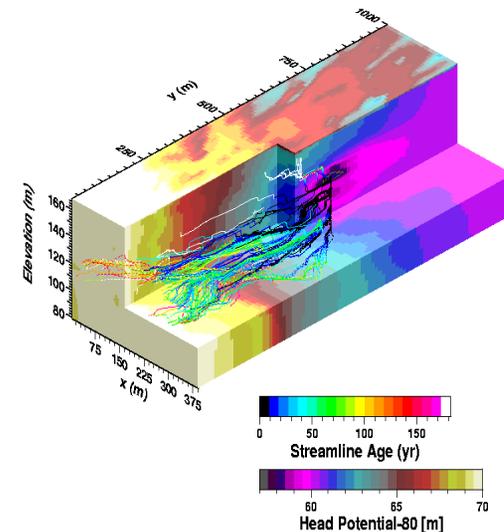


KINSOL enabled modeling variably saturated subsurface flow in numerous contexts

- The Newton-Krylov method in **KINSOL** from SUNDIALS provided the main solver engine for the *PARFLOW* variably saturated subsurface flow solver
- *hypre* structured multigrid preconditioner
- Symmetric approximation to Jacobian for preconditioning
- Line search globalization
- Dynamic linear tolerances

Jones and W., Adv. Water Res., 2001

$$\frac{\partial \theta(p)}{\partial t} - \nabla \cdot (Kk_r(p)(\nabla p - \rho g \nabla z)) = q$$



Variably saturated PARFLOW is used in large-scale, parallel models of many DOE sites

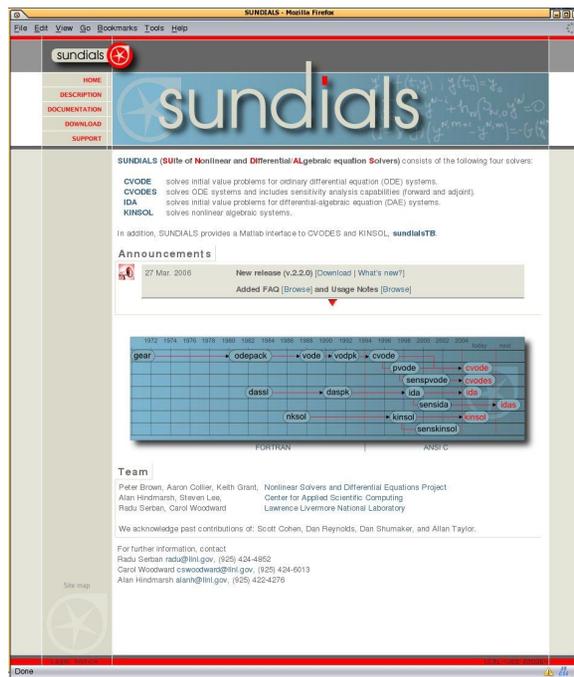
Availability

Open source BSD license

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

Publications

<https://computation.llnl.gov/casc/nsde>



Web site:

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

The SUNDIALS Team:

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

